IBM OS/360:
An Overview of the
First General Purpose Mainframe

CS 550-2: Operating Systems

Fall 2003

E. Casey Lunny

Table of Contents

Introduction

OS/360 was born in an era when systems were designed for a specific environment. The 360 system, however, was the first system capable of supporting both commercial and scientific applications, previously separated to different lines of hardware. It was also the first multiprogramming operating system. Thus, it was the first general purpose mainframe computer.

This operating system was very popular and was financially successful leading to several other lines of operating systems. In a humorous “obituary” written in response to the announcement that production had stopped on the system, its complexity and problems were mentioned but the writer pays tribute by stating “It will be mourned by its many friends” (Jardin, 1972)

OS/360 was developed in a world where different computing environments required different lines of hardware. IBM had the vision to merge these separate lines into one product and developed a new way of thinking about the commonalities amongst processes they were previously thought to irreconcilable.

This paper aims to demonstrate the role that OS/360 played in the development of modern computing. The topics covered here should give the reader an understanding of the operating system's place in history. The reader will gain an understanding of how the system provided enhanced services to its users, as well as the limitations faced by these users.

“A Second Generation OS”

The IBM OS/360 was launched at the end of 1965. 21 releases later, its retirement was announced on August 2, 1972. OS/360, designed to run on IBM's System/360 hardware. The hardware was originally intended to run three separate operating systems: DOS/360 for small machines, OS/360 for mid-range environments, and TSS/360 for high-end, multi-user time-sharing installations. The latter was so wrought with problems, it was never widely used and many users implemented OS/360. (Paulson, 2001)

The name “OS/360” refers to IBM's intention that this system be able to meet all of the needs of its users, “encompassing customer needs” 360 degrees. In hindsight, of course, this was not possible and the next implementations of the systems were known as the 370 and 390, which corresponded to the decades of their release.

IBM saw OS/360 as being what it called a “second generation” operating

system. It described so-called First Generation operating systems as batch-job operating systems where “each job has, more of less, the entire machine to itself” (Mealy, 1966) A separate line of real-time machines was developed. Such computers dedicated all of the resources of the machine to meet the needs of one “real-time” application. “By and large, these real-time systems bore little resemblance to the first generation of operating systems, either from the point of view of intended application or system structure.” (Mealy, 196) The types of systems described as “real time” are not the same as our modern notion of the term. These systems were along the lines of *Sabre*, systems that provided terminal access for users of the system, such as users who needed “instant” access to inventory, or airline reservations. If a travel agent needed to check if a flight was available, he couldn't very well run a batch job and wait around for the results.

The goals of this new generation of operating system was “to accommodate an environment of diverse applications and operating modes” as its primary objective. It would enable flexibility of use. The installation could run large, traditional batch jobs as well as jobs whose function was to allow terminals access to a data set. Secondary objectives included increased throughput, lowered response time, increased programmer productivity, adaptability and expandability.

Secondary Design Goals

Throughput was increased by letting operators setup for jobs that are waiting in the queue. It permitted concurrent use of system resources. **Programmer productivity** was increased by support of a large number of source languages.

As this system sought to meet the needs of both batch job environments and real time environments, **response time** was not defined as a simple element. Times ranged wildly between different types of systems. In a batch environment turn around time was defined as the time between when a user handed over a deck of punched cards to a computer operator and the time the printout was ready to be picked up. At the other end of the spectrum, in a real-time environment, responses were measured down to the millisecond. Priority options and control program options allowed shops to customize the system as needed.

The machine was highly configurable, both in the software and hardware spheres, both at initial installation and post-installation. To enable the flexibility and **adaptability**, programs could be written to be device independent. The system enabled shops to reduce the issues previously associated with device changes, where entire sections of code had to be re-written.

The system had a higher degree of **expandability** that previously

implemented, with the ability to add new hardware, applications and programs. IBM had an eye towards the future and ensure that its system really could encompass its users' needs 360º. (Mealy, 1966)

Processor Modes

Two processor modes are employed - the **supervisor state** and the **problem state**. The supervisor state is equivalent to “kernel mode” and the problem state is equivalent to the user mode. All programs other than those called by the operating system itself run in the problem state. The system can enter supervisor state by use of an SVC, the supervisor call instruction. There is no “superuser” equivalent in this operating system.

Users of this system interacted with the system mainly through use of computer operators who read jobs in through punch cards or other media, and output, in the form of printed reports, was printed by the computer operator and picked up later by the user.

The operating system was expanded in 1968 to include support for multiple processors where symmetric multiprocessing model was implemented. The processors would share all other components and be under the control of a single OS. Jobs in the ready queue would be doled out to the next available processor.

Jobs and Tasks

The OS/360 was a step up from strict batch-mode operating systems. With a more robust notion of what a “job” is, the OS is able to advance from batch processing to something that begins to resemble a multi-threaded system.

“Jobs” in a batch-oriented system are loaded onto the system and given use of nearly the entire system. There are no other jobs to content with and it has all of the resources available on the system until the current job has completed. The OS/360 uses a more complex definition of job, described below. The seeds here are sown for a multi-threaded system.

A **job stream** is the programming code as it is read into the computer. A job stream “defines and characterizes jobs” and may designate many jobs.

Job steps can be created by the programmer by using control statements to

divide a job. A job step is not necessarily sequential and may be conditional upon completion of previous steps. Only one step of a job can be processed at a time.

A **job** is defined a basic unit of work. Its is independent from other jobs. It can't be aborted by another job, it can't be contingent upon the output of another job. It can run concurrently with other jobs and does not share data with other jobs. A job is the “sum of all work associated with its component job steps.”

A **task** is created by the control program when a job step is formally recognized. A task is “the work to be accomplished under the program named by the job step.” Tasks are independent in that they can be performed concurrently but they can be made to have dependent relationships on other tasks. This dependency can take the form of waiting for completion or requiring successful completion of another task. To enable this task dependency a **task hierarchy** is created. Tasks can designate **subtasks** upon which they are dependent and have power over.

“Although a job stream may designate many jobs, each of which consists of many job steps and, in turn leads to many tasks” (Witt, 1966), in an alternate example, one could imagine an environment where the entire mainframe is dedicated to a single job that consists of a single job step. This job step could create many tasks. Each task might be responsible for taking input from operator consoles in a system such as an online inventory system.

Degrees of Multiprocessing

The level of multiprocessing increased during the life of the OS via additional, optional components. These additional components, which greatly increased productivity, came at a high financial cost and would not have been purchased without a definite need.

In the initial, basic version of the OS, the control program was called the Primary Control Program (PCP) and allowed only one task at a time. In this mode, the system could wait a considerable amount of time waiting for I/O, as no other task was allowed access until the current task completed its processing.

Multiprogramming with a Fixed Number of Tasks (MFT) allowed four tasks to proceed concurrently. Memory was divided into a set of partitions. Each partition ran one job at a time. If a job was idle, the partition was still help by that job. A version of MFT that allowed 15 concurrent tasks was offered later. The number of tasks was set at system build and was not changeable thereafter.

Multiprogramming with a Variable Number of Tasks (MVT) was the most

complex option offered. In theory, this option offered a limitless number of tasks. Memory management issues (discussed in the “memory management” section below) required the addition of a job scheduler to manage memory more effectively.

Data Sharing

In reviewing the nature of tasks, IBM reconciled two very different views held by previous OS designers. In first-generation batch systems, the view was that a machine executed and incoming stream of programs. Each program and its associated data corresponded with one “problem”. In first generation real-time systems, the incoming stream consisted of both the program and data.

OS/360 utilized the commonality between the two approaches, that they both were programs used to process data, and viewed data as something separate from the task that is acting upon it. An example might be two tasks that each update a single table. The term for the data is a “data set.” (Mealy, 1966)

Data sharing, too, was an advance that paved the way for the multi threading system that would follow.

Job and Task Management

There is a distinction made between job management and task management.

Job management “primes the pump” - it turns each job step over to the task manager as a formal task. Job management functions are accomplished by two schedulers: a job scheduler and a master scheduler. The **job scheduler** controls three functions: read/interpret, initiate/terminate, and write. The **master scheduler** handles operator commands from the console, and sends messages to the console operator.

The job scheduler is responsible for job management functions. It permits either sequential FIFO scheduling or non-preemptive priority scheduling. Priority scheduling is accomplished via an input work queue. The queue can react to job priorities which are set by the user, consisting of a number from 0-14 in increasing importance. In addition, in a system where a job queue is in use, a unit called the initiator/terminator can look ahead to future job steps and allocate resources in advance such as issuing commands to the operator to load tape.

The **task management** controls the flow of work. Its functions consist of fetching modules, allocating the CPU, storage space, channels and control units on behalf of the tasks. It is responsible for task synchronization. All work submitted for processing must be formalized as a task. Tasks can run in either a

single-task environment where only one task can exist at a given time, or a multi-task environment where many tasks can compete for resources.

Single-task environments create a single task which steps through the program and can has access to all resources on the system. When the job step is completed, the job scheduler is notified of its completion status.

Multi-task systems must assign resources amongst the tasks on the system. It must keep track of these allocations and must queue multiple requests for instances of resources. Each resource is managed by a separate part of the control system. The task dispatcher allocates the CPU. The queue waiting for the processor is called the task queue. This is a priority queue.

Allowable Process States

OS/360 uses only long term scheduling in allocating the CPU to tasks. Scheduling is determined based on jobs, not tasks. A task is defined as “Ready” when it can use the CPU as soon as it gets an opportunity. “Waiting” is a state defined as “if some even must occur before the task again needs the CPU” such as waiting for user input or I/O. “Complete” indicated that a task is finished processing, whether successfully or with an error condition.(Witt, 1966).

Memory Management

Memory management evolved and was expandable with optional schemes which were made available during the life of the product. (See figure 1)

In its original incarnation, PCP (Primary Control Program), OS/360 operated as a batch system, and thus, one task had access to all resources on the system. Thus, the memory model used at this time was a single partition. The OS existed in a portion of main memory and the remaining available memory was used by the task that was currently existing on the system.

Another option for the OS was MFT, multiprocessing with a fixed number of tasks. Recall that this option allowed from 4 – 15 tasks to exist at a time. This option utilized memory that was divided into Multiple Fixed Partitions. If a task was idle, that partition remained held by the task until it was completed.

In the most complex implementation, multiprogramming with a variable (limitless) number of tasks (MVT), partitions were created on the fly. If memory was free, the control module would search the job queue for a job that would fit

into the available space and would create a partition for that task. For this reason, external fragmentation was a problem. An enhancement addressed the fragmentation issue by tagging each task in the ready queue with one of several fixed size tags. When there jobs were released they were fit into predefined slots. This of course would result in internal fragmentation but had the advantage of allowing a very large job to find room on the system, where it might not be able to in the previous incarnation of the scheduler. (Paulson, 2001)

Overview of Memory Management Techniques (Fig. 1)

Component	# Tasks	Memory Management	Comment
PCP	1	1 partition	Batch-processing model – One job had all available memory until complete
MFT	4 to 15	Fixed partitions	Number of partitions = number of tasks
MVT	unlimited	Variable partitions	Created high degree of external fragmentation
MVT with HASP	unlimited	"Buddy system"	Enhanced MVT to reduce external fragmentation and ensure that larger jobs could run

Memory Structure and Implementation

While the 16 KB to 1024 KB main memory available in the OS/360 sounds minute to modern users, there were programs at the time that fit, in their entirety, into main memory. However, even programmers of the era had to use techniques to utilize memory to accommodate programs larger than main memory would allow. A ready to execute program could consist of one or more subprograms called **load modules**. Three techniques are described below. These techniques allowed obvious advantages over the **simple structure**, in which the entire load module can be loaded into memory at once.

Planned overlay structure was utilized when not all of the elements of a program needed to be actively loaded on the system simultaneously. The programmer was able to segment the program into load modules that need to be present simultaneously in main memory. Therefore, one area of memory can be used and reused by many different modules. This makes very effective use of main memory.

Dynamic serial structure was useful when jobs became more complex, as the advantages of planned overlay diminish as job complexity increases. In this structure, load modules can be called dynamically, when they are name in the execution of another module. Memory is allocated as requests arise. Any load module can be called as a sub-module, or **subroutine** of another module that is currently executing. When control is returned to the calling module, the memory taken by the subroutine is released but not overwritten necessarily and if another module calls it, can be re-linked to without the need for bringing it back into memory.

Dynamic parallel structure is the only structure that is not serial. It creates a task that can proceed in parallel with other tasks, but since it uses the ATTACH microinstruction and thus requires the processor to go into kernel, or supervisor, mode, its use needs to be limited. (Witt, 1966)

Deadlock

OS/360 employed little in the way of deadlock management. The system employed what might best be categorized as rudimentary deadlock resolution. The OS had no way of knowing how long an I/O request would take or whether it would ever be completed. Especially in a system using PCP, where only one task ran at a time, this could accidentally (or maliciously) lock up the system indefinitely. Therefore, the OS implemented an arbitrary absolute limit of 30 minutes before canceling the job, this freeing the system for the next job to begin processing. Even in a system running MFT, where a fixed number of tasks is running on the system, this poor deadlock resolution system could result in system resources other than the processor being tied up for a long time interval.

Mutual exclusion and synchronization

The ability to allow sharing and mutual exclusivity to a resource is accomplished more by the programmer than by the operating system. The operating system provides two microinstructions, enqueue (ENQ) and dequeue (DEQ) which allow the programmer to create a queue that enable tasks to share resources in a “serially reusable” model. For example, one might look at several tasks which must update the same table. Each task is given exclusive access to the table and must complete its work before another task is given access. The programmer may create a queue to limit access. The queue for the given resource has a control block which contains an indicator which can be set to declare the resource as busy. (Witt, 1966)

Security

User management in this mainframe OS was virtually non-existent and thus the task of safeguarding sensitive data is limited to the use of a password. A data set can be flagged as “protected”. The correct password must then be entered on the console. Passwords are stored is a control table that has its own security flag set to “protected” can can only be reached via the master password.

Conclusion

The IBM OS/360 was a pioneering operating system that bridged the two arenas of commercial and scientific, of batch processing and “real time” operations, with one general purpose machine. IBM had the vision and knowledge to understand that two diverse computing environments could be incorporated in one mainframe. IBM furthered the state of the art with multiprocessing and arguably hastened the era of modern computing.

Bibliography

Clark, W.A. (1966). “The functional Structure of OS/360, Part III: Data management.” IBM Systems Journal., Volume 6, Number 1. 30-51

Jardine, D.A. (1972) “Operating System/360 1965-1972” http://ldworen.net/fun/os360obit.html

Mealy, G.H. (1966). “The functional Structure of OS/360, Part I: Introductory survey.” IBM Systems Journal., Volume 6, Number 1. 3-11

Paulson, Lars. (2001) http://www.beagle-ears.com/lars/engineer/comphist/ibm360.htm

Witt, B.I. (1966). “The functional Structure of OS/360, Part II: Job and task management.” IBM Systems Journal., Volume 5, Number 1. 12-29