Z77-8153

TECHNICAL INFORMATION EXCHANGE

IBM®

December 19, 1968

OS/MVT PRIMER

A discussion of the facilities of OS/MVT and some techniques of correctly using them. The paper is current with OS/360 Release 15/16.

Mr. Philip P. Grannan
IBM Corporation
250 State Street
Rochester, New York 14614

IBM Corporation, Technical Publications Dept., 112 E. Post Road, White Plains, N.Y. 10601

Z77-8153

AUTHOR: Philip P. Grannan

DATE: December 5, 1968

TITLE: OS/MVT Primer

ADDRESS: IBM Corporation
250 State Street
Rochester, New York - 14614

* * *

There are three programming subsystems available under the full Operating System (OS/360): The Primary Control Program (PCP); Multiprogramming with a Fixed Number of Tasks - Version II (MFT-II); and Multiprogramming with a Variable Number of Tasks (MVT).

This paper is intended to provide prospective users of OS/MVT with an overview of the facilities available with that option. Also, while the paper is devoted primarily to MVT, it should be noted that the other options have many of the same facilities and thus this paper could be used as a partial reference for them.

Currently, the information contained in this paper is distributed throughout several manuals and is not available to the user in one source. The idea of this paper is to present this information in a form that is easily accessible to the user.

OS/MVT is the largest IBM operating system and contains many facilities giving flexibility and power to the user. Let us now look at these facilities and some techniques of using them.

I. USER SOURCE AND OBJECT LIBRARIES - To cut down on unnecessary card handling and to speed throughput, the user can place source programs and object programs in direct access libraries. An IBM supplied utility (IEBUPDTE) can be used to delete or insert statements, renumber the program, etc. The updating procedure is not difficult for the programmer to use.

   Anyone who has handled large card decks, or who has dropped one will see the worth of this facility. Further, it can speed throughput as the card reader is used much less.

II. LOAD MODULE LIBRARIES - The storage on direct access storage devices (DASD) of load modules in LINKLIB or private JOBLIBS is a tremendous step in throughput improvement. This facility

II. LOAD MODULE LIBRARIES - continued -

allows the user to do away with voluminous card handling and card reading and with unnecessary compile and link editing steps.

III. PROCEDURE LIBRARY - The job control language (JCL) associated with running a job can be stored on a DASD in SYS1.PROCLIB, thus enabling the user to cut down on card reading and handling and at the same time establish standardized compile procedures, reader procedures, etc.

The members of PROCLIB can be easily modified by using an IBM supplied system utility (IEBUPDTE). In fact, the JCL for using this utility can reside in PROCLIB.

IV. CORE USAGE - MVT gives us the ability to vary the amount of core that each program step uses leaving the rest for other programs, system tasks, etc. When initiated, the program is placed in as much core space as it specifies in the REGION = parameter of the JOB or EXEC card (there is a minimum size region determined by initiator/terminator considerations). Furthermore, during execution, the program can change dynamically the size of its region. Additional modules can be brought into core using overlay structures or dynamic structuring. MVT allows us to utilize core in a very efficient manner.

V. TIME SLICING - The use of the time-slicing facilities allows us to keep any one program from controlling and using the CPU to the detriment of other jobs in the computer. The time-slicing facility allows the user to set a processing time limit on jobs in specific user determined priority groups (priorities will be discussed more fully later). When the limit is exhausted, the job must give up control of the CPU to the first ready job in that priority group in the CPU. This does not stop a higher priority job from taking control whenever it is posted ready. It also does not allow a lower priority job to gain control of the CPU from the time-sliced job. If there is no other equal priority job ready and no higher priority job becomes ready, the original time-sliced job continues processing for another set time interval. Further, using the CHAP macro instruction to change priorities, a job can be moved out of or into a time-sliced priority during a phase of its execution.

VI. CHANNEL PROGRAMMING - The use of channel programming (the channel is a physical interface between the CPU and its I/O equipment) allows us to perform input-output operations at

VI. CHANNEL PROGRAMMING - continued -

at the same time the CPU is processing something else. Further, by adding command chaining, also known as chain scheduling, to our reader and writer procedures, we can reduce the amount of IOS (Input/Output Supervisor) time used by I/O operations. This can result in a saving of CPU time and improve I/O throughput.

VII. REENTRANT CODING - Reentrant coded modules allow us to keep a single copy of a program in core, while that copy processes multiple sets of information.

By definition, reentrant code (automatically generated by the PL/1 compiler) does not alter itself during execution and thus the same copy can be used over and over and/or concurrently.

Let us take the example of an inquiry type of program used by a teleprocessing control program which brings in the inquiry program for certain requests.

Multiple inquiries can be processed simultaneously by this single copy of the program. The requests treat the inquiry program as if each request had its own copy in core.

The core and time saving due to this facility are very real and important. Compare the above method with having to actually bring in and use multiple copies of the inquiry program or, having core limitations, being forced to process the requests in a sequential manner.

VIII. ATTACH - The ATTACH macro instruction allows one task to create another task which can now compete for control independently with all other tasks in the system. This facility allows the programmer, through his program, to actually create new tasks in his region without the use of the initiator. This is a very important but as yet little used (by programmer) ability of OS/MVT which gives tremendous flexibility to the programmer.

IX. ROLLOUT/ROLLIN - This is a method whereby a program attempting to dynamically gain additional core and finding none free can cause another program to halt execution and be placed temporarily in a DASD. When this program frees the extra core, the rolled out program is returned and resumes execution. This facility must be carefully regulated by programming standards to prevent indiscriminate use.

X. JOB QUEUE - This facility allows us to build a list of jobs on a DASD or in core (the programs are kept on a DASD) so that the operating system, using a priority scheduler, can select jobs for execution. This allows us to perform batch processing and multiprogramming. The benefit using the option is increased execution speed -- sequential schedulers are tied to the speed of the job stream input device which is usually a card reader. Using multiprogramming, we can be placing jobs on the job queue as well as performing other work in the computer concurrently.

XI. STORAGE PROTECTION - Storage protection is actually a combination hardware and software feature of OS/360. It prevents one task from altering the region belonging to another problem program or the supervisor. It should be noted that ATTACHED programs have the same storage protection key as the ATTACHing program.

XII. JOB CLASS - Categories of jobs can be determined by the system analyst and each category can be assigned a job class name (up to 15 different class names). Then certain class types can be assigned to specific INITIATORS (up to eight classes per initiator) in order to control the job in the CPU. An example of this would be to assign all CPU bound jobs to class A and all I/O bound jobs to class B. Starting two initiators, one to class A and one to class B would insure that a Class A and a Class B job would share the CPU.

In the actual production environment, this job class breakdown would be much more extensive, and also very important as we must optimize the job mix in the CPU to fully utilize the CPU and its I/O equipment.

It should be noted that the initiator is assigned priorities for each class associated with it, while priorities within class is the scheduling priority discussed later. Thus an initiator associated with classes A, B and C would initiate all class A jobs, first, according to their scheduling priority, then class B Jobs and finally all class C jobs.

XIII. JOB PRIORITY - Job priority is another important facility of MVT. Scheduling priority determines position on the job queue within job class (assigned by PRTY = , default to reader procedure or can be modified from the console). Once selected, dispatching priority (initially based on scheduling priority) determines the job's priority in the system.

XIII. JOB PRIORITY - continued -

Furthermore, by use of the CHAP macro instruction, dispatching priorities can be altered by the program itself. Thus a program can CHAP itself a higher priority (upper limit is its initial dispatching or limit priority) for an I/O portion and then CHAP down for a long computational (CPU bound) portion.

Also, ATTACHed tasks can vary their priority in their region. The initial limit and dispatching priorities can be established in the ATTACH marco or default to the limit and dispatching priorities (at ATTACH macro execution) of the ATTACHing program. By careful use of priorities and the CHAP macro, the system can be made to run more efficiently and more effectively.

Used with the CLASS facility, the PRIORITY facility can be used to insure both proper job mix and proper execution priorities.

XIV. ENQ-DEQ - This facility helps guarantee data set integrity. At job initiation, the initiator ENQ's upon each data set name used in the program and blocks out all other programs for the duration of the job if DISP=OLD is coded. DISP=SHR can be used on read-only jobs and will allow other programs with DISP=SHR to get at the data set. If, however, the ENQ is issued, other programs with DISP=SHR or DISP=OLD cannot be initiated until the terminator DEQ's on the data set names.

The purpose of this facility is to allow a data set to be used by only one program at a time if the data set is to be modified but to allow several programs to use an unmodified resource concurrently. Inadvertent use of DISP=OLD in a read only job can block out initiation (and tie up an initiator) of another read-only job. Of course, using DISP=SHR incorrectly can also cause problems.

ENQ and DEQ are also macro instruction that can be issued from an executing problem program. This macro facility allows ENQing on any system resource (program, data set, record, track, etc.).

In the case of a mother task (a "mother task" "Attaches" "Daughter" tasks) which could possibly tie up many resources with a DISP=OLD, the use of the macros should be investigated. In fact, most long running programs should use DISP=SHR and the ENQ-DEQ macros. Again, this is an area for system programmer study.

Incorrect use of ENQ-DEQ can often cause subtle problems. Therefore, a well defined set of standards is necessary if we are to keep all initiators running and ENQ only on specific resources for the time that we actually need them.

XV. CORE RESIDENT FUNCTIONS - This is a facility which allows the user to decrease or eliminate DASD access time required to obtain records and/or routines commonly used during operation. These functions can be load modules of non-resident SVC routines, other reentrant load modules from LINKLIB and SVCLIB, and a table (BLDL table) containing directory entries of load modules in LINKLIB. These functions, specified at SYSGEN time, can be respecified at IPL time. They are kept in an area of core storage called the LINK PACK AREA (LPA).

This facility allows us to save access time on frequently used functions. Of course, we are trading core storage (for the LPA) for this time saving. Therefore, it is important that the system programmer review the contents of his LPA in relation to the job stream his installation is running. Consideration should be given to varying the contents of the LPA by Re-IPLing for specific portions of the job stream to speed throughput.

XI. BLOCKING - The blocking of input and output data sets is vital for efficient machine utilization. Blocking has a triple effect:

It speeds up the processing problem by reducing contention on DASD; less IOS time is required for large input and output block sizes; and with blocking more records can be presented to the processing program faster.

As before, we have the core storage versus speed conflict where we trade core storage for processing speed. Further, other factors such as chain scheduling (OPTCD=C) and number of buffers (BUFNO=) combine to make the decision quite complex.

Also, different processors have varying maximum block sizes which they can accept. This makes it impossible to have a standard reader that blocks IEFDATA (disk resident input data set created by the reader) for all jobs to a standard block sizes. Release 16 allows blocking as a job stream to be controlled by the SYSIN DD card. For example:

```
// SYSIN DD *, DCB = (LRECL = 8Ø, RECFM = FB,
                                  BLKSIZE = 16ØØ)
```

SYSPRINT can be blocked (to different degrees) for compilers and the LINKAGE EDITOR. Specify the blocking parameters in the SYSPRINT DD card. For example:

```
// SYSPRINT DD   SYSOUT = A, DCB = (LRECL = 121,
                                    RECFM = FBA, BLKSIZE = 3Ø25)
```

XI. BLOCKING - continued -

SYSLIN and LINKAGE EDITOR output can also be blocked within certain limits.

What makes this so important is that blocking can have a tremendous effect on throughput, perhaps the greatest effect on throughput other than multiprogramming. Therefore, programming standards must specify blocking as determined by the system programmer weighing the combined effects of core size, speed, buffer number, chain scheduling and arm contention on DASD.

XVII. DATA SET ALLOCATION - Data set allocation is especially important in MVT since it uses DASD input and output, thus creating additional data sets requiring servicing by the system. Contention can occur between reader and writer data sets, system data sets, scratch data sets and user data sets.

As for the reader and writer data sets, generic names specified at SYSGEN time and used in the RDR and WTR procedures can place these data sets on separate sets of volumes.

There should be a minimum of contention between SYS1.SYSJOBQE, SYS1.SVCLIB, SYS1.LINKLIB and between these and other data sets.

User data sets should also be allocated with discretion so as to avoid causing contention. Thought should be put into providing multiple DASD packs for programmer use and careful planning and scrutiny should be used in placing user data sets.

The use of split cylinders can reduce conflict caused by scratch data sets.

In installations where there are different types of DASD (2301, 2314, 2311), frequently used files or frequently used portions should be on the faster device. An example might be an ISAM file with just its indices on a drum (2301).

XVIII. MULTIPLE INITIATORS - The use of multiple initiators can insure that an effective combination of jobs is run together to fully utilize the system. Thus the number and class designations for initiators is determined by such CPU resources as CPU time, I/O devices and available core.

Since access to the job queue is sequential, a job stream consisting of short running jobs benefits little from multiple initiators. With longer running jobs, multiple initiators can be used to balance CPU resource use.