A STRUCTURE FOR INTERPROCESS COMMUNICATION IN A DATA COMMUNICATIONS HANDLER

H. F. Tibbals
Computer Unit, University of Durham, Science Laboratories,
Durham. DH1 3LE. England

Special Interest Groups: Operating Systems, Data Communications

The use of an implementation which models a simple single server queuing system to control transfers between processes in a communications handler is described. Exchanges between processes are carried by three word elements, called LINKs, which are used to transfer control information and to assign data to processes. The managements of interactions by the single queue structure is used to coordinate parallel, time-independent operation of the attached processes, with the interface provided by the LINKs giving a simple yet flexible interface between activities. The implementation is a special purpose operating system designed for use in a PDP-11 computer dedicated to handling transfers of control and data between a number of communicating processes, which are connected to the PDP-11 by communications lines, direct memory access interfaces and shared memory.

INTRODUCTION

In a communications handler a large, perhaps variable, number of processes are served by a fixed number of specific tasks. This environment permits a simplified approach to the problem of inter-process control and processor utilization by each process. As the number and nature of the tasks to be done for any process is fixed beforehand, requests for the execution of any task can be accepted as demanded, without suspending the operation of the process, provided:
1) the execution of the current task can be interrupted to provide urgent services of short execution time, and
2) newly arising requests for the execution of tasks can be queued for later processing.

In practice, the proper management of the queue of outstanding task requests may require a queuing structure with multiple priorities. An additional practical limitation is that continued execution of services on behalf of a process which has requests outstanding may be limited by protocols governing that process' communication with other processes. Relative process priorities and interchange protocols are matters dependent on the nature of the processes themselves, and in changeable environments the mechanism for interprocess communication should itself place a minimum of restrictions on process operation.

The mutual exclusion problem (1,2,3,4) at the task execution level may be dealt with by use of a queuing system to control the execution of tasks. (5,6) Mutual exclusion primitives such as semaphores (7,8) are used to control access to the task queue itself. (6)

The LINK Queue

In the THAD (Durham Terminal Access Handler) system the familiar concept of using a queue to impose serial order (9-12) on randomly occurring demands for service is applied in a very direct and literal fashion in the implementation of a dedicated communications processor. Requests for service are placed into a three word structure called a LINK, which is inserted into the queue by a chaining operation. On fulfillment of the; request, or in some cases its cancellation, the LINK is freed by removal from the queue structure. Free LINKs may be the property of individual processes, or may be one of a pool of shared LINKs held on a stack accessible to many processes.

Use of LINKs by Processes, Tasks and Services

A process is considered to be any independent source and/or sink of data and control requests which communicates with the THAD processor. Each process is represented by a control table containing the values of its state variables, dedicated addresses, etc., and a three word linkage element for each data input and output associated with the process. These linkage areas hold the head elements of chains of linkage elements which are used to queue input/output from/to the process. Input and output are chained to processes indirectly, using the same form of LINK structure that is used to transfer control between tasks and processes. In addition to the LINKs used for data chaining, each process has access to one or more LINKs which may be used to carry communications to other processes and tasks.

A task is any one of a set of services which a process may request to have executed on its behalf. Execution of a task may result in control and/or data being transfered to one or more other processes. A task may call for the execution of other tasks or subtasks. Tasks are distinguished from system and process service procedures in that tasks may only be executed by placing a request in a queuing system (called SHAD) which controls the scheduling of tasks and the transfer of control and data. Requests are represented by a linkage element chained into the SHAD queue. The way in which the linkage elements are used provides a simple mechanism for separating the delayed execution of a task on behalf of a process (to resolve conflicts with other processes) and the concurrent execution of services in response to events generated by the process in the meantime. The scheduling of multiple tasks by the same process is permitted along with the scheduling of tasks by another task. The LINKs are used as dynamic extensions of the process control tables, to expand as needed the capacity of the table to represent multiple outstanding control requests and queues of outstanding control requests and queues of outstanding data of multiple types.

The LINK elements may also be thought of as tokens which carry and control the flow of control and data information through the system. Control is acheived through restrictions on a process' access to LINKs, which are necessary for the scheduling of requests for transfer of data. The representation of control requests and data transfers in explicit form, as queues of LINKs, facilitates monitoring of the state of the system.

Service procedures are performed on demand without program scheduling. Their function is to handle events generated by processes or tasks and cause the scheduling of appropriate tasks to carry out any function which makes non-trivial demands on system resources, or which must be coordinated in terms of time and/or resources with other processes or tasks.

Special system services are available through procedure calls and program trap interrupts to insert and remove requests on the SHAD queue, chain data LINKs onto process tables, and perform similar services.

Events cause the execution of services either by generating a hardware interrupt or by setting a flag in a register or in shared memory which is examined on a clock interrupt service.*

To accommodate the roles of task execution, process service and system services, the THAD system divides the PDP-11 processor into two virtual machines. Task execution, running in processor interruptable state, and process servicing, which is interruptable only by a limited number of high priority event services, are given two dedicated registers each. The additional two general purpose registers are shared, along with the system stack pointer and the program counter. The system services use the same virtual machine as the process services. One of the two shared general purpose registers is used as a base register referencing the process control table of the currently executing process. The other shared register is used as a stack pointer to free buffer space which is shared by the two virtual processors.

The LINKs in the THAD system play a more important role in the operation of the system than do mailbox messages (13) or message buffers (14), in that the LINKs are used to carry the actual transfer of control from process service to tasks, and carry assignments of data from one process to another. In addition, the SHAD queue is the primary mechanism for the scheduling of all critical tasks.

Use of LINKs in Transfer of Control

A process is guaranteed one place in the queue of tasks to be executed for each dedicated, unshared control LINK which is contained as a part of the process' private storage area. Requests which rely on using shared LINKs are in competition with similar requests from other processes; hence the number of LINKs provided in common stacks may be used to selectively control the loading of the system. Individually owned LINKs may be used as tokens (15) to indicate completion of a task and control the initiation of new requests by a process. Stack pointers for individually owned stacks of LINKs may similarly be used as indicators of a process' state.

The format of a LINK used to carry a control request is:

WORD 1	WORD 2	WORD 3
W H O	W H A T	W H E R E

Where WHO is a pointer to the process table identifying the process making the request, and containing parameters and data for task execution. WHAT points to the entry point of the task to be executed, and WHERE is the queue chaining word

Use of LINKs in Transfer of Data

LINKs are also used to assign and transfer ownership of data, without relinquishing control from the currently executing process or explicitly requesting the initialization of a new task. This type of transfer is employed to hand data over to interrupt driven output services, for example, and to accumulate batches of data in chains to be dealt with by a task which is initiated by some data-independent means such as a clock interrupt. Asystem service procedure links data LINKs onto output chains and permits events to drive service processes where necessary.

Three data transfer modes are used, buffered, direct and dump. In the buffered mode data is held in fixed length buffer segments which are

* For hardware without interrupt facility (e.g. some microprocessors which are considered for use as sub-concentrators for terminals) registers and flags would be examined as part of the null SHAD task performed whenever no requested tasks were outstanding, or if necessary, examined between executions of scheduled tasks. In such an environment it would be necessary to define tasks so that execution of each separately scheduled task was of short duration.

taken from and returned to a stack of free buffer segments. In the direct mode data is assumed to be held in an unbuffered area; there is no need to transfer data to buffer segments or return freed buffer segments. Using this mode several processes may access the same data concurrently. It is used for shared system data and for transfer of data from a process' dedicated buffers or transmission blocks. Direct mode is also used to reduce overheads in messages, prompts, etc.

In the dump mode no LINK is used. The processes transfer data by directly referencing the data link area in the process control table. This mode is used only in certain system diagnostic and console functions.

In addition to the transfer modes outlined above, a data LINK carries information on the encoding, format, etc. of the chained data. The format of a LINK used for data chaining is:

WORD 1	WORD 2	WORD 3
H O W	W H I C H	W H E R E

Where HOW contains data length and modifier information, (such as mode, encoding special device functions, etc.) WHICH points to the beginning of a data string, and WHERE is a chain word.

Structure of the SHAD Queue and Server

The queue of outstanding task requests at any time is represented by a closed chain, or ring, made up of one LINK element for each request, plus a null LINK element, called the ANCHOR, which belongs to the queue server. The WHO word of the ANCHOR identifies task pointed to by the WHAT word. When no requests are in the queue, the chain word of the null element points to itself.

Tasks are driven by a system service whose basic operation is to unchain the current queue element at the head of the queue, which is always pointed to by the chain word of the ANCHOR element, and replace it as head by the next link in the queue. Having thus disposed of a completed task, or the null task, the server loads the base register with the contents of the WHO word, enters interruptable state, and transfers control to the address referenced by the WHAT word. When no requested LINKs are in the queue, the server operation has the effect of repeatedly executing the background task, which may be a simple cyclic activity such as diagnostic testing or execution of some program for one clock cycle. When any task is completed, it suspends itself by calling a system service which may return freed LINKs to the shared stack (depending on the information in the process table) before invoking the queue server.

Multiple priorities are implemented by means of multiple pointers to entry positions at which items may be chained into the ring queue structure. The basic, single priority queue structure is illustrated in figure 1.

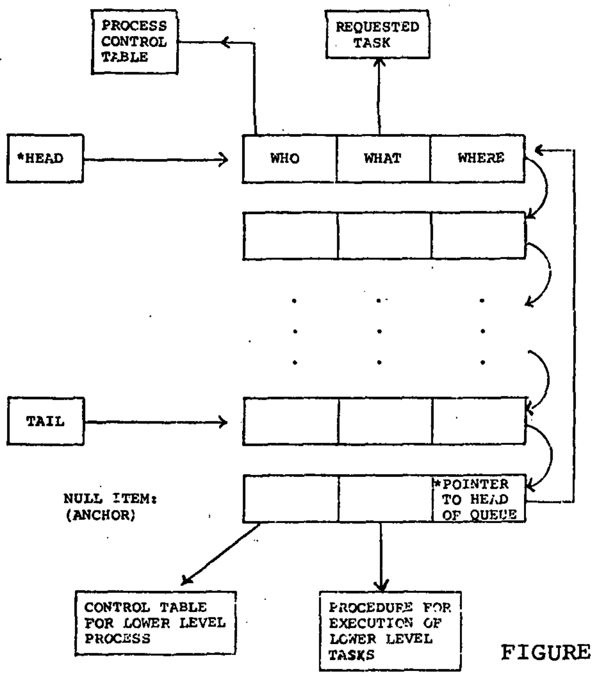
PROCESS CONTROL TABLE
REQUESTED TASK
*HEAD
WHO
WHAT
WHERE
TAIL
NULL ITEM: (ANCHOR)
*POINTER TO HEAD OF QUEUE
CONTROL TABLE FOR LOWER LEVEL PROCESS
PROCEDURE FOR EXECUTION OF LOWER LEVEL TASKS

FIGURE 1

Delays

If a task is unable to carry out the transfer of data from one process to another for any reason, it may reschedule itself onto the queue for a later retry. Alternatively, the data may be assigned to a pseudo-process which will be invoked to do diagnosis and garbage collection when the queue reverts to the null state.

Operation of the System

In operation the THAD system is similar to any event driven communications handler of transaction processor. (16) Interrupts occurring at random times are dealt with by the process service procedures, which make requests on the task scheduler, usually after accumulating data. State variables in the process control tables are set to condition the response of the service procedures based on the outstanding requests and on results of the execution of tasks.

Semaphore primitives are used at the most basic level to control transfers of data through device registers or shared memory. The THAD processor recognizes the data transfer as an event either through a hardware interrupt or through a periodic examination of semaphores by software. An information transfer event triggers the execution of a service procedure which places data into a buffer belonging to a process or inserts a task request into the SHAD queue. Figure 2 gives a schematic of the overall system operation, with process services represented on the left as interrupt service routines and tasks on the right as scheduled routines.

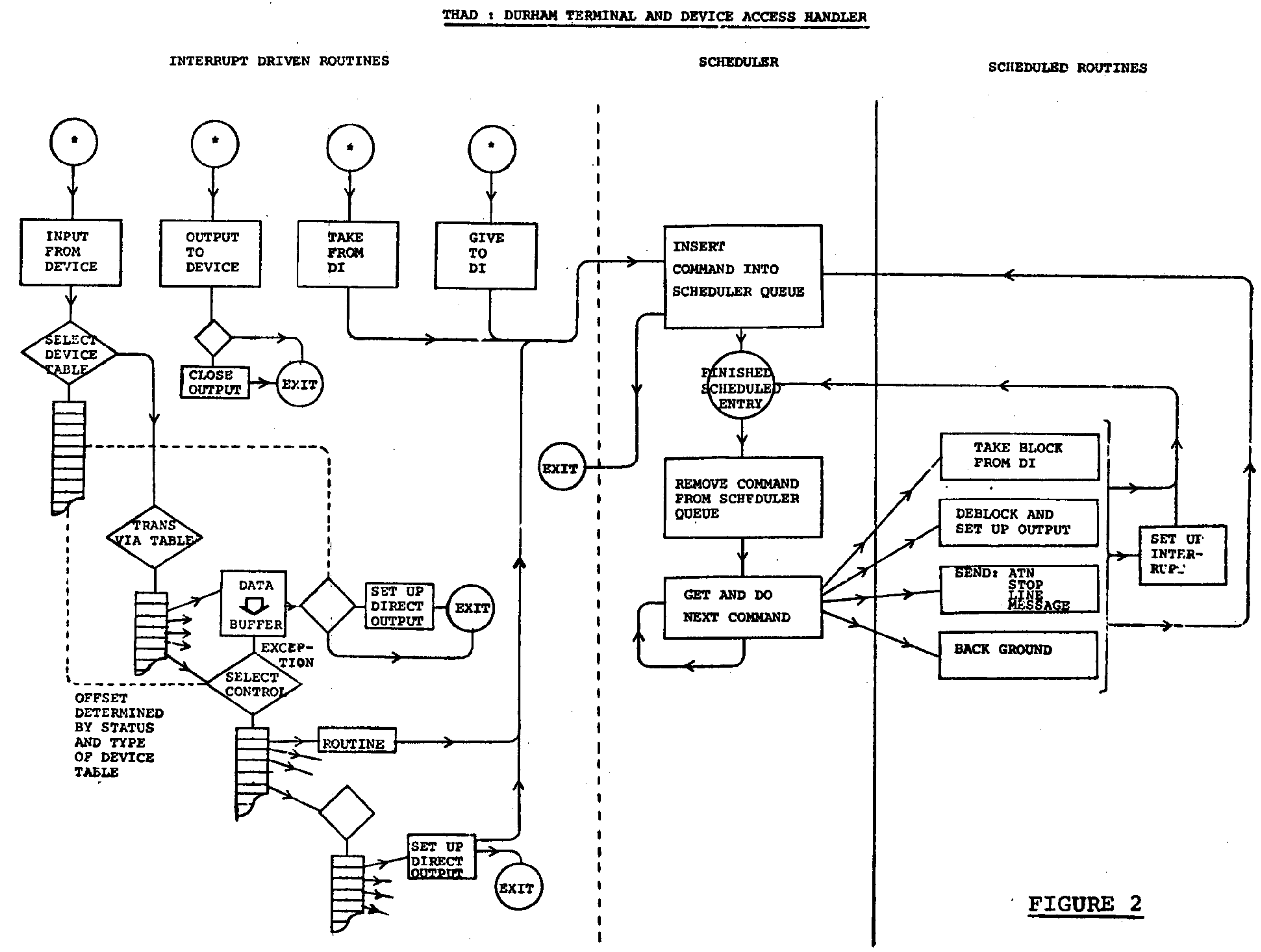

FIGURE 2

Service procedures for input from user terminal type devices are driven by decision tables referenced by combining the input information and the state variables held in the process control table. A character table distinguishes between control and data characters, carries out translations, and converts control characters into word offsets which are used to reference columns of a state function table. Each row of the state function table represents a different device state, with the items across the row being the addresses of procedures to carry out the commands available from each state. For some commands, and some data functions, another level of decision is made by referencing a table which chooses modifications of the command or data according to the device type.

Transfer of data between, for example, a terminal input process and a send process is done by a task which carries out any necessary translation, compression, blocking, etc. on individual messages handed it by the input process, and chains the partially prepared data to the process control table of the send process. Completion of a series of control events associated with the send process triggers the task which does the final preparation of a transfer block and starts up the send service procedures which carry out the transfer.

Processes which have direct memory access or which share memory with the THAD processor use the interchange between services and tasks to maintain an orderly sequence of negociation and carrying out of transfers, according to their various protocols.

Summary and Conclusions

The general mode of operation of the system is not particularly novel. The system provides, through the use of structures which are explicit models of such useful constructs as queues and tokens, events and actions, a tool which makes the organization and understanding of common data communications operations relatively easy.

An implementation of the system was made in assembler on a PDP11-20 and has been in service for the past two years. An earlier version of the somewhate complex environment has been described elsewhere (18). The terminal handler gives remote access to a large central IBM 370 and 360 complex (the Northumbrian Universities Multiple Access Computer) for users on conversational terminals and mini-computers, under conditions of high terminal usage. The current version is based on 16 K of core, supporting a direct memory access to an IBM 1130 computer, 32 multiplexed user devices on asynchronous lines at speeds from 110 to 2400 baud, and 5 directly connected asychronous lines. Devices attached to the lines include teleprinters, visual display terminals, storage tube graphics terminals, various laboratory mini-computers, and an Intel 8008 microprocessor. A schematic of an early implementation is given in Figure 2.

Support for synchronous BSC and full duplex lines is currently being implemented. A command interpreter has been added to the list of scheduled functions, to provide an expandable repertoire of selectable local device functions. Expansions currently being implemented, include addition of a disk-based spooling and backing store processor, distribution of some terminal handling functions onto microprocessor-based sub-concentrators, and alteration of the basic terminal image to conform to the British Post Office Experimental Packet Switched Network standards.

References

1. Dijkstra, E.W. Solution of a problem in concurrent programming control, Comm. ACM 8, 9 (Sept. 1965), 569.
2. Knuth, D.E. Additional comments on a problem in concurrent programming control, Comm. ACM 9, 5 (May 1966), 321-322.
3. Eisenberg, M.A. and McGuire, M.R. Further comments on Dijkstra's concurrent programming control problem, Comm. ACM 15, 11. (Nov. 1972), 999.
4. deBruijn, N.G. Additional comments on a problem in concurrent programming control, Comm. ACM 10, 3 (Mar. 1967), 137-138.
5. Lamport, Leslie. A new solution of Dijkstra's concurrent programming problem, Comm. ACM 17, 8 (Aug. 1974), 453-455.
6. Tibbals, H.F. A treatment of Dijkstra's concurrent programming problem by a formalism based on queuing theory, submitted for publication, Comm. ACM
7. Dijkstra, E.W. The structure of THE multiprogramming system, Comm. ACM 11, 5 (May 1968) 341-346.
8. Habermann, A.N. Synchronization of Communicating processes, Comm. ACM 15, 3 (Mar. 1972), 171-176.
9. Chan, W.C. and Chung, W.K. Computer controlled queuing systems with feedback, Proc. Inst. Elec. Engrs. 118, (1971) 1373-1377.
10. Adiri, I., Hofri, M. and Yadin, M. A multiprogramming queue, JACM 20, 4 (Oct. 1973), 589 -603.
11. Mitrani, I. A queuing model of priority multiprogramming, Univ. of Newcastle-upon-Tyne Computing Laboratory, Tech. Rept. No. 41, (Dec. 1972)
12. Cohen, J.W. Some aspects of queuing theory, Statistic Neerlandica 28, (1974), 55-67.
13. Spier, Michael J. and Organick, Eliott I. The Multics interprocess communication facility, Proc. Second Symp. on Operating Systems Principles (Oct. 1969) 83-91.
14. Brinch Hansen, P. The nucleus of a Multiprogramming system, Comm. ACM 13, 4 (Apr. 1970), 238-241, 250.
15. Furtek, Frederick. A new approach to Petri Nets, MIT Project MAC Computation Structures Group Memo 123 (Apr. 1975)
16. Feinroth, Y., Franceschini, E. and Goldstein, M. Telecommunications using a front end minicomputer, Comm. ACM 16, 3 (Mar. 1973) 153-160
17. Pettersen, Odd. Synchronization of Concurrent Processes, Staford University Computer Science Dept. Rept. No. STAN-CS-75-502 (July 1975)
18. M. Kolar, A.A. Young, P. Shelton and H.F. Tibbals. Sharing a Communications Line between Interactive VDU Terminals and a Batch Remote Job Entry Terminal via a specially designed Interface. Proc. Int. Conf. on Minicomputers in Data Communications (AIM, Liege, Jan. 1975)