Major OS Achievements

Chris Collins

15th October 2006
Introduction

This paper discusses several major achievements in operating systems (OS). We begin by discussing the birth of the operating system with multiprogrammed batch systems. Then we discuss memory management and process management which both of which were required by multiprogrammed batch systems.
Birth of the Operating System

In the very early days of computing, a computer was a single task, single user machine. A user would load their program in source code from punch cards or punch tape, load an appropriate compiler, any required libraries, an assembler or linker, etc, compile the program and run it. This was a very time consuming and error prone process. Since computers were very expensive machines any idle time was a major cost to the computers operator.

Batch systems were introduced with the intention of reducing the setup and tear down time involved with running a program. In a batch system, similar programs would be put in batches. For example, all programs which required the FORTRAN compiler would be batched together. The computer would be set up appropriately and all programs in the batch were run in sequential order. This in itself was a major change from the way the user interacted with a computer. No longer were they required to set up the computer for operation, but now they submitted their jobs to a job manager who took responsibility for running the jobs.

Even though batch processing had some performance gains, the computer still remained a single task machine. Once a job was submitted it had access to all the resources on the computer and had to take it's own responsibility for managing them.

When programs were analysed, it was realised that much of a programs time is spent doing I/O operations between the processor and disk or printer for example. Up to 96% (Stallings 2006, p251) of a programs execution time has the central processing unit (CPU) idle while I/O is taking place. The solution to this was to interleave programs on the processor. Thus the Multiprogrammed Batch System was born.

In this arrangement if Program A requested an I/O operation Program B could be loaded to use the processor until it blocked for I/O or completed, then Program A could resume
operation. This could be taken to the n\textsuperscript{th} degree dependent on system memory size etc., as all programs had to be loaded into memory simultaneously to make use of this switching.

With multiple programs in memory some sort of memory management function was needed. Also with the possibility of multiple programs being read to run at the same time, some program scheduling function was required. These were addressed in the first rudimentary operating system, a program written with the intention of controlling and managing other programs. Even to this day, memory management and process scheduling are among the cornerstones of all modern operating systems.

**Virtual Memory**

In early computers, instructions were read one by one from punch cards or punch tape as they were needed. When stored program computers came along, they kept the same metaphor, though now the next instruction to execute was stored in memory and not on a punched card. Just like punched cards had to be kept in sequence so did the instructions in memory, and programs were loaded into contiguous memory locations for execution.

With the advent of Multiprogrammed Batch Systems were several programs could be in memory at the same time, computer designers were tasked with keeping the CPU busy as much of the time as possible. However they were restricted in this as they could only load so many programs as the computers memory could accommodate.

If the situation arose where all loaded programs were waiting for I/O, then it would be useful if the system could unload one of the programs and load another to make use of the CPU. In essence a swap system was needed. This lead to the development of sophisticated memory management.
The first of these schemes was Dynamic Partitioning (Flynn & McIver, 2006). In this scheme, programs were allocated as much memory as the requested in a contiguous block when they were started. When a program was swapped out, this block could be used by another program who's memory requirement was equal or less. Over time, this lead to external fragmentation. For example, if Program A had been using a 10 Kb block and is swapped out for program B using 8Kb, a 2Kb gap would ensue, which may not be of use to any of the other programs in the system.

To overcome this, Paged Memory Allocation was eventually developed. This scheme divides both the available memory and the incoming jobs into fixed size pages. A program is then allocated enough pages to fulfill its memory requirements. This had the effect of significantly reducing external fragmentation, though it had the downside of increased internal fragmentation. For example, dividing a program requiring 13Kb into 5Kb pages would require a total of 15Kb from memory, with 2 Kb wasted. In reality the flexibility afforded by this scheme more than compensated for this shortcoming.

The operating system had to be a little more complex in this instance in order to keep track of which page belongs to which program, and providing a way to translate absolute memory addresses into paged memory addresses. The flexibility of this scheme also had a greater consequence, programs no longer had to be wholly in main memory. Only the currently used page needed to be in memory, all other pages could reside on a disk and be loaded as needed. This allowed more programs to be interleaved thus keeping the CPU as busy as possible. It also meant that for the first time, programs larger than the total amount of main memory could be loaded and run.
This lead to the term 'Virtual Memory', as the programmer now had access to much larger memory than main memory given that parts of a program could be stored on disk until needed. Stallings (2006) notes “Virtual memory allows for very effective multiprogramming and relieves the user of the unnecessarily tight constraints of main memory” (p. 268).

Process Management

In the multiprogrammed system, the operating system has to manage several programs who are competing for the computer's resources. Not only for CPU time, but also for access to I/O devices.

The part of an operating system which handles these tasks is usually referred to as the scheduler. The scheduler implements a scheduling policy which defines who gets a resource, when and how long they will have use of it.

There are several factors which need to be addressed to make a good scheduling policy. The scheduler needs to maximize throughput, minimize response time, minimize turnaround time, minimize waiting time, maximize CPU efficiency and ensure fairness for all jobs (Flynn & McIver, 2006, pp. 120-121).

In order to achieve these goals, different policies can be implemented by the system designer. For example, the system designer may decide to use a preemptive scheduling policy. In this case, running programs can be interrupted by the operating system so that it may switch programs on the CPU or I/O devices.

Most modern operating systems use a preemptive scheduling policy. This is done through a technique called time slicing. A program will get the use of the CPU for a period of time after which the operating system will regain control and decide if the program is to continue using the CPU or if CPU time is to be given to another program.
For example, the Linux operating system uses a preemptive scheduling policy with dynamic priorities. Processes who have recently had CPU time have a reduced priority, while processes who haven't had CPU time recently are given higher priority. At the end of every time slice, the scheduler decides based on these priorities which process is given CPU or I/O device time. (Bovet & Cesati, 2000, chapter 10). A policy of this type ensures fairness and that processes don't suffer resource starvation.

Conclusion

As computers continue to evolve and as computing touches more and more aspects of our daily lives, it's difficult to see useful applications being developed that don't rely on operating systems and the facilities they provide. Process and memory management are two corner stones of modern operating system design. Without them, computers would be difficult to use, and would probably have remained specialist tools used by people with specialist knowledge. In no small way, the achievements of operating systems have allowed computer to come far, and for us users to benefit from all the exciting applications that computers are being used for in the modern world.
References

