1978

A User's Introduction to the Dual-Mace Operating System

V. Y. Shen

Stephen Tolopka

Report Number:

78-295

A USER'S INTRODUCTION TO THE DUAL-MACE OPERATING SYSTEM

V. Y. Shen

Stephen Tolopka

TR-295

September, 1978

I. Introduction

A modern computer system normally has a large collection of devices which are built for different functions and can often be operated independently of one another. The heart of the system is, of course, the central processing unit (CPU), which is responsible for performing all the arithmetic and logical operations on the jobs stored in its memory. Attached to the CPU are the peripheral devices, whose functions are information storage (magnetic disks, tapes) or input/output (card readers, printers, terminals, etc.) Many of these devices (including the CPU) may be duplicated for additional power and reliability. The computer operating system is a set of computer programs that manage these devices and provide a reasonable environment for the users and their programs. This document describes the basic concepts and facilities of the Dual-Mace operating system at Purdue. It is structured in such a way that each section introduces an additional feature that the operating system supports. The user is assumed to be familiar with at least one higher-level language such as FORTRAN. It is our hope that through the reading of this document the user will be able to compose control statement sequences to accomplish what is needed in most applications.

II. The Basic Computer

Many early computers were designed with just one input device and one output device attached to the CPU (Figure 2.1). The computer took data from prepunched cards or a section of tape, and listed the results on a typewriter-like printer. A modern computer can still be considered as shown in Figure 2.1. The input contains both program and data punched on cards, and the output is printed on a high-speed line printer. A FORTRAN job for the Dual-Mace system may be set up as follows:

```
 0751.                              (Sequence card)
76543,UID.                          (Job card)
PASS=UPASS.                         (Password card)
MNF.                                (Control card)
#eor                                (End-of-record card)
    .
    .
    FORTRAN source program
    .
    .
#eor                                (End-of-record card)
    .
    .
    Data
    .
    .
#eoi                                (End-of-information card)
```

It should be recognized that there are many users at a large computing center. The job card and the password card are used to tell the computer that the user is authorized to use the account. The main purpose of the sequence card is for the orderly filing of the final output, so that it may be retrieved by the user on a self-service basis. (The output will be identified on the listing as UID0751 in large, composed letters.) Since different users may program in different languages, the control card MNF tells the computer that this program is written in a version of FORTRAN (MiNnesota Fortran). The input actually consists of three types of information; control cards, FORTRAN source, and data. They are presented to the computer in three sequences, called "records", separated by special end-of-record cards that have 7,8,9 multi-punched in column 1. The three records make up a "file", whose end is identified by the special end-of-information card that has 6,7,8,9 multi-punched in column 1.

One of the purposes of the end-of-information card is to signal the computer to stop when the program tries to read more data than it is supplied. Note that the sequence card supplied by the computing center also has 6,7,8,9 punched in column 1. This protects the user in the event that the previous user "forgets" to include the end-of-information card and then tries to read too much data.

III. Spooling

The CPU's in the earliest computer systems were built using electronic relays and vacuum tubes. They were bulky, slow, and required a lot of power. Advances in electronic technology have significantly improved the performance of CPU's. A large computer such as the CDC 6500 can perform about a million 14-digit arithmetic operations per second. A modern computer using microelectronics could be a hundred times faster than that. Peripheral devices, on the other hand, are mostly mechanical. Their improvement in speed has not been as dramatic as that of the CPU. A modern impact-type line printer, for example, can operate at 1200 lines per minute. Assuming ten arithmetic results are packed in one output line, such a printer can print at the rate of only 200 results per second. This is hardly adequate to keep up with the CPU if the computer system is organized as shown in Figure 2.1.

The problem of mismatched speed is handled by the introduction of a secondary storage device serving as a buffer between the CPU and the printer and reader (Figure 3.1). The secondary storage device is a magnetic disk which can store data for an indefinite period. Its speed for data transfer (about one million characters per second) is a lot faster than either the card reader or printer. In the computer system as shown in Figure 3.1, a job on punched cards is first copied to the disk. It is made into a file of card images (called the INPUT file) which, practically speaking, behaves like a card reader, but can be read by the CPU at the higher speed of the disk. The dotted arrow in the figure shows that while the copying is taking place, the CPU is actually free to process some other job. Similarly, the line images generated by the CPU are recorded on the disk first, and made into the OUTPUT file. This file is transmitted later, without CPU action, to the line printer when the job completes. Such a process is used in many "batch-processing" systems, and is called SPOOLing (Simultaneous Peripheral Operations On-Line).

To further compensate the speed differences between I/O devices and the disk, additional readers and printers may be attached to the system. The Dual-Mace system supports many of these devices at eight different locations. It maintains a directory of all job files waiting to be processed (called the input queue), and of all OUTPUT files waiting to be printed (called the output queue). It also uses several disks to increase the storage capacity.

IV. Job Card Parameters

A large computing center processes thousands of jobs a day. These jobs have a variety of characteristics ranging from short, instructional types taking a few seconds, to long, research types taking several hours to process. The amount of printed output generated also varies significantly. A first-in, first-out processing schedule is undesirable since it would cause an unexpectedly long delay to short jobs which are submitted just after a long job. The spooling system used in Dual-Mace permits jobs to be executed in a different order from that in which they were submitted. This is accomplished by a program called the scheduler which ranks the jobs in the input queue according to the parameters indicated on the job card.

The sample program in Section II has only two fields on the job card: the account code, and the user identifier (ID). The account code used indicates that this is an instructional-type job. These jobs generally are assigned a lower priority than some research-type jobs. There is also a set of parameters associated with the account, which will be used as "default" values if they are not explicitly included on the job card. These parameters represent resources required during the processing of the job, such as central memory usage, processing time, printed lines, etc. For example, the job card

```
76543,UID,T16,L500,CM15000.
```

indicates that the job needs a maximum of 16 CPU seconds, prints a maximum of 500 lines, and needs 15000 (octal) words of memory initially. All other parameters are set by default. (A complete list of default values for an account may be generated by a PFILES(LIST) control card.) The scheduler ranks the jobs according to a "queue priority", which is computed as a function of these parameter values. It is generally true that the fewer resources a job needs, the higher its queue priority. The scheduler normally selects the highest priority job from the input queue for processing by the CPU. Thus it is strongly recommended that the parameters be set as close to the job's actual needs as possible.

Queue priorities are also "aged" as time passes; i.e., the queue priority of a job is increased periodically by some small amount. This ensures that jobs with identical parameters will be processed on a first-in, first-out basis. Aging also ensures that every job will eventually be run, no matter how low its original queue priority is.

V. Local Files

It is often necessary to store information on a temporary basis during the processing of a job. For example, the FORTRAN job of Section II is processed by the computer in two phases—a compilation phase, in which the FORTRAN source statements are translated into machine code, and an execution phase, in which the user's data are processed. The computer often does not have enough central memory to completely store the machine code during the compilation phase. The magnetic disk described in Section III can be used to store the generated machine code as a file. Thus the computer uses three files during FORTRAN compilation: it reads the source statements from the INPUT file, produces a listing on the OUTPUT file, and saves the machine code generated on a temporary file called LGO. At the end of the compilation phase, the information saved on LGO is brought back into the computer and execution begins. The files INPUT, OUTPUT, and LGO are called "local" files since they are closely identified with the job. They will not be confused with the identically named files belonging to another job in the system. The disk space occupied by local files is normally released at the end of the job.

A user may use additional local files whenever there is a need for them. Continuing with the FORTRAN example, a program may have input/output statements of the form

```
                READ(i,j) list of variables

or              WRITE(k,m) list of variables
```

The integers i and k in the above statements are called "unit" numbers, and j and m are format numbers. The convention is that units 5, 6, and 7 refer to the standard input, output, and punch files, respectively. A user may use any other integer (less than 100) to refer to a local file and perform the desired input/output operations on it. However, the Dual-Mace operating system requires that it be informed of all the local files to be used before the program begins execution. This is done using the special PROGRAM statement at the beginning of the FORTRAN program. A typical FORTRAN program uses the following PROGRAM statement:

```
      PROGRAM MAIN(INPUT,OUTPUT,PUNCH,TAPE5=INPUT,TAPE6=OUTPUT,
     $           TAPE7=PUNCH)
```

This statement, just as other FORTRAN statements, begins at column 7. The PROGRAM statement need not be included with the source deck if the user does not plan to access any units other than 5, 6, and 7. The system would supply the above PROGRAM statement by default.[1] The local file corresponding to unit i is called TAPEi by the Dual-Mace system. These names are aliases of the standard system input, output, and punch files. (It is of interest to note that files in earlier computer systems were stored on magnetic tapes and the use of TAPEi is a relic of those days.) If a certain user does not plan to punch any cards, but wishes to save something on unit 91, the following PROGRAM statement may be used:

```
      PROGRAM MAIN(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT,TAPE91)
```

Data may be placed on TAPE91 using the following statement:

```
      WRITE(91,format no.) list of variables
```

Since the local files are destroyed at the end of a job, special steps must be taken if the information saved is valuable. For example, assume the FORTRAN program is written to process a sequence of numbers which is the record of an experiment over a time period. Some of the numbers are quite different from the rest due to malfunctioning instruments. They are to be rejected using a certain criterion. The rejected numbers are saved on TAPE91 to be printed at the end of the regular program output. The following control card sequence would accomplish these functions:

[1]These remarks pertain to the RUN and FUN compilers on this system. The MNF and RUM compilers supply only the files INPUT and OUTPUT (TAPE5 and TAPE6) by default.

```
MNF.
REWIND(TAPE91)
COPYBF(TAPE91,OUTPUT)
#eor
      PROGRAM MAIN(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT,TAPE91)
         .
         .
   Fortran Program
         .
         .
#eor
         .
         .
      Data
         .
         .
#eoi
```

The control card MNF compiles the FORTRAN program. It also instructs the computer to "load" the file LGO for processing the data cards which follow. When the program writes on TAPE91, data is saved in a sequential fashion similar to the way it would be saved on an actual reel of tape. The control card REWIND effectively rewinds the file TAPE91, and the card COPYBF copies the entire contents of TAPE91, appending it to the OUTPUT file. As described previously in Section III, the desired results are printed after the job terminates.

VI. Permanent Files

The user may sometimes wish to save files on a more permanent basis. For example, the data recorded by the sample program of Section V on TAPE91 may be processed by several other programs later. It would be convenient to store a copy of it somewhere in the system for easy access. The Dual-Mace system has a special disk attached to the CPU as shown in Figure 6.1. The following control cards will create the file TAPE91 and place the data on the permanent file disk (called the PFILES disk):

```
MNF.
REWIND(TAPE91)
PFILES(PUT,TAPE91)
#eor
```

The PFILES control card is a command to the computer to transfer the contents of TAPE91 to the PFILES disk. A new file is established with the same name (TAPE91) under the user-identifier UID. This file is now in a different format to conserve disk space. This is accomplished by replacing repetitive characters in the file (e.g., trailing blanks) by a repeat count. The process is called "file compression". Files so stored are only semi-permanent in that they will be deleted from the system if not accessed within 15 days.

After the data is properly saved on the PFILES disk, the following control card sequence may be used to reference it:

```
PFILES(GET,TAPE91)
MNF(N)
LGO(TAPE91)
#eor
          .
          .
    Fortran program without PROGRAM statement or with a PROGRAM
    statement with INPUT as the first file
          .
          .
#eoi
```

The first control card locates the file TAPE91 on the PFILES disk and makes a local file copy which is also named TAPE91. The second control card tells the computer not to execute after compiling the FORTRAN program. The third control card causes the LGO file (containing the machine code) to be loaded with one "parameter substitution". That is, the parameter TAPE91 is used to replace the first parameter in the PROGRAM statement, which is INPUT. Thus all references by the program to the INPUT file are effectively replaced by references to the file TAPE91, and the program processes the previously stored data.

A file on the PFILES disk may have a name different from that of the local file. For example, the following control card may be used to save a prepunched data deck:

```
PFILES(PUT,DATA,X=INPUT)
#eor
       .
       .
    Data Cards
       .
       .
#eoi
```

Notice that the first file name is the name of the file on the PFILES disk, while the file name given by the X= parameter is the local file name. The same rules apply when GETting a file. Hence, the control card

```
PFILES(GET,MYDATA,X=MODEL)
```

will fetch a file named MYDATA from the PFILES disk,make a local copy of it, and name the local copy MODEL.

VII. Control Card Processing

The examples in the previous sections show that the computer understands commands such as MNF, REWIND, PFILES, etc. There are about two hundred different commands defined in the Dual-Mace system. A command like MNF is quite difficult to perform, as it involves the complicated compilation process. There are also commands that are much simpler, such as the one that tells the computer to stop (EXIT). A set of programs was written by the system programmers to handle these commands. They are stored on a special systems disk using the same names as the corresponding commands. For example, when the computer recognizes the control card MNF, a copy of the MNF program (the compiler) is retrieved from the disk and executed by the CPU. The next control card is treated the same way when MNF terminates, and this process continues until there are no more control cards in the first record of the job file. Thus the sample program of Section V requires the execution of three systems programs (MNF, REWIND, and PFILES). Note that parameters may be passed to these programs easily.

Consider the second sample program in Section VI, which is repeated here:

```
PFILES(GET,TAPE91)
MNF(N)
LGO(TAPE91)
#eor
    Fortran program
#eoi
```

The parameters passed to the PFILES program cause it to retrieve from the PFILES disk a file named TAPE91. A copy is placed in the local file disk. The N parameter to the MNF program stops the automatic loading of the translated machine code, which is kept on a local file LGO. But there is no system program whose name is LGO; the third control card simply loads and executes the translated machine code with one parameter substituted. Thus we can conclude that the first string of characters on a control card is actually the name of a file on the local file disk. It simply instructs the computer to load that file and execute it with the proper parameters. The "keyword" LGO is actually a default file name; the following control card sequence will produce identical results.

```
PFILES(GET,TAPE91)
MNF(N,B=ANALYZE)
ANALYZE(TAPE91)
#eor
    Fortran Program
#eoi
```

In the above program, the B= parameter tells the program MNF to place the machine code (Binary) on the file ANALYZE instead of the default LGO.

Suppose there is another file of test data saved on the PFILES disk under the name TAPE92. Additional control cards (shown following) enable the FORTRAN program to process both data files in one run.

```
PFILES(GET,TAPE91)
PFILES(GET,TAPE92)
MNF(N)
LGO(TAPE91)
LGO(TAPE92)
#eor
    Fortran program
#eoi
```

The control cards are interpreted exactly as before. The following sets of control cards are some variations that produce results identical with those of the preceding job. However, a job will usually be completed in the smallest amount of time if steps using the same amount of central memory (e.g., all of the PFILES operations) are grouped together. Hence, the original control card sequence is preferred.

```
MNF(N)                  PFILES(GET,TAPE91)
PFILES(GET,TAPE91)      MNF(N)
LGO(TAPE91)             LGO(TAPE91)
PFILES(GET,TAPE92)      PFILES(GET,TAPE92)
LGO(TAPE92)             LGO(TAPE92)
#eor                    #eor
```

The first record on a job file is always the control card record. The rest of the records on the INPUT file are used in order by the programs initiated by the control cards. The following example processes two data sets: one is stored in the PFILES disk and the other comes as punched cards with the job.

```
PFILES(GET,TAPE91)
MNF(N)
LGO(TAPE91)
LGO.
#eor
    Fortran program
#eor
    Second data set
#eoi
```

VIII. Library Programs

The computer has been used to solve scientific, engineering, and business problems for about 30 years. Experience has shown that many of the tasks performed by the computer are routine. The most frequently performed tasks have been standardized, coded into library programs by professional programmers and made available for general use. These programs have been extensively tested and are continuously maintained. A user may save a lot of effort in a programming project if he knows how to use the library programs properly.

Most high-level programming languages have a library of "built-in" functions. They often represent the most basic arithmetic and trigonometric functions (such as absolute value, sine, cosine, etc.) and can be referenced by the user in a program. These functions are automatically included during the "load" phase when the translated machine code (called "object" code) is placed in the central memory. When the user wishes to include some more sophisticated library program (such as a sorting procedure), additional steps must be taken.

The Dual-Mace system maintains several distinct libraries in object code form. They are kept as files on the local file disk and are identified as SYSLIB, RUNLIB, PASCLIB, etc. The appropriate name(s) must be given to the loader program before the desired library program can be included. For example, there is a FORTRAN subroutine SORT(A,N) in RUNLIB that sorts an array A of N elements. The following control card sequence may be used to access the library program:

```
MNF(N)
LOAD(LGO,MNFLIB,RUNLIB)
EXECUTE.
#eor
      Fortran program with statements like
      CALL SORT(ARRAY,NUMBER)
#eor
      Data
#eoi
```

Note that LOAD is the name of the loader program. The parameters instruct the loader to first enter the translated machine code of the source program into central memory. Since the program is written for the MNF compiler, it may require the loading of some built-in functions from the library for the compiler, MNFLIB. Thus any needed programs are taken out of MNFLIB first. The loader then tries to find SORT on MNFLIB; since it is not there, the loader next looks for SORT on RUNLIB, finds it, and completes the loading process. The control card EXECUTE starts execution of the user program. Since it is so simple, the LOAD-EXECUTE control card sequence may be replaced by

```
LOADX(LGO,MNFLIB,RUNLIB)
```

The computing center keeps the information on library programs and other documents in room B4 of the Math Building. An excellent set of FORTRAN callable mathematical and statistical routines are maintained in RUNLIB. It is frequently worthwhile to spend some time looking at the library programs before one starts programming.

IX. Multiprogramming

The main computing facility at the Purdue computing center has four CPU's, more than 200,000 words of central memory, eleven disk systems, ten tape drives, four line printers, two card readers, one card punch, and various other peripheral devices connected to the three front-end minicomputers. It is very unlikely that any one job needs to use all this equipment at the same time. To improve the utilization of these expensive resources, the computer is operated in the "multiprogramming" mode. That is, several jobs can be placed in central memory and processed at the same time. It is the responsibility of the scheduler in the Dual-Mace operating system to select the jobs that can best share the resources while also meeting other performance goals. However, certain performance goals may be in conflict with the ideal of best sharing the system's resources. For example, one of the goals of the computing center is for greater user satisfaction through shorter average turn-around time. It is known that shorter average turn-around time would result if the scheduler always attempted to process the jobs requiring the shortest overall processing time first. However, these jobs normally just need a CPU and a small amount of memory, and do not use other devices such as the tape drives. Running them together would leave the resources other than the CPU and memory idle, thus decreasing the total system utilization. Therefore the scheduler should try to make the best selection of jobs while considering all the individual demands.

The scheduler in the Dual-Mace system proceeds by first dividing the jobs that are ready to run into several categories according to their resource requirements. A queue priority is computed for the jobs in each category based on their actual resource requirements and the time they are submitted. A limit is established for each category as the maximum number of jobs in that category that can be processed by the computer at the same time. Thus there is some control over te mixture of jobs that are actually placed in memory. The terminal command or control card CHECKQ displays the current definitions of these categories and the number of jobs queued in each category.

The resource requirements of a job may change during the course of its execution. For example, a job may begin with some PFILES operations taking 15000 words of memory. It may then ask for 50000 words for compilation. It only needs 35000 words for the last step of execution. After the PFILES operations are completed, there may not be enough free memory available to perform the compilation step. The processing of the job is temporarily suspended and it is placed in the queue for its category again. The job is then considered "rolled-out" of memory because it returns the 15000 words of memory it has been using to the system. To avoid unnecessary roll-outs, it is advisable to group job steps that require the same amount of central memory together if at all possible. There is normally no roll-out during a central memory reduction (e.g., between the compilation and execution steps). The CM parameter on the job card should accurately reflect the initial memory requirement; i.e., the memory required by the first job step.

X. Terminals

The Dual-Mace system supports nearly two hundred typewriter-like terminals on campus. These are input/output devices which operate at slower speeds than card readers and line printers (typically 30-960 characters per second), but give the user the advantage of being able to interact directly with the computer. Since the terminals are relatively slow, they are connected to the CDC 6500 system through one of several MODCOMP minicomputers, sometimes called "front end" or "buffer" machines. The MODCOMPs collect characters typed at the terminals (Figure 10.1), feeding them to the main computer whenever a complete command has been typed, and also accept return messages from the main computer which are sent to the terminals at an appropriate speed. In effect, the MODCOMPs take over the time-consuming chores of accepting input and dispensing output, leaving the main computer free to concentrate on the more important arithmetic processing work. (Compare this idea with that of spooling in Section III.)

A user at a terminal may create, edit, and delete files, submit jobs to the system, and retrieve results of submitted jobs. These things are all done by means of various terminal system commands. The commands will be illustrated by tracing the progress of two typical terminal sessions. Everything typed by the user will be underlined. Each user-entered line is ended by hitting the 'return' key, although that will not be explicitly shown.

A terminal can operate in either of two modes. In local mode, it acts just like a typewriter. In on-line mode, everything typed at the terminal is sent to the computer. For our purposes, the terminal should have its mode switch set to on-line. The next thing a user does at a terminal is to "log on" to the system. This is done by typing a Control-B (holding down the Control button and typing a 'B'). The computer will ask for an account number, user ID, and password, which should be supplied by the user. These are checked against a master list for validity; if they are authenticated, the computer may print one or more items of current interest concerning the computing center, and then ask the user which system he prefers to work with. Most users choose PIRATE, which accepts the commands discussed in this section. The computer will then respond with a sequence of three plusses. This is the PIRATE prompt sequence; whenever it appears, it means that the computer is waiting for the user to enter a command.

```
(Ctrl-B)
TCB L201  11.35.12  11/07/78.  FULL DUPLEX
ACCOUNT? 76543,ABC
PASSWORD? MYPASS
THE COMPUTING CENTER WILL CLOSE AT 22:00 TONIGHT
SYSTEM? PIRATE
+++
```

Log-On Sequence

At this point, a beginning user will normally want to create a file containing a program, data, or other information. Files manipulated at a terminal are called terminal files. In the same way that a local file is associated with a particular job, a terminal file is associated with a logged-on terminal. It exists for the duration of the terminal session, and then disappears. Terminal files also behave like local files in that it is possible to GET a PFILES file and make it a terminal file, or PUT a terminal file into PFILES. However, local files and terminal files are distinct; normally, a running job cannot directly access a terminal file and a terminal user cannot access a file local to an executing job. Terminal files also contain imbedded line numbers and pointers to aid in the editing process, and so have a different format than either local files or PFILES. One possible point of confusion is that terminal files are sometimes referred to as "local files". As this is something of a misnomer, this document will always use the term "terminal files".

To create a terminal file, the user types the word CREATE followed by the name he wishes the file to have. The computer will respond with a line number followed by an equal sign. At this point the user should type in the contents of the file. (The computer will prompt with the next line number after each line.) When the entire contents of the file have been entered, type #S to return to PIRATE.

```
+++CREATE LIGHT
   1.000=76543,ABC.
   2.000=MNF.
   3.000=#EOR
   4.000=      WRITE(6,100)
   5.000= 100     FORMAT(1X,'SAMPLE PROGRAM')
   6.000=      STOP
   7.000=      END
   8.000=#S
+++
```

Creating a File

Notice that since there is no way to multipunch on a terminal, the end-of-record mark in line 3 is typed as #EOR. There is no special sequence of symbols for end-of-information; an EOI mark is automatically appended to any file created at a terminal. Note further that the created file contains both a FORTRAN program and the control cards to compile and run it, just as if it were typed at a card punch. It is important to understand the distinction between PIRATE commands and Dual-Mace control cards: the former are typed following a '+++' prompt, and will produce a response of some sort at the terminal; the latter are typed in only as parts of files, and have no immediate effect. Any file which is intended to be used as a job file must contain Dual-Mace control cards as its first record, just as it would if it were a card deck.

The file created in the sequence above is an example of a job file. It contains a set of control cards and a FORTRAN program (and could contain data if it were needed) exactly like any of the card decks shown earlier in this document. (The only difference is that a terminal job file need not contain a password card since the computer already knows that you are an authorized user because you are logged-on. However, the account number and ID in the job file must match those under which you are logged-on.) In order to submit a job file to the system for execution, the user issues an XMIT command. The parameters it takes are the name the user wishes to assign to the job (any one to four letter combination) and the name of the job file.

```
+++XMIT RED LIGHT
  JOB "RED" SENT
+++
```

Submitting a Job

When a user XMITs a job, a copy of the job file is placed in the queue of jobs waiting to be executed by the system. This is the same queue that jobs which were read in via the card reader join. (See Figure 10.2). At this point, the job submitted by a terminal user behaves like a job submitted by a batch user, with one important difference. For terminal-submitted jobs, any output produced is stored on a disk instead of being sent directly to a line printer. Thus the terminal user may examine the output from his job at the terminal before deciding whether or not to print it. If a printed copy is desired, the user may issue commands to place the output from the job into the line printer queue. Notice from Figure 10.2 that it is also possible for the user to change his mind and remove job output from the printer queue.

There are several ways for a terminal user to watch his job's progress through the system. One is by means of the SEARCH command. When a SEARCH command is issued, the computer reports on the current status of all the user's jobs in the system.

```
+++SEARCH
  NAME    QP    TYPE    MESSAGE         11/08  11:41
 ABCRED  3755   RO 1    MNF.
+++SEARCH
  NAME    QP    TYPE    MESSAGE         11/08  11:42
 ABCRED  3777   PR
+++
```

Performing a Search

The QP number is the queue priority discussed in Section IX. The job type will be one of the following: IN (the job is waiting to enter the system), EX (the job is executing), RO (the job is rolled-out of memory), or PR (the job has completed execution and is ready to be inspected or printed). A number following the type indicates the job's size category (again, see Section IX). The message normally consists of the control card that the job is currently trying to execute. Hence, in the first SEARCH above, the job RED had begun execution and was waiting to execute the MNF control card as a size 1 job, while in the second SEARCH the job had completed execution.

A second way to trace a job's progress is with the PREVIEW command. This command is used to trace the progress of a particular job (e.g., PREVIEW RED) by printing information about the job (similar to that given by SEARCH) periodically until the job completes. At this point, however, PREVIEW retrieves the job output from the disk file and displays it at the terminal. First the job's dayfile (control card information) is displayed, followed by all of the job's output. Certain options can be appended to the PREVIEW command to request that only certain portions of the job be displayed: D (display dayfile only), S (summarize dayfile), O (output only), or Ri (i-th output record only; R0 is the first record).

```
+++PREVIEW RED
"RED" DAYFILE:
           .
           .   dayfile listed here
           .

"RED" OUTPUT:
        .
        .   program listed here
        .
        #EOR
        .
        .   program output listed here
        .
        #EOR
+++PREVIEW RED R1

"RED" R1:
        .
        .   program output listed here
        .
        #EOR
+++
```

Previewing a Job

We mentioned earlier that it is necessary to issue a specific command to produce printed output from a terminal-submitted job. That command is 'ROUTE name AS nnn AT site', where name is the job name, nnn is the output bin number, and site tells where the job should be printed (the default site is the Math-Science building).

```
+++ROUTE RED AS 409
  ABCRED  =  ABC0409 AT MATH
+++
```

Routing a Job

If a terminal-submitted job is not routed within a short time (typically thirty minutes), the system removes the job output from the disk so that the disk does not become overcrowded. The user can expedite this process by removing unwanted output files from the disk using the PURGE command.

```
+++PURGE RED
  PURGED RED/PR
+++
```

Purging an Output File

It was earlier mentioned that terminals may interact with PFILES. If the user now wished to save the current copy of his job file, he could do so by issuing a PUT command.

```
+++PUT LIGHT
  LIGHT           16 WORDS
+++
```

Saving a File

Note that the user must not issue a PFILES,PUT command to save a file. PUT will remove all the imbedded pointers and line numbers in the file before storing it away, while PFILES,PUT does not.

This initial terminal session may be brought to an end by issuing a LOG command. This signals the system to terminate the session and delete any terminal files associated with the user.

```
+++LOG
TCB L201  11.59.26.  11/07/78.
ESTIMATED SESSION COST $   .16
PLEASE TURN OFF TERMINAL.  TNX.
```

Terminating a Session

Now suppose the same user begins a new terminal session. After completing the log-on sequence, he decides to retrieve from PFILES the job file he was working on last time in order to make some changes in it. He first GETs the file, and then issues an EDIT command. The system editor returns a '#' as a prompt character. The editor is a powerful and versatile tool for locating, printing, inserting, deleting, moving, and changing lines within a file. The many editor commands will not be described here; the interested reader is referred to the documents listed in the references.

```
+++GET LIGHT
  LIGHT          16 WORDS
+++EDIT LIGHT
#    .
         .   editing commands
         .
#S
+++
```

Retrieving and Editing a File

Notice that the last editing command used is the 'S' (stop) command, and that this is the same way that the CREATE command was exited. In fact, the CREATE command also gives the user access to the editor. The only difference between CREATE and EDIT is that the former builds a new file, while the latter is used to work on an already existing file.

Having the use of an editor makes it possible to inspect output from completed jobs in yet another way. The QAC (access) command retrieves a copy of a job's output and makes it available as a terminal file under the job name. If the SA (save) parameter is used, a copy is still available for later routing from the output disk. Once the output has been retrieved as a terminal file, the user can EDIT the file and use the editor to display selected portions of the output. Note that the file being EDITed is the _output_ file from a job, _not_ the job file itself. The user will normally discover his errors by inspecting this output file, but must then EDIT the actual job file (in our case, LIGHT) in order to make corrections. Using QAC and EDIT is generally preferable to using PREVIEW unless the amount of output generated is very small or only the dayfile is being examined.

```
+++XMIT FIAT LIGHT
  JOB "FIAT" SENT
+++QAC FIAT SA
+++EDIT FIAT
#    .
     .   editing commands to display output
     .
```

Examining Job Output

Occasionally, the user may wish to examine the contents of a terminal file without entering the editor. The DISPLAY command may be used to list any terminal file. (The SUP parameter suppresses the listing of line numbers.)

```
+++DISPLAY LIGHT SUP
76543,ABC.
MNF.
  .
  .
  .
         END
+++
```

Displaying a File

The commands described in this section constitute only a small part of those available on the system, and many additional parameters are available for the more sophisticated user. It should be emphasized that the commands listed here are meant solely as an introduction to the power of PIRATE; the interested user should consult the documents listed in the references for further details.

References

The discussions in this document sometimes oversimplify the true structure of the Dual-Mace system. The serious reader is referred to the following Computing Center documents for additional information.

Sections I - III

ZO-MANUAL

Section IV

ZO-PUCCPG
ZO-SCHEDUL

Section V

LO-CDCMACE

Section VI

QO-PFILES

Section VII

LO-CONTROL

Section VIII

VO-RUNLIB

Section IX

ZO-PUCCEQP

Section X

LO-QED
LO-PIRATE
LO-PROCSY

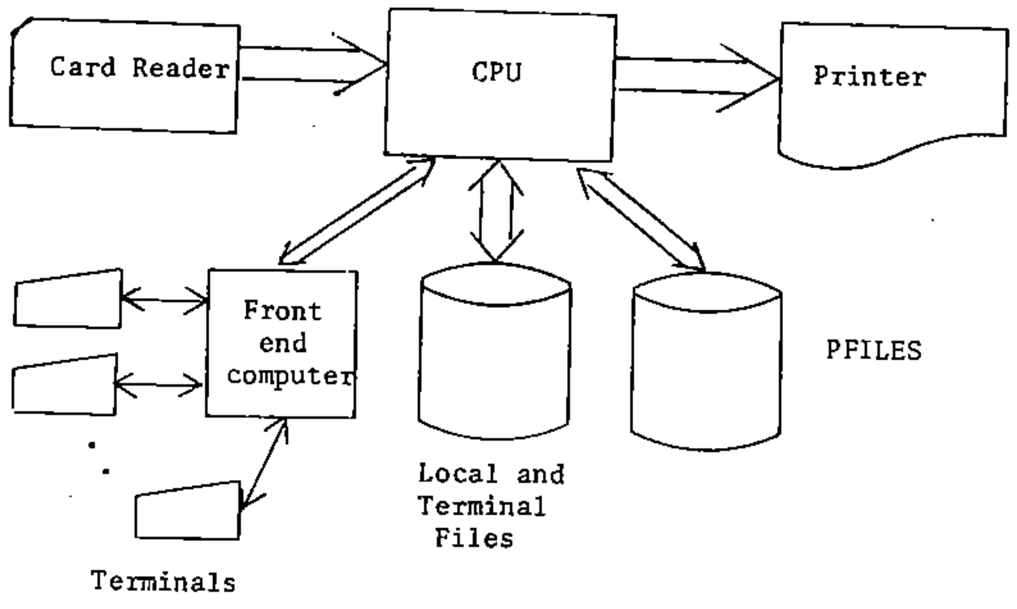

Figure 10.1 Organization with the Terminal System.

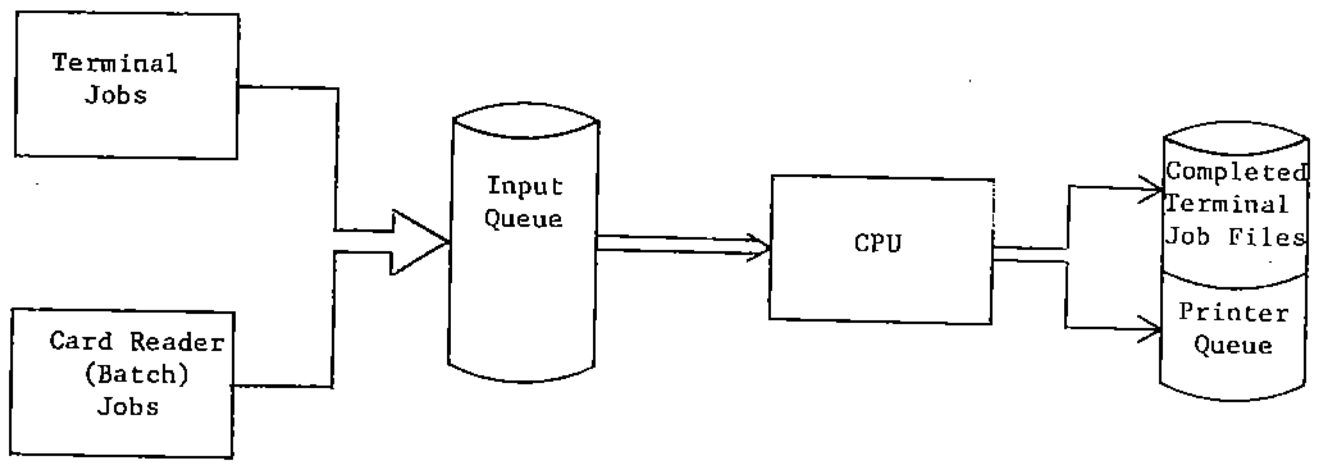

Figure 10.2 Disposition of Output

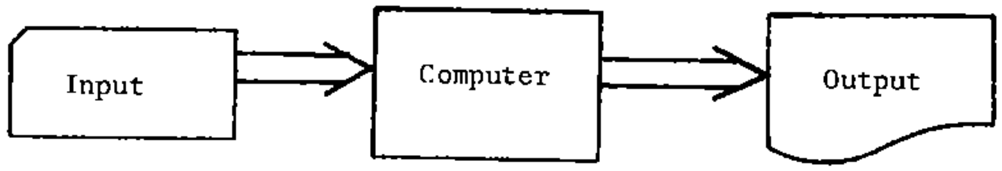

Figure 2.1. The Basic Computer.

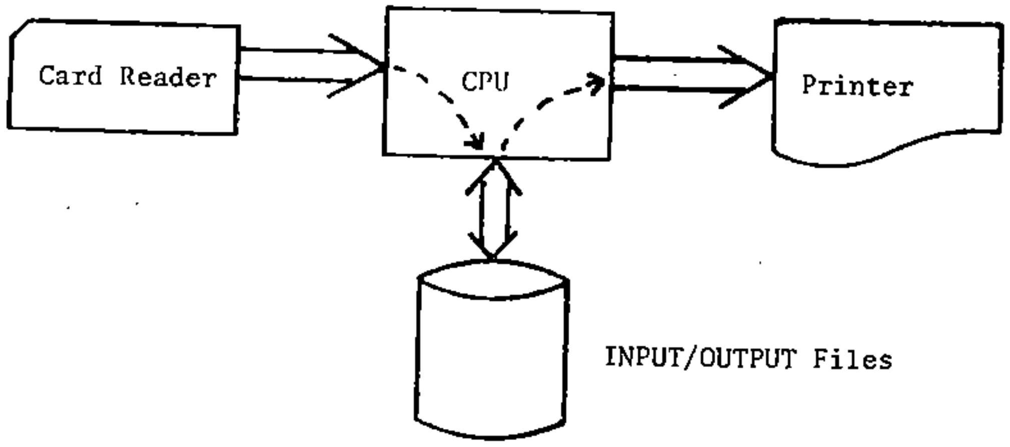

Figure 3.1. Organization for Spooling.

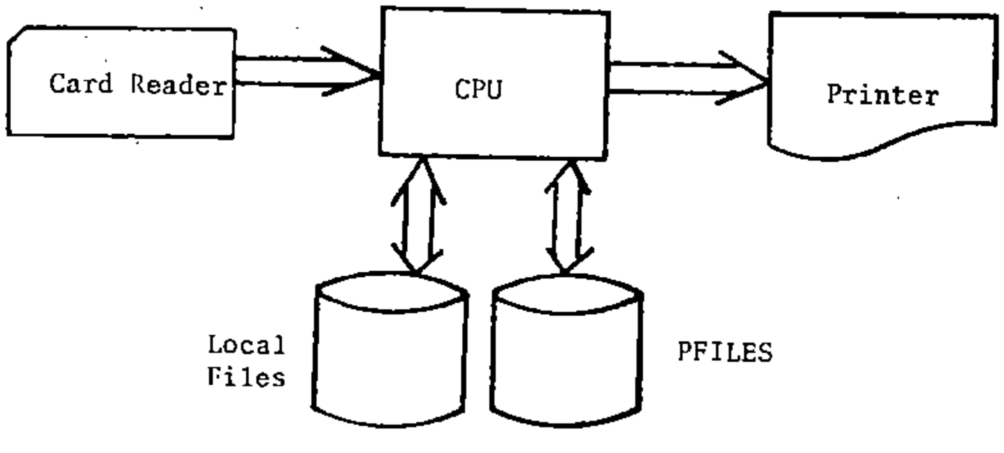

Figure 6.1. The PFILES Disk.