A model for the local area of a data communication network
Software organization

P. T. WILKINSON
National Physical Laboratory
Teddington, Middlesex
England

1. Introduction

A general purpose store-and-forward data communication network is under development at NPL. The background to this work is described in companion papers[1][2] which also detail the hardware environment in which the software of the central message switching computer (MSC) operates.

A user of this system sees it as a star-connected network by means of which his terminal may exchange data with any other terminal via the MSC. Because this centre is stored-program controlled, it is possible to offer the user other communications-oriented facilities in addition to the basic data transmission function. The MSC may be described as a multi-access computer controlled by a timesharing operating system.

Section 2 gives a description of the way in which terminals use the network, illustrating the main principles of operation of the system as a whole and outlining the features provided by software. Section 3 is a description of the operating system of the MSC.

2. The interaction of terminals with the network

The interaction of a terminal with the MSC is controlled by a set of 'status' characters which are tagged so that they can be distinguished from data traffic. These status codes are exchanged between the 'peripheral control unit' (PCU) module at the terminal [2] and the MSC software. Some status codes are automatically generated by the PCU, others may be generated manually at a control panel which provides the terminal user with the necessary network signalling functions. Some of the status characters received by the PCU will cause the state of a group of

whereby a user knows when he may transmit and when he must await a response, is entirely a matter for prior agreement between the subscribers. Consequently, the user is only involved in network signalling procedures during call initiation and termination.

The 'conversational' type of interaction outlined above is expected to be used mainly for communication between multi-access computer services and keyboard/display terminals. Since computers are potentially high data-rate devices, it is possible for them to lock out a keyboard terminal, for example by generating a lengthy printout which the user may wish to suppress without ending the call. For this purpose an INTERRUPT control button is provided to allow a receiving terminal to break in. On receiving this signal, the MSC cancels any blocks which may be awaiting output and allows the interrupting terminal to transmit.

It was stated above that the PCU is responsible for informing the MSC that the input of a block is complete by sending the ETB signal. For peripheral devices such as paper tape readers, the ETB is generated on exhaustion of a character count. For keyboard/display devices, however, a line of type is a more natural unit than a fixed-length block, and interaction between such terminals and multi-access computers is usually organized on a line by line basis. The PCU for such devices is therefore constructed to recognize the device's 'end of line' character and follow this with the ETB code. The users of such terminals therefore need not be aware of the block organized store-and-forward operation of the MSC provided that they interact in units of one or more complete lines.

In many existing multi-access systems, terminals are connected directly (or via a switched network) to a satellite computer. While the satellite and main computers may communicate in line units, the satellite and terminal often do so in character units and advantage is sometimes taken of this fact to provide the user with facilities such as current line editing and with various control functions. For example, in echo-printing systems a user may be allowed to turn the echo off on to suppress the printing of passwords. However, it is nearly always possible to provide these desirable facilities on a line-organized basis. Thus in the Cambridge Multi-access System[3], the user is provided with a heavily printed line over which

display lamps on this control panel to change, to inform the user of the current state of his terminal.

For manually operated terminals, use of the data network is similar to the use of the telephone system. To initiate a "call", the calling subscriber presses a HELLO button causing input of status TRANSMIT. If the MSC can accept a new call, it will return a status code (TX) which causes the SEND lamp on the PCU to light. The peripheral device is then enabled to send data.

Data transfers between terminals and the MSC are organized into blocks of variable length up to 128 (8-bit) bytes. The transfers take place byte-serially to or from buffers in the MSC store, the buffer full (or empty) condition being recognized by hardware at the MSC. However, the end of an inward block transfer is normally indicated by receipt of a status code (ETB) from the PCU. This signal may be generated by the PCU on detection of a special data code from the peripheral, on exhaustion of a character count or after a preset time has elapsed. The particular method adopted will depend on the type of peripheral handled by the PCU. Following despatch of ETB, the PCU extinguishes the SEND lamp and disables further data transfers from the peripheral until the MSC has allocated a free buffer and returned status TX to the terminal. It must be emphasized that network terminals contain single byte buffers, not full 128-byte buffers. The organization of data input/output into blocks promotes the efficient operation of the MSC and a user need not normally be aware of it. However, this point merits further attention and will be taken up later.

The block of data initially input by a terminal is known as a 'heading' and contains commands by means of which a subscriber may specify the address of the the terminal which he desires to 'call', and may select other communication functions provided by the MSC. The MSC is thus 'dialled' by using the peripheral device itself. Provided that the heading commands are acceptable and the called subscriber is available, the MSC returns status TX to the calling terminal which may then proceed to transmit data. Otherwise, the MSC returns status QUIT which causes the PCU REJECT lamp to light, informing the user that his request cannot be accepted.

During the heading phase, the MSC and the subscriber interact through the peripheral device. The MSC must therefore be able to interpret the device's character code. During the communication phase, in which two terminals exchange data, the network is transparent to this information unless the caller has specified a processing function such as code conversion.

Once a 'connection' has been established, the two terminals may exchange data until one of the subscribers ends the call by pressing the GOODBYE button which causes the input of status EM. If one of the terminals contains an input-only or an output-only peripheral, then obviously the transmission of data will be simplex. The sender's data will be assembled into a buffer. When the PCU terminates the block by sending ETB, the buffer will be queued for output to the receiver. When the MSC has another buffer available to accept data, the sender will be re-enabled by means of the TX code.

When a block of data is available for output to a terminal, the MSC despatches a status RX character which causes the PCU to generate a series of status READY codes, requesting each byte from the MSC store as the acceptor peripheral becomes ready for them [2] The RX code also causes the PCU RECEIVE lamp to light. This lamp stays on at the acceptor terminal for the duration of a simplex call, thus an operator is informed that transmission is not yet complete even though the peripheral may be currently inactive.

The transfer of blocks between terminals is double-buffered to smooth the operation of the peripherals. However, the devices are effectively 'tied together', being constrained to operate at the same overall data rate. This method, the storing and forwarding of messages on a block by block basis, is sometimes referred to as 'cut-through'.

If the connected terminals are both capable of input and output, a half-duplex mode of operation is permitted in which the devices exchange blocks singly or in groups. The mechanism of block transfer is the same as for the simplex case. However, the MSC automatically enables a terminal to send whenever there are no blocks awaiting output. This means that the 'turnaround' procedure, the convention

he may type his password. One important exception is the 'escape' function which allows the user to break in while his printer is operating, this being implemented as the network control function INTERRUPT.

The use of network terminals falls into two phases, the heading phase and the communication phase, both of which have been described above. The operation of a terminal is controlled at all times by the interchange of status characters between PCU and MSC, this interaction being referred to as a 'control procedure'. For all simple peripheral devices which may form part of a network terminal, the control procedure is the same. The simple device control procedure is described further in section 3.

In computer terminals, most functions of the PCU will be taken over by the subscribing computer itself. One of the most important differences between computer and simple terminals is that the former can operate in a multi-access mode, engaging in concurrent communication with a number of other terminals. The control procedure is split into two levels, the lowest of these controlling the flow of data blocks between subscriber and MSC by the exchange of status characters. Each data block will contain the address of the other terminal involved in the communication and status information of relevance to the particular interaction, which is used by the second level of the control procedure. This information will be held in a fixed position in each block. It is not proposed to describe the operation of computer terminals in any more detail in this paper.

A number of facilities may be made available to users, by the MSC, through the medium of heading commands. Some of the facilities which could be offered are discussed in[4]. However, the only feature to be offered in the first version of the NPL network will be the permanent storage of heading information. A terminal having a permanent heading is able to set up a call merely by the use of the HELLO control button. This facility will be useful for those terminals which normally access only one destination, being equivalent to having a private wire, and those for which heading input is inconvenient, for example, paper tape readers. Terminals will be divided into two classes, priviledged and non-priviledged, at the discretion of the network management. Terminals belonging to the former category are allowed to input

permanent headings on behalf of non-priviledged terminals. Thus a keyboard device, for example, may be used to control the access to the network of a group of other peripherals.

3. The organization of the MSC software

The MSC is a real-time multi-access system controlling the operation of terminals in a time-shared manner. The software has a generalised structure in which the various system functions are carried out by entities known as 'Processes', which may conceptually run in parallel with each other.

The problems of organizing the co-operation of such Processes have been discussed by Dijkstra[5] in terms of synchronisation operators ("semaphores"). In this system the required synchronisation is achieved by allowing Processes to communicate with each other using fixed format messages. A technique similar to this has been used in the RC4000 multiprogramming system[6].

There are two types of Process in the MSC system. Normal Processes (NPs) operate on an input queue of messages, possibly generating messages for other NPs as a result. Interrupt Processes (IPs) are activated in response to hardware signals (i.e. interrupts) and may generage messages for NPs as a result. IPs may be regarded as acting on an input queue of 'hardware messages', which they may process in their entirety or which they may turn into standard internal messages for the attention of NPs.

The time-shared execution of the NPs is controlled by a primitive operating system called the Monitor, according to a simple strategy based on the state of the NP input queues. The Monitor also controls the allocation of temporary working storage to NPs. In the MSC system these storage areas will be used, for example, to buffer the data passing between two 'connected' terminals.

The operation of the Monitor is described in more detail in Section 3.1. Its function is essentially independent of the hardware and software features, embodied in the IPs and NPs, which give the MSC system its particular characteristics.

When a NP is in execution, the Monitor has no means of forcibly regaining control. In addition, it has no control over the execution of IPs. This lack of

constraint is not inappropriate in a system in which the Processes themselves represent the main operating system, and is desirable in the interests of simplicity and efficiency.

While the standard message interface is the principal means of synchronisation between Processes, it is clearly necessary to have semaphores linked with the operation of IPs, for example where common data areas may be read and written both by a NP and an IP. Since the operation of IPs is tied to a computer's interrupt hardware, these synchronising operations can most simply be implemented by means of the (selective or non-selective) inhibit/enable functions which are provided on most modern machines.

The MSC operating system may be described as 'event-driven'. Thus each status word, for example, originating from the communication network hardware will ultimately cause an interrupt at the central computer. This signal will in turn cause an IP to be activated, which may generate a message for a NP. The NP receiving the message may extend the train of events to embrace other NPs before the system activity triggered by the original signal is completed. Clearly, therefore, the software framework described here is a very natural one for this type of real-time system. However, it is imperative that overheads due to message transfer and scheduling operations should be very low, since the amount of CPU time required to deal with a message will usually be small, i.e. a few milliseconds.

When a real-time system is to be implemented using this queue-processing approach, it is important to ensure that strictly periodic functions, or functions which must be undertaken within a fixed time-limit, are not performed by NPs. One may only talk of the mean time which an NP takes to respond to a message, and this mean will in turn depend on the overall system loading. Such time-related operations are the province of the IPs, and most such systems will require an IP linked to a clock interrupt. The hardware of the NPL network is such that the MSC software has no strictly time-related scheduling requirements.

Sections 3.2 and 3.3 describe the way in which the MSC's operations are implemented within the basic framework outlined above and in Section 3.1.

3.1 The Monitor

All inter-Process messages pass through the Monitor and are the means by which it controls the operation of the system. In addition, messages concerning store management pass between NPs and the Monitor.

All messages are held in queues whose organization is shown schematically in Figure 1. Each NP input queue (NPIQ) is serviced by its associated NP wheneter the latter is executing. The currently executing NP may place messages into the single output queue (NPOQ). The NPOQ is serviced by the Monitor when it regains control; store management messages result in appropriate Monitor actions, while messages for other NPs are added to the ends of the specified NPIQs. IPs place messages into a single output queue (IPOQ) which is serviced periodically by the Monitor; IPs may only generate messages for NPs.

The Monitor maintains a state variable for each NP. A NP can be in one of four states: Idle (I), Waiting for compute time (W), Waiting for a store block (WB) or Computing (C). The transitions between these states are shown in Figure 2.

Once a NP enters state I, it will remain there until the Monitor receives a message addressed to that NP. When this happens, the NP is transferred to state W and its identifier is placed at the end of a 'Waiting for compute time' queue. This queue is serviced by the Monitor on a 'round-robin' basis, the next NP to be executed at each stage being the one whose identifier is currently at the head of the queue. A priority scheme superimposed on this mechanism is introduced below.

When the NP reaches the head of the W queue, it is placed in state C and its execution commenced. When the Monitor regains control, it scans the NPOQ. If a message requesting a store block is found, the NP is transferred to state WB and its identifier placed at the end of a 'Waiting for store block' queue. This queue is serviced periodically by the Monitor, which grants store blocks in order of request until either all outstanding requests are satisfied or until no blocks remain. When a store block is allocated to a NP, a message containing the block address is placed in its NPIQ and the NP transferred to state W.

If the currently terminated NP has made no store block request, the Monitor

will place it in state I if its NPIQ is empty, otherwise it will be returned to state W.

A flowchart summarising the operation of the Monitor is shown in Figure 3. It consists simply of a main loop which is cycled once for each NP activation, and a subsidiary loop which may be cycled when system activity is low. While NPs are only activated in a controlled fashion by the Monitor, IPs may be activated at any time by the system hardware. The Monitor therefore samples the results of IP activity at the beginning of every cycle, scanning the IPOQ into which all IPs place their messages. While the Monitor is cycling the idle loop, the system is simply waiting for the next external event to occur.

NPs are allocated to one of four priority levels according to the importance of their tasks. For each level there is a W and a WB queue. The next NP to be activated is that at the head of the highest priority W queue. Thus the highest priority level having any waiting NPs retains control until all its NPs revert to I or WB states or until a higher priority level becomes active. The WB queues are serviced in a similar manner.

The Monitor changes NP priority dynamically to prevent large message queues building up. Each time that a message is added to a NPIQ, the queue length is checked. If this exceeds a preset limit, the NP priority is raised by one level, the NP identifier being removed from its current position in the W or WB queue and placed at the end of the corresponding queue for the higher priority level. After a NP execution, the length of its NPIQ is again checked. If this is less than a second preset limit, the NP priority is returned to normal.

The Monitor controls the use of a dynamic working store which, in the present implementation, is divided up into blocks of a fixed length. Free blocks are held on a chained list; a NP may 'book' a block by the mechanism described above. When a NP has finished with a block, it returns the block address in a special message. The Monitor does not guard against the possibility of a 'deadly embrace' situation arising[5], in which the pattern of store block requests is such that all NPs end up in the WB state. This decision was taken in the interests of simplicity, the

necessary checking algorithm being rather time-consuming. In systems where the behaviour of all Processes in this respect is known, it is possible to ensure at the outset that such situations cannot logically arise.

Since the number of messages in the NPIQs can grow very large at times of heavy system loading, these queues are implemented as simple chained lists. The NPOQ is similarly implemented. The message queues are backed up by a list of free message cells. When system activity is high, the Monitor may replenish the free list by taking a free block from the dynamic working store. Conversely, when activity is low, a garbage collection phase may be entered, in which references to a selected block are removed and the block returned to the free block list.

The IPOQ is organized as a circular buffer of message cells. The use of a separate mechanism from the NP message queues minimises the number of interrupt lockouts which would otherwise occur. However, since the IPOQ is of fixed length, the system must safeguard itself against overflow by locking out all interrupts when the queue fills. In this connection, a 'software interrupt' flag is set when the number of messages reaches a preset limit. This flag is checked periodically by the currently executing NP and, if set, will cause a return to the Monitor where the backlog of interrupt activity can be dealt with. The probability of total interrupt lockout can be adjusted by changing the maximum length of the IPOQ and the level at which the software interrupt flag is set.

The interface between NPs and the Monitor consists of two global subroutines, one to take messages from the head of the appropriate NPIQ, the other to append messages to the tail of the NPOQ. A third global subroutine is used by IPs to place messages at the end of the IPOQ. This subroutine is also responsible for setting the software interrupt flag and for locking out all interrupts when the queue becomes full.

Other aspects of Process discipline, in particular the decision as to when a NP is to return control to the Monitor, are implemented by the Processes themselves. Thus a NP will return control whenever its NPIQ is found to be empty or whenever the software interrupt flag is found to be set. In addition, NPs have access to a

timer location which allows them to check the (elapsed) time of execution for each activation, control being returned when a set limit is reached.

3.2 The message switching software

Figure 4 shows the main hardware and Process components of the MSC system proper and the communication paths between these components. The principal software components of the system are the Terminal Processes (TNPs). To each network terminal there corresponds a unique TNP which controls all aspects of the terminal's interaction with the MSC.

The remaining Processes implement common system functions, forming a suitable environment for the operation of the TNPs, which are therefore described in the last subsection (3.3).

3.2.1 The logging processes

The Logging NP (LNP) receives coded messages describing various types of network hardware failures from TNPs and system Processes. These messages are turned into textual form and output to an Alarm Printer. This printer is local to the MSC (i.e. it is not itself a network terminal) and its character-at-a-time operation is controlled by the Alarm Printer IP (APIP), which passes a message to LNP when the output of a report to the printer is complete.

Messages giving call accounting information are received from TNPs and cause LNP to output a record to a paper tape punch. The Logging Punch IP (LPIP) controls the output of each record, in a similar manner to APIP.

Lastly, the receipt of a message from the Clock NP (CNP) causes LNP to generate a 'snapshot' summary of the system's current operating status. This information is output to the tape punch; it includes the lengths of the message queues and other parameters of relevance to Monitor operation. The logging tapes will be processed off-line to give a record of MSC operation which should be useful for gauging system performance and adjusting parameters so that this may be improved. In the case of the NPL network no subscriber charging procedures are planned for the present.

3.2.2 The Clock Processes

The MSC system has an interval timer which is set to activate the Clock IP (CIP) every second. CIP maintains a time-of-day indicator which may be accessed by NPs for 'time-stamping' reports or for long-term timeouts. NPs may also access the interval timer location (which is incremented every 20 milliseconds) for short-term timeouts, in particular execution timing, mentioned in Section 3.1.

Every 10 seconds, the CIP sends a message to the Clock NP (CNP), which is responsible for sending messages to all NPs which require to be activated periodically. This task is given to CNP rather than CIP because of the limited capacity of the IPOQ. CNP holds three units of information for each NP, a flag which indicates whether the NP requires timing messages, an integer specifying the interval (in 10 second units) and a current count.

The clock messages received by NPs cannot represent an accurate interval of time because of the queueing inherent in the system. The messages are intended for the periodic triggering of NP activity which does not need to meet rigid time constraints. Thus a signal will be sent to LNP every 5 minutes, say, to trigger a system snapshot; the fact that the interval may vary to some extent is of no importance here, since the record can be time-stamped by reference to the time of day indicator.

3.2.3 The I/O Hardware Interrupt Process (IOHIP)

The IOHIP is activated in response to interrupts from the special purpose hardware which interfaces the communication network proper to the MSC. This hardware in fact consists of two units whose functions are described in[2].

The I/O hardware places 7-bit status signals into a 'status dump' in the MSC core store. Each status signal is accompanied by a 9-bit address which is added by the network multiplexers to uniquely identify the originating terminal. Three types of status signal will be distinguished in describing the operation of IOHIP, (a) 'Inoperable' signals, (b) 'Normal' signals and (c) 'End of range' signals.

(a) Inoperable signals indicate a hardware-detected malfunction in some part of the network. The signal and the address are passed in a message to the Failure NP (qv).

(b) Normal signals are all those (apart from status READY) which may be received from the terminal PCU. The terminal address is applied to a conversion table to obtain the identifier of the associated TNP, to which the status code is then passed in a message. If the terminal address is found to be illegal (i.e. if there is no corresponding TNP), the status code and address are passed to LNP.

(c) End of range signals arise when a data input or output operation terminates, the I/O hardware transferring an EOB or a READY code respectively to indicate that the end of the buffer has been reached.

To each terminal address there corresponds a pair of 'command words', held in core, which are used by the I/O Hardware to control the transfer of data to and from the terminals. Since the hardware cannot be selectively prevented from transferring data following input by an end of range signal, it is necessary for IOHIP to reset the command words to a common 'garbage buffer' to prevent illegal data input from corrupting store (this might occur, for example, if an undetected address corruption has taken place). To make the system completely safe, the hardware ceases to function until it is reset by a signal from the central processor. IOHIP changes the appropriate command words before sending this signal.

Subsequently, the EOB and READY signals are processed as in (b) above.

3.2 The Failure Normal Process (FNP)

Messages indicating hardware-detected failures in the network are passed on from IOHIP to FNP, which is responsible for interpreting them.

Whenever a multiplexer detects the failure of an inferior hardware unit (i.e. a unit nearer to the terminals), it will respond to any outgoing information addressed to the failed channel by returning one of the status codes INOP1, INOP2 or INOP3, depending on the level of the multiplexer in the hierarchy[2]. Whenever a PCU discovers that its attached peripheral has failed, it will respond with the INOP1 code, whatever its level of attachment.

In order to interpret the incoming INOP codes, FNP needs to know the current network configuration. The structure is represented by three arrays, M1, M2, M3,

each containing information on the state of the multiplexer subchannels at the corresponding level in the hierarchy. Three units of information are required: (i) whether the channel is in use (connected to inferior hardware) and, if so, (ii) whether the hardware is a terminal or a multiplexer, and (iii) whether the channel is inoperable.

The level of the failure indicated by the INOP code, and the associated address determine the affected subchannel. FNP marks the subchannel as inoperable and scans the arrays to determine the addresses of all terminals dependent upon it, sending a failure message to each associated TNP. Whenever the failure of a multiplexer is detected, a message is also sent to LNP.

If the subchannel was already marked as inoperable, no further action is taken, thus avoiding repeatedly sending failure messages to the TNPs and to LNP. FNP must be informed when the subchannel is available again, however. It is the responsibility of each TNP to regain contact with its terminal after a failure. When this has been achieved, a message is passed to FNP causing it to clear all inoperable marks on the 'path' to the terminal.

3.3 The Terminal Normal Processes

A TNP deals with all aspects of the communication between its associated terminal and the MSC. A single re-entrant program module acts as a TNP for all terminals of the same class. The class of a terminal is determined by the hardware and the type of control procedure needed to interface the given peripheral device to the network.

The PCU described in [2] and in section 2 of this paper, and the associated control procedure, is capable of interfacing a wide variety of device types. Other information of relevance to the terminal's interoperation with the MSC, such as the peripheral's input-only or input and output capability, character code and so on, can be parameterised. The module dealing with this single device class is described below. The other principal device class relates to subscribing computers, for which a different type of PCU and control procedure, briefly discussed in section 2, will usually be necessary.

The module acting as TNP for all simple terminals is divided into three levels. The Scheduler level is responsible for implementing the required NP discipline and for routing incoming messages to the other two levels. The Master level (ML) deals with the overall aspects of terminal operation, including communication with other TNPs, data buffer management, heading interpretation and the provision of user-oriented facilities. The Control Procedure level (CPL) is concerned with the detailed handling of the terminals, checking the incoming status codes and initiating data transfers.

3.3.1 The Scheduler level

On initial entry from the Monitor, the module has to decide which TNP it currently represents. For this purpose the Monitor conveniently leaves the identifier of the currently executing NP in a globally available location. The identifier is used to load the base address of the appropriate TNP data area into a "signpost", from a table of base addresses. The fact that NPs can return control to the Monitor at a convenient point means that the amount of information which has to be held over between activations of a TNP can be minimized.

The setting of a control variable determines the next step taken by the Scheduler. If the current message processing activity is incomplete, the appropriate level is entered; CPL and ML may exchange internal messages during such activities. Otherwise, the next message is taken from the NPIQ, its destination decided and control again passed to the appropriate level. Signals from IOHIP, FNP, CNP are dealt with by CPL, signals from other TNPs and from the Monitor, by ML.

On re-entry from CPL or ML, the Scheduler performs the termination checks discussed in section 3.1. If control must be returned to the Monitor before the processing of the current message is complete, the Scheduler ensures that the TNP will be re-activated in due course by placing a dummy message addressed to itself into the NPOQ, (provided that the NPIQ is empty). If the TNP can continue execution, the Scheduler recycles for the next phase of operation. The control variable will always be set appropriately an exit from CPL or ML.

3.3.2 The Master level

ML exchanges messages controlling the progress of a call with other TNPs and

messages governing the operation of the terminal with CPL. The principal messages are summarized in Table 1.

The response made by ML to an incoming message will depend on the current state of the call being processed. The operation of ML is controlled by a state variable and a message/state table, which contains the entry address as of 'action' routines to deal with every possible combination of circumstances. The states and transitions are shown in a simplified form in Figure 5. Each transition is labelled with the identity of the incoming message (underlined) which caused it. Messages which are output by ML before the new state is entered are also shown (not underlined).

In state F, ML is free either to set up a call on behalf of its own terminal or to accept a call from another TNP. Receipt of TRANSMIT from CPL signals the former event; a request block (RQB) message is sent to the Monitor and state H1 entered to await the response, BRG, giving the address of the allocated block. The size of the store blocks controlled by the Monitor is chosen to be sufficient to hold two maximum length (128 byte) buffers plus some housekeeping information, since this is their main use in the MSC system. A single block is used to buffer data input and output while a call is in progress, the buffer references being exchanged, as necessary, between the two "connected" TNPs by means of the OPB and EB messages. The TNP initiating a call is responsible for obtaining a store block and for returning it at the end of the call.

When BRG is received in state H1, the appropriate action routine checks to see whether a permanent heading is stored on behalf of the terminal. If so, a CALL message can be output immediately to the TNP whose address is specified in the heading. Otherwise, a TX message is sent to CPL to initiate the input of a heading, ML entering state H2 to await its completion.

This event is signalled by the receipt of the ETB message from CPL. ML converts the stored characters into a standard code and then interprets the heading command. If the command is in error, a QUIT message is sent to CPL to cancel the call and state F re-entered. Otherwise, a CALL request is sent to the TNP whose address is specified in the heading, and state H3 entered to await the response.

The response to CALL may be REJ, in which case ML sends QUIT to CPL, or it may be ACC, when the data transmission phase of the call is commenced. In the CALL and ACC messages, the TNPs exchange information on the terminal I/O capabilities, to determine whether the 'connection' will be simplex or half-duplex. The parameter which specifies the terminal capability must also be used to decide whether a call can be initiated, thus the TNP for an input-only device can never accept CALL requests.

In section 2 it was mentioned that certain priviledged terminals are allowed to input permanent headings on behalf of other terminals. The CALL message is used to impart this information to the specified TNP. If the response is ACC, ML outputs TX to CPL and returns to state H2, allowing the subscriber to input another heading. Thus a user always obtains a positive response if his attempt has succeeded. Not all subscribers will wish their terminals to have permanent headings; a parameter held by each TNP indicates the status of the terminal in this respect.

Having accepted a call, ML enters state O1 to await the first buffer of data from the calling terminal. Having received confirmation of a call, the ML for the calling terminal sends TX to CPL to initiate data input, and enters state I1 to await the completion of this operation. If the call is simplex, ML for the sending terminal will cycle round the I1 and I2 states. Similarly, that for the receiving terminal will cycle round states O1 and O2.

When CPL signals the completion of input, a reference to the data buffer is passed in an OPB message to the other TNP. If the second buffer is available, ML again sends TX to CPL and returns to state I1. Otherwise state I2 is entered to await the return of a free buffer, by means of the EB message, by the other TNP. If the receiving terminal has a higher data rate than the sender, the EB message will be received while ML for the latter is in state I1.

When ML for the receiver, in state O1, is given the OPB message, it passes RX, with a reference to the buffer, to CPL which initiates the output process. ML then enters state O2 to await ETB from CPL. When this arrives, the buffer reference is returned to the connected TNP and state O1 entered to await the next OPB. If,

however, there is already a buffer queued for output, state 02 is re-entered immediately.

If the call is half-duplex, ML will switch between the 0 and the I states, since a terminal is always enabled to send when there is no information available for output (see section 2). Thus, on exit from state 02, ML may make the previous output buffer available for input and enter state I1. While a low speed terminal is sending a number of blocks to a higher speed terminal, the ML for the latter will switch between states 02 and I1. In state I1, receipt of the OPB message causes ML to return the unused buffer. If one of the subscribers breaks the half-duplex operating rules, it is possible for data to be lost.

Receipt of INTERRUPT via CPL while ML is in state 01, causes the output to be cancelled and the buffer made available for input. Other messages and transition paths, not shown in Figure 5, are used to ensure that the previously sending TNP is forced into state 01.

Another aspect of ML operation not shown in the diagram is call termination. The ML receiving an EM message from its terminal, or a failure indication, passes an END message to the connected ML which acknowledges with END. Thus the call initiator, after receiving END, may assume that all references to the store block, e.g. by command words, have been cancelled. This ML then places call accounting information in the block and passes its address to LNP which eventually returns the block to the Monitor.

3.3.3 The Control Procedure level

CPL has a similar organization to ML, its operation being controlled by a message/state table and a series of 'action' routines to deal with every possible circumstance. A simplified state-transition diagram for CPL is shown in Figure 6.*

---

* Table I of companion paper[2] gives the status codes exchanged between CPL and the PCU. While incoming status codes are received by CPL as messages from IOHIP, the output of these codes takes place directly.

CPL is concerned with the detailed control of terminal operation, and in particular with the detection and correction of faults which may arise from the loss or corruption of information transmitted on the serial links, or the malfunction of the various system hardware units.

The states of CPL correspond to the phases of terminal operation. During a simplex call, CPL for the receiving terminal will cycle round the RECEIVE and WAITING TO RECEIVE states. That for the sender cycles round the WAITING TO SEND and SEND states. During a half-duplex call, switching between these pairs of states will occur. If ML requests CPL to switch from the SEND to the RECEIVE state, or vice-versa, status EM is first output to halt terminal operation, followed by status ENQ. When the response, status STATE is received, CPL sets the terminal command words to point to the new I/O buffer and outputs the appropriate status code, TX or RX, to re-activate the terminal. This step cannot be taken immediately because data 'in transit' could corrupt the newly supplied buffer.

CPL is forced back into the IDLE state by the receipt of EM from the terminal, or when ML sends a message to end the call, when CPL will output EM or QUIT to the terminal.

Only 'meaningful' status codes, i.e. those causing CPL to change state, cause messages to be sent to ML. The receipt of invalid or out-of-context codes will result in an error message being sent to LNP. If status NACK is received, indicating that the PCU had been sent an invalid status code, the previous status output is repeated. To guard against a hardware malfunction generating an endless succession of NACKs, CPL counts the number of status repeats and takes a failure action, described below, if the count reaches a preset limit.

An important function of CPL is terminal polling. Each time a message is received from CNP, status ENQ is output to the terminal. The response, status STATE, indicates the state of the PCU lamps. If the state of the terminal does not correspond to the current CPL state, an error message is sent to LNP and a status code output to reset the terminal. If the response is not forthcoming before the next message is received from CNP, CPL again sends a message to LNP. Two successive response failures are taken to indicate terminal inoperability.

If a terminal failure is detected by CPL or if a message is received from FNP, an INOPERABLE state is entered and a message sent to ML which will cause the current call to be cancelled. In this state, CPL polls the terminal each time that it is stimulated by CNP. When two successive STATE responses are obtained from the terminal, it is assumed that the terminal is available again; a message is sent to ML to indicate that calls may be accepted and CPL enters the IDLE state. A message is also sent to FNP at this stage.

4. Concluding remarks

The software of the NPL communication network is still in the development stage at the time of writing this paper, so that no assessment of the strategies described above can be given in the light of operating experience. However, it seems worthwhile to make a few comments on the implementation of the MSC software.

The machine chosen for this system is a Honeywell DDP-516 and the programs are being written in the standard assembly language, DAP-16. The modular operating system concept has proved invaluable. It is doubtful whether the task of implementing such a system could proceed successfully without the imposition of a general framework which allows the detailed specification, coding and testing of modules to be carried out independently once the basic functional divisions have been decided, and the messages interchanged between the modules have been defined.

Thus the Monitor was separately tested using a driver program to simulate the behaviour of IPs and NPs. Each NP has been tested individually by providing a program to interpret commands from the computer control typewriter and pass them on as as standard messages to the NP, intercepting and printing out its responses. In the case of the TNP, the routine sending status codes to the I/O hardware was replaced by one to print the information at the console, so that testing could proceed independently of the hardware. In the case of the IPs, hardware independent testing is not possible. However, the philosophy of the MSC system is that each IP only undertakes the essential interrupt processing tasks, passing messages to NPs which will complete the work under control of the Monitor's priority scheduling system. This approach also simplifies the testing. The implementation of each of the main

levels of the TNP by means of message/state tables has also considerably eased the subsequent detailed design, coding and testing phases of this module. The approach has the important advantage that the designer is forced to consider every unlikely circumstance and cater for it explicitly.

Since the NPL network is intended to be an experimental system, it is likely that the operating system will evolve by modification and addition as experience of its use is gained and as new types of peripherals are added. For example, new types of facilities thought desirable by users will entail changes to ML. CPL may need to be altered so that certain kinds of hardware failures may be detected more readily. This expected evolution of the software will be made much simpler by its modular construction.

Acknowledgements

The author would like to acknowledge the help of Keith Wilkinson, Rex Haymes and Sue Connelly in the implementation of the operating system.

The work described herein has been carried out at the National Physical Laboratory.

References

1. Barber, D.L.A. Experience with the use of the British Standard Interface in computer peripherals and communication systems. ACM Data Communications Symposium, Pine Mountain, Georgia,(October 1969)
2. Scantlebury, R.A. A model for the local area of a data communication network - objectives and hardware organisation. ACM Data Communications Symposium, Pine Mountain, Georgia,(October 1969)
3. Hartley, D.F. (Ed.) The Cambridge Multiple-Acces System, User's Reference Manual. University Mathematical Laboratory, Cambridge, (1968)
4. Wilkinson, P.T. and Scantlebury, R.A. The control functions in a local data network. Proceedings of the IFIP Congress 68, D16, (August 1968)
5. Dijkstra, E.W. Co-operating Sequential Processes. Mathematics Dept., Technological University, Eindhoven, (September 1965)
6. Brinch Hansen, P. (Ed.) RC4000 Software: Multiprogramming System. A/S Regnecentralen, Copenhagen (April 1969)

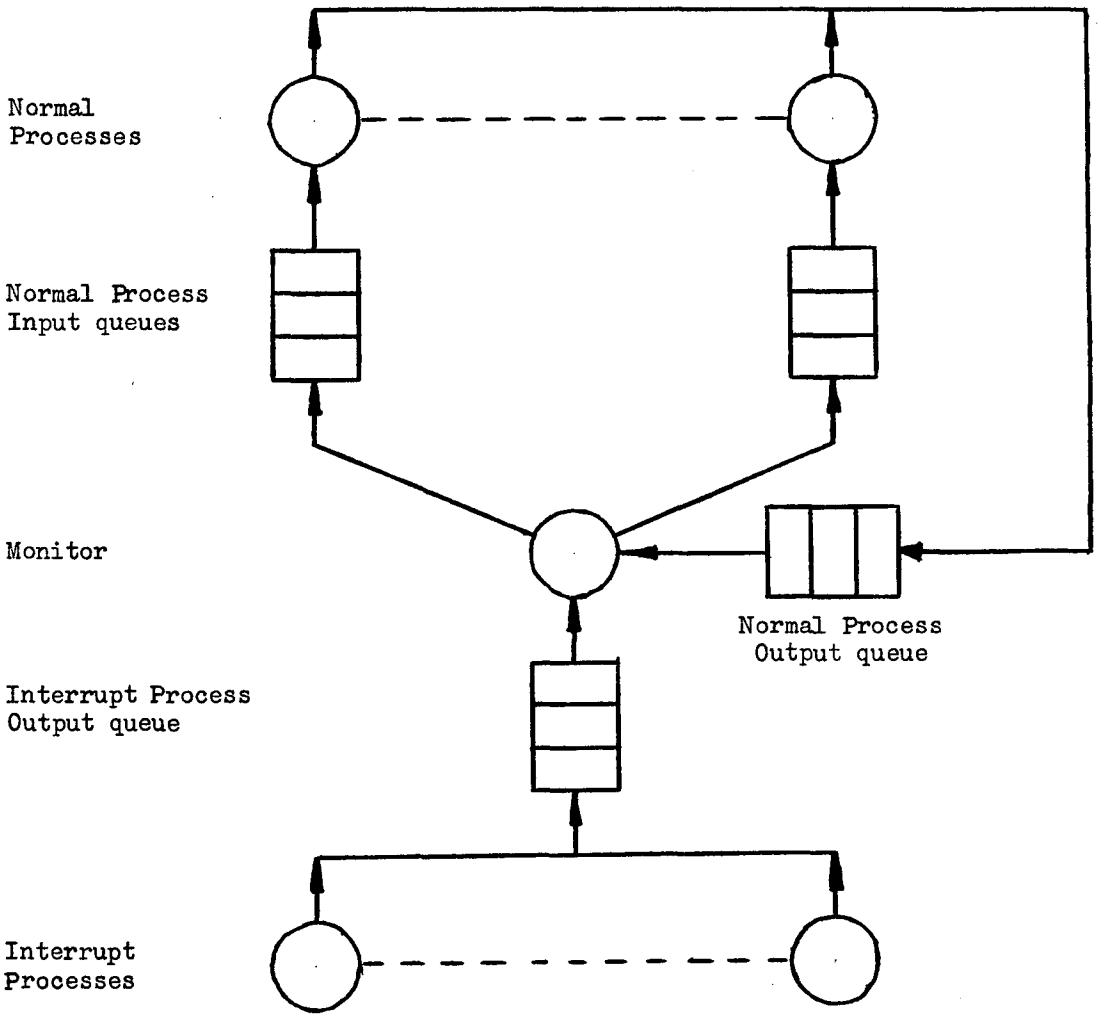

Figure 1 Process message queue organization

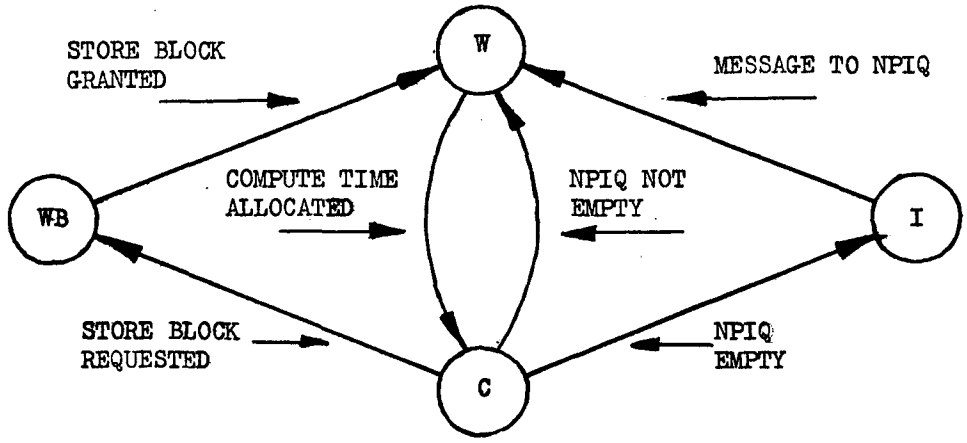

Figure 2 Normal Process state transitions

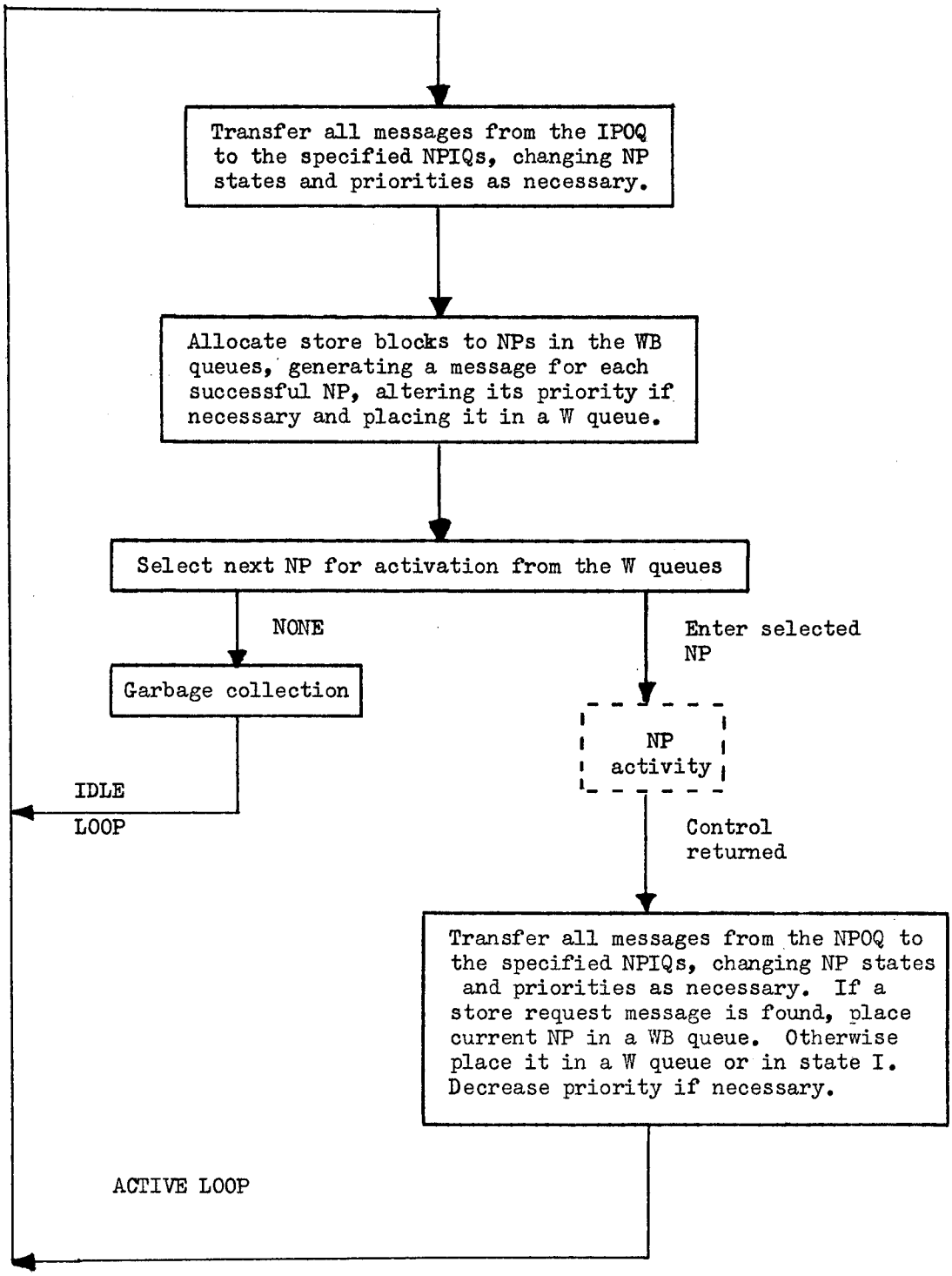

Figure 3 The Monitor operating sequence

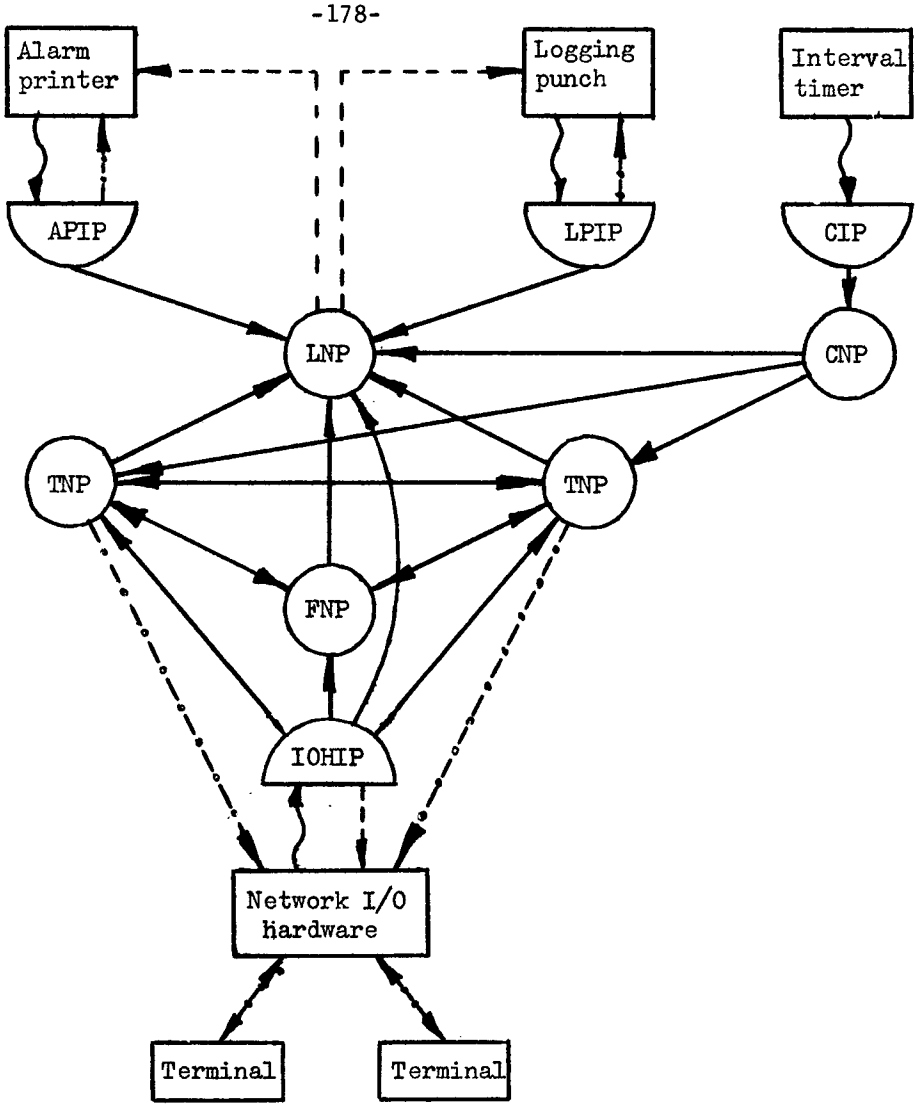

Figure 4 Main Process and hardware components in the MSC system

KEY

Normal Process

Interrupt Process

Hardware units

Normal processes

TNP	Terminal
CNP	Clock
LNP	Logging
FNP	Failure

Interrupt Processes

APIP	Alarm printer
LPIP	Logging punch
CIP	Clock
IOHIP	I/O Hardware

Message path

Control path

Data path

Interrupt path

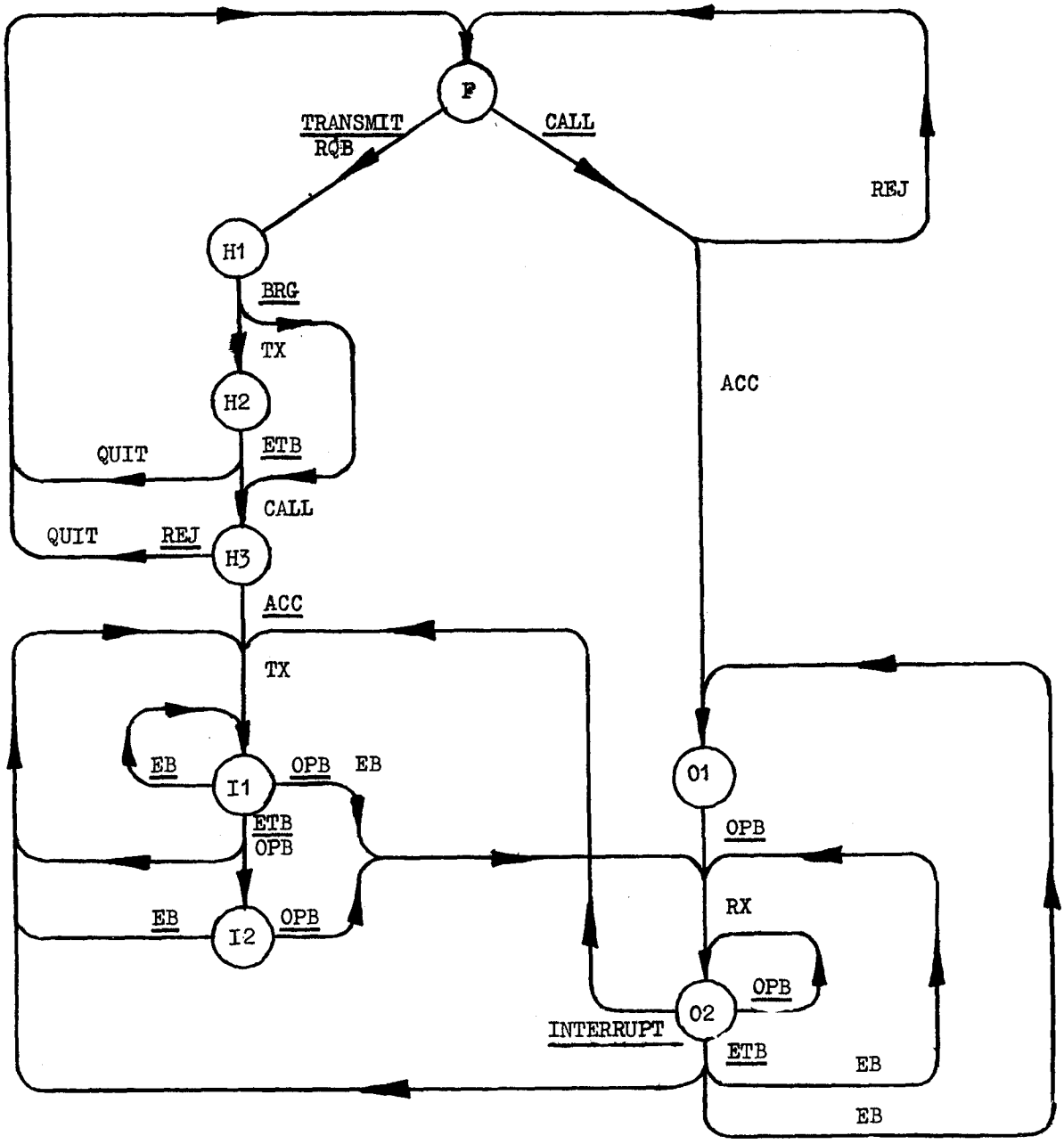

FIGURE 5 Master level state-transition diagram

The circles represent states, the lines transitions. Underlined messages are received, non-underlined messages are output. Refer to the text for a full explanation.

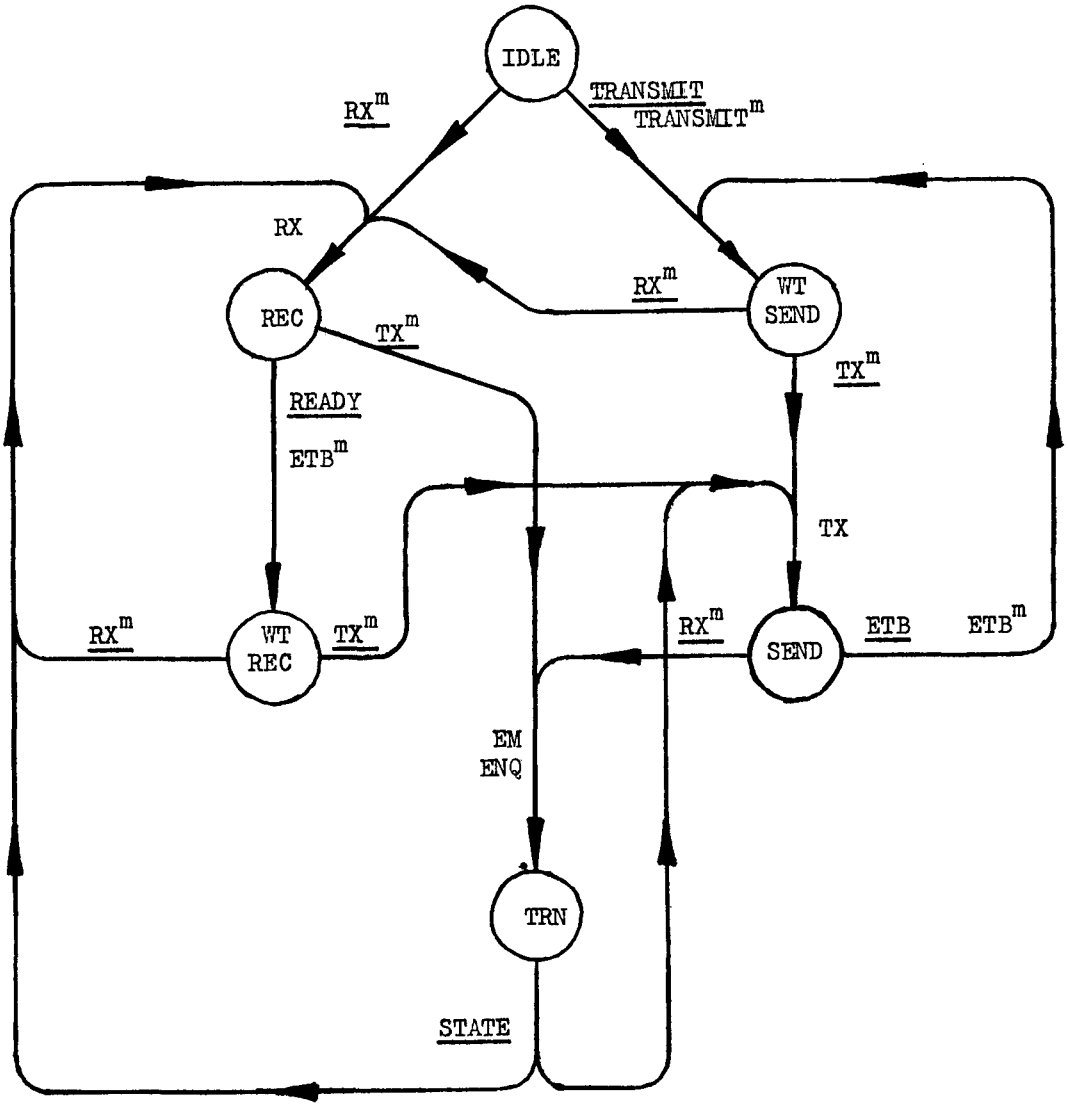

Figure 6 Control Procedure level state-transition diagram

Key as for Figure 5. In this case the superscript m indicates messages which are exchanged with ML. Non-superscripted messages are status codes exchanged with the terminal PCU.

Table 1

Master level control messages

Message	Significance
(a) Exchanged between TNPs	
CALL	Request to set up a call
ACC	CALL request accepted
REJ	CALL request unacceptable
OPB	Output the specified buffer
EB	The specified buffer is available for input
END	Terminate the call
(b) Sent to CPL	
TX	Initiate input to the specified buffer
RX	Initiate output from the specified buffer
EM	Terminate the call normally
QUIT	Cancel the call
(c) Received from CPL	
TRANSMIT	The terminal requests a call
ETB	The current input or output operation is complete
INTERRUPT	The terminal has requested to break-in
EM	Call ended by terminal