COMMUNICATION BETWEEN UNRELATED COMPUTERS

A STUDY OF SOFTWARE DESIGN PROBLEMS

I.H. GOULD

INSTITUTE OF COMPUTER SCIENCE
LIBRARY
NOT TO BE TAKEN AWAY

ABSTRACT

The development of a communication system between computers whose operating systems, as supplied by the manufacturers, are not designed to support such communication, and where the operating system of the larger computer in the partnership cannot be altered, poses a number of problems. After an historical introduction in which the theme of this thesis is set in the context of a research project at the Institute of Computer Science of which it formed a part, the characteristics of four communication systems for which software has been developed are described.

The software developed for the satellite computer common to all four systems is next briefly discussed, with particular emphasis on the problems encountered that seem peculiar to systems involving unrelated computers. The solutions found to these problems lead to some general principles of design, and of the approach to design, which are next considered. These principles, together with the general pattern of use of the satellite machine at the Institute, indicate the need for a specifically-designed operating system for the satellite, and the thesis ends with a discussion of this system showing how it can be designed to accommodate future expansion of communication facilities.

CONTENTS

		Page
	Preface and Acknowledgments	9
Chap.1	The Background to the Project	11
	1. Historical Introduction	11
	2. The PDP-9 Computer and its Software	14
	2.1 Hardware Characteristics	14
	2.2 The DEC Advanced Software System	16
Chap.2	Characteristics of the Indra Communication Systems	18
	1. Introduction	18
	2. Communications Protocol	18
	2.1 The Slow Link	18
	2.2 The Fast Link	20
	2.3 The Atlas Link	21
	2.4 The CDC Link	22
	3. Message Characteristics	23
	3.1 The 360 Link	23
	3.2 The Atlas Link	25
	3.3 The CDC Link	26
	4. Influence of Message Format on Software Organisation	27
	4.1 Outgoing Messages: General Considerations	27
	4.2 Outgoing Text Messages	27
	4.3 Incoming Messages	28
	4.4 Incoming Messages for Display	28
	4.5 A Note on Error Retransmission	29
Chap.3	Indra Utility Software	30
	1. Introduction	30
	1.1 Historical Note	31
	2. Indra Executive Software	31
	2.1 INDR1 Commands	31
	2.2 Utility Routines	32
	3. Local Data Transfer	33
	3.1 360 Link Message Processing	33
	3.2 CDC Link Message Processing	33

		Page
Chap.4	The First 360 Link and the Atlas Link	34
	1. Introduction	34
	2. The Slow Link	34
	2.1 General System Philosophy	34
	2.2 The Environment	35
	2.3 The Handler	35
	2.4 Monitoring and Off-line Testing	36
	3. The Atlas Link	36
	3.1 Error Recovery	37
	3.2 The Atlas Execute File	37
Chap.5	Experimental Techniques in Software Development	39
	1. Introduction	39
	1.1 Two Design Problems	39
	1.2 Off-line Testing	41
	2. Link Software Organisation	41
	3. Driver Programming	42
	3.1 General Conclusions on Drivers	44
	4. Low-level Software	45
	4.1 Reception	45
	4.2 Transmission	46
	5. Monitoring Facilities	50
	5.1 Running Commentary	50
	5.2 Low-level Momitoring	50
	6. Off-line Testing	51
Chap.6	Recent Developments in Indra Software	53
	1. Introduction	53
	2. Fast Link : First Production Version	53
	2.1 Initialisation	53
	2.2 Driver/Handler Control	54
	2.3 Read Interrupt Routine	55
	2.4 Use of the Teletype	55
	3. The Indra Overlay System	56
	4. Revision of Indra Programs	57

		Page
Chap.7	Towards a New System	58
	1. Limitations of the Present System	58
	2. Proposals for Redevelopment	60
	2.1 The Current State of the New System	62
Chap.8	In Retrospect and Prospect	63
	1. Introduction	63
	2. A Record of Achievement	63
	2.1 Operational Software	63
	2.2 Development Techniques	64
	3. Balance Carried Forward	65
	3.1 The Lessons of DECASS	65
	3.2 Unified Control	65
	3.3 Common Software	65
	3.4 Telecommunications Capability	66
	3.5 High-level Software Techniques	66
	3.6 Real-time Monitoring	67
	4. The Next Step	67
	4.1 Conversion of Existing Links	68
	4.2 Text Transmission	68
	4.3 Text Reception	69
	4.4 Job Turnround	70
	4.5 A Proposal for a Segmented Link Handler	70
	5. Longer-term Prospects	72
	5.1 Terminals	72
	5.2 General Telecommunications Applications	73
Appx.1	The Language of Driver Programs	77
	1. Macros	77
	2. The Form of Assembled Code	78
	3. Repertoire for the 360 and CDC Links	78
Appx.2	Proposed Command Structure for the New System	80
	1. General Principles	80
	2. Current Version	80
Appx.3	DECASS to New System Conversion	82
	1. Introduction	82
	2. Assembly-language Programs	82

		Page
	2.1 Operational Differences	82
	2.2 Assembler Limitations	83
	2.3 Seldom-used Facilities	83
	3. Fortran	83
	4. Files and Directories	84
Appx.4	TOPSY: A Communications System Test Program	86
	1. Introduction	86
	2. Programming for TOPSY	86
	3. TOPSY Operation	87
Appx.5	Personal Involvement	89
References		90

PREFACE AND ACKNOWLEDGMENTS

"We are in great haste to construct a magnetic telegraph
from Maine to Texas; but Maine to Texas, it may be,
have nothing important to communicate" - Thoreau

The main theme of this thesis is the development of a design philosophy for software used in the satellite machine of a two-computer communication system, where the satellite is not recognised by the operating system of the main computer as a standard partner. The design philosophy has emerged as a result of the implementation of four specific communication systems, and the main theme develops from a discussion of these systems. Neither the specific design nor the general approach to design would have any relevance except in the context of some wider purpose, and I wish therefore to take opportunity, outside of the formal part of this thesis, to indicate where the work described fits into the larger scheme and to acknowledge assistance received from many of my colleagues, whether directly connected with the work I have myself been engaged upon, or indirectly through association with the same research group.

The context of the work described here has been a research project in the use of coupled computer systems, supported by the Science Research Council, and led by Prof. Peter Kirstein at the Institute of Computer Science, University of London. To the SRC, which provided the means, and to Prof. Kirstein, who provided the impetus both for the project as a whole and, as Supervisor, for my part of the project described here, all thanks are due.

My responsibility within the project was for the development of telecommunications software for use in the satellite computer. Each system developed thus constituted a quite specific task, but from these specific tasks have emerged two topics of more general application: an approach to the design of telecommunications software for coupled systems, and a design for a small-machine operating system capable of supporting telecommunications as well as more conventional computer use. Many of my colleagues have contributed to both aspects of the work, either directly or indirectly, and I am glad to have opportunity to acknowledge my debt to them. I would like also to mention those MSc students of the Institute whose projects I have had the pleasure of supervising; from these projects has come much of value in the software developments for which I have been responsible. An attempt is made in this thesis to give an overall picture of the various systems, at least as far as the software in the ICS machine is concerned. To do so it has been necessary to include descriptions of some of the work of other people; the last appendix contains a statement of my own contribution.

It would have been impossible to complete the development of the systems described without the cooperation of both the administrative and the technical staffs of the three remote installations: the Rutherford High Energy Laboratory, the Atlas Computing Service (now London University Computing Service), and the University of London Computer Centre. To all those at these installations who in various ways have given their help my thanks are due.

Although the development phase of any given piece of software ultimately reaches a conclusion, as long as development continues in any area of software application there will be scope for the design and implementation of new programs. In the particular area of coupled system design as practised at the Institute of Computer Science it is hoped that the techniques described in this thesis will continue to find application, and that they in turn will be further refined as time goes on.

CHAPTER 1 : The Background to the Project

1. Historical Introduction

In 1967 a research group was established at the Institute of Computer Science, University of London, under the leadership of Dr (now Prof.) Peter Kirstein. The group, known as the Indra Group (Institute Display and Remote Access project), had as its aim the investigation of the use of coupled computer systems for computer-aided design. This aim is described in [1]; briefly, the proposal was that the major calculations should be undertaken in a large and powerful batch-processing computer, while the preparation of raw data and the examination of results should be performed with the aid of a remote small computer equipped with interactive graphic display facilities.

One notable difference between the scheme proposed for Indra and those implemented elsewhere was that control of the whole operation should be exercised at the satellite computer. The large machine was to be in no way dedicated. From the point of view of the large machine, therefore, the operation was to be a matter of remote job entry and remote job control in a conventional batch-processing system, with the remote machine appearing as a conventional peripheral. To the user of the small machine, on the other hand, the system should appear equivalent to the interactive use of a large dedicated computer, although it was recognised that the job turnround would be somewhat slower than might be expected of a genuine large interactive machine.

In the conventional use of remote peripherals on satellite computers, the remote system is, to a great extent if not wholly, controlled by the central system, and its characteristics are known to the central system (in the sense that it is an equipment that the software in the central system is designed to control). The proposed system was thus incompatible in two ways with conventional practice: the transmission of jobs from the remote machine and the initiation of their execution was to be under the control of that machine (at least as far as the remote operator was aware), and the satellite would not be of a type known to the central system. At the same time it was clear that the standard operating system of the central computer could in no way be tampered with.

Consequently it was necessary to develop telecommunications software for the satellite that would simulate the characteristics of some terminal equipment known to the central system. At the same time it was necessary to develop software for the central machine to mediate between the batch-processing habit of that machine and the apparently interactive use of the coupled system as seen by the remote operator. Another design aim was to make the operating procedures for the whole system such that it would be acceptable to a user whose interests were in computer-aided design rather than in the subleties of software.

A grant was obtained from the Science Research Council, and a PDP-9 computer was purchased from the Digital Equipment Corporation. This was installed in 1968, and the original specification included 8k of core, three slow magnetic tape units (Dectapes), paper tape input and output, a control teletype, a DEC type 340 display with light-pen, and a communications adapter for use with IBM modems on a private line. (For general information on the PDP-9 see the next section.) An agreement was also made

with the Science Research Council's Rutherford High Energy Laboratory to allow the use of the IBM 360/75 at that laboratory as the central machine for the coupled system. Software development then started in the areas of telecommunications (software being required for both the PDP-9 and the 360/75) and of interactive graphics. This thesis is concerned only with PDP-9 telecommunications and general system software, and other activities of the Indra group are therefore not discussed.

The first system was constrained by available hardware to use asynchronous transmission at 600 baud. At that time none of the members of the research group had had any experience of writing telecommunications software, and consequently the system, although ultimately operational, could not be claimed to have greatly advanced the art. However, lessons were learnt that resulted in later systems being considerably more satisfactory. The initial system is described adequately in [2]; it is only discussed briefly here in order to point out what some of the lessons were (see Chapter 4).

It was soon realised that the original PDP-9 was two small and that Dectape was too slow to provide an adequate backing store, especially when higher transmission speeds became available, so a further 8k of core was added and a 128k drum was obtained from the Vermont Research Corporation at the beginning of 1969.

That such extended facilities were required may seem strange when it is remembered that many terminal computers operate satisfactorily with very limited resources, so it is worthwhile to digress briefly and list some of the reasons. Many of these are discussed or illustrated more fully in later chapters.

(a) The assumption inherent in the design of the project that files would be held and processed in the terminal implies the provision of backing storage, and to transmit or receive such files requires either very large core buffers or a backing store with low latency.

(b) Space is required in core for file-processing routines which must be at least to some extent co-resident with the telecommunications routines if a reasonable efficiency of operation is to be achieved.

(c) The project was broadened to admit of more than one coupled system. It was hoped that these could be co-resident, although this aim has not yet been realised.

(d) Very extensive graphics facilities were to be developed, requiring a powerful machine.

(e) The software developed for the project was to be used in conjunction with the standard manufacturer's software, as far as was appropriate, and this is somewhat bulky and inflexible.

(f) Although it is possible to develop software in a machine that is only just powerful enough to run it, it is extremely inconvenient to do so. Software development is often a major data-processing operation in itself, and can be most efficiently carried out only when adequate facilities are available.

At about the same time a start was made on a more unified software system for Indra activities. This was based on a simple executive, operating under the standard DEC keyboard monitor (the basis of the DEC Advanced Software System for the PDP-9, known familiarly as DECASS). The main function of this executive was to decode user commands and call appropriate programs to deal with them; it is discussed further in Chapter 3.

A direct link to the Atlas computer was established during 1969, using special hardware in the PDP-9 developed by the Indra group's engineering section [3]. This enabled the PDP-9 to have convenient access to a line printer and a card reader, and was later extended to give access also to IBM magnetic tape. The software was developed as an MSc project by J. Shane [4]. The characteristics of the Atlas link put its software somewhat out of the main stream of telecommunications work; a brief description will be found in Chapter 4.

Also in 1969, facilities became available to allow telecommunication with the 360 in binary synchroncus mode at 1200 baud. Software was developed for this link (known as the fast link to distinguish it from its predecessor), and the use of the 600 baud link was discontinued early in 1970. The fast link software has some interesting features introduced to overcome some of the design difficulties encountered while developing the slow link; these difficulties are described in Chapter 4, and the solutions in Chapter 5 and subsequently.

During 1969 the work load on the PDP-9 increased greatly, due in part to the increasing range of activity of the Indra group, and in part to the use of the machine by numbers of MSc students of the Institute, both for course work and for projects. Also the inefficiency of using an Indra executive running under the DECASS monitor and the limitations of DECASS for telecommunications and interactive operation became increasingly apparent. By the end of the year it had become clear that further progress would be restricted unless some alternative, more specifically Indra-oriented, operating system was developed, and that the pattern of work favoured a rudimentary multiprogramming system in which standardised activities such as communication with remote machines could proceed autonomously while the PDP-9 was in concurrent use for stand-alone jobs such as program development. The limitations of the PDP-9, to say nothing of the shortage of system programming effort, precluded the development of any grandiose scheme, but since nothing suitable seemed available or likely to become available from the manufacturer it was decided to attempt to produce a New System for Indra. The New System was to be designed specifically to facilitate the implementation of both existing and projected telecommunications systems, and is thus seen as a natural outcome of the work that has gone before. Work on this system was undertaken, and has continued, as a private venture, not forming part of the Indra project in any way although based on known or anticipated Indra needs. Development work by the author and P.L. Higginson has continued throughout 1970, in a somewhat desultory fashion because of other commitments; a brief account of existing and planned features of the system is given in Chapter 7.

The main Indra telecommunications activity in 1970, as far as the fast link was concerned, was to replace the experimental version of the system described in Chapter 5 by a production version. This work was undertaken as an MSc project by R.M. Humphreys [5]. Subsequently the fast link system was rearranged by P.L. Higginson to make use of the DECASS overlay facility. Both of these developments are described briefly in Chapter 6.

In 1970 also, it was decided to link the PDP-9 to a CDC 6600 at the University of London Computing Centre. One of the reasons for undertaking this was to investigate the applicability of the techniques used in fast-link development, for the telecommunications and operational characteristics of the two systems are quite different. The results of this investigation are discussed in Chapter 5. So far only an experimental version of the CDC link software has been developed.

Another reason for linking the PDP-9 to the CDC 6600 was to gain experience that would be useful in the context of a project mounted by the University of London: the development of software for a Modular One computer to enable it to act as a replacement for a CDC 200 User Terminal. This project started late in 1970, with software development initially the responsibility of the manufacturer's staff, the author being project leader responsible to the University's Steering Committee for the project. More recently the project team has been extended to include staff from the Institute of Computer Science (where the project is based), and from Westfield College. The Indra group is not directly involved in this project, but experience gained in other telecommunications projects and some test programs developed for use on the PDP-9 have proved useful. In the latter context mention should particularly be made of a program written by A.R. Duncan, primarily for the diagnosis of faults in telecommunications systems (see Appendix 4).

2. The PDP-9 Computer and its Software

The brief description given in this section is intended to establish a background against which particular topics can be discussed later. The description relates specifically to the PDP-9 installation at the Institute of Computer Science; this contains a number of unconventional features and is therefore not typical of PDP-9 systems [6,7].

Lest it should seem that the following description is unduly critical of both hardware and software, it would be as well to point out that one purpose of including this section is to highlight those characteristics of the system that have imposed constraints during the development of telecommunications software, and that, in the end, have both occasioned and influenced the development of the New System. The hardware and software provided by the manufacturer are well matched and impose no unreasonable limitations when the PDP-9 is used for conventional, stand-alone purposes, and indeed all the communications systems described here were constructed to operate under essentially the standard system.

2.1 Hardware Characteristics

The PDP-9 is a single-accumulator 18-bit machine with 16k of 1μs store. It is only possible to address 8k of core directly, the bank so addressed being that in which the referencing instruction is located. Indirect addressing is possible, but there is no modification facility. Store protection is available, but the protection boundary can only be set at integral multiples of 1k words; when store protection is enabled, certain classes of instruction (in particular, IO instructions and those referring to interrupts and store protection) are trapped, causing an interrupt.

A conventional repertoire of arithmetical and logic instructions is provided, together with a subroutine call instruction and a range of conditional skips.

Input/output, real-time clock and display control instructions are also included. The real-time clock can be operated at 20 ms (synchronised to the mains supply), 5 ms (synchronised to the drum), or 100 μs.

Permanently-connected peripherals are as follows:

(i) Paper tape reader (300 ch/sec) and punch (50 ch/sec).

(ii) Teletype (KSR 33).

(iii) Three Dectape magnetic tape units on a common controller, each tape holding some 140k words in 256-word blocks, with a transfer time of some 50 ms/block. To economise on space, files are written using every 5th block, the whole tape being scanned in five passes, so the effective transfer time approaches 300 ms/block.

(iv) A 256k-word drum holding four 256-word sectors on each track, with a transfer time of 5 ms/sector. Track switching is possible between sectors.

(v) A non-storing vector and incremental display (type 340), with light-pen and keyboard.

(vi) Two communications adapters for use with GPO Model 7 modems.

(vii) Special interface logic for controlling a direct link to Atlas.

Other equipment that may be connected at will:

(viii) Teletype (KSR 33 or ASR 35 or both).

(ix) Computek storage tube display with keyboard and Rand tablet.

(x) Olivetti teletypewriter.

(xi) A.B. Dick line printer (connected via a communication adapter).

There are three main techniques for data transfer between the processor and the peripherals:

(a) Program interrupt, in which one character at a time is transferred by software in response to an interrupt from the peripheral. Typical use is for paper tape.

(b) Data channel, in which one word at a time is transferred by hardware from or to a buffer in core. A hesitation of 3 or 4 store cycles occurs as each word is transferred, and an interrupt is given at the end of the block transfer. Typical use is for backing stores.

(c) Direct memory access, similar to data channel operation but faster, requiring only a 1-cycle hesitation. This is used for the type 340 display.

Only one interrupt line exists, without priority except as conferred by software; this serves for clock and storage protection interrupts as well as those associated with peripherals.

The communications adapters and the Atlas link interface were originally designed for program interrupt operation, but recent developments have been made to allow these to be used on the data channel. The corresponding software changes have not yet been finalised.

2.2 The DEC Advanced Software System

Briefly, DECASS consists of a keyboard-controlled monitor (KM9), and a number of utility programs including a Fortran compiler, an assembler, a linking loader, an editor and a general-purpose routine called PIP (peripheral interchange package) used for simple file handling [7]. DECASS is originally a Dectape-based system, but has been modified to work from the drum. Somewhat diffuse and store-consuming handlers exist for the standard peripherals, but there are no handlers for drum, display or any of the special peripherals mentioned above. All these have been developed, where possible according to the DECASS recipe, but the recipe has not been found applicable to telecommunications work.

A clear distinction is drawn in DECASS between system programs and those few routines (mainly handlers) that are available at run time. It is not, for example, possible to include PIP in a user program, and dynamic loading is not available.

A limited overlay facility allows the subdivision of a set of software into separate segments called chains. Each chain is built by a process essentially equivalent to the use of the normal linking loader, and a set of chains form a file called an execute file, held on backing store. A chain from an execute file is loaded in much the same way as a system program, and commences execution automatically. The initial load from an execute file is the first chain placed in it.

Any program in a given chain can call another chain into core and transfer control to it. Data may be passed from one chain to another by the Common mechanism of Fortran, or by making use of a free store area (by non-standard artifice) in assembly language.

Chains must be complete, in the same way as conventional program loads must be; that is, the loader will not accept any unresolved global labels or any missing device handlers. Also, all the programs in a given chain are overwritten when a new chain is loaded. Hence common data must be kept outside, as has already been stated, and common programs, including handlers, must be repeated in each chain requiring them. Thus the execute file may become very large.

An execute file must be planned in advance, so that the chains can be constructed appropriately. It is not possible to modify the contents of a chain, or the circumstances in which one chain calls another, during the course of a run. Note also that all handler allocations are fixed at the time of constructing the execute file and cannot be altered subsequently.

It may be noted here that the overlay facility was not available until Indra work was well advanced. Recently there has been considerable use made of overlays in an attempt to overcome some of the storage problems, and it has

been found necessary to make some radical alterations to DECASS in order to achieve reasonably satisfactory results (see Chapter 6).

Operation is slow, especially with a Dectape-based system, mainly because of the necessity of reloading the non-resident part of the monitor between each step. It is not, for example, possible to call the loader directly from the assembler.

The assembly language (Macro-9) is a very satisfactory vehicle for system programming. It includes a range of system macros by means of which DECASS-controlled IO and other operations can be called, and it also has a facility to allow the programmer to define his own macros, although little use has been made of this facility. Part of the slow link system was written in Fortran (by F. Huber), but on the whole Fortran has not proved to be suitable for telecommunications system programming, partly because of a poor implementation, although it is possible (and, indeed, necessary for space reasons) to combine Fortran and assembly language in the same program, and partly because it is not possible to make efficient use of the facilities provided by the hardware.

The standard representation for alphanumeric data in DECASS is known as 5/7 ASCII, that is, five 7-bit ASCII characters packed into two words. This is in many ways an inconvenient format to handle because of the comparative complexity of packing and unpacking routines, and results also in the equally inconvenient DECASS practice of using word pairs rather than words as the unit in all data transfers. No conversion facilities exist for codes other than ASCII.

Some particular limitations may be noted. The assembler makes two passes over the source code and in consequence is somewhat slow; it produces a diffuse intermediate code that requires extensive further processing by the loader. As a result, the size of the loader often severely limits the size of program that can be loaded. The monitor tends to take control irrevocably if errors occur, making recovery difficult. The teletype keyboard, if used as a control device, is wholly owned by the system, and control functions are not readily available outside the monitor. There is no provision for the use of store protection. The routine within the monitor for handling the real-time clock is of limited utility, especially where independent activities have to be timed concurrently. Various expedients have been adopted to overcome these and other difficulties, but the effect has often been to increase the size and complexity of programs and to reduce running speed, quite apart from the reduction in support given by the system.

CHAPTER 2 : Characteristics of the Indra Communication Systems

1. Introduction

Four communication systems of differing characteristics have been developed: two to the IBM 360/75, one to the CDC 6600, and one to the Atlas. All but the Atlas system have been true telecommunication systems, that is, they have involved the use of modems and the public network. The Atlas system is little more than a hardware interface, but is worthy of inclusion here because the special purpose hardware developed for the PDP end of the link, and the software used to control this hardware, together from a simulator of conventional Atlas peripheral equipment. The use of simulation techniques of this kind is common to all the Indra communication systems, as will be shown in this chapter.

Communication systems have much in common with interfaces, as far as their overall operation is concerned, although they are commonly more complex and offer a greater variety of modes of operation. Thus, for example, control of data transfer is exercised in a hand-shaking manner:: in a communication system it is usual to employ control messages for this purpose where in an interface simple signals would usually suffice. Again, the format of transmitted messages is defined, at least partially:: for example, the character set to be used, the maximum number of characters in a block of text, and any techniques of error control.

From the point of view of the system software designer, differences in communication system hardware are comparatively unimportant, as are also differences of speed, provided always that the speeds are not too great. The significant characteristics are the two mentioned already, namely the rules governing the control of transmission (called the communications protocol), and the format of messages. An account is given in this chapter of the protocols and message formats of the three true telecommunications systems developed, and for completeness, an account of the corresponding, but much simpler, characteristics of the Atlas interface.

2. Communications Protocol

2.1 The Slow Link

The first system set up between the PDP-9 and the 360/75 used asynchronous transmission at 600 baud. In this system, 360 software existed to support a passive terminal (type 1050) which could contain a variety of equipment, card readers and line printers being typical. The PDP-9 software was therefore designed, as far its transmission characteristics were concerned, to simulate the operation of a 1050 terminal.

Slow link protocol is summarised in [2]; a little more detail is given below. In the following, upper case letters are used for characters sent by the 360 and lower case for those sent by the PDP; transmission control characters are underlined.

In the system to be described, the central machine (the 360) polls or addresses (selects in CCITT terminology) its terminals. Polling is an invitation to the terminal to send any data it may have available, addressing indicates to the terminal that data is ready to be sent to it.

The polling sequences used are

C C C A 5 for normal messages

C C C A 0 for diagnostic messages

The addressing sequences are

C C C A 1 for normal messages

C C C A 9 for diagnostic messages

After polling, the interchange of control characters and text strings depends upon circumstances, e.g.

C C C A 5 n	(if nothing to send)
d block1 b lrc1 N block1 b lrc1 ...	(if block incorrect first time)
Y block2 b lrc2 ...	(if block correct and more to send)
Y c	(if block correct and no more to send)
D BLOCK B LRC ...	(if block correct and 360 wishes to send)

Similarly, various alternative sequences are possible after addressing, e.g.

C C C A 1 n	(if not able to receive message)
y D BLOCK1 B LRC1 n BLOCK1 B LRC1 ...	(if incorrect first time)
y BLOCK2 B LRC2 ...	(if correct first time)
y C	(if correct and no more to follow)

The significances of the transmission control characters are:

C c end of transmission

D d end of address

B b end of block

Y y positive acknowledgment (i.e. message correctly received)

N n negative acknowledgment (i.e. message incorrectly received, or no message to send)

The LRC (lrc) is a longitudinal parity check accumulated from the text character immediately following D (d) up to and including the B (b).

It will be seen that in this system of operation, overall control is with the central machine, but it is possible to maintain the illusion that control is with the satellite if the central machine polls sufficiently frequently. The system was in fact arranged to operate in this way.

2.2 The Fast Link

The successor to the slow link as a means of telecommunication between the PDP-9 and the 360/75 used synchronous transmission at 2000 baud. By the time work started on the fast link, software had become available in the 360 to support IBM 1130 machines as terminals, and so the PDP software was designed to simulate the telecommunications characteristics of the 1130.

The main features of this telecommunication systems are its use of point-to-point contention mode (that is, the possibility of either terminal making a bid for control of the line, with a contention being resolved in favour of the satellite), and the possibility of code-transparent transmission (that is, a mode in which all possible bit patterns in a syllable can be used to represent data) [8]. Both features affect protocol; the latter also affects message format as will be explained later in this chapter.

Transmitted strings have essentially the same format, irrespective of which terminal is currently sending. The protocol is thus symmetrical, unlike that of the slow link.

Reference [9] contains an initial discussion of protocol and other matters, derived mainly from the IBM literature available at the time. In the event, a slightly simpler protocol was used, the main features of which are indicated below. Here mnemonics for transmission control characters are used. SYN represents a string of synchronising characters (5 or 6 in practice; the exact number is not critical) that must prefix all transmitted strings to ensure correct synchronisation of the hardware.

Bid for the line:	SYN	ENQ	PAD			
Response:	SYN	NAK	PAD			if the receiving terminal is unable to accept the message
	SYN	ACK0	PAD			if the receiving terminal is able to accept the message
If ACK, send text:	SYN	STX	Text	ETX	BCC	PAD
Response:	SYN	NAK	PAD			if errors are detected
	SYN	ACK1	PAD			if no errors are detected

Repeat as above until all text has been sent, and then after the final response send: SYN EOT PAD

The transmission control characters are:

SYN synchronising character

ENQ request for control of the line

NAK negative acknowledge (e.g. after detecting an error)

ACKO, ACK1 } positive acknowledge (e.g. no errors detected). Note that a poll is always acknowledged with ACKO, but text blocks are acknowledged alternately ACK1, ACKO, ACK1,.... as a check on the loss of a complete block.

STX start of text

ETX end of text

EOT end of transmission

PAD a filler used to allow time for hardware switching

The BCC is a two-syllable check code obtained by applying a cyclic redundancy check (CRC) over all characters following STX up to and including ETX [10].

ACKO and ACK1 are also two-syllable codes, the first in each case being DLE (data link escape). In transparent mode, where all bit patterns may represent data, DLE is additionally used to indicate where a transmission control character is to be interpreted as such and not as part of a text string. Thus:

DLE STX the following text is in transparent mode

DLE ETX end of text in transparent mode

DLE SYN a SYN character inserted into a transparent mode text string to confirm synchronisation (in code-dependent transmission (normal mode), the pair SYN SYN is used for this purpose)

DLE DLE the second DLE is part of the text string

These prefixed DLE characters are not accumulated in the BCC.

In contrast to the situation that obtained with the slow link, it can be seen that contention mode readily allows the overall control of operations to be exercised at the satellite.

2.3 The Atlas Link

The Atlas supervisor does not support remote peripherals in the same way as, say, OS/360, but a spare paper tape channel was available. Consequently the PDP hardware and software for the Atlas link were designed to simulate the action of a conventional directly-connected paper-tape reader and punch.

The Atlas interface is a parallel hardware interface [3], and, as already pointed out, there is thus no protocol in the usual sense. The greater part of the control of the interface is exercised at the hardware level (in response to IO instructions from the respective central processors), and a variety of control signals each on its own line takes the place of the usual variety of transmission control characters.

A command input at the PDP to send data to Atlas is equivalent to the action of loading a tape into the Atlas reader, and results in the Atlas supervisor taking control of the reading operation. Thereafter the PDP software must merely respond appropriately to control signals from the Atlas.

The whole process is entirely asynchronous, and it is worth noting that if the Atlas is busy it may defer reading for an arbitrary length of time.

If the Atlas detects errors in the incoming data stream it does not communicate with the PDP, of course, since to Atlas the PDP is a tape reader. The only indication the PDP operator has that anything is amiss is that data transfer is excessively slow while error messages are sent to the Atlas operator. For this reason, and for general operational convenience, it has been found necessary to instal an intercom between the Atlas and the PDP operators.

2.4 The CDC Link

Software exists in the London University CDC 6600/6400 complex to support both batch input terminals (200 User Terminals) containing card reader, line printer and display console, and interactive terminals (Trend Terminals) containing one or more display consoles. In both cases, telecommunication uses synchronous transmission at 2400 baud. The PDP software (which at the time of writing is in an experimental rather than its final form) is designed to simulate the transmission characteristics of the 200 User Terminal, [11], but the differences between these and the characteristics of the Trend Terminal are small. In particular, the protocols for the two systems are almost identical. Very little difficulty is anticipated in expanding the PDP software at a later stage to allow the PDP to simulate either the batch or the interactive terminal at will. It might even be feasible, given two consoles connected to the PDP, to operate them concurrently, one in batch mode and the other interactively. To do this would probably require the connection of two lines to the PDP, because software in the 6600/6400 is not designed to support both types of terminal on the same line.

The description of protocol given here, and the later discussion of PDP software, are based upon the properties of the CDC software as understood at the time of writing. Changes to this software have recently occurred, and understanding may not yet be complete. Although these changes are not likely to affect any property of the system that is conventionally provided by hardware, they may result in the use of standard facilities that were not used in the earlier system and are not catered for in the current version of the PDP software.

The CDC system polls its terminals, somewhat as the slow 360 link, but does not select in the same way : data is sent from the central machine to the terminal without warning. All messages, including polls and responses, contain two control characters used for addressing : the site address, here denoted by SIA, and the station address STA. It is possible for a given line to serve more than one station : in our case polling usually alternates between two stations, although there is in fact only one on the line, and distinct values of STA are used to distinguish between them. (It appears that the latest CDC software only polls one station on the line.) Also, one bit in the given STA code takes alternate values for successive blocks of text as a protection against loss of a complete block (c.f. the use of alternate ACK0 and ACK1 in the fast link).

Thus every message (in either direction) starts with

SYN SOH SIA STA

where SYN represents a group of SYN characters, typically four in number, although in some circumstances many more may be sent by the central machine

SOH is the start of header

Every message ends with

EOT PAR

where PAR is a block parity syllable, parity being accumulated over all characters from SOH to EOT inclusive, but omitting any SYN characters.

A control character follows STA; in messages from the CDC this is either POL (poll) or CWT (clear write). The latter is the prefix to a write message, that is one or more blocks of text. The control characters that may be sent by the PDP are

ACK acknowledge, used in response to a poll when the terminal is ready to send data

REJ reject, used in response to a poll when the terminal has no data to send

ERR error, used when messages received at the terminal are corrupt

STX start of text (read message, i.e. message sent by the terminal)

The format of messages is discussed in the next section; it is sufficient here to note that an escape character (ESC) followed by a control character known as an E code (EC1, EC2 or EC3) is placed before the EOT at the end of write and read messages to indicate the source or destination of the message, or that the message serves some system control purpose.

As far as is known at present, the only differences between the protocols for the 200 User terminal and the Trend terminal are in the codes used for certain control characters, and in the fact that messages referring specifically to the card reader or line printer are not sent to Trend terminals.

3. Message Characteristics

3.1. The 360 Link

The choice of format for messages on the 360 link was largely made to suit the purposes of the Indra project and the convenience of the PDP software, since the 360 system appeared to improve few constraints [12]. The code used for alphanumeric data was, however, determined by 360 software requirements; for the slow link a 6-bit BCD code was used, and for fast link 8-bit EBCDIC.

In the early stages of developing the slow link the message format used was intended to be appropriate for store-and-forward operation, even although only two machines were involved. The various sources and destinations that would be actual pieces of equipment in a true store-and-forward system were in this case the various data sets, files and processing programs. This approach was abandoned when the fast link came to be developed. Apart from the presence of source and destination identifiers in the slow-link message headers, the formats of messages on both versions of the 360 link were similar, and only the fast link version is described here. Some notes on slow link message organisation will be found in Chapter 4.

The original design has been modified in various ways as the project has developed and as experience has been gained; a more orderly arrangement could be achieved by redesign, but as far as present use of the 360 link is concerned there does not seem to be sufficient need for this to justify the effort that would be involved.

Interest in store-and-forward systems is currently on the increase again, with the possibility of practising a genuine store-and-forward operation involving, say, the 360 and the CDC 6600. For such operation to be feasible the message format will certainly have to be redesigned, but for that matter so will many other aspects of the PDP-9 telecommunications procedures.

The unit of data transfer is the block, which contains up to 500 8-bit syllables, each syllable containing either an EBCDIC character (for alphanumeric data) or two hexadecimal digits (for binary data). On the PDP-9 backing stores, a form of packing known as double-image is used, in which each word is divided into two 9-bit syllables, each containing one character (or two hexadecimal digits) right justified. 500 syllables is thus a convenient number of syllables to hold in a 256-word drum sector or dectape block, leaving room for header data. In the general case of file transmission, each file may occupy a number of blocks, and for ease of data handling the blocks are arranged to be coterminous with records for the file.

500 syllables also seemed to be a convenient length from the point of view of transmission errors. When development began it was far from clear what the transmission error rate might be; indeed, it is still far from clear, since few if any of the errors that have occurred in practice can be attributed with certainty to transmission faults. The cyclic redundancy check (in the fast link) is effective for block lengths up to 4096 syllables or beyond, and it may well be that, from the transmission point of view, the block length chosen is unnecessarily short, but the convenience of handling 500-syllable blocks rather than larger units in the PDP-9 is not to be dismissed lightly.

Each block is prefixed with a type code (an EBCDIC numeral) used by the software at the receiving terminal to control processing. The type codes are:

0	intermediate block of a multi-block message	message used for conveying data
1	first block of a multi-block message	
2	last block of a multi-block message	
3	single-block message	

4	chat message	messages for system control and operator information
5	command message	
6	'Good day' message	

Chat messages were originally intended to include means for operator-to-operator communication via the link. This has proved troublesome to implement, as discussed later, and in the present system chat messages are program-selected. They are typically error messages or other information for the operator at the receiving terminal. The 'Good day' message is a particular example used only when communication between the terminals is first established. Command messages are sent from the PDP to the 360 to initiate particular activities; they are either held in core, computed (or completed) at the start of a run, or input at the teletype during the run.

Note that the decision to put the type identifier at the front of each block on backing store makes pre-processing unavoidable, for otherwise the correct choice of type could not be made.

Messages of types 0-3 are sent in transparent mode, to allow the use of either alphanumeric or binary data. Messages of types 4-6 are sent in normal mode for historical reasons, although there is now seen to be no good reason for this.

3.2 The Atlas Link

Since the Atlas link operates as a bidirectional paper-tape channel, no particular formatting is required. Data are always transmitted in Atlas code, because although Atlas will recognise ASCII there are certain difficulties in its use as explained in [4].

It is necessary to precede data transfer (in either direction) by sending Atlas a short program to control the transfer, since, unlike the 360, Atlas does not contain permanent software for servicing the link to the PDP. The two programs (one for each transfer direction) are held as data within the PDP software, and the appropriate program is sent automatically, following a job description that causes Atlas to compile and then execute the program. To add further facilities to the Atlas link system only involves writing the appropriate Atlas routine and arranging that it is selected and sent in response to a suitable command. The Atlas link could therefore be fairly easily extended to provide a general remote job entry facility.

The only remaining point to be noted in the context of message format is that Atlas uses sequences of the form ***X, where X is one of a number of different characters, for supervisor control purposes. It is therefore necessary to avoid repeated asterisks in transmitted text; this is done during the code conversion process by following each * with an erase character. The erase character is ignored by the Atlas software, but serves to break up sequences of asterisks. Source text sent to Atlas must be terminated by the string ***Z; this is appended after code conversion so that the asterisks are left intact.

3.3 The CDC Link

In the terminology of the CDC telecommunication system, the term 'message' refers to all transmitted strings, including control sequences such as those for polling and responses. We are here concerned only with messages conveying text, that is, with write messages (CDC to PDP) and read messages (PDP to CDC).

Ignoring for the moment certain special write and read messages used for control purposes, the transfer unit (block) consists of up to 1000 characters when writing to the printer or reading from the card reader, or up to 50 characters (equivalent to one line) when writing to the display (with certain exceptions) or reading from the keyboard. When reading the card reader, the block contains simple concatenated card images (i.e. up to 12 cards, giving 960 characters). Write messages to the line printer may include ESC characters followed by a variety of codes indicating new line, compressed spaces, or compressed zeros. The first character of each line sent to the printer is used for format control. Lines written to the display or read from the keyboard terminate with new line, and do not commence with a format control character.

In the PDP-9, a deck of cards (without control cards) is represented by a conventional file which is pre-processed into a double-image format on drum.

All write and read messages end with an ESC and a special control character known as an E code, immediately preceding the EOT. The significance of the E codes, here denoted by EC1, EC2 and EC3, is as follows.

In write messages

EC1 means that the message is to go to the display

EC2 means that the message is to go to the printer

EC3 means that the message refers to the card reader (see below)

In read messages,

EC1 means that the message comes from the keyboard

EC2 means that the message comes from the card reader and includes the last card (but see below)

EC3 means that the message comes from the card reader, and that further cards are available (but see below)

The EC3 write message, which has the form +ESC EC3, is a request to read more cards. The response to this is either a block of cards ending with EC2 or EC3 as stated above, or the 'automatic read with E2' indicating that there are no cards to read, or that there is some error condition (e.g. a card jam).

The 'automatic read with E2' is also used as a response to write messages for the printer to indicate that the printer is not ready to accept the data. If it is ready, the response is an 'automatic read with E3'. The automatic read messages have the form ESC ECn.

The 200 terminal is used for remote job entry, and the file for a job therefore consists of two or three sections: the job description, the program (in source language), and any required data. Between each section an end of record card is sent, and at the end of the job file an end of file card is sent. These cards contain ESC EOR and ESC EOF respectively.

4. Influence of Message Format on Software Organisation

Comparison of the message formats described above for the 360 and CDC links indicates that the software for these links requires somewhat different characteristics.

4.1 Outgoing Messages : General Considerations

Considering outgoing messages first, it will be seen that all messages have one or more standard sections, determined by the protocol, in which most of the characters are independent of the kind of message to be sent. There are also in general certain characters that must be present, but whose value is dependent upon the message (e.g. the ACK or REJ of the CDC link) or upon some previously determined condition (e.g. the choice of second character of an ACK pair in the 360 link).

Some messages have no other components (e.g. responses), and are thus fairly easily dealt with. They are either held entire in core, or are composed under control of some parameter or switch while transmission is in progress. The standard parts of messages including text are also easily handled in this way.

Broadly speaking, messages input at the teletype for transmission are handled similarly in the two systems: they are code-converted and held in double image in a core buffer until they can be sent. In the CDC system the regular poll ensures that opportunity will occur for transmission; in the 360 system, since all transmission is initiated by the PDP, opportunity must be made, and this requires a higher level of control.

4.2 Outgoing Text Messages

The major differences between the two systems appear when the transmission of text held on backing store is considered. In the 360 system, the fact that text is pre-processed into 500-character blocks makes transmission simple, since each block is prefixed with a syllable giving its length. It is however necessary to have a buffer large enough to hold a complete block.

In the CDC system, on the other hand, pre-processing is much simpler, and could probably be omitted altogether except for the necessity of ensuring that the output files are on drum. (Dectape has in general too high a latency.) It was felt that to provide 1000-character buffers was unreasonable, so a complete message block is never present in core; instead, one line (corresponding to one card) is read into core at a time. Two line buffers are used, so that a card can be read into one while the previous card is being transmitted from the other.

As a consequence, the transmission routines in the CDC link software are considerably more complicated than those for the 360 link. The complication is increased by the requirement to convert each line into card image, and to insert end-of-record or end-of-file card equivalents since these are not part of the files as held on backing store.

In synchronous transmission systems it is necessary to maintain the rate of output of syllables if synchronisation is not to be lost. It is fairly common practice to insert SYN syllables into a message string for this purpose if part of the text is not ready in time. The 360 will accept SYN syllables inserted in this way, although as a full 500 syllable buffer is maintained in core it is in fact never necessary to insert any. The CDC 6600 frequently inserts strings of SYN syllables; several hundred have been observed at a time when main store access has been unduly delayed. But it will not accept inserted SYN syllables sent from the terminal, so it has been necessary to design the software so that each card-equivalents record is ready to be sent as soon as its predecessor has gone. This constraint has further complicated the transmission routines of the CDC link, as will be shown in Chapter 5.

4.3 Incoming messages

As far as the standard components of incoming messages are concerned, similar routines are used in the 360 and CDC systems to check and identify the various transmission and message control characters. Incoming text strings are assembled into double-image, held in most cases on backing store for subsequent post-processing.

The CDC system is in this case somewhat simpler than the 360 system in its requirements for text handling. As has been explained already, it is possible to transmit either alphanumeric or binary data from the 360; both kinds are assembled into 500-character blocks. The only complication in assembly is catering for the slightly different requirements of normal and transparent modes, but post-processing is a major undertaking. The CDC transmits only alphanumeric data and has no transparent mode; all transmission is essentially in lines destined for the line printer or display. As far as data for the line printer is concerned, it is simple to expand compressed zeros or spaces as the line is being received, and the only post-processing required is to remove displayed header pages (unless the data is in fact to go to a line printer, when there is no overwhelming need to do so), and to handle the formatting characters. Post-processing software has not yet been developed for the CDC link, as no major problems are anticipated.

4.4 Incoming Messages for Display

In both systems as currently implemented, any received messages to be displayed are converted immediately into DECASS format and output. In the 360 system, messages are identified by the prefixed type code, and hence the destination for display messages is known before the message is assembled. In the CDC system, on the other hand, the destination of a message is given by the E code at the end of the message. This presents no problem with single-line display messages, but with multi-line messages it is necessary to write each line to backing store as soon as it has been assembled, so as to free the core buffers, and then to re-read the whole message and display it if an EC1 is ultimately detected.

4.5 A Note on Error Retransmission

If the response from the central machine indicates that a block sent from the PDP has not been received correctly, it is necessary to retransmit it. In the 360 system this is straightforward, since each block is retained in the core buffer until it has been correctly acknowledged. The CDC link, on the other hand, cannot retransmit a block because, as already explained, the block does not exist as a recognisable entity. This is a recognised short coming of the present experimental system, which will be corrected in due course.

It may be noted in passing that DECASS does not allow other than sequential reading of the records in a file. Hence to retransmit a block to the CDC would involve reading through the file until the first card-equivalent record of the block in question was found. This would be a time consuming process, and if transmission errors are often encountered in practice it may be necessary to adopt the use of pre-assembled blocks as in the 360 link. No data is yet available on the incidence of transmission errors in the CDC system.

CHAPTER 3 : Indra Utility Software

1. Introduction

The software for the 360 slow link was developed as a stand-alone system. As Indra work developed, and the fast link and Atlas link were planned it was realised that this approach had severe limitations and that a more unified software system for Indra was to be preferred. In particular, it was realised that

(a) there was a need for some simple and extensible command language to cater for Indra activities, initially in telecommunications and local file handling, and possibly later being used e.g. for graphic work.

(b) local file-handling processes were required in their own right and also as an adjunct to telecommunications, e.g. for automatic preprocessing of message texts. The processes were similar to some of those possible with PIP (see Chapter 1), but with certain extensions, and would be required at run time (in the DECASS sense, i.e. when PIP would not be available).

(c) the various telecommunications systems that were envisaged would probably have some common requirements for utility routines.

(d) it was desirable as far as possible to minimise loader delays by having as much Indra software resident as could be arranged. Note here that the DECASS overlay system, used in the latest versions of Indra software as described in Chapter 6, was not at that time available.

(e) as a result of (d) there would in general be a chronic shortage of store.

Since the decision had been taken to base Indra software on DECASS, it was next necessary to determine how much standard software could be used, and hence what items it would be necessary to develop. As was pointed out in Chapter 1, the manufacturer's operating system for the PDP-9 (that is, the resident keyboard monitor) does not support telecommunications activities. In fact, except where DECASS device handlers exist, or are specifically written, the only significant use that can be made of DECASS software at run time is for the identification of interrupts from remote machines, and for conventional input/output operations involving standard (local) peripherals.

Consequently, even when use was made of DECASS facilities wherever possible, it was found to be necessary to develop software for the following purposes:

(i) Overall control of telecommunications, i.e. the PDP operator's interface with the system.

(ii) Preprocessing of data to be transmitted and post-processing of data received.

(iii) Run-time control of transmission and reception, including both the correct administration of protocol and the lower-level detail of IO.

(iv) Run time communication between the PDP operator and the remote system, or its operator.

(v) Miscellaneous utility routines

of these, item (iv) does not apply to the Atlas link at present, since no operator-to-operator communication has been implemented; Atlas supervisor facilities would permit the PDP to send messages to the Atlas console. It did not apply to the slow link, where chat messages were only generated by the system at one terminal or the other. In the fast link, two-way operator-to-operator communication should be comparatively straightforward, since contention mode is standard, but in fact it has not yet been successfully implemented, mainly because of software constraints (see also Chapter 6).

Items (i) and (ii) are, in part at least, independent of the particular link considered, and are discussed in this chapter, together with item (v). Items (iii) and (iv) are specific to each link, and are discussed in Chapters 4 and 5.

1.1 Historical Note

Although the slow link required facilities for operator control and message pre- and post-processing, of course, those facilities, like the rest of the slow link software, were developed in ad hoc fashion, and it was not until work was in hand on the development of the fast link that any serious attempt was made to establish any systematic and extensible body of software for them. Consequently we shall here be concerned with the executive and utility software developed initially for the fast link and subsequently used in the Atlas and CDC systems.

2. Indra Executive Software

A simple executive routine known as INDR1 was written to run under the DECASS keyboard monitor [13]. The main purpose of this was to provide extensible command decoding facilities so that a uniform command structure could be used for the various activities in local and remote file handling. Commands decoded by INDR1 are checked for validity, and INDR1 then calls the appropriate program or sequence of programs to execute the command. There is thus, for example, no need for the operator to give explicit commands for the pre- or post-processing of messages.

Programs such as the link handlers run under INDR1, and return to INDR1 when they have completed their process. The value of the accumulator content on return is an indication of the success or failure of the process, and INDR1 prints error messages if failure occurs.

2.1 INDR1 Commands

All commands to INDR1 are input at the teletype and begin with two single-letter fields denoting the system involved (360 link, Atlas link, CDC link, or local) and the function required. The remainder of the command consists of a variety of fields according to system and function, used to specify details. Full specifications of the commands will be found in the references; the following examples will serve to indicate the types of facility available.

P T DT1 FILE SRC DR FILEX BIN / AC

A local command, that is, one involving the PDP only, that transfers a file called FILE from dectape 1 to drum, changing its name to FILEX and changing the data representation from 5/7 ASCII (A) to double-image coded (C). The target code may be either EBCDIC or CDC code depending on the version of the processing program loaded (see section 3 below).

C C DR <program> DR <data> /

This sends a job to the CDC 6600 (the second C implies cards), where the card deck is simulated by two files on drum: <program> and <data>. In later development, job description parameters could follow the /, but this facility has not yet been implemented.

A T DT2 FILE SRC L

This sends FILE from dectape 2 to Atlas for printing. The last field was provided to allow insertion of some parameters of the job description, but the facility is not currently used.

R S <job name>

This is a command to the RHEL 360, used to start the designated job. The format of <job name> is specified to suit the requirements of the 360 software, and the command results in a command message containing <job name> being sent to the 360 (see also Chapter 6).

2.2. Utility Routines

INDR1 contains a number of utility routines, callable by processing programs, some of which may be briefly mentioned.

One of the problems in using DECASS is the amount of storage space occupied by programs and their data areas. In the simple use of DECASS, i.e. without overlays, all required programs must be loaded together before the run. The loader will fail if any programs are omitted whose existence is indicated by the presence of global labels. The loader itself occupies considerable space that could be used after loading is complete, but that cannot be allocated beforehand (in assembly language programs). A simple free store routine was therefore included in INDR1 so that processing programs could obtain and release working space as they required it, instead of being written with the working space incorporated in the code.

Data handling, whether in command decoding, message processing, or local formatting, involves frequent packing and unpacking. Three formats are in regular use: 5/7 (five characters in two words) and sixbit (three characters in one word) for DECASS, and double image (two characters in one word) for telecommunications; packing and unpacking routines for these are provided, although not all in INDR1.

In telecommunications work it is often neccessary to time processes, as will be seen in later chapters, and independent timing of concurrent processes may be required. The timing routine available in DECASS is not suitable in the latter case, and a special routine was developed to allow up to eight independent timing operations to be performed concurrently, all synchronised to one of the real-time clocks of the PDP.

3. Local Data Transfer

Run-time local data transfer, with the option of a variety of code conversions, is an essential for message processing, whether before, during, or after transmission or reception. Consequently a program called PTRANS [14] was developed, incorporating some of the facilities of PIP, but with several extensions. PTRANS is called explicitly from INDR1 to execute a P T command, and can also be called by other processing programs when required.

PTRANS will accept store or any local peripheral as source or destination, and will handle alphanumeric or binary data. When PTRANS was developed, the only code used other than ASCII was EBCDIC, for which a code convertor was written, and the PTRANS command was therefore designed to specify only ASCII or EBCDIC. When the CDC link was started, a code convertor for CDC code was developed for use with a slightly modified form of PTRANS; although a lack of generality is apparent here it is of no consequence since it is impossible to operate the CDC and 360 links concurrently.

PTRANS may also be used to convert binary data into formatted and packed ASCII octal. This is of use typically for diagnostic purposes, for example when examining the content of a buffer in store.

3.1 360 Link Message Processing

The pre-processing of files for transmission to the 360 is done by a special call to PTRANS which generates a file called FLOUT on drum. In this, the file to be transmitted is held as a set of blocks of double-image EBCDIC, with up to 500 characters in each block. The first word of each block contains the number of syllables in the block, for the use of the transmission routines, and the second word contains the block type (see section 3.1 in Chapter 2).

PTRANS is also used to post-process alphanumeric data received from the 360, being called after a run to deal with job result data, or during a run to deal with chat messages. In the former case the call is the last step in executing an R R (receive) command, and the converted file is written away as indicated in the command. Chat messages are converted to ASCII and sent to the display. In this case PTRANS is used with core buffers as source and destination, because the display is also in concurrent use for other purposes (see Chapter 5), and it is more convenient to use a common display call.

3.2 CDC Link Message Processing

The version of PTRANS with CDC code conversion is similarly used for pre-processing source files, leaving them in double image on drum. Here no special formatting is required, as the conversion of DECASS records (lines) to card image is done during transmission.

Post-processing of received data could be done similarly (write messages to the line pointer are sent to backing store), but so far an automatic call to PTRANS for this purpose has not yet been incorporated.

As in the fast link, PTRANS is also called from the link handler during the run to code convert and format messages destined for the display.

CHAPTER 4 : The First 360 Link and the Atlas Link

1. Introduction

As has already been mentioned, the experience gained during the development of the slow link to the 360 lead to the introduction of an approach to telecommunications software development that has greatly benefited later work on the fast link and the CDC link. The account of slow link software given here is intended to point out what were the major lessons learned, and thus to provide a starting point for the discussion of improved techniques in later chapters. A more detailed description will be found in [2].

The Atlas link is included here also for completeness, since it is a component of the totality of software available under the general Indra executive discussed in the last chapter. As the Atlas system is radically different from the several true telecommunications systems, there would have been little advantage in applying telecommunications software development techniques, and hence the development of the Atlas link owes relatively little to other work. The Atlas link was in fact the first major program developed to run under INDR1, and it has been in regular use ever since it was completed in the autumn of 1969. For more details than are given here, see [3,4].

2. The Slow Link

2.1 General System Philosphy

The slow link software had two main sections: the environment, which attended to PDP files and the user's commands, and the handler, which attended to the receive and transmit operations of the data adapter. There was no pre- or post-processing; the transmission block corresponded to one record in the DECASS or Fortran sense: that is, a single line or card. An arbitrary maximum block length of 100 characters was set, including five header characters (source, destination, block length and block type).

It was originally intended that the environment should be written in Fortran, and that data and control parameters should be passed between it and the assembly-language handler in a common block. Thus all packing, unpacking and code conversion were to be provided by the handler. The common block included two buffers, each big enough for an unpacked transmission block; packing and unpacking routines were designed to operate within the confines of a single buffer.

A hand-shaking interface was established between the environment and the handler by means of four flags (treated as logical variables by the Fortran environment). These were:

AT (able to transfer) : set TRUE by the handler when an error-free block had been assembled in the read buffer;

TC (transfer complete): set TRUE by the environment when the received block has been disposed of;

AL (able to load) : set TRUE by the handler when a block has been transmitted, and correct reception has been acknowledged by the 360;

LC (load complete) : set TRUE by the environment when a block has been placed in the write buffer ready for transmission;

2.2 The Environment

The original aim of writing the environment wholly in Fortran had to be modified when it was found that the DECASS object-time system for Fortran was too big to be loaded with the rest of the software and could not be segmented. Consequently all the IO routines for handling files and teletype commands had to be specially written in assembly language. and only the central part of the environment (some 70 statements) remained in Fortran. These were almost exclusively used to select subroutine calls according to the values of various flags (including those mentioned above), operations that could have been almost as simply written in assembly language with considerable savings in storage space.

The environment was written in three sections, the first of which merely called the initialisation section of the handler (see below), and awaited the 'Good day' message from the 360. The second section waited for the operator to input a command at the teletype, and then prepared the local peripherals for reading or writing a file according to the requirements of the command.

The third section contained the receiving and sending loops for text (i.e. files). In the receive loop the environment waited for AT = TRUE, and then sent the received record to backing store or teletype according to block type. When a received file was complete, it was closed, and control was returned to the command section. In the transmit loop the environment tested AL, and when AL = TRUE a record was read from the PDP file and placed in the write buffer, with source and destination identifiers in the header according to the command parameters. The environment calculated the block type, and after the last block had been sent the input file was closed. The environment then awaited a chat message from the 360 (e.g.'awaiting commands'); this was output to the teletype and the command section was then re-entered.

It will be observed that all the activities of the environment were strictly sequential: there was, for example, no interleaving of the command section with the receive/send section.

2.3 The Handler

The initialisation section of the handler set up the connection between the interrupt identification routine in the DECASS monitor and the corresponding interrupt handling routine in the handler proper. It then set initial values for the various flags, established operating and monitoring conditions (see below), and, in on-line operation, primed the communications adapter.

The transmission section of the handler began by unpacking the outgoing block and converting the code from ASCII to BCD. It then inserted the block length component of the header (two decimal characters). The block was then transmitted, preceded and followed by the various control characters required by protocol.

The reception of a block was essentially the converse process. The initial control characters were checked, and the rest of the block was then assembled. The header was left unpacked, for the use of the environment, and the rest was code-converted and packed into a conventional DECASS record.

Two classes of error were distinguished. Recoverable errors included, for example, parity errors in received blocks, and gave rise to an automatic n response (reject). Retransmission of the block from the 360 was then awaited. Similarly, an N response from the 360 caused the handler to repeat transmission of the last block sent. Irrecoverable errors included, for example, receive interrupts from the 360 while the PDP was sending, or failure to receive a good block after 3 retries. The handler was then set to a condition such that it would only accept polling, and when a poll was received a preset error message was sent. The assumption was that if all else failed, the 360 would ultimately revert to polling, and the 360 software was written on this basis.

It will be seen, even in this brief description from which much detail has been omitted, that all matters of protocol were dealt with at the low level of the handler, and that error recovery procedures were somewhat primitive.

2.4. Monitoring and Off-line testing

An eight-unit code was used for the asynchronous transmission at 600 baud, corresponding to about 75 ch/sec. A reduced speed of transmission is acceptable to the hardware of an asynchronous system, and hence it was possible to monitor outgoing messages by sending them also to the 50 ch/sec punch. Received characters could also be monitored (up to a maximum of about 1000, due to core limitations) by insertion into a circular buffer. It was not at that time thought feasible to hold a monitor record on backing store. The monitoring options were selected before a run by appropriate setting of console switches on the PDP. This amount of monitoring, although useful, was exceedingly limited, and gave very little help in identifying software bugs. It was also not possible with certainty to relate transmission with reception, and no information on the relative timing of events was deducible.

For off-line test purposes, the punch was used for output in the same way as described above, and the tape reader was used to simulate input from the 360. With the aid of a simple program it was possible to prepare data tapes containing a character string thought to be in accordance with what the 360 would send in the conditions of a given test; the characters were interspersed with groups of ignorable blanks to give approximately the correct data rate. Off-line testing or on-line operation was selected by a console switch. This method of testing proved satisfactory, within the limits imposed by imperfect knowledge of protocol and other transmission characteristics, and was gradually refined as development proceeded.

3. The Atlas Link

In common with all programs designed to run under INDR1, the Atlas Communications Routine (ACR) is called as a subroutine of INDR1. The content of the accumulator on entry specifies the transfer direction, and the calling instruction is followed by the address of a parameter block. This contains a specification of the PDP peripheral involved as source (in A T commands) or destination (in A R commands), and of the local file name. It could also contain parameters of the Atlas job description, but in ACR as implemented this facility is not used, and the complete job description is held as data within ACR.

The initialisation section of ACR incorporates the command parameters into the various DECASS file-handling calls, and sets up the IO connections for the ACR interrupt handlers. The program then branches according to the command direction.

The Atlas interface in the PDP-9 is so arranged that the link behaves, from the PDP software standpoint, somewhat as a pair of paper tape channels, operating half-duplex. Consequently the low-level link IO activities of ACR are similar to those of a conventional reader or punch handler, accordingly to direction. At a higher level, the interaction of the Atlas supervisor with its peripheral channels makes the operation of ACR less regular, as will be shown.

When an input tape is loaded into an Atlas reader, the operator must engage the reader manually. An IO instruction in ACR causes the interface to send the corresponding engage signal. The supervisor will then read the first part of the data, so as to be able to analyse the job description. Further reading may be delayed if the Atlas queue is full, and ACR will then also be unable to proceed. The supervisor may also disengage the reader, for example if a parity error is detected or if a control sequence such as ***Z is detected.

The state of engagement of peripherals is checked at regular intervals by Atlas, and if the PDP interface is disengaged it is necessary to wait awhile before re-engaging it so that the supervisor is able to detect that re-engagement has taken place. ACR inserts a timed 1·5 sec wait before re-engaging. The cause of disengagement is not checked, since it is to some extent data dependent, so if transmission errors are the cause, the only evidence the PDP operator has is that the transfer is proceeding unduly slowly.

When reading from Atlas, the first action of ACR is to send the 'read cards' program. It is then up to the Atlas supervisor to commence output when it is ready, and ACR must wait. The technique adopted is to issue a read IO instruction, which engages the channel, and wait for 100 ms. If no character has been received in that time, an instruction is issued to disengage the channel and ACR waits a further 1·5 sec before re-engaging.

3.1 Error Recovery

As the Atlas link has no protocol, the operator is more closely concerned in ensuring its correct functioning. As the Atlas supervisor sends error messages to the Atlas console, but does not communicate with the PDP on its own account, the use of a direct intercom between the two operators is a necessity if errors occur.

On the whole, the link has functioned very reliably, but when errors do occur the only course open is to start again at the last command issued. Occasionally even this course can give rise to unexpected results: if the PDP fails during transmission, for example, the Atlas already has an incomplete file, and when the command is repeated the retransmitted file will be appended to the part that is already in Atlas. The solution is to send the document terminator ***Z as a file from the teletype, so as to dispose of any partial file before repeating the original command.

3.2 The Atlas Execute File

The limiting case of the DECASS overlay facility discussed in Chapter 1 is an execute file containing a single chain. It is a simpler matter to load such a file than it is to load a set of separate routines, and consequently an execute

file for the Atlas link was constructed. This contains ACR and its code conversion routines, together with a version of INDR1 from which all reference to routines other than ACR had been removed. This modification was made to avoid the necessity of including a wide variety of unwanted routines so as to satisfy loader restrictions.

The Atlas link has also been incorporated as one overlay in the general Indra overlay system described in Chapter 6. The facilities of this link are therefore available to all PDP users, whether involved with Indra activities or not.

CHAPTER 5 : Experimental Techniques in Software Development

1. Introduction

When work on the fast link was first started, the approach used was similar to that of the slow link described in the last chapter, except that the longer block to be used necessitated pre-processing, and that the use of Fortran for any part of the environment was discontinued. It had already become clear that the slow link approach was not altogether satisfactory, but no alternative had at that time been considered.

It was soon found that the 360 support software for 1130 terminals was considerably less reliable than that for 1050 terminals used previously, and that less assistance was forthcoming in resolving its inadequacies. The result was that attempts to establish on-line connection were frequently bedevilled by inexplicable failures that could not with certainty be attributed to one terminal or the other, and there was an obvious danger that development would degenerate into a frenzied and aimless series of arbitrary software changes.

Consequently a radical reappraisal of the approach to terminal software development was undertaken. The arguments are presented in this chapter, together with an account of the organisation of the experimental form of telecommunications software that resulted. The success of the approach in the fast link application led to its adoption later for the CDC link; the extent to which the approach had to be modified to suit the differing characteristics of the latter link is also considered.

1.1 Two Design Problems

The two main problems in the design and implementation of telecommunications software, at least where unrelated machines are involved, and perhaps generally, may be informally stated as

(a) what should happen next?

(b) what actually did happen?

In principle one might expect that, given a specification of the protocol of a telecommunication system and a knowledge of the transmission code used, it would be simple to construct a simulator. In practice, unfortunately, it seems that a sufficiently precise, detailed, and even comprehensible specification is not to be had, and even transmission codes may not be exactly as suggested in the literature. These uncertainties were more pronounced in the 360 system then in the CDC system, perhaps because the 360 telecommunications software was less well established when work commenced on 360 fast link development than the corresponding CDC software was when the CDC link was started.

It is therefore not possible to be certain, *a priori*, what interchange of control and other messages to expect in any given circumstances, and this is particularly the case when errors occur, especially as a simulator will

not in general fail in the same way as the system simulated. Hence it is imperative to collect as much evidence as possible during a run. If some of the evidence can be made available while the run proceeds, it may be possible to take corrective action, or at least to minimise the effects of system collapse; in any case the evidence should be protected as far as possible against system collapse, so that it can be analysed afterwards.

It is worth noting also in this context that it is extremely difficult to introduce low-level monitoring at the central computer because of the all-pervading operating system. In the circumstances of fast-link development, where a considerable body of Indra software was developed for the 360, it was ultimately found possible to bypass OS/360 to a fair degree for monitoring purposes, but in the CDC link no information is available at the centre at a level below complete messages. Consequently the onus for providing detailed monitoring is on the satellite. In the early stages of fast-link work, while the only trace available in the 360 was the standard, relatively high-level 'line interrupt trace', it was frequently the case that the PDP facilities described later in this chapter yielded a detailed account of the history of a run before the line interrupt trace at the 360 had even been printed.

To make changes in an assembly-language program, consequent perhaps upon obtaining a better understanding of the performance required, normally involves detailed analysis, possibly extensive editing, reassembly and reloading before another attempt at communication can be made. These are time-consuming processes, and there is great danger that errors will be edited in due to some unappreciated side effects of a change. Thus much time can be wasted, and in situations where one or both of the machines involved has a full work load it is undesirable for too many runs to be abortive.

Telecommunications software is necessarily interrupt-driven, at least in part. (One of the objections to attempting to implement link handlers according to the recipe for DECASS device handlers is that remote machines, unlike most wholly-owned peripherals, are likely to cause unsolicited interrupts.) In a simple system, such as the PDP with DECASS, it is difficult to go far towards multiprogramming, and consequently any facilities incorporated for run-time monitoring add to the general overhead and increase the risk that the time available for interrupt servicing may at some point be exceeded with disastrous results.

The foregoing considerations lead to certain desiderata for telecommunications software, at least during the development phase.

(i) The area of greatest uncertainty should be identified in advance of any coding being done, and steps taken to facilitate changes to routines serving this area. Uncertainty is likely to be greatest in matters to do with protocol.

(ii) Extensive monitoring should be included, covering as far as possible all actions of the telecommunications software, whether resulting from local or remote causes, and listing in detail all messages received or sent.

(iii) Some real-time monitoring should be included, perhaps as a subset of (ii).

(iv) Control facilities should be provided so that the operator has a means of modifying the telecommunications process, perhaps as a result of (iii), of attempting to avert impending system collapse, and of retreiving the monitoring record when a collapse occurs.

(v) The overheads of monitoring and operator control should be spread as evenly as possible, and where concentrated overheads are unavoidable they should be arranged to occur at insensitive times in the sequence of operations (e.g. when neither machine is expected to be transmitting).

1.2 Off-line Testing

The successful development of the types of telecommunications system discussed here depends greatly on the good will of the owners and system programmers of the large machines to which connection is made. Goodwill cannot be expected to survive frequent system collapses in a large and busy machine, caused by a malfunctioning simulated terminal, and all reasonable steps must be taken to prevent such collapse. The software in the central machine is, in principle at least, probably designed to be capable of surviving terminal failure, but even so it would be foolish to use that as a pretext for carelessness.

Ideally the simulator would be designed to fail in the same way as its hardware counterpart; this is hardly a feasible design aim, but it is possible to go some way towards ensuring that the behaviour of the simulator is in accordance with one's current understanding of the intended behaviour. One further desideratum may therefore be added to the list.

(vi) Facilities for testing the terminal simulator without requiring on-line connection to the central machine should be provided. Off-line operation should be designed to test as much as possible of the software to be used on-line, under conditions approximating as closely as possible to those expected to obtain on-line.

2. Link Software Organisation

The software used to control the details of telecommunication is organised in two main sections: a low-level section looking towards the hardware, and handling interrupts and the transmission of individual characters, and a higher-level section looking towards the file structure and command levels of the system. These sections are broadly equivalent to the handler and environment of the slow link, but, as will be shown, there has been some redistribution of detailed activities in the later systems.

The overall mode of operation is as follows, it being remembered that data manipulation is in general done either beforehand e.g. for T commands on the 360 link or afterwards (e.g. for R commands). The link program as a whole is a subroutine of INDR1, and is called with a set of parameters according to the command to be executed; only one command is executed at each call.

An initialisation routine interprets the parameters, and then calls a driver program. The driver is the main part of the higher-level section, and is used to break up the overall communications process into a sequence of steps as required by the protocol. Each step is executed as a call to the low-level section, the handler, where appropriate character strings are transmitted and responses or messages are received.

Incoming control strings are analysed character by character, and message blocks are assembled and packed. All necessary parity and block checking is done, and each call returns to the driver with an exit parameter indicating the type of message or control string received, and the presence of any errors. The exit parameter then influences the choice of the driver's next call.

The overall operation of the driver and the low-level handler may be likened to that of a sequential machine, where the states correspond to the various calls made by the driver, the inputs correspond to the received messages, and the outputs to the transmitted messages. The low-level operations of the read interrupt routine in the handler are also conveniently described in sequential machine terms, as will be seen later.

Interwoven with driver and handler routines are various routines used for monitoring. These include switch-controlled probe routines used to detect particular items of interest, and common routines for assembling monitor files and displaying the real-time running commentary.

Calls to backing store (drum in most cases, but sometimes dectape) are made through the DECASS system routines, and run concurrently and asynchronously with link operation. Run-time operator contact is also made asynchronously, using the teletype.

The foregoing general description is equally applicable to the fast link and the CDC link, and the organisation will be seen to bear a family resemblance to that of the slow link. The framework was established firmly during fast link development, and was carried over entire to the CDC system. Indeed, a significant amount of detailed code was also carried over, especially in the monitoring routines. The experimental form of the fast link is described in [15], the results of a preliminary investigation into the CDC link are presented in [16], and the experimental form of the link at the time of writing (apart from minor changes) in [17].

3. Driver Programming

One of the problems in system design is, as shown earlier, the lack of certainty about the protocol. The uncertainty is greater at the level of sequences of messages than at that of sequences of characters within a message, and the difficulty of program amendment at this level is greater too. The control of message sequence, as distinct from character sequence, is the specific function of the driver: each call it issues to the handler results, in general, in a particular message being sent. It follows that there is an advantage in making a driver program easy and quick to alter, while keeping it insulated from other parts of the software so that changes to the driver do not produce side effects elsewhere. These considerations suggest that a language of higher level than assembly language could well be used.

The possibilities open to us within DECASS were to use Fortran or the macro facilities of the assembly language. The need for efficient object code precluded Fortran, and the desire for speedy conversion from one driver to another precluded both Fortran and macro. (To change a conventionally coded routine requires editing, reassembly and reloading, and may take 10-15 minutes even for a trivial change.)

A simple language was therefore devised specifically for writing fast-link driver programs [18]. This is discussed in some detail in Appendix 1, and summarised here. The main requirements were

(i) it should be capable of defining calls to the handler, and of using the handler's exit parameter to control the sequence of calls

(ii) it should be readily compiled and reasonably fast in execution

(iii) it should allow easy design and amendment of drivers

(iv) it should be simple to implement

Driver programs are usually held (in source code) on a backing store, and the particular driver required for any run is specified by the operator at the beginning of the run, all the rest of the software being already in core. Thus no unnecessary time is lost if the driver is to be changed between runs. It is also possible to compose a driver directly at the keyboard. A special assembler for driver programs is incorporated into the link software and is called automatically during system initialisation.

A driver written in this language may contain up to some 15 labelled lines. Each line contains one macro name, corresponding to the call to be made by the driver, and a switch table which specifies for each possible value of exit parameter the label of the macro next to be executed. A few special facilities were added: the label S means stop, W means wait (i.e. for the operator to select which macro should be executed next), and L means loop (i.e. repeat a given macro or set of macros some predetermined number of times).

There are eleven values of exit parameter that can lead to some further action, and a few that indicate catastrophic failure. Not all parameter values are possible for all calls, but for simplicity no account was taken of this in designing the language or its assembler. The assembler translates each macro into a subroutine call followed by a set of jumps corresponding to the exit switch, followed in turn by the parameters of any loop specified in the macro. The translated macro is efficient, but no attempt is made to economise on core since this type of driver is only used during development of telecommunications software. Once the correct protocol has been established, it is a simple matter to hand-code the correct set of calls at the same time as tidying the program generally and removing any other unwanted experimental features.

The driver language was originally devised for the fast link, and the macros and exit parameters are listed in Appendix 1. One point of interest when starting work on the CDC link was to discover how well or badly the driver language would serve in the very different conditions of that link. It will be remembered that in the fast link the normal contention-mode protocol was modified by a higher-level convention that the PDP would always start any interchange of messages; the driver approach of "Send something and wait for one of a variety of responses" was clearly appropriate.

The CDC link, on the other hand, operates in almost the opposite manner: the central machine polls repeatedly, or sends write messages only a few of which are wholly dealt with automatically by the terminal. The response to a poll, and to a lesser extent the response to a write message also, depends largely on internal conditions of the terminal software: whether it has a command or text to send, and so on, and hence the sequence in which driver macros are executed is only partially determined by the nature of the messages received from the 6600.

The CDC link driver macros and exit parameters are also given in Appendix 1; comparison of these with those used for the fast link will show that the same general pattern has been followed. As will be shown later, a common routine is used by all driver macros to await the response for the CDC, and it is this routine that generates the exit parameter corresponding only to the type of response received, and taking account of any error conditions detected. The exit parameter range is insufficient to take account also of prevailing internal conditions, so following a poll or write message control passes to one of two special switch macros in which these conditions are used to give a subsidiary control of the choice of macro to be executed next. These switch macros are also specified in the Appendix.

3.1 General Conclusions on Drivers

There can be no doubt that the use of a suitable high-level language for driver programming is well worth while. In the two systems for which the language described has been used, it has never been necessary to write a driver of more than 16 lines, and it is easy to rewrite a driver completely in 10-15 minutes. It is also almost impossible to make coding errors in a driver (the assembler contains checks for errors such as the omission of a line), although logical errors due to misunderstanding of the protocol can, of course, be made. There also appear to be no low-level side effects.

The driver language as described is far less well suited to the needs of centre-controlled telecommunications systems such as the CDC link than it is to those of satellite-controlled systems. The reason is simple: the driver language is essentially designed for situations in which the satellite takes the initiative in any interchange of messages, that is, where the role of the central machine is mainly confined to issuing appropriate responses. In a centre-controlled system it is the satellite that has the more limited role, and the more appropriate form of software would be one that is driven rather than one that drives.

But there seems to be no reason to suppose that the design of a language for one type of system would be any more difficult than for the other. The reasons for not redesigning the language for the CDC link were simply that the need for change was not known until the language had been tried, and that there was a certain software investment in the assembler.

Ideally, of course, one could wish that there were system support available to allow the use of special-purpose languages at a level slightly higher than assembly language, and so save diverting effort on to comparatively unproductive tasks such as the development of an *ad hoc* assembler. This point will be taken up again later.

4. Low-level Software

Insofar as differences of transmission protocol and message format will allow, the low-level software for the two links is similar. The data adaptor in the PDP-9 gives an interrupt for each syllable received or transmitted, and it is therefore convenient to examine the characters of an incoming control sequence as they are received. (Hardware changes have been proposed, and partially implemented [19], to allow some automatic character identification, under program control. Further changes will allow message blocks to be packed or unpacked by hardware and to be received or transmitted by way of the data channel. This will reduce the number of interrupts to one per block, but will necessitate certain changes to software organisation, especially to deal with check syllables.)

The development phase for the 360 link software was completed during 1970, as already described, and the organisation of the software is described in the references. Work is still in progress on the software for the CDC link, and less has therefore been published on this system. The general structure of the software has been established, however, and in the absence of a fully-detailed published account some features will be discussed here in order to give specific examples of techniques used in both 360 and CDC systems.

4.1 Reception

It will be remembered that after any message has been transmitted from the PDP the software waits for the next message from the CDC, and that the exit parameter of the driver macro is determined by the nature of this received message. During reception the program is in a wait loop, and apart from monitoring and any keyboard input the only interrupts expected are those caused by syllables entering the data adapter.

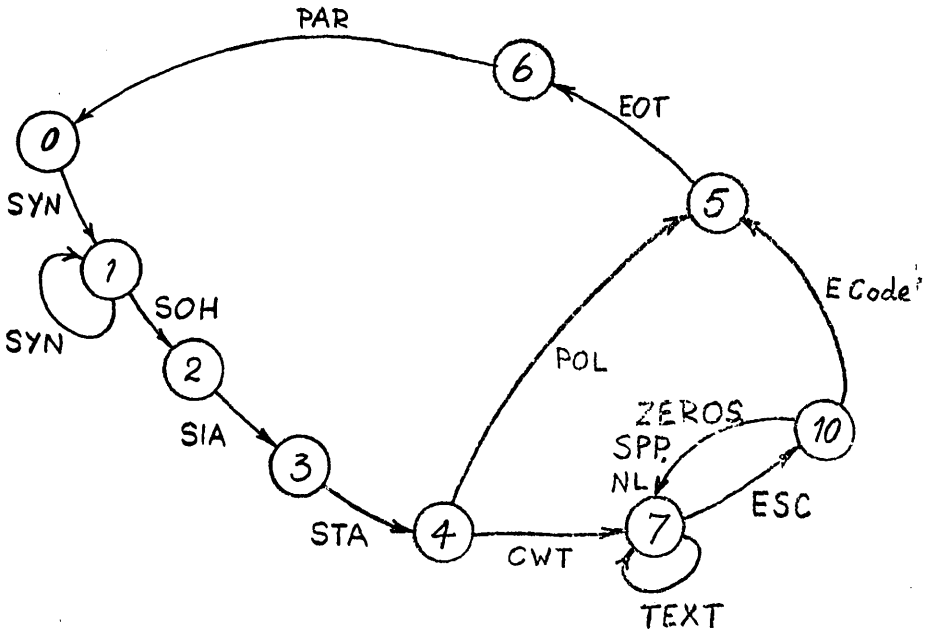


Fig.1 State Diagram for CDC link Reception

A simplified state diagram for the receive routines is given in Fig.1. The initial state 0 is distinguished from the others by the fact that the data adapter is set to a quiescent condition in which it will only respond to SYN. Once a SYN has been recognised, the routine moves from state to state on receipt of the various characters shown. The CDC is liable to insert strings of SYN more or less arbitrarily, so except in states 0 and 6 a received SYN is always ignored. Incoming control characters are checked for validity; invalid characters in the early part of a message cause a return to state 0, not shown in the diagram for simplicity.

The message type is determined by the character received in state 4. Any subsequent error causes the routine to move to, or remain in, state 5, where EOT is awaited. Text is assembled and packed in states 7 and 10, the latter being used to cover the various possibilities following an ESC.

When a message is expected, and the wait loop entered, a 5 sec timeout is set to provide an error exit if no message is received. This timeout is increased to 30 sec if the message proves to be a write message. There is also a hardware check on line failure: a break in the carrier will cause an interrupt. Timeout and line failure both result in flags being set which are tested in the wait loop and cause immediate exit. In normal operation, exit from the wait loop occurs when the parity syllable has been received in state 5.

The three types of write message require slightly different treatment. The simplest case is the E3 message ("read more cards"): this consists of + ESC EC3 only, and the identification of the E code in state 10 causes an exit from the wait loop with the appropriate exit parameter. Messages for the printer contain in general more than one line; each ends with a newline character which is identified in state 10 and causes a flag to be set. This flag is detected in the wait loop where it causes the assembled line to be written to drum (assuming no errors have been detected). After the last line there may be formatting characters, and the whole ends with ESC EC2 which causes the file to be closed. Messages for the display may be of one or more lines; multi-line display messages are written away in the same manner as messages for the printer, since they are indistinguishable from printer messages. The last line (or only line in the case of single-line messages) ends with ESC EC1; if any lines have been written to drum the last line is too, and the file closed, otherwise the line remains in the receive buffer until it can be displayed. Any number of write messages can be accepted, and if written to drum are arranged to occupy separate files.

4.2 Transmission

Although the CDC will insert SYN strings into write messages whenever the next message component is temporarily unavailable, experiment showed that it would not accept SYN strings in read messages from the PDP. Consequently the transmission software had to be written in such a way that the next syllable to be sent was always ready when the transmit interrupt occurred.

From the point of view of the transmit routines there are three kinds of message:

(i) Fixed messages (ACK, REJ, ERR, EC2, EC3), held wholly in core

(ii) Commands, assembled in core as a one-line message

(iii) Text, read line by line from drum, transmitted in blocks of up to 12 lines, each line being converted to card image.

A common routine for all transmit macros, with one parameter to define the message required, is used to initialise the transmission process, send a special character (in fact a 360-link SYN) to activate the communications adapter, and place the first CDC-link SYN in a one-syllable buffer. Thereafter all transmission is interrupt-controlled, the syllable in the buffer is sent, and the next syllable is selected in one of a variety of ways according to the point reached in each particular type of message. The program remains in a wait loop until transmission is complete.

The parameter passed by the macro defines an execute table that specifies the sequence of message components required, in terms of the routines that supply them, and gives particular syllables to be incorporated into the standard sections of a message. The TXT macro is somewhat more complicated than the rest, since it is in general called a number of times to control transmission of the job. The initial contents of the execute table for this macro are set at the first call to TXT to control transmission of the job description cards (held in core); subsequent calls to TXT use the execute table as it was left at the end of the previous call.

The essential features of the interrupt routine are shown in Fig.2. The flag MORE is set > 0 before the transmit wait loop is entered, and the program remains in this wait loop as long as MORE ≠ 0. MORE is set to -2 when EOT is about to be transmitted, because only two more interrupts are required to complete the process: those for EOT and the parity syllable.

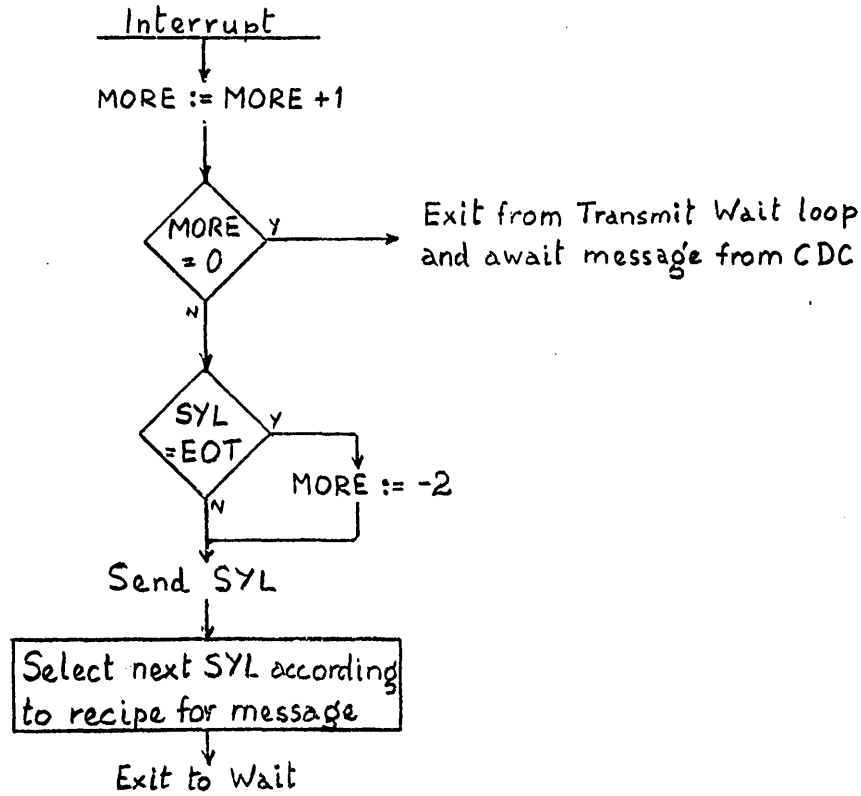


Fig.2 Flow Diagram for CDC link transmit interrupt

A standard buffer SBUF1 contains the first part of all sequences (SYN, SYN, SYN, SYN, SOH, STA) and is completed during initialisation by inserting the current value of STA and the control character appropriate to the message (ACK, REJ, ERR or STX). A second buffer SBUF2, contiguous with SBUF1, has space for ESC, E code, EOT and PAR (the parity syllable is computed in this space).

For fixed messages SBUF1 and SBUF2 are treated as a single buffer and transmission is controlled only by the MORE mechanism described above. (In responses there is no ESC, E code; the word allocated to these is left null, and the unpacking routine is designed to ignore a null word.)

Text lines are terminated by ESC, which is the character substituted for ASCII carriage return by the code converter. This ESC is detected, but not transmitted.

For commands the transmission recipe is as shown in Fig.3. TBUF is a special buffer used to assemble commands input at the teletype.

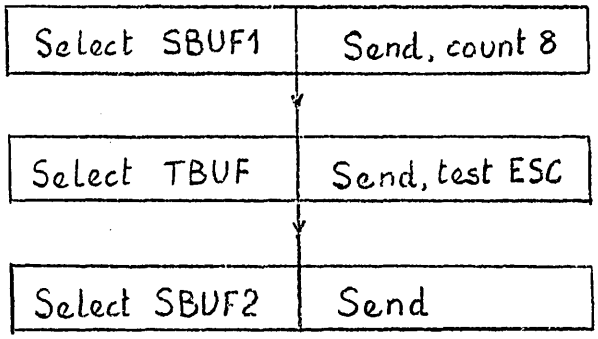


Fig.3 Recipe for Command Transmission

In this diagram, and in Fig.4 following, the boxes labelled SELECT.... correspond to routines specified by the execute table. Two final selection routines are used, one concluding when a preset total is reached, and the other when ESC is detected. In both cases, conclusion of part of the message causes application to be made to the execute table in order to determine the next selection process. The last SEND box uses the counting routine with the count preset to a value that will not be reached, since at this stage the final conclusion is determined by the MORE mechanism.

Fig.4 shows the recipe for text transmission. This diagram is somewhat idealised, but gives a general impression of the process. JCARD is one of three buffers containing the job description card equivalents, fixed in the experimental version of the system, but in later versions to be constructed according to parameters input at the start of a run. COUNT 80 means count until the total number of characters in the card is 80, i.e. pad with spaces to card image.

The read card flag is detected in the wait loop between interrupts, and causes the next line to be read from drum. If it does not arrive in time due to some error condition, or if the file on drum is found to be corrupt, transmission is terminated in the manner specified for a card jam in a 200 terminal, that is, ESC EC2 is sent.

The DECASS drum handler has a special return code to indicate end of file. This is used to set a flag which is detected when the next card is due to be transmitted. (The flag is identified as EOR in the diagram because in CDC terms each drum file corresponds to one of the records comprising the job.)

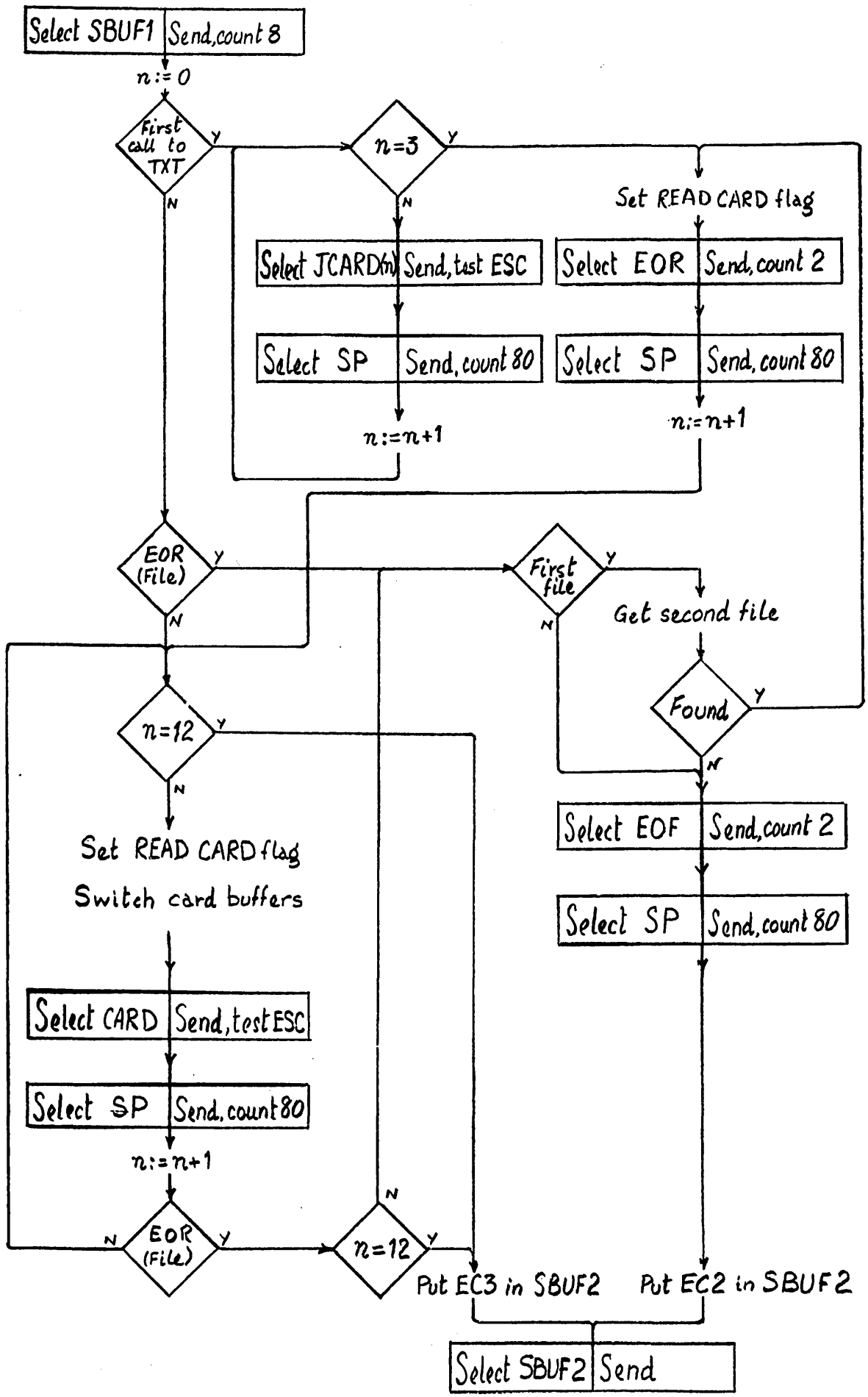


Fig.4 Recipe for text transmission

Depending on whether the drum file just sent is the last or not, an end of file card (ESC EOF) or end of record card (ESC EOR) is sent next, either in the current block, or, if twelve cards have already been sent, as the first card of the next block (i.e. at the next call to TXT).

5. Monitoring Facilities

The slow link monitoring facilities were grafted on as an afterthought, and were in consequence neither as effective nor as elegant as could have been wished. When the fast link software was planned, extensive monitoring was designed in from the outset, and proved so useful that very similar facilities were later put into the CDC link. Some of the monitoring routines are identical in both systems; the rest differ only in minor details.

5.1 Running Commentary

Each driver macro leaves a record of its call and its exit parameter in the form of a short comment on the display screen. A rolling display is used, to keep flicker within bounds, and at any time the last 10-15 macro records are visible, interspersed with any messages that may have been received for display.

The display is controlled by a routine, common to all macros, that determines which macro has just been executed and displays its name and its label in the driver. The existence of a message for the display (e.g. E1 messages in the CDC link) is detected at this stage, and the message is added to the screen. The same routine also selects the next macro, according to the current exit parameter, and takes care of irrecoverable errors in an orderly manner.

5.2 Low-level Monitoring

The results of the low-level monitoring processes are intended for post-mortem analysis and are accordingly assembled into a file on backing store. Dectape is used in the 360 system, but the rate of accumulation of a monitor file in the CDC system requires the use of the drum. The higher rate is due to the larger number of short messages handled, and the persistent polling by the CDC.

The events to be monitored in any given run are selected by presetting handswitches on the PDP console. There are minor differences between the facilities in the two systems, but broadly speaking the following list is applicable to both.

(a) All, or a selection of, received syllables

(b) All, or a selection of, transmitted syllables

(c) Internal states of the receive routines

(d) Error indications, recorded when the error is detected

(e) A code indicating each driver macro as it is called

(f) The exit parameter from each macro

(g) Elapsed time from the beginning of the run, in units of 100 ms

A separate subroutine handles each monitor point and generates a 5-character packed ASCII word-pair. These are combined into lines by a common routine. It is simpler to operate in this way than to attempt to post-process binary data, because the separate monitor subroutines can insert identifying characters as required. The disadvantage is that these routines increase the software overheads, in some cases at points where the timing may be critical, but care has been taken to design the routines so that the overheads are spread as evenly as possible. It is for this reason that each subroutine generates packed ASCII directly, since otherwise a line-by-line call to a packing routine would be required.

Sixteen line buffers are held in core. These are filled circularly by the monitoring routines, and emptied circularly by a backing store writing routine. This latter routine is only called when the system is expected to be reasonably quiescent, for example, immediately after receiving or transmitting a message.

In the early stages of development, the full set of monitor points is certainly required, and the monitor files, although tedious to analyse, have often been the only possible means of discovering the reason for collapse. They have the additional advantage that they demonstrate the correctness or otherwise of assumptions made about protocol, whether collapse occurs or not, and are hence valuable even if the software is constrained only to listen for transmission from the remote machine. This constraint has been usefully applied (by choice of a suitable driver) when first putting a new set of link software on line (see also Appendix 4).

As development proceeds, it is possible to omit some of the monitor points and thereby reduce the length of monitor files so as to simplify analysis. For example, once the receive routines are reasonably proven there is little need to record their internal states.

6. Off-line Testing

Testing telecommunications software off-line imposes an additional burden on the computer, that of simulating transmission from the remote machine. If tests are to be realistic, they must satisfy four conditions:

(i) the simulated input must correspond to that expected from the remote machine in a comparable on-line test

(ii) the rate of input (syllables per second) must be at least as high as the telecommunications rate

(iii) as much as possible of the software used on-line should be capable of being tested off-line

(iv) the additional overheads of off-line testing should be minimal.

The first of these conditions has two aspects: the content of an input string should correspond to that of a comparable message, and the relative timing of input strings and of responses output by the local software should correspond to the relative timing of an on-line interchange of messages.

In practice, it is the first condition that has the greatest influence, especially in the early stages of development when transmission characteristics are only imperfectly understood. Evidence obtained from monitor files made when receiving on-line, even without transmission, allows improved simulations to be made. Use has also been made of A.R. Duncan's TOPSY program to obtain information on transmission characteristics (see Appendix 4).

Condition (iv) implies that the additional software used should be as simple as possible, even at the cost of limiting the scope of tests. In the PDP system it is convenient that the paper tape reader, when up to speed, reads at some 300 ch/sec; this corresponds closely to a 2000 or 2400 baud transmission rate. Hence the easiest approach is to construct a data tape containing the desired input string, and to read some of it whenever the simulated remote machine would be expected to transmit. The off-line interrupts from the tape reader can be serviced in the same way as on-line interrupts from the data adapter, thus largely satisfying condition (ii).

The technique described has been successfully used in all the telecommunications systems developed. A slight amendment was required for the slow link: the characters on the data tape were spaced out with ignorable blanks to give a data rate corresponding more closely to 600 baud.

There is no need to use any device to simulate transmission from the local machine; the fact that the syllables that would be transmitted on-line are monitored provides sufficient evidence of their correct selection. It is, however, desirable to simulate the transmission rate; this is readily done by incorporating a short delay loop into the off-line software.

The conditions for off-line or on-line testing are set up during initialisation of the link software, and may be reset between runs without reloading being necessary. As an example of the extent to which condition (iii) can be satisfied, the part of the experimental CDC link software that is active during a run amounts to some 2000 words, of which all but about 10-15 are used in both on-line and off-line operation. The control of off-line operation requires about 30-40 words of special routines.

CHAPTER 6 : Recent Developments in Indra Software

1. Introduction

The experimental stage of fast-link development, using high-level drivers, was completed early in 1970 when a definitive driver had been established, and the low-level handler had been apparently freed from bugs. The next step in development was to replace the high-level driver by an equivalent organisation of low-level software, and to tidy up certain details; this work was undertaken as an MSc project by R.M. Humphreys [5], and the first part of this chapter contains an account of the link software as it appeared in the autumn of 1970.

Subsequent work (by P.L. Higginson) has been devoted to improving the general utility of the Indra system by making use of the DECASS overlay facility discussed in Chapter 1. At the same time there has been further revision of the fast link system to satisfy some changes in user requirements. The second part of this chapter contains an account of the software as it exists at the time of writing; it is expected that few if any further changes will take place while the system continues to be based upon INDR1 [20,21]. Some of the possibilities for future reorganisation of the software under the New System will be discussed in Chapters 7 and 8.

2. Fast Link: A First Production Version

The main part of the work in developing the production version of the fast link system was to replace the high-level driver. The result was to produce a unified handler running as a subroutine of INDR1. At one stage it was planned to remove all monitoring and off-line facilities from the handler, making them available as an add-on extra to be selected at load time. Experiment showed however that the resulting software was very clumsy, and it was decided that since some monitoring (if only a displayed running commentary) would usually be required, the necessary routines might as well be left in place. These routines are essentially as described in Chapter 5, and are not further discussed here.

A brief account of the final organisation of the combined driver and handler follows. A more detailed description will be found in [5].

2.1 Initialisation

The handler is called from INDR1 with a set of parameters defining function (R,T,S,C or D in the command), PDP file description and 360 job description. The first stage is to build the command to be transmitted to the 360; this is held as a double-image packed message in core. The formats of these messages are:

R (receive) : <data set> 0. <first 3 chars of file name>

T (transmit): <data set> 1. <first 3 chars of file name>

S (start) : 377 2 <job name>

C (cancel) : 377 3 <job name>

D (display) : 377 4

(Note: There has been a recent change to command structures, for which see section 4 below.)

The initialisation section then sets up the various internal connections for handling interrupts, selects on-line or off-line operation, inserts monitor probes as specified, initialises local peripheral handlers, and starts the clock. It also transfers the teletype from being under the control of KM9 and makes it available for run-time input of chat messages.

One practical problem that was encountered during earlier development was that if the 360 software was not loaded when the PDP sent its first ENQ, the 360 end of the link was never able to recover, and the whole process had to be re-started. (A similar problem applies in reverse too, since the PDP collapses if it is interrupted by a unknown device - i.e. one that has not been initialised. This situation cannot occur with the 360 link, since the PDP sends first, but can easily occur with the CDC link if the correct manual operating sequence is not followed.)

The normal practice was to agree by telephone with the 360 operators when a run may start. Even so, loading the PDP software (even under DECASS) is liable to be quicker than loading the 360 software, so on the first command after loading the software the initialisation section inserted a one minute pause to ensure that the 360 was ready and waiting (see also section 4).

2.2 Driver/Handler Control

A set of routines is provided at the low-level end of the handler to control transmission of the various message sequences, and to await and assemble the response. These routines are substantially the same as were used in the experimental system, and similar to those described in the previous chapter; they will not be described here. They have a common entry point, and are called with a parameter selected from a table.

Three tables are used by the driver. One of these contains the call parameters, which are 9-bit integers made up of three fields. The least significant four bits are zero in the call parameter; the handler inserts a code into this field according to the outcome of the call. (These codes are the same as the exit parameters discussed in Chapter 5 and Appendix 1). The next three bits in the call parameter define the action required (c.f. the call parameter corresponding to the driver macro). The most significant two bits distinguish different circumstances in which a given action may be performed.

On return from the handler, the composite call and exit parameter is used as a key in a table-lookup routine; this is a standard routine in INDR1, used also, for example, in INDR1 command decoding. Failure to find the parameter in the table indicates an irrecoverable error; files are closed, and the handler exits to INDR1.

If the key is matched against an entry in the table, the serial number of the entry is used to select the next call parameter from the table already mentioned. A third table is also selected similarly to determine any pre-processing required before the handler call is made.

2.3 Read Interrupt Routine

The organisation of the read interrupt routine is very similar to that described in Chapter 5 for the CDC link, and, as already explained, supplies the exit parameter field in the sequence control parameter. Received text is packed into blocks in double image, for subsequent writing by the normal DECASS drum handler.

One special point is worthy of mention. It is possible for a character to be received from the 360 at a time when the PDP software is not able to deal with it properly: for example, while the PDP is transmitting. This is usually due to some error condition causing the two sets of software to get out of step. In some circumstances the condition may be recoverable, or ignorable; appropriate action must be taken at the driver level rather than the handler level, since the action required is dependent upon the stage reached in the overall transmission sequence. Consequently the handler sets a flag if such an interrupt occurs, and on the next call from the driver level an immediate exit is caused, with a distinctive exit parameter.

2.4 Use of the Teletype

In normal circumstances, that is, when the teletype is owned by KM9, the effect of typing control C is to return abruptly to KM9, and that of control P is to return to INDR1. When control of the teletype is transferred to the link program, control C continues to have the same effect, but control P allows all files to be closed before exit, so that any successfully received data and any monitoring records are preserved.

It may be noted in passing that control P is a dangerous facility to use: it causes an interrupt, like any other character input at the teletype, but the interrupt is not serviced properly in that exit is not back to the interrupted program. Thus the programs may be corrupted, and reliable restart can not be guaranteed. The effects are most serious in handlers, where a peripheral operation may not be able to terminate properly.

The teletype is taken over so that chat messages may be composed during the run. Operator-to-operator communication has not been implemented; in fact, the characteristics of the 360 software seem to be such that it is not possible to arrange for the 360 operator to communicate with the PDP, although communication the other way is feasible. So chat messages are only sent from the PDP to the Indra software in the 360; chat messages received from the 360 are displayed.

A chat message is assembled in ASCII; it is not converted to EBCDIC as it is being keyed in because the code converter might concurrently be in use, so the message is converted by a preprocessing step before transmission. Transmission takes place before the EOT of the current (or next) outgoing message, and hence there will be an indeterminate length of time after one chat message has been keyed in before input of the next may start. Various notices are displayed to indicate to the operator the availabilty or otherwise of the teletype.

3. The Indra Overlay System

The DECASS overlay facility is described in Chapter 1, and a brief mention of its use for the Atlas link was made in Chapter 4.

By the autumn of 1970, a considerable body of software had been developed by the Indra group, including the various local and remote systems described here together with a quantity of display software and an interactive editor based on the use of the Computek. If conventional loading is used, it is not possible to combine these various parts, that is, one may have only one of Atlas link (under INDR1), fast link (under INDR1), display software (independent of INDR1) or Computek editor (DECASS-based) at a time. The various local utilities described in Chapter 3 could be loaded with either the Atlas link or the fast link. It was further envisaged that more display software and the CDC link would become available.

Conventional loading is somewhat slow and laborious, and must be manually-controlled. It is therefore not possible to load, say, the Atlas link or the fast link as a direct consequence of entering an A or R command to INDR1. The use of overlays was thus indicated, but it was found that fairly drastic alterations to the DECASS overlay facility were required.

To be able to use INDR1 as a general control program for those parts of the system that had been designed to run under it, it was necessary to locate it in the resident area of core, that is, below the boundary established as the bottom limit of the overlay area. It could not be built into the correct position when the chain containing it was set up, because the DECASS chain-generating program occupies the same area and more. Hence a modified, pseudo-relocatable version called INDR2 was constructed, together with a core-to-core loader that would move it down in core and adjust the position of the boundary. Various tables were also set up to allow communication between chains and INDR2.

The first version of the execute file to be established occupied some 200 blocks on dectape or drum, that is, nearly 50,000 words. One reason for the large size of this file is the DECASS requirement that every overlay should contain all relevant device handlers. A later modification was made so that many devices are loaded initially by the DECASS system loader and thereafter remain resident. Several advantages have resulted from this change, among which may be noted

(a) that the execute file has been reduced to about one third of its original length, thus releasing space on the backing stores,

(b) that overlay loading is faster, and

(c) that the user has freedom of choice of local peripherals, since they can be assigned conventionally before a run.

It is worth noting that in order to obtain the present overlay capability it was necessary not only to alter Indra software, as has already been mentioned, but also to change KM9, the system loader and the overlay loader. The changes to DECASS programs were fairly small, but involved a great deal of investigation of unfamiliar software whose documentation was by no means as complete as that of the Indra software.

4. Revision of Indra Programs

In the course of preparing the Indra overlay system, opportunity was taken to augment the facilities offered, and to make some changes to the fast link command structure. These modifications are noted briefly below.

(a) It is possible to use either the teletype or the Computek as control console, the choice being made either initially or at any time during a run.

(b) A new command has been included to allow the listing of a backing store directory.

(c) Some problems due to an incompatibility between ASCII and EBCDIC have been resolved by amending the code converter, and a further amendment converts unidentifiable characters into a recognisable form instead of terminating conversion as hitherto.

(d) Some trouble was experienced with the fast link handler when messages were to be printed on the teletype during link operation. This has been overcome by only allowing a message to be set up for printing if printing is complete for the previous message. Error indications and 360 chat messages that previously only went to the display are now printed as well (subject to the reservation above), so that a hard-copy record remains after the run. Some additional error indications are also given, for example, if an irrecoverable error causes exit to INDR2.

(e) An optional <job parameter> field has been defined for fast link commands. This precedes (or replaces) the <data set> field in R and T commands, and follows the <job name> field in S and C commands. To distinguish the various fields, <data set> (if used) is preceded by #, and <job name> is followed by . if <job parameter> is included.

(f) The arrangements for an initial pause have been modified. Current practice is either to wait five minutes after the agreed starting time, or to ensure by telephone that the message LINE OPERATIONAL has appeared at the 360 console. The initial pause in the link handler has been retained as an additional precaution, but is now only 20 sec. It occurs after overlay changes as well as on the first occasion of loading, since these conditions are indistinguishable by the link handler.

CHAPTER 7 : Towards a New System

1. Limitations of the Present System

Some of the disadvantages of DECASS for the telecommunications projects discussed above were referred to in Chapter 1, and others have been mentioned in passing. All the software developed so far has been designed to operate under DECASS, but the necessity of bypassing system restrictions has resulted in some loss of elegance and efficiency. A few examples, some already discussed in earlier chapters, are mentioned briefly here to illustrate this point.

The resident part of the DECASS monitor (KM9) includes facilities, based on the use of the CAL instruction, to initiate activities of the local peripheral handlers provided in DECASS. The CAL handler is interruptable, and consequently no action based on the use of CAL may be given in an interrupt service routine. This limitation introduced some problems into the design of link transmission routines such as those described in Chapter 5.

The teletype keyboard handler in KM9 is similar in most respects to handlers for other input peripherals. In particular, it will only accept input data if a CAL-based read command has been given, and the system then waits in the teletype handler until a line or a character has been input. There is thus no means of accepting input at the operator's discretion, the mode of operation required for the telecommunications system. Consequently it was necessary to build special-purpose keyboard handlers for the 360 and CDC systems, and to arrange that control of the keyboard is transferred from DECASS to link software at the start of a run. Control must be transferred back at the end of the run, which is straightforward if the run terminates in an orderly manner but is almost impossible if software collapse occurs. The KM9 keyboard handler remains in core, of course, and thus represents so much wasted space.

The presence of the resident KM9 also wastes space on account of other routines contained in it that are not used by Indra software : the clock routines are examples.

KM9 traps a variety of errors that commonly occur during software development, such as attempts to obey data, or illegal peripheral commands. It is, of course, reasonable that these should be trapped, but the trapping mechanism does not allow easy recovery in cases where recovery would be possible, and is liable to give misleading indications of the cause of failure. Errors that could with advantage be trapped are attempts to reference non-existent store; unfortunately store protection is not implemented in KM9, so no trapping occurs: this type of error is wholly irrecoverable and often most difficult to trace.

The Indra executive described in Chapter 3 is designed to allow the use of all permanently-connected local peripherals through the normal DECASS handlers. As dynamic loading is impossible in DECASS, and as the loader cannot be constrained to accept the omission of programs that may not be required on a given occasion, it is necessary to load a full set of the standard handlers. These are all comparatively large programs since they provide a general set of facilities many of which are of little or no use in Indra work, and they include permanently-assigned buffer areas. They thus occupy much space, often to little purpose, and their mode of operation tends to result in a frequent need for largely redundant core-to-core data transfers. The associated problem of the interruptable CAL handler has already been noted.

The problem of storage has been alleviated to some extent, as mentioned in Chapter 3, by eliminating buffers from the various Indra programs and using a simple store allocation routine to obtain working space from the free area available once the loader has completed its operations. But. the buffers of standard handlers cannot be supplied in this way, of course, since the handlers are used in other contexts where the store allocation routine is not available. In any case, the task of modifying a standard handler, or any other item of DECASS software for that matter, is not one to be undertaken lightly.

The result of these limitations on space has been that it is only possible to load the software for one of the links at a time. The operator wishing to use more than one link has therefore more work to do than could be wished. Some relief has been obtained in recent times by means of the DECASS overlay facility, as described in Chapter 6, but the would-be user of the Indra system still has a considerable amount of keyboard operation to do.

As long as the operator is a skilled programmer, or is at least moderately familiar with the intricacies of the system, operational complexities may be held not to matter. But the original intention was to develop a system that would be of use to the designer wishing to use CAD techniques, and it it unreasonable to expect a user in this category to become expert in the subleties of computer operation. The ideal system would have a simple user-oriented command structure, readily learned, and be so arranged that the user could concentrate upon the problems of his own application. Few systems, if any, attain to this ideal, but that is no reason to forego the attempt to develop one that does.

Another problem very different in kind, results from the pattern of usage of the ICS PDP-9 installation and the characteristics of telecommunication systems. The PDP is very extensively used, mainly for program development, and part of its usage is by research groups and students at the Institute who have little or no connection with the work of the Indra group. Even when use of the machine by the Indra group and its associates is considered on its own, only a comparatively small part of the total involves connection to one of the three remote machines. Yet the time spent connected to these machines is disproportionately large, and for a large proportion of this time the PDP may be more or less idle.

The reason for this state of affairs differs in the three systems. The 360 is in many ways the best of the three: provided that connection can in fact be made at the time agreed with RHEL (no small proviso this), telecommunication is reasonably efficient. Files can be sent without undue delay, and jobs started remotely in the 360; the connection can then be broken and the results of the run will be held in the 360 until they are requested at some later time from the PDP.

This mode of operation is possible because the Indra software in the 360 was designed to permit it. The CDC link, in contrast, is designed to simulate a wholly-owned terminal of the CDC which, in normal operation, is on-line all the time. Jobs sent for processing take their turn at the central machine, and results are automatically returned in the fullness of time to the terminal. If the PDP system were to operate in this way, the CDC link software would have to remain resident and idle until the results were ready; the probable time delay could often be intolerable. In the experimental work done so far it has not been necessary to face this problem, but clearly it must be faced if a fully operational system is to be developed. Enough has been said

about the DECASS system for it to be clear that software cannot be loaded automatically as a result of an interrupt from a remote machine.

Considerable delay is also possible with the Atlas link, for a different reason. Here the first part of transmission always takes place promptly, because the Atlas supervisor reads in the job description as soon as the tape reader is engaged. But the reader may then be forced to wait if the input queue is full; the delay may be a matter of minutes, and in unfavourable circumstances there may be further delays before input is complete. The PDP has no option but to idle if transmission is delayed; it is not possible to suspend file transfer and later resume from the point reached.

To summarise this discussion of present limitations it may fairly be said that the development of communications software and of operational procedures using communications cannot proceed any further within the framework of the current approach to PDP system software; that the user facilities available are restricted; that operation of the system is unduly complicated; and that machine time is used inefficiently.

2. Proposals for Redevelopment

It had become apparent by the end of 1969 that the difficulties discussed above would in due course limit development and lessen the usefulness of the Indra system. A paper published early in 1970 argued the case for a radical reappraisal of the whole approach to system software design [22]. Work on the New System, as it has come to be called, has continued intermittently since then, as other duties have permitted, one major advance having been a MSc project by P.L. Higginson [23]. The New System is still far from complete, and in any case this is not the place for a lengthy description of it; a brief account will be given, so as to indicate some of the ways in which it is hoped that the New System will alleviate the shortcomings of the present system and open the way for future developments. The account is not wholly hypothetical; various items of software have been written and at least partially tested.

Applications of the PDP can be divided into two classes: those in which the user requires more or less continual contact with the machine, as for example in program development or interactive graphics, and those in which he does not, as for example in most file processing, including telecommunications. The New System allows these two classes of activity to be practised concurrently in a simple multiprogramming environment. Simplicity is an essential concept in New System philosophy: the PDP is a simple machine in many ways, simplicity is the keynote of the operational procedures required of the user, and the facilities to be provided are kept as simple as possible consistent with usefulness and adequate flexibility. It would be ridiculous to design a large, complex and time-consuming operating system for a machine of this kind.

The supervisor, or resident part of the system, will consist of a set of routines to handle interrupts, administer queues, and provide primitive communication with an operator. It is hoped that this will not occupy more than 3k of core, leaving 13k to be shared between an autonomous system and a user system. The autonomous system will run all processes that merely require initiation; the user system will provide hands-on facilities for a user who wishes to interact with his software.

A console will be shared by the supervisor and the autonomous system; this will in general be the directly-connected teletype of the PDP, but any other available console could be assigned at will. The supervisor will also own part of the drum, on which most non-resident programs used by the supervisor or the autonomous system will be held, together with system programs, such as the assembler, provided for the use of the hands-on user. All other local peripherals may be allocated at will to either the autonomous or the user system.

The DECASS distinction between system and user programs will no longer apply. All system facilities will be freely available to the user, either by loading the appropriate system program, or a segment of it, into the user's area of core (in conjunction with his own programs, if he so desires), or by way of supervisor calls made by the user's programs. A wide range of data-handling routines will be provided, as they are required in any case for the autonomous system, and hence the user will not as a rule need to construct his own routines for these purposes. All input and output will be handled by the supervisor in response to calls from programs in either the autonomous or the user system. Storage protection will be used, partly to limit interference between one system and another, partly to provide a mechanism for supervisor calls that is free from the disadvantage of the CAL instruction.

Dynamic loading is a feature of the New System. A single-pass assembler has been written to accept conventional source programs, with a few restrictions where conventional facilities are not applicable, and a number of extensions. The extensions will later include means of assembling in-line code corresponding to higher-level instructions, for example, the driver macros described in Chapter 5. The assembler produces an intermediate code that is much simpler to load than that of DECASS. The loader is under development; it will be considerably smaller and somewhat faster than the DECASS loader. No explicit overlay facility is planned, but dynamic loading can give the same effect if the programs concerned are suitably designed. The loader can be called by explicit command, by the assembler (for assembler-and-execute operation in the user system), by the supervisor to load the routines required for the execution of an autonomous system command, or, again by the supervisor, to bring in non-resident routines required to deal with an adventitious interrupt from a local console or a remote connected machine. Finally, a supervisor call for the loader can be written into any program, thus providing the basis of a flexible overlay facility.

The hands-on user will not have direct access to remote machines; these will be handled only by the autonomous system. This is no serious restriction, since the user can always move to the system console and give remote-machine commands. There will be a privileged mode of operation, specifically for use when developing system programs, in which the distinction between autonomous and user systems will be removed and hands-on access to all facilities of both systems, together with some special facilities, will be made available.

The operator's calls upon the autonomous system will be made by means of a simple command language. The aim is to make this language as free from 'computerese' as possible; a provisional form has been in limited experimental use and is described in Appendix 2. Commands can be input either singly at the system console, or in batches at any convenient input device.

As a counterpart to the provision of a user-oriented command language, the New System is designed to be communicative, at least if the user so desires; to the user who is not a systems specialist few things are more irritating than a system that ruminates interminably, and occasionally delivers itself

of a few cryptic characters. It will also be possible for the user to seek guidance on system operating procedures; a simple conversational routine will be developed for this purpose, allowing the user to gain access to relevant sections of the user guide, held as a tree structure on dectape.

2.1 The Current State of the New System

At the time of writing, software at the supervisor level has been developed by P.L. Higginson to provide comprehensive device and file-handling facilities which, among other useful properties, will allow simultaneous control of all local peripherals and permit many files to be concurrently active on the various backing stores. Techniques for handling remote machines are under consideration, but not yet implemented.

In parallel with this work, the author has developed a command decoder for the type of command discussed in Appendix 2, the assembler, and a context editor, all running under a temporary DECASS-based executive. The loader and the autocode extension to the assembler are under development as student exercises.

The next step is to connect the two hitherto separate parts of the New System. The changes to the higher-level software required by this step are essentially the same as those required to incorporate any DECASS-based software into the New System, and it is therefore of interest to consider the magnitude or difficulty of any changes required.

Broadly speaking, the New System assembler will accept any program that can be assembled by the DECASS assembler, with the one exception that IO instructions will require alteration. In most DECASS-based programs, IO is handled by DECASS system macros, and essentially equivalent supervisor calls are provided in the New System. Hence the task of conversion requires, in principle, only simple editing to convert DECASS IO macros into supervisor calls. Again, the utilities in the INDR1 system have their New System counterparts, so that conversion is also reasonably straightforward in respect of these facilities.

It should be observed, however, that the simplest conversion is not necessarily the best, because many existing programs contain artifices designed to circumvent shortcomings in DECASS. As one of the aims of the New System is to provide general relief from these shortcomings, such artifices are at best irrelevant, and at worst incorrect. Care will therefore be needed in converting programs to run under the New System, but it is reasonable to expect that they will run more efficiently after conversion than they do under DECASS.

A somewhat more extended discussion of the incompatibilities between DECASS and the New System, with the arguments that have led to their adoption, is given in Appendix 3.

CHAPTER 8 : In Retrospect and Prospect

1. Introduction

Previous chapters have traced, in approximately chronological order, the development of software for a particular class of communications application, namely one in which two computers of unrelated and largely incompatible types are interconnected. The three main examples of communications differ in many respects : one (the Atlas link) is a direct connection in which the satellite machine behaves as a conventional peripheral; one (the 360 link) has associated special-purpose software in the central machine, although the interconnection uses one of the conventional telecommunications techniques; one (the CDC link) is a telecommunications system in which the satellite machine behaves as conventional remote terminal of the main machine.

In this chapter only the Atlas, 360 (fast) and CDC links will be considered. The slow link is now obsolete; were a similar slow link required at some time in the future it would be developed in a manner closely similar to that used for the CDC link since these two systems have common telecommunications features. We consider first the current position in Indra telecommunications work, so as to summarise the most significant achievements while not ignoring the inadequacies that remain. The remainder of the chapter contains a discussion of the possibilities for future work; this discussion can only be tentative, since the future is not to be determined solely on technolgical grounds; but it is suggested that past experience, allied to developments now in hand, provides a sound technological basis for further exploration of small-machine applications in coupled-computer systems.

2. A Record of Achievement

The New System will not be considered in this section, because it has had no influence on the telecommunications software so far developed; the influence of telecommunications attempts on the design of the New System has already been mentioned.

Enough experience has been gained in the development of communications software for it now to be appropriate to draw up a balance sheet, so to speak; to consider how far the software developed has been successful, and what has been achieved in the establishment of sound development techniques.

2.1 Operational Software

On the positive side we have

(i) a fully-operational link to the Atlas, in regular use for obtaining listings, and occasional use for other purposes.

(ii) an operational link to the 360, still with a few details to be improved, but providing a useful service for users wishing to develop and run software in the 360 under remote control.

(iii) a body of local software extending the facilities of DECASS and providing a unified command structure for local and remote file handling.

(iv) an experimental link to the CDC which, although not yet fully developed, is well on the way to being so and will conform, from the user's point of view, with the other telecommunications systems.

(v) software of general application, that is, not specific to any one link, for use in investigating engineering aspects of telecommunications.

On the negative side we have

(i) an operating procedure more complicated then could be desired, especially if more than one link is involved in any schedule of operations.

(ii) inefficient use of PDP time.

(iii) severe constraints on any attempt to expand the scope or usefulness of the system as a whole, and to a lesser extent of any individual link.

Insofar as users are willing and able to use the system effectively despite its shortcomings, the balance is positive. It is worth observing that among users of at least some of the facilities offered have been people who were not involved in system development, and therefore cannot be suspected of using the system merely to save face.

2.2 Development Techniques

On the positive side we have

(i) an understanding of the limitations of the PDP and of DECASS, and a variety of artifices for circumventing these.

(ii) a method of using a comparatively high-level language, readily adaptable to different telecommunications systems, to overcome problems of inperfect understanding of protocol.

(iii) an approach to the design of low-level software in terms of distinct easily-defined processes corresponding closely to the distinct types of telecommunications activity.

(iv) an understanding of the need for real-time monitoring, and a body of applicable techniques.

On the negative side we have

(i) some difficulty in applying the preferred techniques due to hardware and standard system restrictions.

Here the balance is clearly positive, although it must be pointed out that a body of development techniques, however excellent in itself, is sterile unless it can be applied to further development.

3. Balance Carried Forward

Any future development will involve a more or less radical departure from the methods of the past, at least as far as actual software is concerned. It is therefore of interest to see what can be carried forward, since this represents the longer-term benefits of the work done in the last three years.

3.1. The Lessons of DECASS

DECASS-based procedures, and artifices developed to overcome DECASS limitations, are of no direct relevance for the future, since we shall not in general be using DECASS. But they are indirectly relevant in that they have indicated what should not be done in the New System. Many implementers of small-machine telecommunications systems have begun by abandoning the manufacturer's software in favour of their own, but we feel that this is unwise unless there is considerable previous experience both of the small machine itself and of telecommunications work. The Indra group did not have such experience three years ago, and we have no doubt that it was well worthwhile to operate initially under DECASS. Had we begun by attempting to implement an alternative system we would have been in grave danger of frittering away scant resources to little purpose. Now that experience has been gained, and we are familiar with the PDP-9 and with telecommunications, we are in a far better position to develop a system that is superior to DECASS for our purposes.

3.2 Unified Control

Although we have not altogether succeeded in creating a truly user-oriented operational procedure, we feel that we now understand far more clearly what characteristics such procedures should have. The command structure of INDR1, which was intended to be applied by suitable extensions to all Indra activities has not been so applied for various reasons, but this fact does not invalidate the intention. Insofar as the concept of a unified command structure has been applied, namely to the whole body of local and remote file handling, it has been found to serve very well; current ideas may be seen as a straightforward generalisation of the earlier approach, with extension of facilities and relaxation of restrictions.

3.3 Common Software

The notion of common utility routines in INDR1 was introduced originally partly to overcome DECASS limitations as has been noted already, but partly also to reduce the tedium and waste of time caused by the frequent need to write essentially the same routine. From these utilities, and from DECASS system macros which serve a similar purpose on a larger scale, has developed the idea of supervisor calls in the New System, together with a measure of expertise in such matters as parameter handling.

This last, in its turn, has influenced the design of the command decoder.

Utility routines and parameter-handling techniques may seem trivial matters, but the overall efficiency of a system is greatly influenced by the extent to which common software facilities may be employed in a variety of circumstances, and the ease with which different parts and levels of software can communicate with each other.

3.4 Telecommunications Capability

Turning now to matters more directly relating to telecommunications, it is fair to say that the work of the Indra group in both hardware and software areas has resulted in a body of techniques of great potential, as well as actual, power. It has recently been stated [24] that "We have not found a computer using a serial synchronous interface with which we could not communicate (from the hardware) at up to 50kbps", and a similar claim could be made for communication with asynchronous devices. This power is due, in part, to the variety of hardware that has been obtained or developed for telecommunications purposes and is discussed in the reference quoted above. Mention was made in Chapter 1 of the range of communications adaptors on the PDP-9, and of the extended facilities developed locally for them. In association with these adapters there is a range of modems and line connections of various kinds, and, more important than the hardware itself, the Indra group has a good understanding of the characteristics, merits and demerits of the various hardware facilities available. The software developed specifically for investigating engineering aspects of telecommunications systems is also important in this connection, and its obviously one of the items to be carried forward.

3.5 High-level Software Techniques

At the level of specific telecommunications software development there are two particular matters to be mentioned. (We ignore here the work done on telecommunications software in the 360, as it does not directly influence future development in the PDP, at least in the sense considered here; this is not in any way to diminish the importance of the work that has been done, either as an essential part of the development of the 360 link, or as a valuable source of relevant experience in central-machine communications techniques.) The first, and more significant, is the use of special-purpose high-level software in conjunction with conventional assembly language. The importance of this approach derives mainly from the facility it confers for simple and speedy revision, and the consequent reduction in the need for accurate knowledge of protocol before experimental communication can start. The use of run-time assembly for the high-level sections of telecommunications software, although orignally introduced merely for simplicity, has been a positive advantage, for it has materially increased the speed and flexibility of experimental work. The idea of an autocode extension to the New System assembly language derives directly from the idea of high-level drivers for INDR1 links, and is a natural extension. So also is the possibility of incorporating parts of the New System assembler into other programs, although it may well be that the provision of efficient dynamic loading will remove the necessity for run-time assembly.

3.6 Real-time Monitoring

Finally, the facilities used in the past for real-time monitoring will be carried over into any future work. The history of monitoring in the various telecommunications systems extends from the meagre facilities grafted on, very much as an afterthought, to the slow link, up to the comprehensive choice of options available in the experimental CDC link. The inclusion of some of these options was originally made on a basis of guesswork; some of the guesses proved themselves to have been inspired. We intend to provide some generally useful monitoring options in the New System, together with general means of assembling monitor files and either displaying or storing them; these facilities will be accessible by means of supervisor calls.

One of the problems, where experimental facilities such as monitoring are intimately associated with other software, is how to remove such experimental sections when they are no longer required. It is intended to provide options in the New System assembly language to allow the inclusion or omission at will of designated sections of code, and in this way, given reasonably careful design, the amount of editing required when passing from experimental to production software should be minimised.

4. The Next Step

We come finally to the question of what future development may be expected to stem from the advances in telecommunications techniques that have been made so far. We shall assume that these are in fact to be further developments, and based on current discussions within the Indra group and elsewhere we shall make certain assumptions about the general form that these might take, largely without comment on their possible validity or usefulness, since such questions are not to our present purpose.

Arguments have been presented in previous chapters to show that little significant further development is possible under DECASS, and we shall not repeat these arguments here. We shall therefore tacitly assume that the New System, or some further development of it, will be the basis for future software development. We shall further assume that the various hardware developments to enhance the communications adapters are completed, so that hardware character detection and data channel transfers are fully available.

The important question that has to be answered is to what extent these hardware facilities and software techniques will be able to satisfy assumed future needs. It will be realised that any answers offered here must necessarily be extremely tentative, depending as they do on an as yet incomplete operating system; it will not be possible to test the validity of these answers until the performance of the New System has been properly evaluated.

It should be unnecessary to point out that any opinions or proposals expressed or implied in the following discussion are the author's own ideas, and do not necessarily represent the views of other members of the Indra group.

4.1 Conversion of Existing Links

It is appropriate to begin the discussion by considering what useful amendments could be made to the present 360 and CDC links. Although the CDC system is still in its experimental phase, the general form of a DECASS-based 200 terminal simulator (or Trend terminal simulator) is now clear, and for the purposes of the present discussion may be assumed to exist. Thus any future development of the 360 and CDC links may be assumed to have the same starting point.

In the following discussion it is assumed that the characteristics of the control software in the central machines (i.e. the operating system and any superadded telecommunications or file-handling programs) does not change significantly. This assumption is probably valid for the CDC system, since it is committed to serving passive terminals, but is probably invalid for the 360. However, without making such an assumption it is impossible to proceed with the discussion, and it may be hoped that the general principles to be enunciated will survive any central system upheavals.

Even when no link handler is loaded, the supervisor can be set up to provide a primitive response to a call from a remote machine: enough to respond REJ to a CDC poll, for example. Dynamic loading of programs from drum will be fast enough to allow a reasonable amount of segmentation: to load a 500-word system program, for example, should not take much over 30ms assuming the worst case of drum latency. Hence it should be quite feasible to load a handler (or at least part of it) on demand from a remote machine.

4.2 Text Transmission

Consider the problem of transmitting a block of text to a remote machine, assuming that the source text is held in a conventional file on drum. The following main processes are required:

(a) preprocess enough data from the file to form a block. For simplicity this may be taken to be an integral number of records (e.g. lines). Preprocessing may typically involve formatting (e.g. into card image), code conversion, provision of headers, provision of transmission control characters, and calculation of check bits or characters. The data is left, after preprocessing, in the form of a syllable string packed in double image.

(b) transmit the block by data channel action. Unpacking from double image is performed by the hardware of the communications adapter.

(c) await the response, and then either to (a) if the block was accepted, or to (b) if it was rejected.

Once the block has been loaded into the transmission buffer, step (b) requires no resident software. Entirely separate programs are required for steps (a) and (c), and these need not be co-resident. Thus step (a) for block (n+1) can be performed while step (b) for block n is in progress, but it would be sensible for step (a) to write its output to drum so as to avoid the need for two permanently-assigned large core buffers, since

step (a) will be complete long before step (b). In this case step (b) must be initialised by loading the core buffer from drum, but that is a trivial process and cannot take longer then 25ms for a 500-syllable block or 30ms for a 1000-syllable block.

Block-by-block pre-processing interleaved with transmission appears to offer several benefits. Firstly it overcomes the retransmission problem noted earlier in the context of the CDC link. Secondly it is more economical of drum space than is full-message pre-processing, and incidentally may make better use of connection time to a remote machine. Thirdly, if two areas are reserved on drum for pre-processed blocks, there is no need to maintain a buffer area in core while step (c) above is in progress, since if retransmission is found necessary the last block sent can be quickly reloaded.

This last point becomes particularly important when one considers response delays. In the 360 system, a 2 sec timeout is imposed on responses; as far as is known there is no response timeout in the CDC system. Operating systems are liable to take a somewhat cavalier attitude towards delays: the long strings of SYN characters that may be sent by the CDC are a case in point. If the central machine operating system develops (e.g. for the convenience of batch or on-line teletype users) into a hierarchy of operating systems with a multi-user file handler on top, the delays in receiving responses may become quite unpredictably large, and the whole concept of timeout errors may have to be rethought. In such circumstances it is clearly pointless to lock out large core areas in the satellite more or less indefinitely, and the use of demand loading becomes imperative.

4.3 Text Reception

Turning now to the process of receiving a block, there are again three steps which, in the 360 link at least, are complementary to those involved in transmission.

(a) await an ENQ and send an appropriate response. The routine required here is the same as that used when awaiting a response after block transmission.

(b) assemble the block in double image by means of the data channel.

(c) Post-process into a set of records and, if correct, write away to a file on drum, or display, or print, according to circumstances.

Post-processing at least as far as block parity (or BCC) checking must be done before the response can be sent, but in the operating conditions of the PDP there is plenty of time for this within any likely timeout constraint. The post-processing software need not be loaded until block assembly is complete, and hence may be selected if necessary according to the type of message that has been received.

The situation is different in the current implementation of the CDC system, because the central machine sends text without warning. In this case, the routines for response detection and for poll detection must be combined, so that if a write message is detected instead of a poll the data channel can be activated. (There is, in fact, a facility available to send what is known as an Alert message, similar in purpose to selection on the slow link; that is, it gives warning that a write message is about to be sent, thus giving the terminal the option of refusing it. Up to the present, alert messages have not been used, but they may be introduced in due course.)

4.4 Job Turn-round

When a job is submitted to a central machine (as in the CDC system), or started by command from the terminal (as in the 360 system), the question of throughput delay becomes significant. In current implementations there are two possibilities. In the 360 system, the results of a run are held locally until called for by command from the terminal, and it is possible to determine when the results are ready by calling for a logging data set in which the status of all work for the terminal is recorded. It is therefore quite unnecessary to keep the link active while the job is processed, especially as the system is constrained so that message interchange is always initiated by the PDP, never by the 360. Hence the time of calling for results can be left to the discretion or convenience of the remote operator. It would be possible, of course, at the expense of some system complication, to arrange that the logging data set was automatically called for after a few minutes, if throughput was expected to be reasonably quick, but since the operator would have to be present to see it there seems very little point in providing the facility.

In the CDC system, on the other hand, the usual assumption is that the terminal is on-line (we consider batch terminals in particular at this point), and hence results are automatically sent when they are ready. The throughput may, of course, be very slow indeed if the central machine is heavily loaded. Where the conditions of PDP use make it acceptable to keep the basic CDC handler active, the method outlined above will satisfy this mode of operation. Alternatively, it is possible to simulate a terminal that has been switched off, by the simple expedient of ignoring all polls, and in this situation the central machine will hold the results of a run until the terminal becomes active again.

4.5 A Proposal for a Segmented Link Handler

From the foregoing discussion it can be seen that the software for a given link can be segmented and operated along the following general lines. The arrangement suggested is appropriate to the CDC link more particularly than to the 360 link.

(i) A control string handler is loaded whenever the terminal is intended to be active. This can analyse all control strings up to and including the message type identifier (STX, POL, etc.), and initially is set to respond REJ to POL and ignore all other types of message.

(ii) Once a poll has been rejected, the subsequent section of the control string handler is set according to prevailing circumstances, and it becomes able to deal with the full generality of responses. This may require loading (as a result of the poll) of a further segment, but there might not be much advantage in carrying segmentation thus far. Alternatively, the extended command string handler might be loaded with the next item to be described.

(iii) The functions of the experimental driver are taken over (in part) by a scheduling routine loaded either at the outset or when the link has been shown to be active. This reacts to messages received, and also to the existence of messages for transmission, and controls the loading and execution of the following routines in an appropriate sequence.

(iv) A message pre-processor capable of dealing with job control records (assumed to have been input or prepared previously) or job files, both on drum, and with commands input at the keyboard. It leaves correctly formatted transmission blocks on drum.

(v) A routine to load a core buffer from drum and initiate data-channel transmission.

(vi) A routine to initiate data-channel reception. These last two are so small that they would probably be incorporated into the scheduler, as there would be little point in making separate segments for them.

(vii) A message post-processor capable of dealing with assembled blocks after reception, including error checking and means for writing the records produced either to a file or drum (or printer, if there is one fitted), or to display.

(viii) Optional monitoring routines used in conjunction with supervisor monitoring facilities.

Two final points remain to be made. The first is that the scheme proposed above is intended to provide the most efficient form of link handler under the New System; it is by no means the only way in which a link handler could be implemented. The existing form of handler could be run quite well under the New System, merely by making the changes indicated in Appendix 3.

The second point is that to implement a scheme such as that discussed, although it would involve extensive recoding, is not as formidable a task as might appear at first. The logic of an existing handler carries over directly into the revised scheme; existing handlers are already largely segmented in the way described, although the segments are not assembled separately. Given the New System facilities assumed in the description, the development of a segmented handler should be no more difficult than the conversion of an experimental handler into a production one; it will be remembered that this conversion was done, for the fast link, as an MSc project by a student with no experience in telecommunications and little prior practice in PDP assembly-language programming.

5. Longer-term Prospects

Consideration of what longer-term developments are possible depends upon an understanding of what types of system are potentially viable. This might well be a contentious topic, but an attempt will be made to discuss it so that some questions of technical feasibility can be examined. The assumptions and disclaimer made at the beginning of the last section apply also here, if anything with greater force.

5.1 Terminals

The original Indra project was to couple two computers in such a way that a complex CAD process could be efficiently shared between them: major calculations were to be performed in a large batch processer, application-specific data manipulations were to be performed in a small dedicated interactive processor, with a telecommunications system linking the two machines. The limitations imposed by low transmission rates were to be minimised by reducing the amount of redundant data transmitted: for example, in a series of runs on a given design problem only the changes to source data were to be sent before each run. It must be admitted that these aims have not yet been wholly realised, due mainly to the magnitude of the software problems and operational difficulties that were encountered, but had not been anticipated. Success or failure of the project as a whole is a topic outside the scope of this discussion, but it is appropriate to consider whether the aims were valid, since these aims influenced the approach to telecommunications system design.

There are three facets to the question: firstly, is the use of a comparatively expensive satellite computer justifiable when much cheaper remote terminals are available; secondly, is it sensible to subdivide the job into distinct interactive and batch components; and thirdly, is there sufficient advantage in reducing the amount of data transferred to warrant the additional system complication introduced.

It should be borne in mind that at the time the Indra project was planned (in 1966), cheap remote terminals were not available, so the first question did not arise. Such terminals now exist, and it is probable that for many purposes a terminal based, for example, on a storage tube display, keyboard and a graphic input device such as a hand tablet would be entirely adequate, used in conjunction, of course, with suitable interactive software in the central machine. In some circumstances it might be convenient to have a small computer associated with the terminal to allow greater flexibility in the choice of remote peripherals for special applications (e.g. automatic drafting machines), or to provide some local data-processing facilities, or to facilitate multiplexing of a number of simple passive terminals. Such active terminals are now readily available, in a variety of sizes and elaborations. One objection to the entirely passive terminal is that it would need to be designed to operate with the protocol of some specific telecommunications system, and could not therefore readily be switched between main machines using different protocols. This objection is overcome by the use of an active terminal containing a small computer, and it would be operationally feasible, although possibly not economic, to deal with protocol switching at the microprogram level rather than by conventional software.

Since a small remote terminal with no backing store has no file-handling capabilities, the problem of sending large volumes of data to and fro does not arise. In this case, the system in the central machine must allow for files to be maintained and processed in response to commands from the terminal.

It appears, therefore, that software-controlled telecommunications systems in terminals are not rendered obsolete by recent developments, but rather that for many purposes a much smaller system than the PDP-9 will suffice. The problem of software development still exists, however, and there is still scope for comparatively large systems in which developing software may be allowed to spread itself before being constrained to fit into a smaller production system. One might even consider the feasibility of two-level simulation here, so that, for example, a computer telecommunications system in which a small machine A simulates a terminal B is itself developed by simulating the whole issue in some third machine C.

5.2 General Telecommunications Applications

Turning now from terminal applications we observe that a requirement for small computers remotely connected to large ones exists in the context of computer networks, where the small machines are required to mediate communications between the large machines. In a fully-operational network one would expect that the small machines would be dedicated, with hardware probably specially adapted to their role, and run without operators. However, in any experimental work on networks it would be appropriate to use more conventional machines and to require close attention by a skilled operator.

It would be out of place to enter into a discussion of computer networks and the problems in telecommunications that they would present. A few general points having bearing on the topics discussed here may, however, be made.

Network telecommunications, from the standpoint of a small computer servicing the network, reduces at its simplest to the necessity of handling two large remote machines concurrently, instead of only one machine as in the various links developed hitherto. It is clearly more complicated to handle two machines than one, but we may initially suppose, for the sake of discussion, that the software in the remote machines that are to communicate with each other by way of the small machine are designed to permit such communication. Ideally this would imply a common protocol throughout the whole system, but in a more general situation this might not be possible, and the task of protocol conversion would be imposed upon the small machine.

The expression 'handling concurrently' deserves some consideration. If the protocol and the data rates were identical, and if a control interchange (e.g. ENQ .... ACK ....) took place effectively between the main machines, it should be possible for the small machine to pass data syllable by syllable. In such a situation there would be little point in having the small machine in the system at all. In practice it seems more likely that separate control interchanges will be required between the main machines and the small machine, and that variable latency and differing data rates and protocol will necessitate the small machine

being capable of buffering any amount of data up to a complete message at a time. It may nevertheless still be required to service both links concurrently and asynchronously.

The question is whether the PDP is capable of operating in this way with any conceivable software system, in view of its relatively simple hardware organisation, and if so, what the necessary characteristics of the software system would be. A positive answer to this question cannot be given without considerable experiment, but a certain amount of indirect, and essentially qualitative, evidence is available from the work that has been done on two-machine links.

To begin with, there is very little difficulty if a complete message is buffered in the PDP between reception from one machine and transmission to the next. We may assume that both links operate in contention mode, and the flexibility inherent in this affords sufficient leeway for the PDP to handle simultaneous requests from both main machines. In an experimental situation, at least, it would probably be reasonable to limit somewhat the frequency of demands from the main machines, either by agreement or by allowing the PDP to refuse demands if it became temporarily overburdened. This would simplify the inclusion of the full generality of monitoring, and allow for any inefficiencies inherent in the use of experimental software.

The other possibility is to allow concurrent reception and onward transmission on a block-by-block basis, with the possibility that an arbitrary number of blocks might have to be buffered. If the block format and the transmission code are common to both links it seems likely that little difficulty would be encountered in operating at speeds considerably greater than those employed so far, using a software system resembling two interleaved segmented handlers of the type suggested in section 4.5 above, although it might be necessary to reduce the amount of monitoring.

The next level of complication comes where a change of block format is required, e.g. from about 500 syllables of unformatted data to about 1000 syllables of card image. Here the internal software control required is far more complex, but there is still plenty of time in the inter-block gaps, when the transmission software is inactive, to perform the necessary data processing. There is probably sufficient leeway in the response time-out constraints for it to be possible to force the remote machines more or less into step, if it were necessary to do so.

Since most of the delay in telecommunications appears to be due to the software in the large machines, and since the use of more extensive remote file handling will only aggravate this delay, a large increase in baud rate may only give a modest increase in effective transmission rate. Hence the inter-block gaps may become greater for the same block size. and the software problems will be eased, since transmission and reception use the data channel, and thus require almost no software.

A still more complicated situation occurs where the remote large machines have largely or totally different approaches to telecommunications usage, e.g. where one operates on file transfer and job initiation, as the present 360 link, and the other operates on remote job entry, as the present CDC link. Here the problems of intermediate data processing can grow almost without limit. The only redeeming feature of this situation is that if,

for example, a grand syntax analysis of a received message has to be undertaken before an "equivalent" message can be transmitted, it is probably necessary for the whole message to be in before onward transmission can start. In this case, from the telecommunications point of view there may well be less need to design for too much simultaneous activity on the lines. But if such a system were contemplated, telecommunications problems would fade into insignificance.

In conclusion, then, there is no doubt that there is still plenty of work to be done, and we believe that the facilities and techniques described here will provide a sound basis for development. Of course, it may be, as Thorean suggested, that they have nothing important to communicate, but at least they will be able to communicate that nothing very well.

APPENDIX 1: The Language of Driver Programs

1. Macros

A driver program is written as a series of labelled macros. Each macro specifies (i) an action to be performed by the link software, and (ii) the label of the next macro to be executed after any of the possible outcomes of the prescribed action. An example of a driver macro (taken from the 360 link repertoire) is:

	next macro if exit para =												
Label	action	0	1	2	3	4	5	6	7	10	11	12	loop comment
A	ENQ	B	L	S	A	A	S	S	A	S	A	A	(A 40 S)/GOOD DAY SEQUENCE

In this example, the action is to send the ENQ sequence. The various responses or error conditions that may occur as a result of this action are indicated by the exit parameter value; in this case if the parameter is 0 the next macro to be executed is that labelled B. If the parameter is 3,4,7, 11 or 12 the macro A is to be repeated an indefinite number of times.

The label S is one of three special labels that cannot be used to identify a macro. S signifies Stop; in this example exit parameters 2,5,6 and 10 correspond to error conditions from which no recovery is considered possible, so link operation is made to stop if one of these should occur.

The label L is the second special label; it signifies that a controlled loop is to be performed. The loop parameters are given in brackets at the end of the line; in this example, if the exit parameter is 1 the macro A is to be repeated up to 40 (octal) times, assuming that the same exit parameter occurs repeatedly. If no other exit parameter occurs, the program is to stop (S) after the 40th repetition. In the driver from which this example is taken, the ENQ of macro A is sent at the start of a run; if the 360 does not respond, exit 1 occurs due to timeout, and the loop gives a convenient means of abandoning transmission if, for example, the link software in the 360 has not been loaded. A loop may encompass a group of macros.

The third special label is W, which signifies Wait. The word WAIT is displayed to the PDP operator who may then enter at the teletype the label of the next macro to be executed. This gives a facility for altering the sequence of events during a run, but it must be used with discretion to avoid system failure due to timeout at the central machine.

Loop parameters are only required if there is an L in the line. Labels need not be supplied for exits towards the right-hand end of the line that cannot occur; for example, EOT in the 360 link expects no response and hence only has exit 0. The assembler will fill out any incomplete line with S labels.

A driver program is terminated by a line consisting of the label S or W only. If S is used, once the driver has been assembled, execution starts automatically at the first macro in the program; if W is used, WAIT is displayed when assembly is complete, and the operator then enters the label of the desired starting macro.

2. The Form of Assembled Code

For simplicity, the assembler allocates a fixed length of core for the expansion of all macros, sufficient to contain the most complex case. The expansion of the example quoted above is as follows, MA and MB being the addresses in core corresponding to the macros whose labels are A and B. The * in subroutine calls and jumps to standard routines indicates that indirect addressing is used.

```
MA  JMS* ENQ     /Subroutine call to send ENQ sequence
    JMP  MB      /Jumps corresponding to labels specified
    JMP  LA      /for each exit parameter
    JMP* STOP
    JMP  MA
    JMP  MA
    JMP* STOP
    JMP* STOP
    JMP  MA
    JMP* STOP
    JMP  MA
    JMP  MA
LA  JMS* LOOP    /Subroutine call to loop subroutine with parameters;
    -40          /Loop count constant
    Ø            /Working space
    JMP  MA      /Exit while loop not terminated
    JMP* STOP    /Exit when loop terminates
```

Note that the loop subroutine is common to all macros using it, and hence keeps its working space outside itself. On first entry to the loop the working space contains zero; the subroutine initialises the count at this time. If the macro is left other than by an L exit, the loop working space is cleared by action in another part of the link software so that the full number of repetitions is again available if the macro is used subsequently.

3. Repertoires for the 360 and CDC Links

The macros defined for the 360 link are:

NON	Send nothing (i.e. wait for the 360 to send)
ENQ	Send ENQ sequence
ACK	Send ACK0 or ACK1 sequence as appropriate
NAK	Send NAK sequence
EOT	Send EOT sequence
GDY	Send text of 'Good day' message (held in core)
COM	Send text of command according to 360 function required
TXT	Send the next block of the main data file.

The macros defined for the CDC link are:

LSN	Initialise transmission and listen for a poll
REJ	Send REJ sequence
ACK	Send ACK sequence
ERR	Send ERR sequence
EC2	Send 'automatic read with E2'
EC3	Send 'automatic read with E3'
COM	Send command input at teletype
TXT	Send the next block of the main data file
PSW	Switch after poll received, according to internal flags
WSW	Switch after write message received, according to internal flags.

The internal flags referenced by PSW and WSW indicate the availability of commands or text to send and the readiness or otherwise of the local peripherals.

APPENDIX 2 : Proposed Command Structure for the New System

1. General Principles

The aim of any proposed command structure for the New System is that it should be simple to use. Commands must therefore be reasonably self-explanatory to the user, be free from formatting quirks, and offer a useful set of default options. Self-explanatory commands tend to be longwinded, but the experienced user may freely abbreviate any reserved word in the command language to not less than its first three characters. A free format and the provision of default options complicates the command decoder, but since this is non-resident there is little objection to it being lengthy.

Some commands are specifically for the autonomous system, some for the user system, and some are common to both. Commands may be input either simply at the appropriate console, or, for the autonomous system, in batches typically by way of the paper tape reader. In a batch each individual command may be prefixed with an indication of the next command to be executed following the success or failure of the command in question.

2. Current Version

A first version of a command language satisfying the foregoing principles has been implemented for the experimental DECASS-based software discussed in Chapter 7. As details of this language remain to be decided, a formal description will not be given. In any case, the general philosophy is subject to change: for example, in the present scheme file editing is a primary activity that may have file transfer as a byproduct, whereas it might be better to treat file transfer as a primary activity that may have file editing as an intermediate process. The following examples indicate the principal characteristics of commands. Each command consists, in general, of a command identifier followed by a series of clauses each introduced by a clause identifier. Clauses may be entered in any desired order; some clauses are optional in certain commands and can be omitted if the facility offered is not required.

(a) Transfer a file from a named dectape to the Atlas, and monitor the process. The use of named dectape allows the system to check that the correct tape is loaded; each tape may carry a unique name and serial number. This command is given in extended form.

TRANSFER TO ATLAS '<JOB DESCR>' FROM DT 'MYTAPE' 351 +
'<FILE NAME>' <MODE> WITH MONITOR

The + is a continuation symbol for commands that are too long for a teletype line. <JOB DESCR> is either the complete job description in the format required by the remote machine, or a set of parameters to be incorporated locally into a standard skeleton job description. <MODE> identifies the format and code of the source file.

(b) Transfer three files within the PDP. Default options are applied where necessary components are omitted. This command is given in abbreviated form.

TRA FRO DT1 'NAME' DIA, PPR, DIF, TT2 TO PPR, DR 'FILE' DIA, ID

The effect of this is to send the file 'NAME' from dectape 1 to punch, read a paper tape in Atlas flexowriter code and write it as a file 'FILE' on drum in double image ASCII, and display data typed at the second teletype on the 340 display.

(c) Assemble two programs into a single file of intermediate code on drum, and output a symbol listing on the user console. This command is given in extended form.

ASSEMBLE FROM PPR SRC, DT1 'PROG' DIA TO DR 'PROG' INT LIST + UC WITH MERGE

Note that the source programs need not have the same format or code. INT is the mode for intermediate code; it could be omitted. Note further that if the TO clause is omitted the process serves only to check for errors and, in this case, produce a symbol listing. If the WITH clause is omitted, the two programs are assembled separately (and the TO clause, if present, must have two components).

(d) Load and execute a program, with system-supplied debugging facilities.

RUN DR 'PROG' INT WITH DEBUG USE 1 DT3, 6 CTK

The USE clause redefines channel-to-peripheral assignments given in the original program. If the mode of the source file is not INT, the assembler is automatically called first.

(e) Start a job in the 360 by sending the appropriate start command. Output any message returned by the 360 (on the system console).

START AT RHEL '<JOB DESCR>' WITH LOG

APPENDIX 3: DECASS to New System Conversion

1. Introduction

When development of the New System began, it was clear that it would not be possible to obtain all the desired system characteristics while remaining strictly compatible with DECASS program and data conventions. At the same time, there was no merit in departing from these conventions merely for the sake of doing so, since the Indra group has a considerable investment in software, and, to a lesser extent, in files that might be expected still to be required after the New System had come into being.

Conversion of programs and data from DECASS to New System is only necessary insofar as the two systems are incompatible. In this Appendix we consider what conversions are required, and why, rather than how they should be performed. Software will be provided to allow systematic conversion, e.g. of files: such software is in any case required during the development phase of the New System itself, since until a rudimentary system is available most development has to be done in the DECASS environment.

To prevent this Appendix becoming unduly lengthy, a rather greater familiarity with DECASS is assumed here than in the main text.

2. Assembly-language Programs

In this section we ignore the extensions to the assembly language provided in the New System, since these are not relevant when existing DECASS programs are to be converted. It is only necessary to consider features of a DECASS assembly language program that would not be accepted by the New System assembler.

The differences between the languages fall into three categories:

(a) those due to the operational differences between DECASS and New System,
(b) those due to the use of a single-pass assembler, a different intermediate code, or a simplified loader,
(c) those due to omission of seldom-used facilities.

The following discussion covers the major items under these headings, but omits a few minor matters.

2.1 Operational Differences

The most important of the operational differences concerns peripheral control and input/output operations. All IO is handled by the supervisor, partly for overall system efficiency and partly for protection. File-handling procedures are radically different in the New System, as explained later. IO is achieved by means of supervisor calls, broadly equivalent in function to DECASS system macros, but based on the use of IOT instructions (operation code 70) rather than CAL, so that they are trapped and uninterruptable.

A related difference is concerned with device assignment. In DECASS, devices are assigned to IO channels (DAT slots) at load time, and a program must contain a .IODEV directive if any IO macros are used so that the loader may load the handlers corresponding to the DAT slots quoted in the macros and listed in the directive. New System IO supervisor calls quote channel numbers, which are essentially equivalent to DAT slots, but the .IODEV directive

gives the device assignment mapping. This mapping serves as a default option at load time, but can be amended by explicit assignment in the loader call.

The only other important difference is that the run-time position in core of a program is determined by the loader, and hence absolute loading is not possible. Programs must therefore be written with symbolic rather than absolute addresses.

2.2 Assembler Limitations

The only difference due to the use of a single-pass assembler is that user symbols must be assigned before they are used; otherwise they are treated as labels.

The intermediate code in DECASS has a somewhat peculiar format and includes a control word preceding every three words of code. The loader therefore has a considerable amount of data processing to do, and is consequently larger and slower than necessary. In the New System, the length of a program in intermediate code is precisely the same as its length in executable form, and hence the loader can cause the program to be read directly into the space it will occupy while it is running. In consequence, all the label linkages must be contained within the 13-bit address fields of store access instructions; to allow a reasonable number of bits for the linkage field it has been necessary to limit the range of offset x in instructions such as LAC LABEL+x to $-20 \leq x \leq 17$ octal. This is a very minor restriction in practice, since it is usually unwise to use large offsets.

In order to simplify the loader, increase its speed, and reduce the amount of data required in a loader command, the DECASS practice of letting the loader select programs on the basis of a loader list (in the command) together with a library search to match unsatisfied globals has been abandoned. Instead, a special directive has been introduced which allows required routines to be explicitly named. The assembler augments the list of names by adding any system routines that may be required to satisfy supervisor calls. By judicious use of this directive it is possible to arrange that only one program need be identified in a loader command. Except where libraries are concerned, this facility is an extension of DECASS rather than a restriction.

2.3 Seldom-used Facilities

In this category are certain features of the DECASS assembly language that Indra programmers appear seldom if ever to use. The most notable of these is the user macro; it seems to be generally held that the small time penalty in using a conventional subroutine is more than compensated for by the reduction in storage space.

The other examples are mainly certain unimportant optional coding conventions, and a reduction in the amount of listing output that can be obtained during assembly.

3. Fortran

DECASS Fortran compiles to the same type of intermediate code as the assembly language, and is in consequence incompatible with the New System. No Fortran implementation has yet been planned.

The only significant use of Fortran in the Indra group has been for graphics, and the inadequancies of the DECASS implementation have resulted in the use of only a small subset of the language. In particular, Fortran IO is not used at all. To implement this subset in the New System should not prove too difficult; it might conveniently be done by way of symbolic assembly language. It would then be simple to combine Fortran and assembly language in any convenient way, and some benefits might accrue to the New System assembly language as well: for example, extensions necessary to allow the compiler to deal with, say, arrays or common could be made explicitly available to the assembly-language programmer.

4. Files and Directories

The methods of file management developed for the New System are incompatible with those of DECASS. In DECASS, alphanumeric files are made up of packed ASCII lines, with five characters packed into two words; all files that are not in this format are treated as composed of 18-bit words, also taken in pairs. In the New System we wish to be able to use 8-bit as well as 7-bit characters, and we have the possibility of hardware packing and unpacking on the data channel for telecommunications. Consequently, alphanumeric files will be packed two characters to a word (double image packing), and the data may be in any one of a number of codes, e.g. ASCII, EBCDIC, BCD, Atlas Flexowriter.

A DECASS record header gives the record length in word pairs; this is most inconvenient with packed alphanumeric records because of the unknown number of ignorable characters at the end of the record (it is not desirable to have to search for a code-dependent terminating character). In the New System the record header will give the number of units (characters or words) in the record, according to type, and the directory entry for a file will indicate the format and code of the records in the file.

The maximum record length is low in DECASS, and irrecoverable errors occur if the maximum is exceeded. It is not possible to read part of a record, and hence buffer space must be provided to accomodate the maximum length. In addition, the backing store handlers have a separate permanent 256-word buffer for each file that may be open at a time; this is very wasteful of storage space. In the New System a record may be of any length up to 32K words: an assembled program, for instance, may be treated as a single record. The programmer can choose how many words he wishes to read at a time, and need only assign enough buffer space to contain them; there are no permanent handler buffers.

DECASS does not as a rule make full use of the space available in a drum sector or dectape block; the New System will do so, that is, in the New System the physical block structure of backing stores has no influence on the format of a record or file. One useful result is that the increase in record size due to the use of double image rather than 5/7 packing is considerably less than might be expected, and it is even possible for a New System file to occupy less space than its counterpart in DECASS.

Since records can be written or read piecemeal it is necessary in general to know the length of a record before writing begins, so that the length can be put into the header word and so be available when the record is later read. In DECASS there is a second word in the header, intended for a sumcheck, although the facility is not always used. In the New System the sumcheck will always be used, but it cannot be placed at the beginning because the whole record may not be written at one call. Hence the New System record header is of one word only, and the record is followed by a one-word sumcheck.

The internal structure of a DECASS file library is quite different from that of a set of independent files, and hence special system programs must be used to access or modify the contents of a library. Also the number of libraries that may be defined is severely limited. In the New System there are no structural differences and no restrictions on the number or contents of libraries. A library is identified in the main directory of, say, drum as a single file (and may therefore be transferred as an entity, as in DECASS); within each library there is a sub-directory of similar structure to the main directory containing the names and locations of the files in the library. A New System file name is up to six characters in length, as in DECASS; in place of the DECASS three-character extension there is a three-character mode identifier (defining format and code, or any special characteristics of the file). If the file is contained in a library the name has a further three characters that specify the name of the library, and the presence of these characters is sufficient to ensure that the file is read from, or written into, the library concerned. If no library name is given, or if the mode indicates that the file is in fact a library, it is referenced through the main directory of the drum or tape.

A DECASS directory on, say, a dectape occupies eight 256-word blocks, and every reference to the directory requires that one of these must be read into core for the file name, and then that another, identified by the content of the first, must be read to give the location of the file on tape. As will be realised, this is an extraordinarily time-consuming process: PIP, for example, takes some 15 secs to close a file on dectape. In the New System, the complete directory occupies only two blocks.

All directories will be held on drum for speed of access, so the space occupied by one DECASS directory will hold all the directories for drum and three dectapes. The system will attend to all matters of tape directory updating on demand, or at the end of a run, as may be required. Also, the small size of a directory makes it possible to load a complete directory into core, and only one transfer operation is necessary in place of the two required with DECASS directory. Since the whole directory has been read in this way, instead of only a part of it, any number of references to the directory can be made without additional transfers being required.

APPENDIX 4 : TOPSY, A Communications System Test Program

1. Introduction

TOPSY is a general-purpose test program for communications systems developed by A.R. Duncan [25]. It provides a means of investigating the low-level performance of a communications system and has so far had three main uses. The first of these is to check the hardware of a communications system, typically by looping the transmission circuit back on itself at some convenient point, transmitting syllable strings and monitoring the syllables returned. The transmission circuit may be looped either locally to the PDP, on the line side of the local modem, at the far end of the line, beyond the remote modem, or even, in some cases, within the data adapter or similar unit of the remote machine. In this way it has been possible to find local hardware faults, or to assist engineers elsewhere to find faults in their own equipment.

The second use is to investigate the operational characteristics of a system, either simply by listening to remote transmissions or by transmitting simple messages in order to elicit responses. This use of TOPSY has obvious application in the preliminary stages of developing the software of a new link.

The third use is to simulate the action of the central machine of a coupled system. This use has found application in the development of the Modular One simulator mentioned in Chapter 1. The form of the simpler messages sent by the CDC system was known from work done on the PDP to CDC link; these messages were sent by TOPSY from the PDP to the Modular One to assist in debugging the Modular One software. The monitoring facilities of TOPSY provided information on the actions of the Modular One that could not have been obtained had the Modular One been put directly on-line to the CDC.

A full account of TOPSY will be found in the reference; this Appendix contains only a brief description, sufficient to indicate the general characteristics of the program and of the facilities it offers.

2. Programming for TOPSY

In order to use TOPSY it is necessary to prepare a program, either on paper tape, or on-line at the teletype. The program is written in a high-level language having some of the syntactical characteristics of a simple subset of Fortran, in which the statements refer specifically to telecommunications matters such as the transmission or reception of messages, the analysis of received message strings, and the like. Thus the program resembles in some respects the driver programs discussed in Chapter 5, although its format is quite different and several facilities are provided that do not have their counterparts in the driver language.

As an example, the following program may be used to exchange the "Good day" message with the 360. The commentary in brackets is given for explanation only; it is not part of the program.

```
TEST PROGRAM TO EXCHANGE GOOD DAYS

1  HEAR                      (This enables TOPSY to receive one complete
                             message of any type; in this case an ACKØ will
                             be expected after the ENQ has been sent.)

2  SEND ENQ                  (The string defined for the ENQ messages is sent.)
   DELAY 12                  (Wait 1 sec. The 12 is octal; the time unit is
                             100 ms.)
   IF INMES = 0/2            (If no message has been received go to 2)
   IF INMES (1) = 160/3      (If the ENQ is correctly acknowledged, go to 3)
   GO 1                      (Otherwise start again at 1)

3  SEND GDY                  (The "good day" message is prestored)
   HEAR
   DELAY 12
   IF INMES = 0/2            (If there is no response, send the ENQ again)
   IF INMES (1) = 141/4      (If the message is correctly acknowledged, go to 4
   GO 1                      (Otherwise start again at 1

4  SEND EOT

5  HEAR
   IF INMES = 0/5            (Wait for ENQ)
   IF INMES (0) = 55/6       (Go to 6 when ENQ is received)
   GO 1                      (If anything else is received, start again)

6  SEND ACK0                 (Acknowledge the ENQ)
   HEAR
   DELAY 24                  (Wait for a message; in this example it is not
                             tested in any way.)

   END
```

The foregoing example does not contain all the types of statement available. Other statements include LISTEN, which is similar to HEAR, but is not limited to one message at each call; IDLE, which is equivalent to an untimed DELAY; and DO loops, which may be nested. The programmer may also use certain standard variables for flags or switches, and clear, increment or test them at will. One use of such a variable would be in the selection of ACK0 or ACK1 on the 360 link.

A repertoire of standard messages is provided for each link. These are stored in the appropriate target code (that is, EBCDIC or CDC external BCD), complete with all transmission control characters and check syllables. A development being planned at the time of writing will allow the standard repertoire to be augmented by messages read in through the tape reader.

3. TOPSY Operation

TOPSY consists of two main parts : a compiler and an operating system. The compiler generates machine code corresponding to the source program, mainly in the form of subroutine calls with parameters. The operating system includes a skeleton link handler used, in conjunction with the subroutines just mentioned, to control low-level communications activity, an executive

section to handle the user's commands, and facilities for general monitoring and for assembling received messages (i.e. for HEAR or LISTEN statements).

The initialisation of a TOPSY run is done at the teletype. The user may specify the system (360 or CDC), on-line or off-line operation, and monitoring requirements. At the end of a run TOPSY returns to the initialisation phase where further facilities become relevant : he may send the monitor file to display or to, say, dectape, output the file of received messages, and display status information such as the values of the internal variables mentioned above, or the number of messages received or sent. It is also possible to rerun the current program, either under the same conditions as before or with a change of conditions, e.g. from off-line operation to on-line operation.

When the source program has been compiled, the user is invited to enter the identification number of the first statement to be executed. A rerun command also comes to this point, so the user has the option of starting a rerun at a different statement. After specifying the starting statement the user may insert a comment; this will be entered at the head of a monitor file and can therefore provide a useful means of identification later.

The monitor file contains a record of received and transmitted characters in time sequence, and optional timing markers. When the run terminates, the reason for termination is also entered into the file, followed by the status information mentioned above. Runs may terminate for one of three reasons : the program has run to completion, a console switch has been set to force termination, or the monitor buffer has overflowed.

APPENDIX 5 : Personal Involvement

In order to give a reasonably complete picture of the telecommunications and other systems programming work of the Indra group, this thesis includes discussion of a number of topics in which members of the group, other than the author, were concerned. The purpose of this appendix is to specify the author's own personal contribution to the work described, and items in which the author had no personal involvement are not mentioned.

(a) Slow Link. The original specification was outlined by T. Fenner towards the end of 1967, but from the beginning of 1968 the author was responsible for the PDP-9 telecommunications software. The environment was written by F. Huber; the author wrote the rest.

(b) Indra Utility Software. The original versions of these programs were written by the author; some later modifications have been made by a number of other workers.

(c) Fast Link. The author was responsible for the PDP-9 telecommunications software. In the first attempt to write a system in a form similar to that of the slow link, the environment was written by N. Lynch. The author introduced the idea of high-level drivers, and wrote all subsequent experimental software with the exception of a post-processing routine which was written by P. Rakity. Some modifications to this and to the pre-processing routines were made by J. Shane and N. Lynch.

The conversion from the experimental form to the first production form was done by R. Humphries as an MSc project under the author's supervision. Subsequent work has been done mainly by P. Higginson, and the author has only been concerned in an occasional consultative capacity.

Hardware changes to the data adapters were made by D. Elliott as an MSc project under the author's nominal supervision, but technical supervision was by A. Duncan.

(d) Atlas Link. A. Duncan was wholly responsible for the hardware interface. The software was developed as an MSc project by J. Shane under the author's supervision.

(e) CDC Link. All the software described in this thesis was developed by the author, with some assistance from M. Beattie in the early stages. Further development is now in the hands of A. Zacharacopoulos, under the author's general direction.

(f) New System. The original proposal and the subsequent general control of the specification of the system have been the author's responsibility. A preliminary study of the properties of the storage protection facilities of the PDP-9 formed part of an MSc project by M. McLean under the author's supervision.

All low-level software development, including the design of file management techniques, has been by P. Higginson, part of this work being done as an MSc project under the author's supervision. The preliminary DECASS-based versions of the executive, command decoder, assembler and editor were written by the author.

REFERENCES

1. Kirstein P.T. et al, Teleprocessing using a PDP-9 coupled to an IBM 360/75 over a Telephone Line, Institute Of Computer Science, October 1969.

2. Gould I.H. and Huber F.X., Final Report on the Slow Link Control Systems, INDRA Internal Note No. 95, September 1969.

3. Duncan A.R., PDP-9 to Atlas Communication Adapter, INDRA Internal Note No. 16, November 1967.

4. Shane J., Some Software for Atlas/PDP-9 Interface, MSc Project 1969, Institute of Computer Science.

5. Humphreys R.M., A Driver Program for a Communications Link, MSc Project 1970, Institute of Computer Science.

6. -, PDP-9 User Handbook, Digital Equipment Corporation, 1967.

7. -, Advanced Software System Monitors, Digital Equipment Corporation, 1968.

8. Eisenbies J.L., Conventions for digital data communication link design, IBM Systems Journal, 6, 4, 1967.

9. Gould I.H., Factors affecting the Design of the Fast Link Handler, INDRA Internal Note No. 64, January 1969.

10. Peterson W.W. and Brown D.T., Cyclic Codes for Error Detection, Proc. IRE, January 1961.

11. -, 200 User Terminal Hardware Reference Manual, Control Data Corporation, July 1968.

12. Kirstein P.T., Message Format, Data Structure and Proposed Link Usage, INDRA Internal Note No. 58, November 1968.

13. Gould I.H., General-purpose Command String Processor for Indra, INDRA Internal Note No. 89, August 1969.

14. Gould I.H., Code and Format Conversion Routine, INDRA Internal Note No. 90, August 1969.

15. Gould I.H., Interim Report on PDP-9 Software for the RHEL Link, INDRA Internal Note No. 123, June 1970.

16. Gould I.H., Progress Report (May 1970) on CDC Link Software Development, INDRA Internal Note No. 128, June 1970.

17. Gould I.H., Notes on the Operation of CLX2 - December 1970, INDRA Internal Note No. 172, January 1971.

18. Gould I.H., Bespoke Driver Programs for the Fast Link, INDRA Internal Note No. 113, December 1969.

19. Elliot D.M., Some Modifications to the Communications Adapter of the PDP-9, MSc Project 1970, Institute of Computer Science.

20. Higginson P.L., Systems Program Guide to the Overlay System, INDRA Internal Note No. 166, February 1971 (with revisions to be published).

21. Shane J., A Users Guide to the Overlay System, INDRA Internal Note No. 170, to be published, 1971.

22. Gould I.H., On a Feasible PDP-9 System for Indra, INDRA Internal Note No. 119, March 1970.

23. Higginson P.L., To Investigate System Software for the PDP-9 with reference to Coupled Computer Facilities, MSc Project 1970, Institute of Computer Science.

24. Kirstein P.T. et al, Investigation of Computer-to-Computer Communications Systems, presented at Datafair, 1971.

25. Duncan A.R., An Introduction to TOPSY, INDRA Internal Note No. 173, to be published, 1971.