DESIGN OF OPERATING SYSTEM KERNEL FOR A MICROCOMPUTER SYSTEM

By

SYLVANA KRISTANTI-SARI

Bachelor of Mathematics

University of Waterloo

Waterloo, Ontario

1977

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December 1979

DESIGN OF OPERATING SYSTEM KERNEL FOR A MICROCOMPUTER SYSTEM

Thesis Approved:

J. K. Phillips
Thesis Adviser

D. D. Fuhr

M. D. Grace

Norman N. Durham
Dean of Graduate College

PREFACE

This thesis is a design of an operating system kernel for a microcomputer system. The design is used mainly for submitting and retrieving batch jobs executed by the host computer.

The author would like to thank Dr. J. Richard Phillips, my major advisor, for his guidance and assistance to make this project an enjoyable and memorable experience. The author would also like to thank Dr. Donald D. Fisher and Dr. Donald W. Grace, the other members of the committee, and Dr. M. Folk for their suggestions.

A special appreciation is offered to my brother, Agustinus, for his assistance in typing my thesis during his visit to Stillwater. I would also like to thank my parents and grandmother for their encouragement, love and confidence which made this thesis possible.

TABLE OF CONTENTS

TABLE

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Objective

Computer technology has advanced rapidly since the first computers were built in the late 1940s and early 1950s. These computers were built primarily to solve scientific and engineering problems. The latest technological developments are in the area of large scale integrated circuits. This has resulted in the development of the microprocessor, a device which can function as a central processing unit (CPU).

Previously, the emphasis of computer development was on larger and more powerful machines, but these were expensive. Microprocessors are now readily available in large numbers and at low cost. In comparison with large computers, a microprocessor has limited capability with respect to scientific and engineering problems; however, it is very useful for controlling information flow from one or more sources.

A typical application for a microprocessor is the control of tasks for a larger computer. For example, a microprocessor can be used to control various types of input/output. It is also possible for system designers to

use multiple microprocessors for certain types of dedicated processes that are now handled by a general purpose computer. The extent of such applications is just beginning.

Conceptually, a multiple microprocessor system could be constructed. With such a system, it is easy to see that one of microprocessors could fail and still allow the other processors to operate. This kind of system is still in the development phase because of the unsolved problem of interprocessor control. A limited multiple microprocessor system, however, can be considered and some of the software problems can be addressed.

The main objective of this thesis is the design of an operating system kernel for a microcomputer system to allow multiple users to create programs, store them on a disk file and execute them on a host computer. In other words, the host computer views the input as a remote job entry station. A user may keep his program on a permanent disk file and retrieve it at any time.

The idea of using a microcomputer to "off-load" a host computer is not new; this design, however, is a first step in understanding the more general problem associated with designing an operating system for a system containing more than one microcomputer in a multiuser environment.

Overview

Chapter II introduces an overview of the kernel by describing the major functions of the kernel. This chapter

also discusses how a user communicates with the system, the functions that are available to a user and the limitation of the kernel.

Chapter III presents the structure of the kernel. It discusses the operating system components at each level and how these components communicate with each other. A table of the basic modules of the kernel and their function is given in this chapter.

Chapter IV presents a detailed description of all system modules and their data bases. The description is categorized into several sections: they are the input/output system, the command interpreter, file management and memory management. The input/output system consists of several modules such as Logon, Logoff, the console scheduler, IOS, IOCS and the communication module. File management consists of the file manager and the disk manager. The design of file management or memory management is not considered in this thesis; only the design of the interfaces are presented. The data bases are described in terms of PL/I-style structure declarations in which each field is described in detail. This chapter also contains a few diagrams illustrating how the system modules communicate with one another. At the end of the chapter a diagram showing the overall system structure is presented.

In order to give a better overview of the design of the kernel, a discussion of how the system works for a particu-

lar command is presented in chapter V. This involves a discussion of the operations of the command interpreter, the file manager, the disk manager, IOS and IOCS.

The appendices include PDL descriptions of the input interrupt routine and the output interrupt routine. There is also a discussion of the characteristics of the I/O interface controller.

CHAPTER II

OVERVIEW OF KERNEL

Overview

The kernel is a multiprogramming operating system designed for a microcomputer system. It supports multiple users in which each user has a terminal. Any type of I/O devices can be supported, but the current system is configured as illustrated in Figure 1.

The present system design only supports background jobs executed in a foreground manner. As shown in Figure 1, each user communicates with the microcomputer system. Prior to submitting a job, a user creates a file which is stored on the disk. The user may then submit the job for execution on the host by using the capabilities of a command interpreter.

Communication with the host is accomplished by using a special protocol. This involves the formation of messages which are composed of a header, text and trailer, each of which is several characters in length. For an IBM system, one protocol that is often used in communication with the host is the byscyn (BSC) protocol. The other protocols are asynchronous and synchronous data link control (SDLC) protocols. A detailed description about communications' protocol may be found in Schoeffler (6).

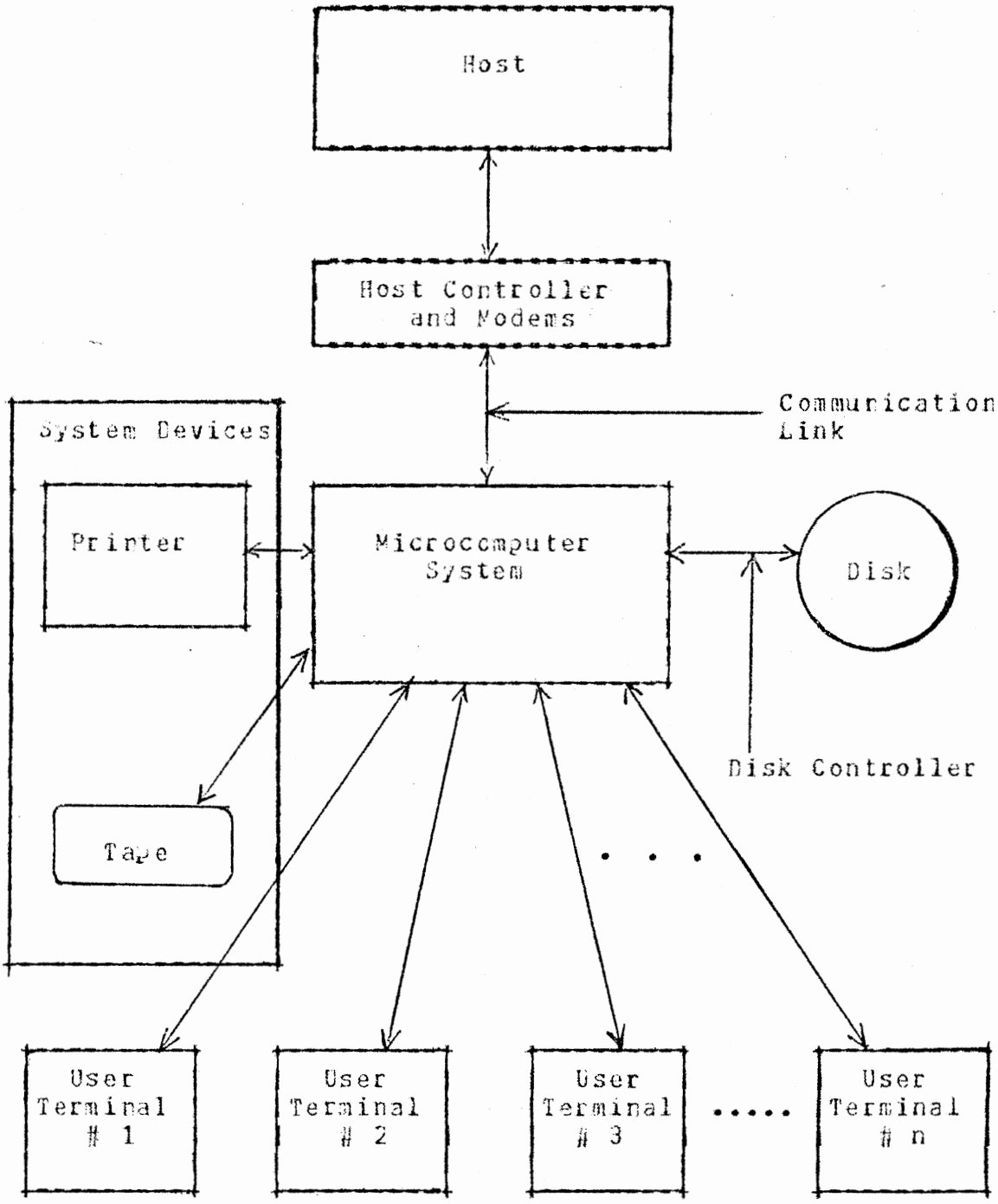

Figure 1. Overview of the System

From the host point of view, the kernel is just another card reader and printer. When a user submits a batch job, the kernel passes the information to the communication module (a module which is responsible for setting up the protocol) and the latter sets up the protocol in such a way that the host recognizes the input as a remote job entry station. In a similar way, the output from a batch job may be retrieved. For instance, the kernel may request a signal from the communication module to allow the host to send the output to the microcomputer. The host interprets this request as a signal to output some information to a line printer.

A user has an option of either storing the output of a batch job on a disk file or printing it at the local printer. If the user chooses to store the output on disk, the output file is stored as a permanent file. An authorized user may request that the file be deleted at a later time.

The system also provides a mechanism for retrieving the status of all batch jobs currently in the system. This allows a user to know whether the job has been submitted or stored on a disk output file.

The kernel maintains the status of all batch jobs currently in the system. The status bit is zero if the job has been submitted to the host, and one if the job is completed. Upon system initialization the status of all previcusly sub-

mitted batch jobs are purged. This initialization may be done on a daily basis.

There is only one communication line connecting the microcomputer system to the host. The kernel schedules a job for submission to the host as long as the corresponding user is in a running state and the line to the host is not busy. Otherwise this user is placed in a blocked state and must wait to use processor time.

There are several different states that a user may be in. A user is in a running state if he is using processor time. Upon an I/O request a user is moved from a running state to a blocked state. Figure 2 gives a transition diagram showing how the states of a user change. A user is said to be dispatched when the kernel gives the user the CPU.

The terminology of multiprogramming may be a misleading concept (as defined in most operating system books). Multiprogramming, in a timesharing system, refers to the concurrent execution of two or more user processes in a single computer system. In the context of the kernel, the concurrency occurs in interpreting user commands, not in executing several processes.

To support multiprogramming the kernel handles the sharing of system resources among executing users. Some resources, such as the line printer, are exclusively allocated to a user until the task is completed. Other resources, such as the CPU and memory, are shared dynami-

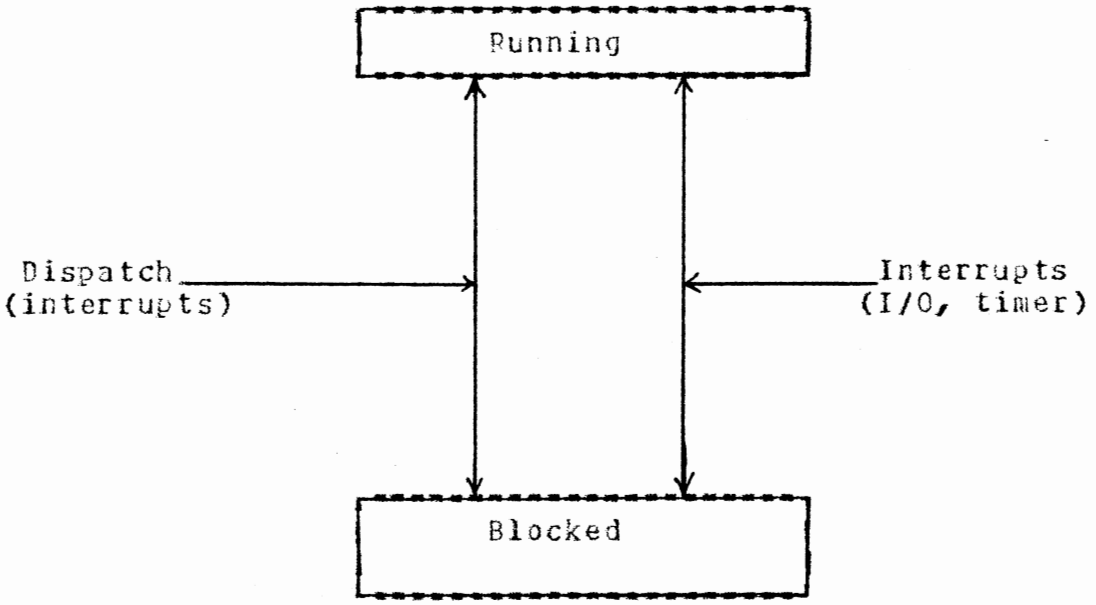

Figure 2. Transition Diagram of User State

cally. The kernel supports time slicing (sharing CPU time among several users by alternately giving each a short interval of time) by using a timer.

Each terminal may have a different rate of information transmission. The kernel provides an adjustable system parameter that allows terminals to be set at different rates depending upon terminal characteristics. This provides a mechanism for adding new terminals to the system. There are also operator commands which are able to display or change the current baud rate of any terminal. These commands are useful for diagnosing communication problems.

Memory allocation for the user is partially handled by hardware using bank allocation. The memory is partitioned

into several memory banks. The system provides a mechanism that allows the kernel to select one or more banks. One of the banks is used by the system and is referred to as the system bank; the others are referred to as user banks. The kernel controls memory bank selection and assigns a free bank to every user. The maximum size of a bank is about 64 k bytes.[1]

Summary

The major functions of the kernel are:

1. to perform input/output operations,
2. to interpret commands from users,
3. to assign storage for users and system functions,
4. to schedule batch jobs, and
5. to retrieve batch jobs.

Functions Available to User

The main objective of the present design of the kernel is to allow a user to submit batch jobs to the host computer. To support this objective the system provides multiple terminals for users. A user can create a file containing the program from the terminal using a text editor and store it on a disk. The text editor allows users to manipulate files in the usual way. For example: save a file,

---

[1]1K is equivalent to 1024.

recall a file, delete a file, change one character or few characters in a text (from a file), and delete lines from a file.

The system also provides user commands which serve as a convenient tool for users to request services of the system. Briefly, the basic user commands are:

LOGON: The user establishes contact with the system. With this command the user gives a valid user identification and password.

LOGOFF: The user breaks contact with the system. The system then displays the accounting information.

SUBMIT <input file name>: The user requests the system to submit a batch job to the host. The input file name is the name of the file containing the program to be executed. The system gives the user a unique job number for each job submitted to the system. This job number is actually assigned by the host. The user could examine the status of a job using STATUS command (discuss below).

STATUS <userid|job#>:[2] The system displays the status of a job for userid|job# specified.

OUTPUT <userid|job#> <destination>: The user requests the output of a batch job identified with userid|job# to be printed at the designated destination.

---

[2] '|' signifies concatenation.

Communication with the System

The user communicates with the kernel primarily by means of a terminal. When the user sits down at the terminal and turns it on, the system issues a logon message requesting the user identification. This identification is a userid and a password. The kernel checks this identification in a table of authorized users and if a match is found, it tells the user to proceed. The system module which allows the user to be able to make contact with the system is named Logon.

Input and output from a terminal is handled on a character-by-character basis. When the user strikes a key, the corresponding character is transmitted to the I/O interface module in the microcomputer system and an interrupt is generated. This character must be removed from the I/O data transmission buffer before the user strikes another key, or else the first character is overwritten by the next character.

A terminal is a combination of two devices in one unit, a keyboard and a printer. When a key is struck the terminal acts as input device and places the character in a data transmission buffer. When the terminal receives a character from the system, it acts as output device, and prints the character on paper or displays it on a video screen. The kernel is responsible for controlling the transmission of character to and from the terminals. The system module that

controls the transmission is called IOS (Input Output System).

Associated with every active user is a workspace and several I/O buffers. For input processing there are two buffers; for output processing a single buffer. Both types of buffers are fixed in size and reside in fixed locations in the system area. Their capacity is one record, where the length of a record is variable. The actual size is determined by the physical characteristies of the device. The system also provides a system parameter that allows an operator to adjust the size of the record accordingly.

IOS is an interrupt driven system. It consists of several submodules. Logon is one of the submodules. The other submodules are Logoff, Input Interrupt System, Output Interrupt System, the communication module and console scheduler. The basic design of input interrupt system is given in Figure 3.

All the designs are written in PDL (7) (Program Design Language), a pseudo language with structured programming technique as a replacement of the flowchart.

The kernel has an input interrupt service routine to handle all terminals as well as input from the host. Figure 3 describes the skeleton of the input interrupt routine. The actual implementation of input interrupt routine is device-dependent.

```
save all registers used;
input a character;
echo input character;
IF end of request is detected THEN
    request IOCS to place the data in user's
    workspace;
ELSE
    place the character received in one of the
    input buffers to form a line (record);
FI;
restore all registers;
enable interrupt;
return;
```

Figure 3. A High-level PDL Description of an Input Interrupt Routine

The interface between a user program and the I/O controller consists of a procedure called IOCS (Input Output Control System). IOCS has the responsibility of copying data from input buffers to a user workspace or from user workspace to an output buffer. For the latter case IOCS only transfers the data one record at a time. When the output buffer is full, IOCS does the proper initialization to allow the device to output characters to the corresponding device. The PDL for the output interrupt routine is shown in figure 4.

Users communicate with the system using user commands via a terminal. These commands are used to communicate with the file system, with memory management, or for gaining access to subsystems such as text editor and other similar system resources. The kernel contains a command interpreter

to interpret these commands. The command interpreter interprets a user command in a user workspace and transfers control to the appropriate routine for further actions.

```
save all registers used;
output a character from output buffer;
IF output buffer is empty THEN
    mark output buffer empty;
FI;
restore all registers;
enable interrupt;
return;
```

Figure 4. A High-level PDL Description of an Output Interrupt Routine

One other important task of the kernel is to maintain all the user files. This is the responsibility of the file management module. It consists of the file manager and disk manager. The disk manager controls the actual movement of data from the device (disk) to a user area (in a system bank). A user bank is divided into two partitions. One partition is used as a workspace for terminal I/O operation, and the other is used as a workspace for text editing. The user area is part of a system bank which is allocated by the memory management module to serve as a transfer station. The length of a user area is determined by the physical characteristics of the disk and is chosen to be a multiple of a record.

With the existence of the file manager a user can perform several functions:

1. The user may create, change or delete files.
2. The user may control the access to the files and the type of access allowed, such as read or write.
3. The user may provide a file back up in case a file is accidentally deleted or damaged.

The process of a user breaking contact with the system is taken care by the logoff routine. Upon termination of a user session the logoff module:

1. updates accounting information,
2. calls memory management to deallocate all storage used during an active session,
3. calls file management to save temporary files, if any.

Limitation of the System

At the present time, the design of the kernel does not include file management, memory management, communication module or text editor. These are self-contained modules specified in terms of functional attributes. The main concern here is the system and interfaces design.

When the command interpreter is given a user command requesting services of the file manager, the command interpreter sends a request to the file manager along with the function specified by the user. The file manager might com-

mand the disk manager to do the appropriate actions, if necessary, but it cannot directly communicate with the disk manager.

The system performs no error checking on the byte being transmitted except in the protocol. Error detection and recovery are on reception of characters. The module that provides the error detection and recovery are different depending upon the device type. There are only three sources of input, either from terminals, the host or cassette tape.

IOS detects and recovers overrun errors from terminals. The I/O channel provides a mechanism to detect these errors. As each character is input to the system, IOS checks for a possible overrun error by interrogating the status register in the I/O interface. If there are any errors then IOS prompts a special character to request the user to retransmit the entire current record. The communication module detects and recovers errors for input from the host.

The size of a user workspace might not be large enough to contain a huge user program. At the present time there are no facilities provided by memory management to enlarge the size of the workspace allocated. A solution to this problem is to allow the editor to copy the user program to the disk when the workspace is full.

CHAPTER III

STRUCTURE OF KERNEL

Levels of the Kernel

The structure of the kernel is designed using hierarchical concepts. The construction of the kernel is such that a given level is allowed to call upon services of lower levels, but not on those of higher levels. In other words, each successive level, from the bottom up, depends only on the existence of those levels below it and not on those above it.

The kernel consists of 3 levels as follows:

1. level 0 - Input/Output System
2. level 1 - File Management
3. level 2 - User Domain

These levels are diagrammed in Figure 5.

The hierarchical approach has the advantage of distributing the system functions. Each level does not really have to know whether other levels are in the same processor. The most important consideration is the design of the interfaces between each level. These are described thoroughly in the next chapter.

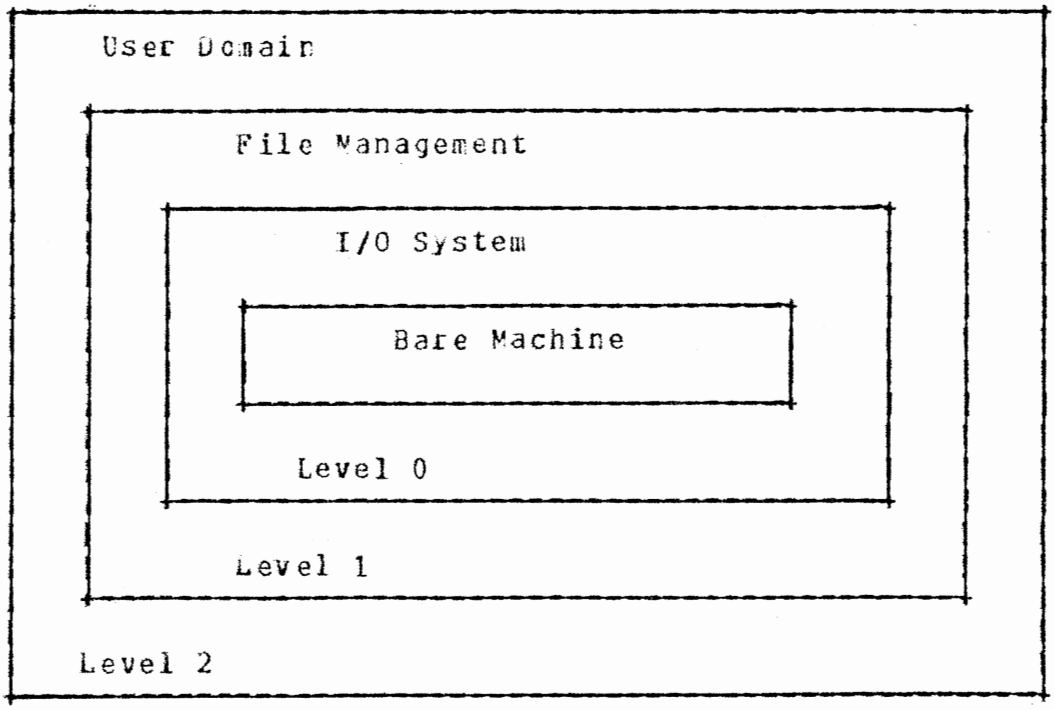

Figure 5. Hierarchical Kernel Structure

From Figure 5 memory management does not appear in any level; this is because all levels need the memory management services to allocate storage within their own processor. In order to distribute the system functions over several processors, memory management must exist on all levels and on all processors. Chapter IV will discuss the functions of memory management in more detail.

Description of Level Dependencies

In the following section each level of the kernel and its dependencies on other levels below it are described. The data structures for interfacing among levels are described in more detail in the next chapter. Table I lists each module of the kernel and the general function associated with it.

Input/Output System

This system consists of several submodules. They are logon, logoff, IOS, IOCS, the console scheduler and the communication module.

When a user is making contact with the system, the logon takes control of the system. It searches a table of authorized users and if a match is found then the user may proceed; otherwise the system gives several trials to the user to resubmit the identification. Logon uses memory management to allocate a free user bank. Before the logon module transfers its control to IOS, the console scheduler places the user identification intc a blocked user queue. This allows the command interpreter to service users by dispatching the first user in the queue. This queue, along with others, is described in Chapter IV.

As each character is input, IOS assembles the characters into a record in one of the input buffers. When the end of record is detected (for example for a terminal it is a

TABLE I

KERNEL SYSTEM MODULES

Module	Function
Memory Management	Allocate and deallocate memory space for other modules and system bank
Logon	Authenticate user and logically connect terminal to the system
Logoff	Disconnect terminal from the system and update user accounting information
IOS	Control channel input/output operations
IOCS	Control data flow from I/O sources
Communication module	Establish protocol for communication link with IBM 370
Console Scheduler	Schedule new users and allocate processor time
File Manager	Maintain all user and system files
Disk Manager	Control disk input/output operations
Command Interpreter	Interpret user commands
Editor	Text or program manipulator

carriage return), IOS puts a request in a request queue (REQQUE) for servicing by IOCS. This allows IOCS to copy the record to the appropriate workspace. The request queue is a circularly linked queue. Allocating and deallocating nodes for REQQUE is the responsibility of memory management.

In servicing a request from the host, IOS calls the communication module to perform the task. The communication module is responsible for setting up the appropriate protocol in a form recognizable by the host before copying the data to the indicated output buffer.

The two modules, IOS and IOCS, communicate by means of a control block which contains information such as the address of the input or output buffer, address of the user area, the status of buffers and other pertinent information. All devices in the system have static control blocks, except for dedicated devices such as the line printer or a tape drive. In these cases, dynamic control blocks are allocated and deallocated by memory management. Whether static or dynamic, all control blocks are of fixed size.

File Management

This module consists of two submodules. They are the file manager and the disk manager. The interface between file management and I/O system is similar to the interface between IOCS and IOS. Upon an implicit request[1] from a user, the file manager initializes another kind of control block,

similar to the I/O control block, referred to as the disk control block (DKCB). The file manager commands the disk manager to control the data movement by placing a request in a disk request queue (DRQUE) to allow the disk manager to move the data to and from the disk. Both DRQUE and FRQUE are implemented as circular linked queues and requests are serviced by the disk manager and file manager, respectively.

The file manager calls upon memory management for several services. For example, the disk control blocks are allocated for each user request and deallocated at the end of the task. Memory management also provides an availability list for allocating and deallocating nodes of DRQUE and FRQUE.

The disk manager examines DRQUE to determine if it is empty or not. If DRQUE is empty then the disk manager is idle because no service is requested from the file manager; otherwise, the disk manager performs the appropriate tasks defined in DRQUE. The structure of DRQUE and FRQUE are discussed in more detail in the next chapter.

Similarly, the file manager is idle if FRQUE is empty. There are three possible types of requests in FRQUE originating from the command interpreter, ICCS or text editor.

---

[1]A user requests the kernel, through the command interpreter, to manipulate files. The command interpreter services the request by placing an entry in a file request queue (FRQUE). The file manager examines FRQUE and does the proper actions.

IOCS requests the file manager to copy data from the user area. This occurs when a batch job is being retrieved. The command interpreter requests the file manager to manipulate files, such as print a file stored on a disk at the local line printer. This causes the disk manager to place a request in REQQUE to allow IOCS to copy the data from the user area to the output buffer. The text editor also requests the file manager to copy the content of a file to a user workspace for editing.

User Domain

At this level the command interpreter schedules users for command execution. A user is said to be eligible for execution if he is not waiting for completion of an input/output operation or for a new time slice or for the completion of an I/O operation from other users. If one of these conditions occurs the command interpreter blocks the user and places the user in a blocked queue. This queue consists of a save area containing the status of the processor for a particular user. The save area includes information such as the contents of processor's registers and the program counter containing a pointer to the next user command to be interpreted by the command interpreter for resuming the control.

As mentioned in the previous section (file management), the user domain (the command interpreter and text editor)

and the file management (the file manager) communicate by means of a file request queue (FRQUE). Upon user request, the text editor and command interpreter place their requests in FRQUE.

The text editor is used by users to enter, delete, and modify text and write it on the file. The file name is usually requested by the editor and the user can change the name if desired. In creating a new file the text editor does not have to place a request in FRQUE. In order to allow users to manipulate old files the text editor requests the file manager, which in turn requests the disk manager, to copy the contents of the file from the disk to the indicated user workspace.

CHAPTER IV

KERNEL DATA BASES AND ROUTINES

This chapter presents a more detailed description of the modules in the kernel and their data bases. The data bases are described using PL/I-style structure declarations. At the end of this chapter a diagram showing the overall system structure of the kernel is given.

Input/Output System

At the present time the Input/Output system only supports three types of communication devices: terminals, a line printer (local) and a communication link to the IBM 370 called the host. The disk is not considered as part of the I/O system since it has its own disk controller. It does, however interface with the Input/Output system.

Interrupt System

The word interrupt has been mentioned several times in the preceding chapters. There are various ways to implement and define interrupt mechanisms. Madnick and Donovan (4) define an interrupt as:

1. A response to an asynchronous or exceptional event that

2. Automatically saves the current CPU status to allow later restart, and

3. Causes an automatic transfer to a specified routine called an interrupt handler.

Only the third definition of an interrupt is explored in depth in this section.

One interrupt method that Z80 microcomputer uses is called vectoring. This means that each interrupt source provides data (an address) that the CPU can use to identify the source of the interrupt. It uses fixed memory locations for storing an array vector of addresses and is therefore called an interrupt address vector.

In mode 0 the interrupt address is at location 00xxx000 where xxx is a 3-bit binary number that is part of the instruction which has the binary form 11xxx111. Since the Z80 is an 8-bit microprocessor using 16-bit addresses, the contents of the new program counter are set to 0000000000xxx000. This is equivalent to the standard RST instruction used by the 8080A.

In mode 1 no interrupt vector is needed. The Z80 interrupt response logic automatically assumes that the first instruction executed following the interrupt response will be a restart which branches to memory location 0056 (in hexadecimal).

In mode 2 it is possible to set up an array of 16-bit interrupt addresses anywhere in addressable memory. These 16-bit addresses identify the first executable instruction

of interrupt service routines. The formation of the interrupt service address is as follows: the Z80 CPU combines the I (interrupt vector) register contents with the interrupt response vectors from the external logic (the I/O interface logic). These 16-bit addresses are used to access the address in the interrupt address vector table. Since 16-bit addresses must lie at even memory boundaries, only seven of the eight bits provided by the external logic are used to create the table address; the low order bit is set to zero (5). Figure 6 illustrates how an address is formed and how it points to an address in the interrupt address vector.

Each microcomputer system has its own I/O interface conventions. The following discussion is based upon the Cromemco system (1).

There are 16 possible interrupt sources on each I/O board. Among these, ten of those are timer interrupts with different hardware priorities. Loading the interval timer with a value of zero causes an intermediate interrupt. The 8-bit interval timer provides intervals that vary in duration from 64 to 16,320 microseconds. Longer intervals can be implemented in software.

In order to be able to use all possible interrupt sources, the kernel must use interrupt mode 2 which allows the I/O controller to generate a unique response without the need for chaining the interrupt request and polling the response.

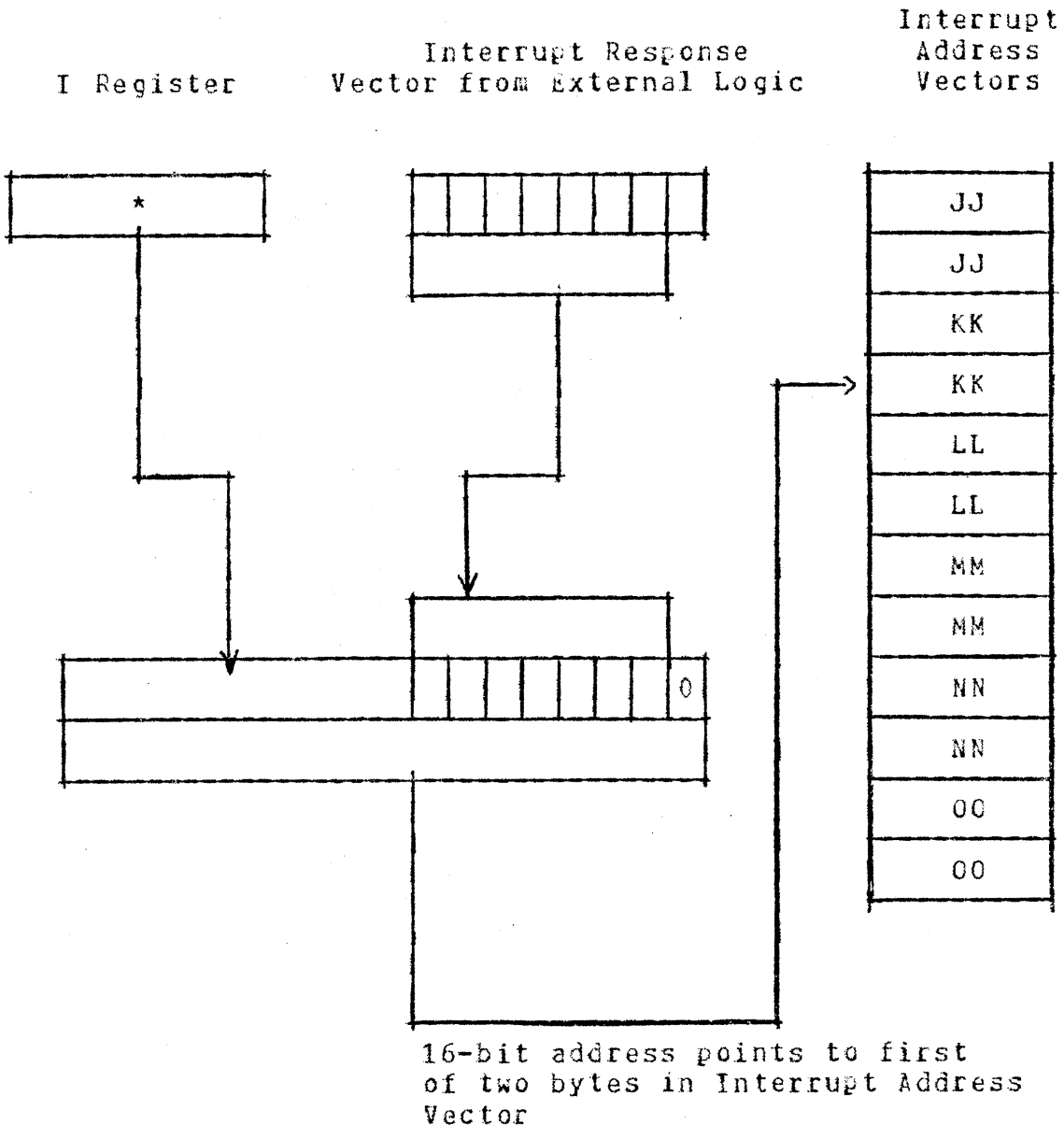

Figure 6. Interrupt Service Address Formation

The I/O interface controller prioritizes interrupts in the order shown below:

1st - Interval Timer 1

2nd - Interval Timer 2

3th - External Sensor

4th - Interval Timer 3

5th - Receiver Buffer Loaded

6th - Transmitter Buffer Emptied

7th - Interval Timer 4

8th - Interval timer 5 or an External Input

The fifth interrupt is an input interrupt and the sixth is an output interrupt.

All modules of Input/Output system (IOS, IOCS, Logon, Logoff, Console Scheduler) use the previously described hardware capabilities. In order to apply the present design of the Input/Output system to a microcomputer system with different I/O interface controller the system must have at least an interrupt system and timer interrupt. Most modules within the kernel's I/O system, such as IOS, Logon, Console Scheduler are interrupt driven routines. Appendix A discusses in more detail the characteristics of the I/O interface controller currently used.

LOGON

The data structure for the logon and logoff is in form of a table named LONTAB (Logon Table). The size of this

table depends on the number of terminals existing in the system. The structure is defined by:

```
DECLARE
1 LCNTAB(*),
  2 STATUS BIT(2),
  2 UID CHAR(8),
  2 PSWD CHAR(4);
```

where the fields are defined as follows:

STATUS: This field contains the status of the device (terminal). It is zero if the terminal is inactive and one if an authorized user is allowed to use the system. It has the value of two if the user is in a blocked user queue waiting to use the system facilities. The value of 3 is undefined.

UID: This field consists of an 8-character string for a user identification. This identification is compared with the entries in a table of authorized users.

PSWD: This field contains the password associated with UID. In order to be eligible to use the system both UID and PSWD should be correctly specified by the user.

The user begins a session by turning the device on. In order to recognize an interrupt, even if no device is physically connected at the beginning, requires that the I/C interface controller be initialized to accept characters from the device (terminal). To initiate logon a user must

hit the carriage return three times. This allows the logon module to do the following:

1. If STATUS is zero (logoff) then issue a logon message such as 'LOGON - ENTER USERID AND PASSWORD', else prompt a rejected logon message. The latter can only happen when the terminal is disconnected improperly following another logon (hitting carriage return).

2. Once a correct identification is given, USERID and PSWD are stored in LONTAB and STATUS is set to indicate that this device is now in the logon state. The logon module gives several trials if wrong identification is typed in. If the user fails to identify himself then a logoff message is prompted. A user may try again by repeating the process beginning at 1.

3. If logon is successful then

   a. Call memory management to allocate a free user bank and update the table (BANKTAB) containing this information.

   b. Load the interval timer of lower priority than the device (timer 4) with a count of zero to cause an immediate interrupt (1). This allows the console scheduler to search LONTAB to find an active user (STATUS is one) and place this newly active user in a blocked user queue. It then sets STATUS to "in progress."

   c. Put a different address in the Interrupt Address Vector. This address will be the address of the actual input interrupt routine. Any characters typed, at this point of time, cause a jump to the IOS module.

Step 3b allows the system to trigger a lower priority interrupt level and exit from the higher routine. The lower-

priority routine then performs the functions indicated but is interruptable by higher-priority signals.

Console Scheduler

The console scheduler is called by the timer interrupt routine set up by the logon module. It searches LONTAB to find an active user with STATUS equal to one and places this user identification in a blocked user queue. This queue contains all active users which are waiting for the command interpreter to schedule them. Finally, the console scheduler sets STATUS to the value of two to indicate that the user is now in a blocked user queue ready to use system facilities.

Logoff/Disconnect

When the logoff module is called by the command interpreter it does the following:

1. it updates the accounting information,
2. it calls memory management to deallocate the user bank,
3. it prompts the user with a logoff message if the status of the device is still in progress,
4. it puts a special address in the Interrupt Address Vector which points to the logon module.

When a disconnect occurs (for example the terminal is momentarily turned off) or the terminal is not being used for a period of time, the logoff occurs automatically. This

is done by a submodule of the input interrupt routine for terminals. It is implemented with another timer interrupt routine which checks the time the last character was input.[1] Once it passes a certain interval, this timer routine sets STATUS to zero (logoff) and places a logoff command in a user workspace. The reason for this is to let the command interpreter finish interpreting the current line. Once the command interpreter interprets a logoff command it calls the logoff procedure.

Input and Output Interrupt System (IOS)

IOS and IOCS (Input Output Control System) communicate by means of an Input Output Control Block (IOCB) and a request queue (REQQUE). Each device in the system has its own IOCB except for dedicated devices such as the line printer and cassette tape. IOCB for devices other than dedicated devices have a fixed structure and location. The kernel reserves some memory locations in the system bank for all fixed IOCB. For dedicated devices their IOCB have a fixed structure but variable in location. It is allocated by memory management when it is needed and deallocated when the task is completed. Therefore, it is necessary to store the address of IOCB in REQQUE to allow IOCS to examine the cor-

---

[1]Without a timer interrupt there is no way for the kernel to detect whether a terminal is still active or inactive.

responding IOCB. Figure 7 describes the communication between IOS and IOCS.

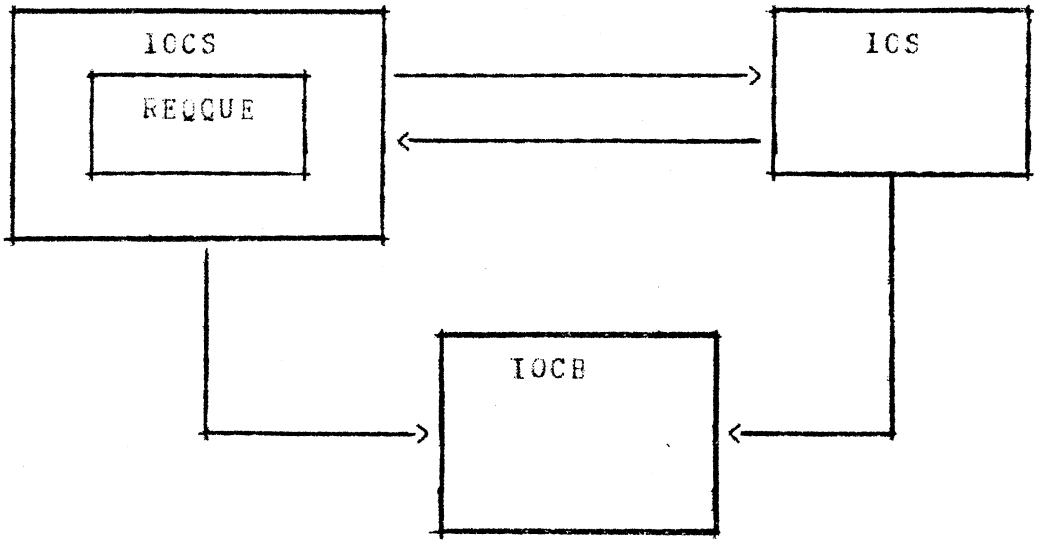

Figure 7. Communication between IOS and IOCS

IOCB is defined by:

```
DECLARE
1 IOCB,
  2 STATUS BIT(1),
  2 DEVTP FIXED BINARY(15,0),
  2 DEVNUM FIXED BINARY(15,0),
  2 ADRUR POINTER,
  2 ADRIB POINTER,
  2 ADROB POINTER,
  2 TOTAL FIXED BINARY(15,0),
  2 ADRDKCB POINTER,
  2 EOD BIT(1);
```

where the fields are defined as follows:

STATUS: This bit is zero if the request is inactive; otherwise it is active. This field is used by the Input Output Control System (IOCS) to determine whether the output buffer is empty of not. The request is said to be inactive if the output buffer is empty and active otherwise. In the latter case, IOS is in process of transmitting characters to a device. This field is set and reset by IOS and is not used for input since IOCS copies the data from an input buffer to the indicated area only if the input buffer is full. Hence, IOCS does not have to delay servicing an input request.

DEVTP: This field contains a binary number which defines the type of device that is connected to the system.

DEVNUM: This field contains the device number for devices in the same category.

ADRUR: This pointer is the address of a user area or a user work area. It contains the address of a user workspace if the input is from a terminal.

ADRIB: This pointer is the address of the input buffer which in progress. Note that there are two buffers for input and only one is being used for the process of accepting characters.

ADROB: This pointer is the address of the output buffer.

TOTAL: This field contains the number of characters in a user area or a user workspace, depending on the value of ADRUR.

ADRDKCB: This pointer is the address of a Disk Control Block (DKCB). This field is used only for transmitting data from or to a file. By following this pointer IOCS informs the disk manager whether the user area is empty or not. In other cases this pointer might contain a null pointer.

EOD: This bit is one if the end of data occurs, zero otherwise. This field is only used for transmitting data from a file and for the host input/output.

For output, EOD is used to indicate whether a file to be printed or to be sent to the host is empty or not. The disk manager sets or resets this field accordingly.

For input, EOD is used to indicate whether more data is coming from the host or not. The communication module sets or resets this field accordingly.

In the case of a dynamic IOCB, IOCS needs the location of IOCB in order to acknowledge IOCS concerning the status of the output buffer.

Memory management is responsible for reserving a fixed location for the address of dynamic IOCB's once they are allocated. The symbolic name of the memory location is:

PRINFO POINTER - this location contains the address of the printer's IOCB

CTINFO POINTER - this location contains the address of the cassette tape's IOCB

There are similar "fixed locations" reserved for storing the address of any dynamic IOCB. This allows IOS to follow the pointer and be able to set and reset the STATUS field in IOCB depending on the condition of the output buffer.

The request queue (REQQUE) is a circular linked queue that is used to request IOCS to transfer data from a user area (or a user workspace) to an output buffer, or from a full (active) input buffer to a user area (or a user workspace). The respective modules, that need this service, insert their request at the end of the queue. IOCS services the requests, one at a time, from the front of the queue. For this purpose only one pointer, Front, is needed to manipulate the REQQUE. Figure 8 describes a typical REQQUE containing three requests, the first request is an input request, and the others are output requests. The fields of REQQUE are described in the next paragraph.

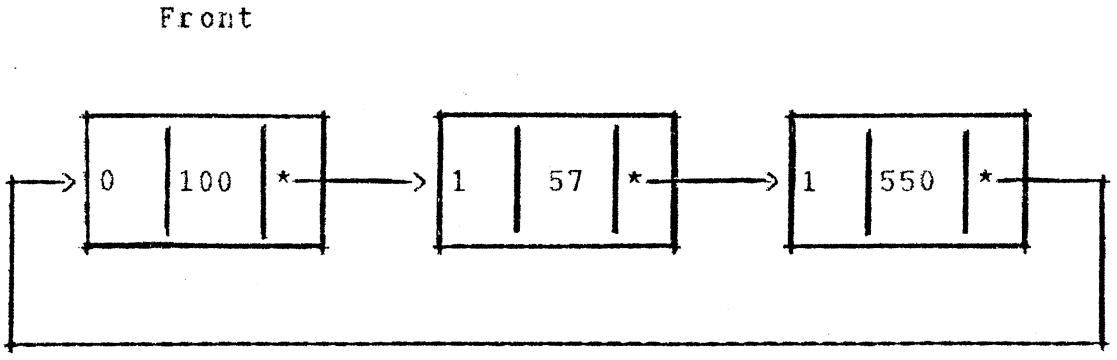

Figure 8. Example of REQQUE

The queue is empty if Front is a null pointer. For the case when IOCS needs to service the next request because the status of the present request is active, IOCS moves the Front pointer to the next request by following the link field. This has the same effect as placing the current request at the end of the queue (because of the characteristic of a circular linked queue).

Memory management provides an availability list for allocating and deallocating nodes for REQQUE. The stucture of a node for REQQUE is defined by:

```
DECLARE
1 REQQUE,
  2 DIR BIT(1),
  2 ADRIOCB POINTER,
  2 NEXT POINTER;
```

where the fields are defined as follows:

DIR: This field contains the direction of the movement of the data. As mentioned above this bit is zero if IOCS moves the data from a full input buffer to a user area (or a user workspace); otherwise it is one.

ADRIOCB: This pointer is the address of IOCB. It is used to point to a dynamic IOCB that has been allocated by memory management for dedicated devices.

NEXT: This is a pointer to the next request, if any.

As mentioned in previous chapter, there were two input buffers and one output buffer. The structure of an Input Buffer (INBUFF) is as follows:

```
DECLARE
1 INBUFF,
  2 BUFUSE BIT(2),
  2 LENGTH FIXED BINARY(15,0),
  2 BUFPTR FIXED BINARY(15,0),
  2 BUFSP CHARACTER(255);
```

where the fields are defined as follows:

BUFUSE: This is an input buffer indicator. It is zero if the input buffer is empty, one if full and two if ICCS is using this buffer. ICS is responsible for setting the field to one (buffer full) while the other two conditions (empty,busy) are set by ICCS.

LENGTH: This field contains the length of the record. Its value depends on the physical characteristics of the device.

BUFPTR: This field is used to point to the next location of BUFSP.

BUFSP: This field is reserved for buffer space. The maximum size is 255 charaters.

The structure of an Output Buffer (OUTBUF) is similar to INBUFF except the BUFUSE field does not exist. OUTBUF is defined by:

```
DECLARE
1 OUTBUF,
  2 LENGTH FIXED BINARY(15,0),
  2 BUFPTR FIXED BINARY(15,0),
  2 BUFSP CHARACTER(255);
```

where the fields are defined as in INBUF.

The kernel provides one input and one output interrupt routine for each I/O device in the system. The description of the input interrupt module is divided into two sections, one is an input from terminals and the other is from the the host.

Input Interrupt Routine from Terminals

As each character is typed in by a user IOS does the following:

1. Place the character in the input buffer.

2. Load the interval timer with the lowest priority (timer 5) with a count of zero to cause an immediate interrupt. The function of this timer is to check on the time when the last character was input. This checking is needed when a disconnect occurs or when the terminal is not being used for a period of time. The choice of the timer priority is based on the urgency of the task. The author thinks that this task is of low priority and can be interrupted by any other modules.

3. Interrogate the status register of the device to check for possible overrun error. An overrun error is an error where the receiver buffer of the I/O controller has been loaded with a new byte before the previous contents have been read from the buffer. If there are any errors then IOS prompts with a special character requesting the user to retransmit the current line (record). Meanwhile, IOS resets BUFPTR and LENGTH to the initial condition.

4. If the end of a record is detected (detection of a carriage return) then

   a. Call memory management to allocate a node for REQQUE. IOS places this request at the end of REQQUE after initializing the DIR and ADRIOCB fields.

b. Initialize the IOCB fields: DEVTP, DEVNUM, ADRUR, ADRIB and TOTAL.

c. Set BUFUSE to one to indicate that the input buffer is full.

d. If the other input buffer is also full then load the interval timer of the highest priority (timer 1) with a count of one. This timer interrupt allows the Input Output Control System (IOCS) to serve REQQUE with an "input request" (the DIR field in REQQUE is zero). The choice of a highest priority for the timers is a guarantee that IOCS finished copying the data from the input buffer before the next character being placed in the input buffer. Since the priority of the input interrupt is lower than the timer 1, the interval timer counter is set to one. This gives enough time for the input interrupt module to finish its task and return from the interrupt before the timer 1 generates its interrupt.

Figure 9 shows how the data flows from the terminal to the user workspace.

The input interrupt routine also provides a mechanism to detect a break key hit by a user while the output interrupt routine is printing. If a user hits this break key the input interrupt routine sets the LENGTH field in OUTBUF to zero to indicate that the output buffer is empty, sets the TOTAL field in IOCB to zero to indicate that there are no more characters in the user area or user workspace, and sets the STATUS field in IOCB to zero to indicate that the request is in active. With all of these conditions, IOCS

will delete the request by removing this request node from REQQUE. This input interrupt routine also sets the I/O controler to disable an output interrupt.

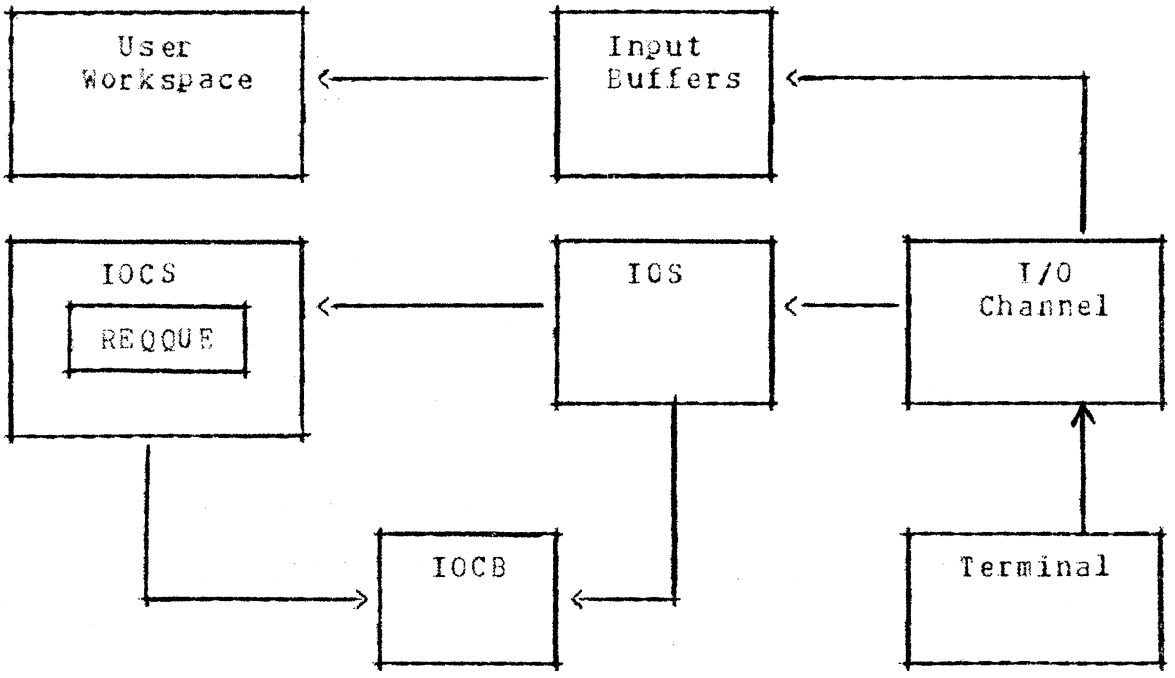

Figure 9. Diagram of Input from a Terminal

As mentioned earlier the kernel provides one input interrupt routine for each I/O devices in the system. This allows the system to recognize different break codes for different terminals.

Input Interrupt Routine from the Host

This input interrupt routine is similar to the interrupt routine for input from a terminal. This interrupt is a result of a user request to retrieve a batch job.

The command interpreter calls memory management to allocate a user area and stores the address of the user area in the IOCB control block that controls the host input. The command interpreter then requests the communication module to send a signal to the host. This signal is recognized by the host as an output to a remote entry station.

When the host inputs a record to the kernel IOS performs the following functions:

1. it calls memory management to allocate a node for REQQUE after initializing the DIR and ADRIOCB fields;
2. it initializes the IOCB fields: DEVTP, DEVNUM, ADRIB and TOTAL;
3. it sets BUFUSE to one;
4. if the other input buffer is full then it generates an immediate interrupt to empty the full buffer by using a high priority interval timer.

Output Interrupt Routine

The main function of this routine is to transmit a sequence of records from a user area (or a user workspace) to a device designated by an IOCB field. IOS does not need to know that the host is the destination. The communication module has the responsibility of formatting the record in such a way that it is recognizable by the host.

Once a character is transmitted, IOS marks the request as "active" (the STATUS field in IOCB is set to one). When the end of record is detected (the output buffer is empty), IOS marks the request as "inactive" and then initializes the I/O controller to disable the output interrupt. For dedicated devices, for example a line printer, IOS needs to follow the pointer specified at the "fixed location", called PRINFO, to get the address of the corresponding IOCB before actually setting and resetting the STATUS field.

The decision to continue transmitting another record is made by IOCS. Once IOS marks the request as inactive and there are still more records to output, it fills the output buffer with the next record. Figure 10 shows how IOS controls an output interrupt.

Input and Output Control System (IOCS)

IOCS serves the requests placed in REQQUE. There are two types of requests, input and output. The type of request is determined by the value in the DIR field. An "output request" asks IOCS to copy the data to an output buffer whereas an "input request" requests IOCS to move the data from an input buffer to the indicated area determined by the value in ADRUR of the corresponding IOCB.

For input, a request can be removed from the queue as soon as the input buffer is empty. While IOCS is transferring a record from the input buffer, the BUFUSE field in the

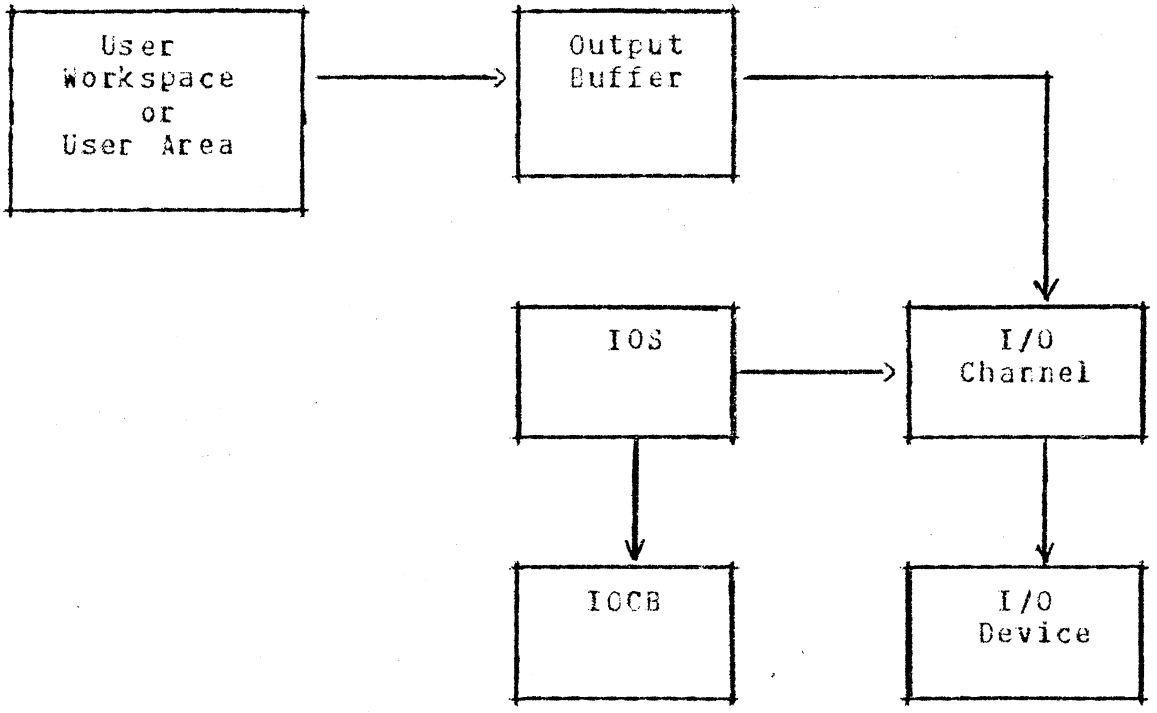

Figure 10. IOS Control of an Output Interrupt

input buffer is set to the value of two (buffer in use). This is used by IOS to determine whether the input buffer is full or being used by IOCS. At the end of the transfer IOCS marks the BUFUSE empty.

If an output request pointed to by Front is active IOCS serves the next request by updating the Front pointer to point to the next request in the queue. For output, a request is deleted from the queue when both of the conditions below are satisfied:

1. TOTAL = 0 (no more records)
2. STATUS = 0 (request is inactive)

A description of the functions of IOCS falls into several categories depending on the type of request and the destination of the output.

Output Request to Terminals

When the request is inactive (the output buffer is empty), IOCS copies a record from a fixed area pointed to by ADRUR to the output buffer and updates TOTAL in IOCB to denote the number of remaining records to output.

The data to be displayed at a terminal has three possible sources: a user workspace, an editor workspace or a user area. The latter is actually a block of data from a file that the disk manager has copied from the disk to the user area.

When the user area is empty IOCS uses ADRDKCB to determine whether there is a null pointer or a valid pointer to the disk control block. In the latter case, ICCS can use the pointer to indirectly inform the disk manager that the user area is empty. If more records are to be output, the disk manager places another request in REQQUE and copies a block of data to the user area. Figure 11 shows how an output request to a terminal is handled by IOCS.

In order to let IOS transmit the characters, IOCS reinitializes the I/O controller which allows the output interrupt to be enabled.

Output Request to Dedicated Devices

IOCS performs a similar task in servicing an output request to a terminal except the destination is different. If the destination is a line printer, the only possible source of data is from an editor workspace or a user area which contains the data from a disk file. The source of data depends on whether the file is "active" or not. A file is said to be active if it is currently in the editor workspace. If the file is active, the disk manager does not have to copy the data to the user area. The procedure on how the kernel outputs a file to a line printer is discussed in Chapter V.

If the destination is a cassete tape the sources of the data are a user workspace, an editor workspace or a user area. IOCS services the request placed in REQQUE by the disk manager in the same way it services the request from IOS.

Output Request to the Host

This operation is used to submit a batch job to the host. Normally, a user has created the program and stored it on a disk file.

The actual output of the records are handled by the communication module and IOS. IOCS still controls the activity of the request queue (REQQUE) as discussed in the previous section. As shown in Figure 12, IOCS does not copy the data to the output buffer, instead the communication module does the copying.

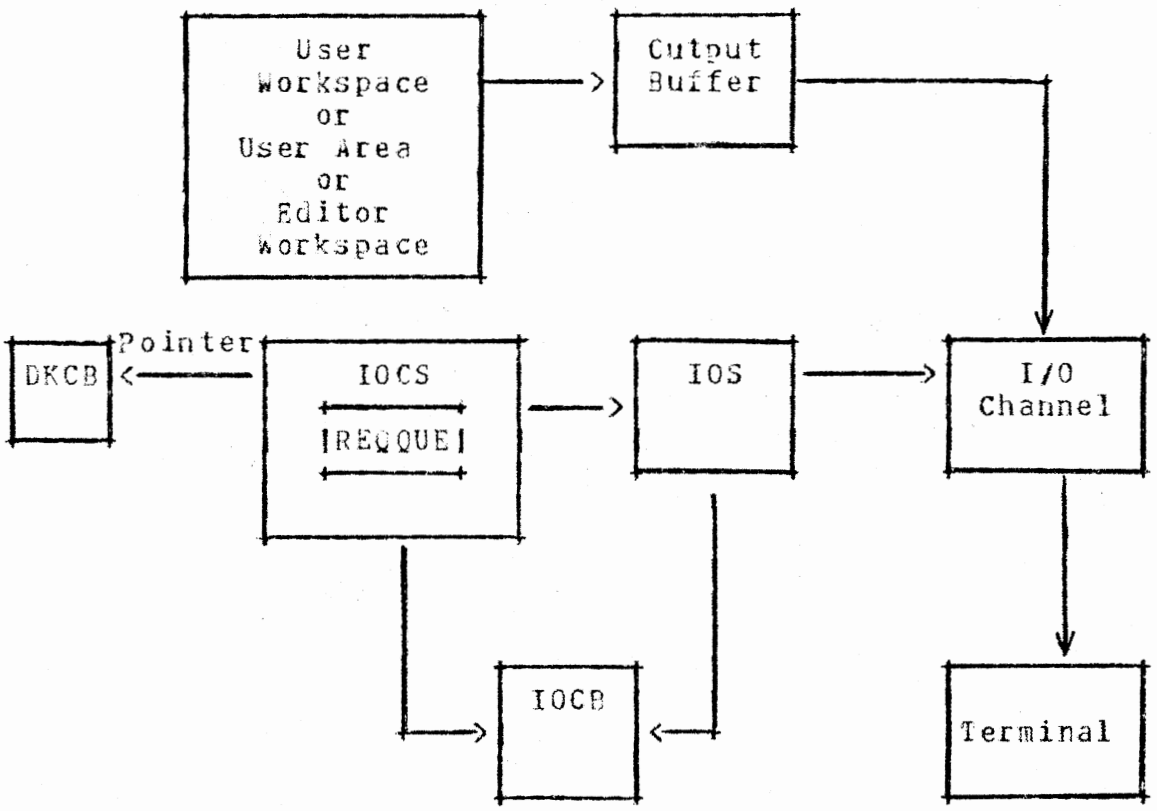

Figure 11. Diagram of Output to a Terminal

The communication module is called by IOCS with the following parameters:

1. CMTP: This is a command type. If it is one, it is a special command and the rest of the parameters are irrelevant. This special command is used to request the status of a batch job. If it is zero the rest of the parameters are examined.

2. OPTP: This is an operation type that is requested by IOCS. Zero indicates an input request; one indicates an output request.

3. Address of user area.

4. Address of input or output buffer (depending upon the value of OPTP).

5. Address of IOCB (for input operation only).

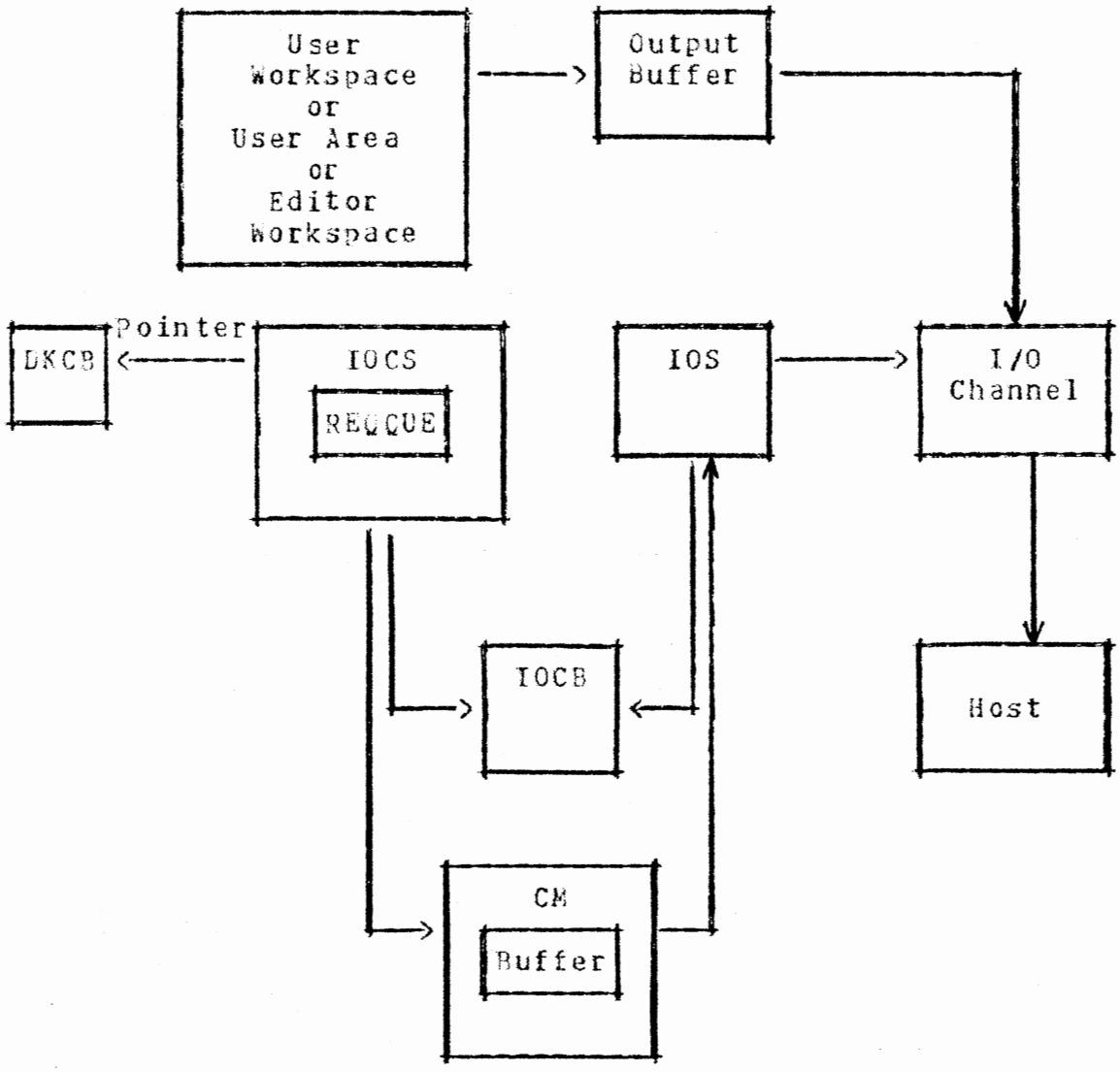

Figure 12. Diagram of Output to the Host

In servicing a request, IOCS calls the communication module with the appropriate parameters. This allows the communication module to set up the protocol before copying the data to the output buffer.

If the user area is empty, IOCS informs the disk manager of this condition when TOTAL in IOCB is zero. The EOD field in IOCB will let IOCS know whether there are more records to be sent or not. IOCS sends a special command to the communication module if EOD is one (the end of data). This allows the communication module to send a special request to the host to obtain the job number. This identification and the user identification together form a unique identification for the coresponding batch job. IOCS sets the I/O controller to allow IOS to transmit the records.

Input Request from the Host

This operation is used to retrieve a batch job. In servicing an input request IOCS calls the communication module with appropriate parameters. IOCS requests the communication module to tranfer the data from the input buffer to the user area. Before this transfer takes place, the communication module reformats the buffer data in such a way that is recognizable by the kernel. The communication module also detects the end of data indicator and sets or resets the EOD field in IOCB to reflect this. Once EOD is

set to one, IOCS informs the disk manager. This enables the disk manager to "close" the file. The size of a user area depends on the physical characteristics of the disk. If the user area is full, IOCS requests an "immediate attention" to the file manager which implicitly requests the disk manager to move the data from the user area. This can be accomplished in a manner similar to that used by IOCS when both input buffer are full. In other words, IOCS generates a high priority interrupt requesting immediate services by using an interval timer. This timer interrupt allows the file manager to request the disk manager to perform the tasks.

Communication Module

The system maintains the status of all batch jobs using a linked list (BJSTAT). The BJSTAT is defined by:

```
DECLARE
1 BJSTAT,
  2 USER# CHARACTER(8),
  2 JOB# CHARACTER(4),
  2 COND BIT(1),
  2 LINK POINTER;
```

where the fields are defined as follows:

USER#: This field is a valid user identification in making a contact with the system.

JOB#: This field contains the job number assigned by the host initiator (Job Entry Subsystem).

COND: This condition bit is zero, if the job has been submitted, one if the job is completed. The latter implies that the host has spooled the output onto a file.

LINK: This pointer points to the next batch job, if any.

The BJSTAT linked list is created by the communication module when the host returns the job# requested. This list is maintained in ascending order by user# and job# and memory management allocates nodes for BJSTAT.

At the time BJSTAT is created the COND bit is zero. At a regular timer interval the communication module takes control of the system to search BJSTAT for all condition bits that are zero. It interrogates the host to determine the status of previously submitted batch jobs. If the host has finished executing the job, the communication module updates the COND field. A user may request the batch job only wher the condition bit is two.

Command Interpreter

The command interpreter is a module that provides services to the users. It interprets user commands in their own workspace and transfers the control to the appropriate system modules for further actions.

We shall assume that all users are serviced in a single processor so that at most one user can use the CPU at a time. The switching of active users occurs when:

1. their time slice has elapsed ,

2. there is a request to do input/output or,

3. an I/O operation is completed.

The kernel has the responsibility of saving the status of running users in a save area when users are switched.

The kernel maintains several types of queues for saving the status of users that are in the same state. Figure 13 shows the only processor queues that the kernel managed (2). Each rectangle in Figure 13 represents a save area containing the status of the processor for a particular user. The save area includes the following information:

1. the contents of the processor's registers;

2. the program counter containing a pointer to the next user command to be executed for resuming the control.

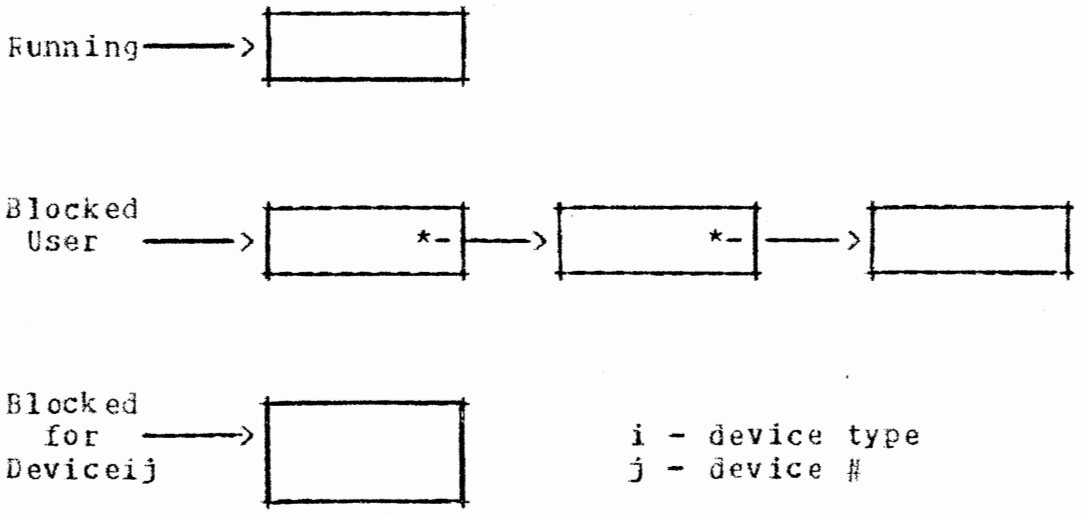

Figure 13. Diagram of Processor Queues

If a user's time slice has elapsed, its privilege of using the CPU is removed and the processor's status is stored in the "blocked user" queue in FIFO (first-in-first-out) order.

As it can be seen in Figure 13 there is only one save area for Blocked for Deviceij. This is because only one request may be serviced by the command interpreter at a time. For example, there may be a request to print a file to a local printer, but the printer is busy. Hence, the command interpreter delays servicing this request by changing the user's state from "running" to "blocked." The kernel maintains a structure describing the activity of the dedicated devices and the host. Each dedicated device has a structure defined by:

```
DECLARE
1 DEVACT,
  2 DEVFREE BIT(1);
```

where the field is used as follows:

DEVFREE: This bit is zero if the device is free, and one if it is busy. For the host, this field is an indication of the status of the communication link between the host and the kernel.

To service an I/O request the command interpreter blocks a running user by placing the status of the corresponding user in a blocked queue for the corresponding I/O device (Blocked for Deviceij) and then dispatches a new

user from "blocked queue for users". At the end of an I/O request the disk manager, which is in charge of the I/O operation, stops the running user from using the CPU and dispatches a user in "blocked for I/O device". In implementing the processor queues, a pointer is needed for each type of queue. The save area for "Blocked for Deviceij" is statically allocated by the kernel for each type of the I/O device in the system. Other types of save areas are allocated and deallocated by memory management when requested by the command interpreter or disk manager. A user is said to be dispatched when the processor's status can be restored by manipulating the Running pointer to point to the appropriate save area. For example, a user changes his status from "blocked for line printer" to "running". This can be done by blocking the running user (insert the save area pointed by the Running pointer to the end of Blocked User queue) and planting the Running pointer to point to the save area pointed by Blocked for Line Printer. This causes Blocked for Line Printer queue to be a null pointer.

In interpreting a user command, the command interpreter may have to initialize something before actually transferring the control to the appropriate system module. As an example, to print a file the command interpreter performs the following operations:

1. Check the device activity (except for terminals); if the device is busy, block the user temporarily;

2. otherwise,

a. Set DEVFREE for the desired device to one to indicate that the device is going to be used.

b. Call memory management to allocate user area.

c. For a request on a dedicated device, call memory management to allocate an IOCB.

d. Initialize the ADRUR field in IOCB with the address of the corresponding user area.

e. Request the file management system to print the data to the indicated I/O device (this will be discussed in more detail in the next section).

f. Once the file management takes over the control, the command interpreter places the processor's status of a user in the running state in "blocked for printer" queue. The command interpreter then dispatches another user from the "blocked user" queue.

A user has several options regarding where to print the output of a batch job. The output can be printed at a remote printer or it may be routed to the kernel. If the output is handled by the kernel, then it may be directed to the terminal without storing it permanently on the disk or it may be stored permanently on a disk file and retrieved by the user at a later time. In the non permanent case the command interpreter requests the file marager to create a temporary file on the disk before directing the output to a terminal. This is to insure a uniform procedure for handling the data flow from the host to the kernel.

Therefore, in both instances where the output from the host is routed to the kernel a disk file is created.

File Management

In this section only the interface with file management is discussed. File management consists of two modules: the file manager and the disk manager. The file manager is responsible for maintaining all user and system files and the disk manager is responsible for controlling all disk input/output operations.

The command interpreter and the file manager communicate by means of a file request queue (FRQUE). FRQUE is defined by:

```
DECLARE
1 FRQUE,
  2 REQSR BIT(1),
  2 ISFN CHARACTER(FILE-NAME.LENGTH),
  2 OSFN CHARACTER(FILE-NAME.LENGTH),
  2 ADRUR POINTER,
  2 ADIOCB POINTER,
  2 LINK POINTER;
```

where the fields are defined as follows:

REQSR: This field is used as a request source indicator. It is zero if the request is from ICCS; one if it is from the command interpreter. IOCS requests the file manager to copy the data from the user area when it is full. This is in conjunction with retrieving a batch job.

ISFN: This field is the input symbolic file name. The length of this field is fixed by the file manager and determined by how it builds its file directory. This is the name of the file given by a user in submitting his batch job.

OSFN: This field is the output symbolic file name. This is the name of a disk file to contain the output of a batch job.

ADRUR: This pointer is the address of the user area that is used to transfer data to or from the disk.

ADIOCB: This pointer is the address of IOCB. This field and ADRUR are passed to the disk manager.

LINK: This is a pointer to the next request, if any.

FRQUE is maintained just as REQQUE and it is a circularly linked queue. Memory management provides an availability list for allocating and deallocating nodes for FRQUE.

The only time that the command interpreter communicates with the file manager is when there is a request from the user to do the following:

1. Output a file (stored on the disk) to one of the I/O devices or
2. Submit a job stored in a giver file to the host (the ISFN field contains this file name) or
3. Retrieve the output of a job executed by the host and store it in a file or display immediately at the terminal.

For the third case the ISFN field contains the name of a file specified by the user if the file is a permanert file. Otherwise, the command interpreter initializes ISFN with a dummy name and the data type of this file is TEMP.[2]

In servicing a request the file manager directs the disk manager to either move the information from the disk to the user area or from the user area to the disk, depending on the nature of the request. For this purpose file management has another queue for the file manager to request the disk manager to control the disk I/O (DRQUE) and a disk control block (DKCB). DRQUE is defined by:

```
DECLARE
1 DRQUE,
  2 FLOW BIT(1),
  2 ADRDKCB POINTER,
  2 DKACS BIT(1);
```

where the fields are defined as follows:

FLOW: This bit indicates the flow of the data. If the data is to be stored on the disk FLOW is zero (input), otherwise FLOW is one (output).

ADRDKCB: This pointer is the address of disk control block (DKCB) which in turn is used by the disk manager for finding more information about a request from the file manager.

---

[2]A file name may consists of the name and the type of the file. This restricts the user not to use TEMP as the data type for a file which will contain the output of a job.

DKACS: This is zero if disk access is not needed. In other words, the file is currently in the memory. It is one if a disk access is needed. This field is used only when FLOW is one (output).

The disk control block (DKCB) contains the necessary information for the disk manager to perform its task. There is a different DKCB for each request and it is dynamic. The file manager requests memory management to allocate a DKCB when it places a request for the disk manager in DRQUE. When the disk manager finishes its task, it removes an entry from DRQUE and calls memory management to deallocate the corresponding DKCB. DKCB is defined by:

```
DECLARE
1 DKCB,
   2 PHYADR FIXED BINARY(31,0),
   2 NOPB FIXED BINARY(31,0),
   2 ADRUR POINTER,
   2 ADIOCB POINTER,
   2 USIND BIT(1),
   2 EOI BIT(1);
```

where the fields are defined as follows:

PHYADR: This field is the physical (device) address if DKACS in DRQUE is one. Otherwise, it contains the address of a memory location.

NOPB: This field contains the size of the file.

ADIOCB: This pointer is the address of IOCB. The disk manager needs this information to initialize the IOCB fields in requesting IOCS to move the data from a user area to the output buffer.

USIND: This bit is zero if a user area is empty, one otherwise. This indicator is used when FLOW is one (output only). When the disk manager moves the data from the disk to the user area, it sets USIND to one and IOCS will reset it to zero when it is empty.

EOI: This bit is an end of file indicator. If it is one (no more data) the disk manager "closes" the file, zero otherwise. This field is set and reset by IOCS depending on EOD in IOCB.

In servicing a request in FRQUE, the file manager searches the file directory to find the physical address of the corresponding file. The file manager then places a request in DRQUE, calls memory management to allocate a DKCB, initializes DKCB and deletes the current request in FRQUE.

The interface betwen the disk manager and IOCS is similar to the interface between IOCS and IOS. While IOCS is copying the data from the user area to the output buffer (this is reflected by USIND being one) the disk manager services the next request, if any. There will not be any request for the same I/O device in DRQUE because the command interpreter will not schedule such a request.

When this input/output operation is completed, the disk manager dispatches a user from the "blocked for I/O device" queue. Figure 14 describes the relationship between the

file manager and disk manager and its control block (DKCB). It is also shown how the disk manager places a request in REQQUE and indirectly initializes some of the fields in IOCB.

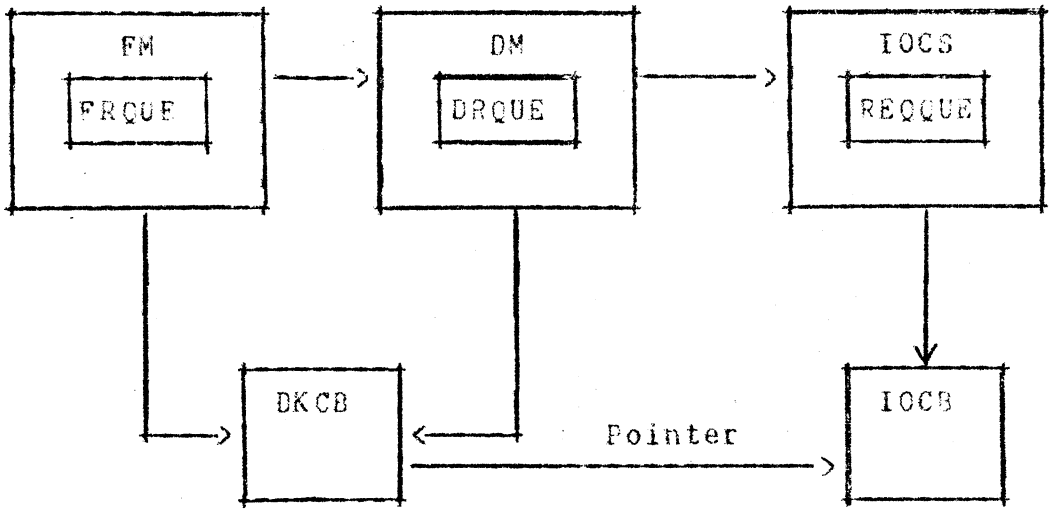

Figure 14. File Management and its Control Blocks

Memory Management

The kernel is an operating system designed specifically for a microcomputer system. For a real-time application it is more desirable to distribute the system functions over several processors. For the above reason and because most modules of the kernel need services from memory management, some of memory management functions exist on all processors.

This allows modules of the kernel to request memory management to allocate space within its own domain (processor).

As a summary, memory management performs the following functions:

1. It allocates and deallocates various control blocks such as the disk control block (DKCB) and the input/output control block (IOCB).
2. It allocates and deallocates user areas.
3. It allocates and deallocates nodes for REQQUE,FRQUE, and DRQUE.
4. It allocates and deallocates "blocked user" save areas.
5. It updates BANKTAB for allocation and deallocation of user banks.

In allocating or deallocating nodes for request queues memory management provides an availability list for each type of queue. Every time a new node for a particular request queue is needed, a call to a procedure named GETNODE is made. GETNODE examines the corresponding availability list and returns the first node on the list, if there is one on the availability list. When a node is not needed anymore it is inserted at the front of the availability list by a procedure named RETNODE. There is a similar GETNODE and RETNODE for REQQUE,FRQUE, and DRQUE.

Memory management maintains two tables, BANKTAB and TERMTAB to keep track all user banks in the system. The I/C interface controller has a bank select feature which can be used to select one of the memory banks. The size of BANKTAB

depends on the number of user banks in the system and the number of the terminals cannot exceed the number of user banks. The structure of BANKTAB is defined by:

```
DECLARE
1 BANKTAB(*),
  2 TERMNO FIXED BINARY(15,0);
```

where the field is defined as follows:

TERMNO: This field contains a terminal number if the bank is occupied; otherwise, it contains a value of zero to indicate that a user bank is free.

The structure of TERMTAB is defined by:

```
DECLARE
1 TERMTAB(*),
  2 BANKNO FIXED BINARY(15,0);
```

where the field is defined as follows:

BANKNO: This field contains a bank number for an active terminal; otherwise, it contains a value of negative one to indicate that the corresponding terminal is not active. In kernel "initialization" all entries of this table are initialized to negative one. Note that zero might be used to identify system bank. The numbering system for user banks starts with the number one.

When the logon module calls memory management to allocate a free user bank, memory management searches this BANKTAB to find the first free bank. It updates the entry

of BANKTAB with the corresponding terminal number and the entry of TERMTAB is updated to contain the corresponding bank number. At the end of a user session the logoff module calls memory management to deallocate a user bank. This causes memory management to search TERMTAB to find the bank# associated with the terminal# and updates the entry of TERMTAB to negative one. Using the bank# obtained from the TERMTAB memory management updates the entry of BANKTAB to zero to indicate that this user bank is free.

When modules of the kernel need to request space memory management must be able to allocate a block of contiguous storage of the correct size. For example, the command interpreter requests memory management to allocate a user area and an IOCB for the printer when a user needs to output a disk file to the printer. Memory management uses dynamic storage management to allocate and deallocate a dynamic IOCB control block, the save areas for "blocked user" and user areas. The size of IOCB control blocks and of save areas are fixed but the the size of user areas are variable and depend on the length of a record (the physical characteristics of I/O device determines the length of a record). For this reason a dynamic storage management is preferred to static storage management.

Dynamic storage management uses algorithms to reserve and free variable size blocks of storage which are in contiguous memory locations. Knuth (3) discusses dynamic

storage management algorithms such as first-fit,best-fit, liberation with sorted list, boundary tag method and buddy system.

First-fit and best-fit are methods for searching and reserving a block of storage if there are any blocks with the required size (say size of N). The first-fit method chooses the first area from the available space that is greater than equal to N. On the other hand, the best-fit method chooses an area with size M where it is the smallest which is N or more. This usually requires searching the entire list of available space before a decision can be made. The disadvantage of using either one of these method is that there are certain situations in which the first fit method is better than the best-fit method and vice versa. Knuth (3) demonstrates an example for a situation in which first-fit is better than the best-fit; suppose there are two available blocks of memory of sizes 1300 and 1200, and suppose there are subsequent requests for blocks of sizes 1000,1100,and 250:

memory request	available areas "first-fit"	available areas "best-fit"
-	1300, 1300	1300,1200
1000	300,1200	1300,200
1100	300,100	200,200
250	50,100	unallocated

Liberation with sorted list is a method for freeing blocks and inserting the block at the appropriate location of the sorted available space list when they are no longer

needed. It also merges two adjacent free areas into one. In fact, when an area is bounded by two free areas, all three areas are merged together.

All the three methods discussed above require an extensive searching through the availability list. The boundary tag method or the buddy system eliminates most of the searching when storage is reserved.

The boundary tag method requires fields for control information at both ends of each block. One of the fields is a TAG field which is used to control the collapsing process (it is easy to detect whether or not both adjacent block are available). This method is perhaps too much of a penalty to pay in situations when the blocks have a small average size. Another approach to dynamic storage management is the buddy system. The overhead in each block is less compared to the boundary tag method and it requires all blocks of size of a power of two. The buddy system keeps separate lists of available block of each size 2**K (0 <= K <= M) and the entire block is of size 2**M. When a block of size 2**K is desired, and if nothing of this size is available, a larger available block is split into two equal parts called buddies. Later when both buddies are available again, they coalesce back into a single block. The disadvantages of the buddy system are internal fragmentation and allocation of unused space.

Diagram of System Structure

In order to give a better overview of all the system modules and how they communicate, a diagram showing the overall system structure is given in Figure 15.

The communication between the modules is described below. The numbers below refer to the diagram numbers in Figure 15.

1. IOCS requests the communication module to set up the protocol and transfer the data from a user area to an output buffer (for output to the host only).

2. IOS requests IOCS to transfer a record from the input buffer to the user area by placing a request in REQQUE. IOCS then requests the communication module to perform this task (for input from the host only).

3. IOCS copies a record from the user area to the output buffer and initializes I/O channel to generate an output interrupt (for an output request to any system devices, except the host).

4. IOS transmits a record character by character to the I/O device after I/O channel generates an output interrupt.

5. The I/O device sends a character and stores in the input buffer. The I/O channel places the input character in the receiver buffer.

6. IOCS copies the data from the input buffer to the corresponding user workspace upon request from IOS (for input from terminal only).

7. For an output request, IOS sets STATUS in IOCB to "INACTIVE" when the end of record is detected and "ACTIVE" otherwise. For an input request, IOS places a request in REQQUE and initializes IOCB when the end of record is detected.

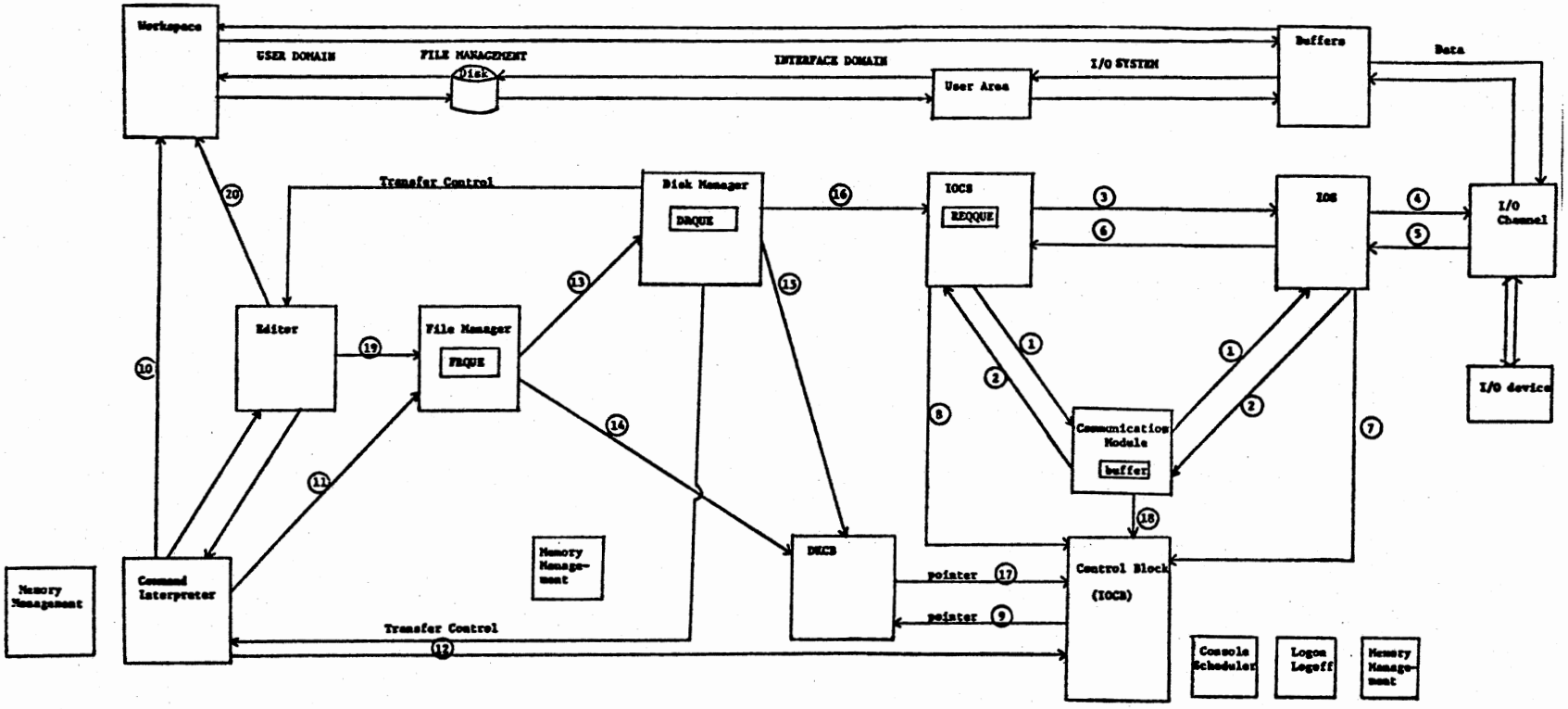

Figure 15. Diagram of Communication of Kernel Modules

8. For an output request, IOCS checks the STATUS field in IOCB. If it is inactive and the user area is not empty then IOCS copies a record from the user area to the output buffer.

9. For an output request, IOCS follows the pointer in IOCB to set the USIND field in DKCB to zero when the user area is empty. For an input request, IOCS checks the EOD field in IOCB. If EOD is one (end of data), then IOCS follows the pointer in IOCB to set the EOI field in DKCB to one. This is mainly for input from the host.

10. The command interpreter interprets a user command contained in the user workspace.

11. The command interpreter places its request in the FRQUE queue.

12. The command interpreter initializes IOCB (ADRUR) for the request made in 11.

13. The file manager places its request in DRQUE associated with the request in 11.

14. The file manager initializes DKCB associated with the request in 13.

15. For an output request, the disk manager checks the USIND field in DKCB. If it is zero (the user area is empty) then the disk manager copies a block of data from the disk to the user area and sets USIND to one to indicate that the user area is full.
For an input request the disk manager checks the EOI field in DKCB. If it is one then it transfers control to the command interpreter; otherwise the disk manager waits for an urgent request from IOCS which indicates that the user area is full.

16. For an output request the disk manager places its request in REQQUE to order IOCS to copy the data to the output buffer.

17. For an output request the disk manager follows the pointer in DKCB to initialize the TOTAL and ADRDKCB fields in IOCB (other fields have been initialized by the command interpreter in 12). When the file is empty the disk manager sets EOD in IOCB to one; otherwise it resets to zero.

18. For "input from the host" the communication module sets ECD to one when end of data is detected; otherwise it resets to zero.

19. The editor requests the file manager to copy a file to a user workspace.

20. The editor does its processing in a user workspace.

CHAPTER V

DISCUSSION OF A KERNEL COMMAND

The Procedure to Output a Disk File to the Local Printer

The procedure of how the modules of the kernel communicate with each other in performing a task such as outputting a file stored on a disk to a local printer is presented in this chapter. The purpose is to give a better overview of the design of the kernel. The choice of this user command is reasonably good because it involves most of the system modules in the kernel.

The syntax of the command is defined as follows:

OUTPUT <input file name> <destination>.

The user is required to supply the input file name and designate the destination to be the line printer.

The command interpreter checks the activity of the line printer. If it is being used the command interpreter delays the servicing of this request by placing the status of processor for the user in a "blocked user" queue; otherwise the command interpreter does the following:

1. It sets the printer activity to one to indicate that the printer is being used.

2. It calls memory management to allocate a user area and an IOCB for the printer.

3. It initializes the printer's IOCB.

4. It places a request in FRQUE.

5. It blocks this user by placing the processor's status in a "blocked user" queue.

In servicing a request placed by the command interpreter in FRQUE the file manager searches its file directory to find the physical address of a named file. The actual data movement is carried out by the disk manager. The file manager communicates with the disk manager by placing a request in DRQUE and by initializing its DKCB. Before the disk manager assumes control the file manager removes the request from FRQUE.

Upon receiving a request from the file manager, the disk manager performs the following operations:

1. Get the necessary information for a disk transfer from DKCB.

2. If USIND in DKCB is still one[1] then delay the request by moving the FRONT pointer to point to the next request; otherwise move the data to the user area and set the USIND field to one to indicate that the user area is full.

3. Call memory management to allocate a node for REQQUE.

4. Place the request at the end of the queue.

5. Set TOTAL field in IOCB to the length of the user area. Also initialize ADRDKCB to the address of

---

[1]a one in USIND indicates that IOCS has not finished moving the data from the user area to the output buffer

the disk control block (the command interpreter has initialized other IOCB's fields).

6. When the user area is empty,[2] repeat steps 2,3,4, and 5 above until there is no more data to be printed.

7. If there is no more data to be transferred and the user area is empty then:

   a. Call memory management to deallocate the user area and the printer's IOCB.

   b. Delete the request in DRQUE and deallocate its DKCB.

   c. Reset the device activity to zero (it is free now).

   d. Dispatch the user from a "blocked for printer" which was created by the command interpreter when the user requests an I/O operation with the printer.

Miscellaneous

The procedure for printing a file to the local printer can be shortened if the file is a current file. A current file is defined as a file that is currently in a user workspace.

When the editor requests the file manager to copy the data to the user workspace, the file manager should mark the "activity" field of a file in the file directory as "current." This information determines the value of DKACS field in DRQUE. If the file is in the user workspace, DKACS has the value of zero and PHYADR in DKCB will contain the

---

[2]This indicates that IOCS has finished moving the data from the user area to the output buffer.

address of memory location instead of the physical (device) address. The memory address can be obtained by searching a "system table" provided by memory management (memory management keeps track of all active users in the system and assigns both types of workspace to each users).

The kernel also provides a mechanism for cancelling the output printed at the local printer. To support this, a fixed location is reserved to contain the address of a disk control block. The symbolic name of the location is LPREQ. This location is used by the disk manager and command interpreter. If there is a request to output a file to the local printer the disk manager initializes LPREQ with the address of the disk control block.

The actual cancellation is recognized by the kernel as an operator command which has higher priority than any user command. When the command interpreter interprets an operator command to cancel the output printed at the local printer, it sets the NCPB field in DKCB to zero to indicate that there is no more data to be printed. The address of DKCB is found from LPREQ. At the time NCPB is zero, the process of printing a file to the printer could be in several different states; IOCS might be transferring data to the output buffer, IOS might be transferring data from the output buffer to the line printer or the disk manager might be copying data to the user area. In any of the above cases several lines of output might be printed at the printer before the disk manager stops the process.

CHAPTER VI

SUMMARY, CONCLUSIONS, AND FUTURE WORK

Summary and Conclusions

The kernel is useful for a real-time application where every users communicate with the system by means of terminals. The kernel provides a user with the following functions:

1. it allows a user to create, delete and keep a file on a disk;
2. it allows a user to print a file on a local or remote printer;
3. it allows a user to submit a batch job to the host computer;
4. it allows a user to retrieve a batch job from the host and store it on a disk file or display the output just retrieved immediately at his terminal;
5. it allows a user to directly print the output of a batch job at the host printer.

At present, the design of kernel only supports background jobs executed in a foreground manner. It does not allow a user to execute a job within the microcomputer system. By expanding the system to include more than one microcomputer and by distributing the operating system over

these processors, it would be possible for the system to execute some jobs locally, for example, a compile step could be done locally followed by a debug session, or possibly an incremental compiler could be accessed locally. It is evident that multiple processors are necessary to achieve a rapid response from the system.

Future Work

The design of the kernel could be expanded to allow some of the features described above. This would require the addition of other modules to the kernel. One such module would be the processor management module which would perform at least the following functions:

1. schedule jobs to be run on various processors;
2. allocate a processor to a user in the ready state;
3. determine the maximum time a user may use an allocated processor;
4. initiate a user into a running state on an allocated processor;
5. monitor the status of all users in the system;
6. swap users that enter a blocked state;
7. deallocate a processor;
8. allocate the necessary resources;
9. deallocate these resources when the job is done;
10. protect users from each other and the operating system from users.

An extension of this type will add another level to the present hierarchical kernel structure. The decision on, the placement of the processor management module with the hierarchical structure and how the operating system should be distributed, has to be made. The design of the interfaces between the processor management module and other modules of the existing system must be determined as well as how the processor management module takes control of the system.

BIBLIOGRAPHY

(1) Cromemco TU-ART Digital Interface. Instruction Manual. Cromemco Incorporated, 1978.

(2) Holt, R.C., G.S. Graham, E.D. Lazowska and M.A. Scott. Structured Concurrent Programming with Operating Systems Applications. University of Toronto, Computer Systems Research Group, Toronto, Ontario. Massachusetts: Addison-Wessley Publishing Co., 1978, 139-159.

(3) Knuth, Donald E. The art of Computer Programming, vol 1, second edition. Massachusetts: Addison-Wesley Publishing Co., 1973, 435-451.

(4) Madnick, Stuart E. and J.J. Donovan. Operating Systems. New York: Mc Graw-Hill Book Co., 1974, 66.

(5) Osborne, Adam and Associates. An Introduction to Microcomputers, vol 2, Some Real Products, June 1977 revision. Adam Osborne and Associates Inc., 4.24-4.25, 7.20-7.21.

(6) Schoeffler, James D. The Small Computer Concept. IBM Series/1. International Business Machines Corporation, General Systems Division, Atlanta, Georgia, 1978, 402-436.

(7) Van Doren, James R. Notes on Software Design Methods. (Unpub. supplementary class notes). Stillwater, Oklahoma: Oklahoma State University, 1978.

APPENDIX A

CHARACTERISTICS OF THE I/O INTERFACE CONTROLLER

The present hardware configuration consists of a Z80 CPU, multiple Cromemco TU-ART boards for input/output and multiple Cromemco 16 KZ memory boards (16K RAM). The TU-ART (twin universal asynchronous receiver and transmitter) can perform the following functions:

1. It converts output data from parallel to serial form and input data from serial to parallel;

2. As a transmitter it adds start and stop bits, generates parity, and clocks the data out at the required baud rate. As a receiver it recognizes and deletes start and stop bits, check parity, and clocks the data in at the required rate;

3. It provides indicators that tell whether it has received data or is ready to accept data for transmission. Other indicators are used to detect errors in the received data.

The TU-ART has two channels of duplex serial data exchange, two channels of parallel data exchange and ten intervals timers in which each interval timer can activate an interrupt. It contains two TMS 5501 (Texas Instruments) I/O Controller chips which will refered to as "Device A" and "Device B".

In order to support interrupts on a priority basis, multiple TU-ART boards may be connected together. Each I/O board is provided with two lines, J1 PRIORITY OUT/ and J1 PRIORITY IN/. Priority is set by the location of the I/O board in a daisy chain configuration. This is done by connecting J1 PRIORITY OUT/ from the highest priority TU-ART to J1 PRIORITY IN/ of the next highest priority TU-ART, then connecting J1 PRIORITY OUT/ of the second TU-ART to J1 PRIORITY IN/ of the next TU-ART until all TU-ART are connected. The J1 PRIORITY IN/ of the highest priority TU-ART is left unconnected. This priority configuration insures that a higher priority device will be serviced before a lower priority device when two or more interrupt requests occur at the same time. Device A is internally prioritized over Device B.

For a TU-ART to have priority its J1 PRIORITY IN/ must be high. When an I/O board needs service, it will prevent downstream I/O boards from interrupting by pulling low on its J1 PRIORITY OUT/. The next I/O board in the chain sensing a low at the J1 PRIORITY IN/ will pass this priority signal on the next I/O board by pulling low on its J1 PRIORITY OUT/ and so on.

The base address of I/O ports are determined by the position of a switch called a DIP switch. This base address consists of the high-order bits of an 8-bit I/O port address. At the present time the disk controller uses zero

as the I/O base address of Device A, 50H (hex) as the I/O base address of Device B and 40H is dedicated for memory bank select feature.[1]

The memory space is organized into 8 banks of 64K each. Each memory bank(s) may be enabled under software control by addressing I/O port 40H. The 8-bit output from port 40H enables or disables the corresponding bank(s) in memory. A set bit '1' in the corresponding bit position will enable the memory bank and a reset bit '0' will disable it. On power up the active memory bank is bank 0; this is used as a system bank in the design of the kernel.

Each TU-ART uses 14 different ports for data and control. They are:

status register	- input port
baud rate register	- output port
receiver data register	- input port
transmitter data register	- output port
interrupt address register	- input port
interrupt mask register	- output port
parallel port	- input port
parallel port	- output port
timer 1	- output port
timer 2	- output port
timer 3	- output port
timer 4	- output port

---

[1]Cromemco 16KZ RAM. Instruction Manual. Cromemco Incorporated, 1978.

timer 5 - output port

The block diagram of TMS 5501 containing the above registers is described in TMS 5501 Multifunction Input/Output Controller.[2]

Certain features of the kernel are dependent upon the above hardware. Timer and priority interrupts are necessary for designing an interrupt driven I/O system. Most of the modules of Input/Output System such as IOS, Logon and the console scheduler are interrupt-driven routines. The ability to isolate the hardware dependencies in any software system is important for achieving a reasonable degree of portability. In the design of the kernel, the hardware dependencies are confined to the IOS module. The implementation of the other modules can be done in a system programming language.

---

[2]TMS 5501 Multifunction Input/Output Controller. Texas Instruments Inc., 1976.

APPENDIX B

OUTPUT INTERRUPT ROUTINE DATA STRUCTURES

The following describes the data structures used in the output interrupt routine and its driver:

```
1 OUBUFi       - output buffer for user# i
  2 NOCHAR     - # of characters to be output
  2 BUFFPTR    - buffer pointer contains the
                 address of next location of
                 output buffer space
  2 ADDR_OUTB  - output buffer space (the maximum
                 is 257 characters - the last 2
                 storages are used for imbedding
                 carriage return and line feed)

1 USRARi     - user workspace for user# i
  2 NOCHAR   - it contains the length of user
               workspace
  2 USRPTR   - user workspace pointer contains the
               address of next location of user
               workspace
  2 UWKSP    - user workspace

1 UCBT1      - output control block for terminal 1
  2 TOTAL    - # of characters to be output
  2 STATUS   - it contains the status of a request
               0 - request is in waiting line or
                   completed
               1 - request is being serviced (in
                   progress)
  2 DEVTP    - device type
               1 - terminal
               2 - disk
               3 - tape      etc
  2 DEVNUM   - device#
  2 EOR      - end of record indicator
               0 - no
               1 - yes
```

NOTE:
Naming convention for I/O control block :
UCBXY where X is device type
Y is device#
Example: UCBT1 is the output control block for terminal# 1

1 OU1REQ(*) - table contains output requests
2 REQ - request indicator
0 - no request
1 - there is a request
2 ADDR - it contains the address of output control block (UCB) if REQ is one

RQTSIZ - it contains the number of requests which have been serviced

ORPTR - it contains the address of next request to be serviced

NOTE:
Both RETSIZ and ORPTR are used for implementation purposes only.

APPENDIX C

PDL DESCRIPTION OF OUTPUT INTERRUPT ROUTINE

The following section contains a Program Design Language (PDL) description of the output interrupt routine and its driver. The data structures used are described in Appendix B.

```
OUINDR : PROC ;
   /*
   This is an output interrupt driver. This driver controls
   the flow of data from a user workspace to an output
   buffer. This transfer of data is done record-by-record.
   Assume that a block of characters to be sent to a device
   are in the appropriate user workspace and the correspond-
   ing UCB entries are filled with the information needed.
   This routine searches CUTREQ table to find whether there
   is a request. If no request is found then this routine
   does nothing. Otherwise, it checks the status of this
   request. If the request is still in progress (the output
   interrupt has not finished printing the data), then it
   delays servicing this request by scanning the next entry
   in OUTREQ. Otherwise it checks whether the end of record
   indicator is on or not. If it is not on then this routine
   scans the next entry in OUTREQ; otherwise it checks
   whether there are more characters to be printed. If not
   then it indicates that the request is completed; other-
   wise:

      1.    Call BUFADR to find the address of user
            area and output buffer for the
            corresponding device type and device#.

      2.    Call CP1 to load the cutput buffer with
            the characters to be printed.

      3.    Call SETOI to update the status of the
            request and initialize the I/C controller.
```

```
*/


DO WHILE('1'B);
    Search OUTREQ  table for a  request and  store output
    control block address in UCBADR, if any;
    IF STATUS¬=1 & EOR¬=0 & TOTAL¬=0 THEN DO;
        CALL BUFADR(UCBADR,OUBUFi,USRARi);
        CALL CP1(USRARi,OUBUFi,UCBADR);
        CALL SETOI(UCBADR);
    END;
    ELSE IF TOTAL = 0 THEN STATUS = 0;
END;


BUFADR: PROC(UCBADR,OUBUFi,USRARi);
    Get DEVTP and DEVNUM from UCBADR;
    IF DEVNUM = i THEN DO;
        Set USRARi to contain the address of user
            workspace for device# i;
        Set OUBUFi to contain the address of output
            buffer for device# i;
    END;
    Return;
END BUFADR;


CP1: PROC(USRARi,OUBUFi,UCBADR);
    Initialize OUBUFi.NOCHAR to zero;
    DO WHILE(USRARi.NOCHAR ¬= 0);
        Move a character from MEMORY(USRPTR) to
          MEMORY(BUFPTR);
        Increment OUBUFi.NOCHAR by one;
        Decrement USRARi.NOCHAR by one;
        Advance USRPTR and BUFFPTR to point to the
           next location;
    End;
    /*
    At the end of a record, a carriage return and line
    feed characters are imbedded into the character stream
    to let the device (terminal) skip to the next line.
    */
    Store carriage return and line feed characters in
        MEMORY(BUFFPTR+1) and MEMORY(BUFFPTR+2),
        respectively;
    Set TOTAL to OUBUFi.NOCHAR;
    Return;
END CP1;


SETOI: PROC(UCBADR);
    Set UCBADR.STATUS to one; /* in progress */
    Set UCBADR.EOR to zero;
    Initialize I/O controller for UCBADR.DEVTP and
```

```
            UCBADR.DEVNUM specified;
        Return;
    END SETOI;
END OUINDR;


OUINT: PROC;
    /*
    This is an output interrupt routine. Each device has
    its own output interrupt routine.
    */
    IF OUBUFi.NOCHAR = 0 THEN DO;
        Set UCB.EOR to one; /* end of record */
        Set UCB.STATUS to zero; /* request is completed */
    END;
    ELSE DO;
        Output a character from MEMORY(OUBUFi.BUFFPTR);
        Advance OUBUFi.BUFFPTR to point to the next
            location;
        Decrement OUBUFi.NOCHAR by one;
    END;
    Restore all registers;
    Enable interrupt;
    Return;
END OUINT;
```

APPENDIX D

INPUT INTERRUPT ROUTINE DATA STRUCTURES

The following describes the data structures used in the input interrupt routine & its driver:

```
1 INBFAi      - input buffer A for user# i
  2 NOCHAR    - # of characters
  2 BUFFPTR   - buffer pointer contains the
                address of next location of
                input buffer space
  2 ADDR_INB  - input buffer space (the maximum
                is 255 characters)

1 INBFBi LIKE INBFAi - input buffer B for user# i

1 USRARi - user workspace for user# i
           (the fields and their descriptions
            are found in Appendix B)

1 ICBT1     - input control block for terminal# 1
  2 DIR     - it contains the direction of data
              movement.
                0 - input request (the data is copied
                    from input buffer to user
                    workspace)
                1 - output request (the data is copied
                    from user workspace to output
                    buffer)
  2 STATUS  - it contains the status of a request
                0 - request is in waiting line or
                    completed
                1 - request is being serviced (in
                    progress)
  2 DEVTP   - device type
  2 DEVNUM  - device#
  2 ADRUW   - it contains the address of user
              workspace
  2 ADRIB   - it contains the address of active
              input buffer
```

NOTE:
For input interrupt routine the value of ICBT1.DIR is zero at all times.

```
1 INREQ(*) - table contains input requests
  2 REQ    - request indicator
             0 - no request
             1 - there is a request
  2 ADIOCB - it contains the address of input
             control block (IOCB) if REQ is one
```

The following data structures are used by the input interrupt driver to check for input requests.

```
MRTSIZ - it contains the number of requests which
         has been serviced
MVRPTR - it contains the address of next request
         to be serviced
```

The following data structures are used by the input interrupt routine to get the next empty entry in INREQ table.

```
PTNERT - it contains the address of the next
         empty entry in INREQ
NOREQ - # of requests in INREQ
```

NOTE:
MRTSIZ, MVRPTR, PTNERT and NOREQ are used for implementation purposes only.

```
INPTi - a pointer used for swithching between
        two input buffers (for user# i)
        1 - indicates that the first buffer
            (buffer A) is active
        2 - indicates that the second buffer
            (buffer B) is active
            Initially, it contains the value of one.
```

APPENDIX E

PDL DESCRIPTION OF INPUT INTERRUPT ROUTINE

The following section contains a Program Design Language (PDL) description of the input interrupt routine and its driver. The data structures used are described in Appendix D.

```
INITDR: PROC;
   /*
   This is an input interrupt driver. The driver examines
   INREQ table for a request from the input interrupt rou-
   tine. The input interrupt routine requests INITDR to copy
   data from an active input buffer to the corresponding
   user workspace.
   This routine searches INREQ table to find whether there
   is any request or not. If no request is found then this
   routine does nothing; otherwise:

     1.      Reset the buffer pointer of an active
             input buffer to point to the beginning of
             input buffer.

     2.      Call CP0 to copy data from the input
             buffer to the corresponding user
             workspace.


   */


   DO WHILE('1'B);
       Search INREQ table for a request;
       IF there is a request THEN DO;
           Store the address of input control block
              in ICBADR;
           CALL CP0(ICBADR);
           Set REQ to zero; /* request is completed */
       END;
```

```
    END;


    CP0: PROC(ICBADR);
        Get the address of input buffer(INBADR) and user
           workspace(UWSADR) from ICBADR;
        Initialize UWSADR.NOCHAR to zero;
        DO WHILE(INBADR.NOCHAR ¬= 0);
           Move a character from MEMORY(INBADR.BUFFPTR)
              to MEMORY(UWSADR.USRPTR);
           Increment UWSADR.NOCHAR by one;
           Decrement INBADR.NOCHAR by one;
           Advance INBADR.BUFFPTR and UWSADR.USRPTR to
              point to the next location;
        END;
        Return;
    END CP0;
END INITDR;


ININT: PROC;
    /*
    This is an input interrupt routine. Each device has
    its own interrupt routine.
    */
    IF INPTi = 1 THEN INBACT = INBFAi;
                 ELSE INBACT = INBFBi;
    Input a character from a device;
    Echo input character;
    IF end of record is detected THEN DO;
        IF INPTi = 1 THEN INPTi = 2;
                     ELSE INPTi = 1;
        Get an empty INREQ entry;
        Set REQ = 1 and ADIOCB to contain the address
           of the corresponding input control block;
        /*
        Initialize fields in input control block.
        */
        Set DIR to zero; /* input request */
        Set ADRUW to USRARi;
        Set ADRIB to INBACT;
    END;
    ELSE DO;
        Place the character received in
           MEMORY(INBACT.BUFFPTR);
        Increment INBACT.NOCHAR by one;
        Advance INBACT.BUFFPTR to point to the next
           location;
    END;
    Restore all registers;
    Enable interrupt;
    Return;
END ININT;
```

VITA

Sylvana Kristanti-Sari

Candidate for the Degree of

Master of Science

Thesis: DESIGN OF OPERATING SYSTEM KERNEL FOR A MICROCOMPUTER SYSTEM

Major Field: Computing and Information Sciences

Biographical:

Personal data: Born in Surabaya, Indonesia, on November 5, 1954.

Education: Graduated from St. Agnes High School, Surabaya, Indonesia, in December, 1972; received Bachelor of Mathematic from University of Waterloo, Waterloo, Cntario, in August, 1977; completed requirements for Master of Science degree at Oklahoma State University, Stillwater, Oklahoma, in December, 1979.

Professional Experience: Graduate research assistant at Oklahoma State University, Electrical Engineering, October, 1977-May, 1978; programmer for Department of Parasitology at Oklahoma State University, Summer, 1978; graduate teaching assistant at Oklahoma State University, Department of Mathematics, Fall 1979; graduate teaching assistant at Oklahoma State University, Computing and Information Sciences Department, Spring 1979.