

CPU Scheduling

by *Simon Salomon*

An operating system is a program that manages the hardware and software resources of a computer. It is the first thing that is loaded into memory when you turn your computer on. Without the operating system, each programmer would have to create a way in which a program will display text and graphics on the monitor. The programmer would have to create a way to send data to a printer, tell it how to read a disk file, and how to deal with other programs.

In the beginning programmers needed a way to handle complex input/output operations. The evolution of computer programs and their complexities required new necessities. Because machines began to become more powerful, the time a program needed to run decreased. However, the time needed for handing off the equipment between different programs became evident and this led to programs like DOS. As you can see the acronym DOS stands for disk operating system. This confirms that operating systems were originally made to handle these complex input/output operations like communicating among a variety of disk drives.

Earlier computers were not as powerful as they are today. In the early computer systems you would only be able to run one program at a time. For instance, you could not be writing a paper and browsing the internet all at the same time. However, today's operating systems are very capable of handling not only two but multiple applications at the same time. In fact, if a computer is not able to do this it is considered useless by most computer users.

In order for a computer to be able to handle multiple applications simultaneously there must be an effective way of using the CPU. Several processes may be running at the same time, so there has to be some kind of order to allow each process to get its share of CPU time. In the figure below, the diagram shows the possible states of a process.

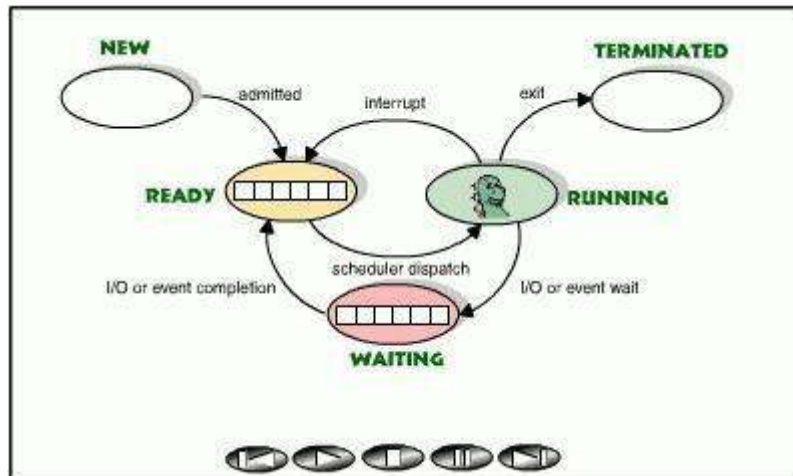


Fig. Process States

When a process is created its state is set to new. Once the process is ready to use the CPU its state is set to ready. It is inserted into the ready queue awaiting its turn to be assigned CPU time so that its instructions can be executed. Once the CPU is available the process next in line in the ready queue is set to running. This means that the process' instructions are being executed.

Once the process is being executed two things can happen.

- 1) The process' instructions are all executed in which case its state will be set to terminated.
- 2) While the process is running an I/O interrupt or event wait is executed which stops the running program.

In the event the first action takes place, the program finishes executing and then terminates. This means the all the instructions of the process have been executed and it has no more need for the CPU. However, this can also happen if there is some kind of error in the program that requires the process to become terminated prematurely.

In the second case the actions taken are much more complex. For example, let us say that there is a process that is currently occupying the CPU. As the instructions of this process are being executed the program needs to get input from the user at the keyboard. This causes the process to stop executing. In this situation the process will enter the waiting state. This means that the process will lose control of the CPU and be inserted into the waiting queue. Once the input is received from the user at the keyboard the process must go back to

the ready state. The process cannot take hold of the processor; it must wait in the ready queue until it is assigned the CPU.

Once the process is assigned the CPU again, it will continue executing its instructions. Once again two things may happen. If there is need for more I/O then the process will once again enter into the waiting state. If not, then the process will complete and will become terminated once the final instructions are executed.

As stated earlier a process may enter several states in its lifetime. However, where is all this information stored? The answer is in the process control block (PCB). The process control block is a representative of each process. It contains information about the process which it is associated with. The information it contains is the process state, program counter, CPU registers, CPU-scheduling information, memory management information, accounting information, and I/O status information.

CPU-scheduling information is information that includes process priority, pointers to scheduling queues, and any other scheduling parameters. This is the basis of multi-programmed operating systems. Because the CPU is able to switch from process to process the operating system is able to make the running programs seem as if they are being executed simultaneously.

Whenever the CPU has to wait for an I/O operations to occur there are CPU cycles that are being wasted. The idea behind CPU-scheduling is to be able to switch from process to process when the CPU becomes idle. This way while a process is waiting for an I/O request to complete, the CPU does not have to sit idle. It can begin executing other processes that are in the waiting state.

There are two scheduling schemes that are available. There is the non-preemptive scheduling scheme and there is the preemptive scheduling scheme. I will explain the different algorithms that are used in each scheme and discuss how they work to decide which process in the ready queue should be allocated the CPU.

Non-preemptive scheduling is a scheme where once a process has control of the CPU no other processes can preemptively take the CPU away. The process retains the CPU until either it terminates or enters the waiting state. There are two algorithms that can be used for non-preemptive scheduling. The first is the First-Come, First-Served

algorithm. In this scheduling algorithm the first process to request the CPU is the one that is allocated the CPU first.

The First-Come, First-Served algorithm is very simple to implement. It can be managed using a First-In, First-Out (FIFO) queue. When the CPU is free, it is allocated to the first process waiting in the FIFO queue. Once that process is finished the CPU goes back to the queue and selects the first job in the queue. Any process that requests the CPU must go to the back of the queue. You can imagine it as students waiting in line to pay for their lunch. When one student is ready to pay for their meal, they must go to the back of the line and wait their turn. This is the idea behind the First-Come, First-Served algorithm.

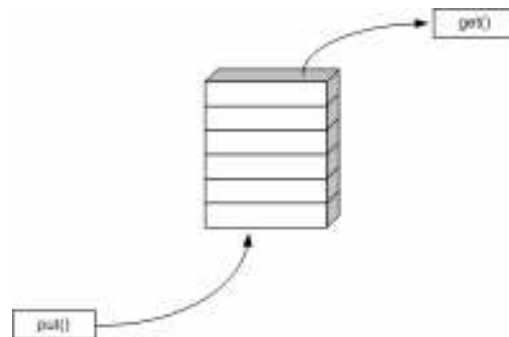


Fig. FCFS Scheduling Scheme

The second non-preemptive scheme is the Shortest-Job-First (SJF) scheduling algorithm. In this scheduling scheme the process with the shortest next CPU burst will get the CPU first. By moving all the short jobs ahead of the longer jobs we decrease the average waiting time. However, it is impossible to know the length of the next CPU burst. One approach we can use is to estimate its value.



Fig. SJF Scheduling Scheme

Another implementation of the SJF is the priority scheduling algorithm. In this scheduling scheme a priority is associated with each process. Depending on the implementation of the algorithm there can be a range of priorities. The job that has the highest priority will be the one that is selected from the ready queue. This process will be allocated the CPU and the lower prioritized jobs will have to wait.

The major problem with SJF is starvation. Starvation is caused when a process that is ready to be executed is not because it is still waiting for the CPU. For example, let's say that we have a lot of jobs who have high priority and one that has low priority. The processes that have high priority will all be executed and if jobs continue to be inserted into the ready queue with high priorities, then the low priority job will never be allocated the CPU.

The solution to this problem is aging. Aging increases the priority of a process gradually. For example, we can increase the priority of a process from low to high, thereby guaranteeing that it will be executed at some point.

Preemptive scheduling is the second scheduling scheme. In preemptive scheduling there is no guarantee that the process using the CPU will keep it until it is finished. This is because the running task may be interrupted and rescheduled by the arrival of a higher priority process. There are two preemptive scheduling algorithms that are preemptive. They are the Round Robin (RR) and the Shortest Remaining Time First (SRTF)

The Round-Robin scheduling scheme is similar to that of FCFS except preemption is added to it. In the RR scheduling scheme the CPU picks

a process from the ready queue and sets a timer to interrupt after one time quantum. During this scheme two things may happen.

- 1) The process may need less than one time quantum to execute.
- 2) The process needs more than one time quantum.

In the first case when a process is allocated the CPU it executes. Because the time required by the process is less than one time quantum, (the unit of time assigned to every process), the process gives up the CPU freely. This causes the scheduler to go and select another process from the ready queue.

In the second case if a process needs more than one quantum time to execute it must wait. In the RR scheme each process is given only one time quantum. The only way for the process to gain access to the CPU for more than one time quantum is if it is the only process left. If that is not the case, then after one time quantum the process will be interrupted by the timer. This will cause the process to go to the end of the ready queue. The next process in line will get allocated the CPU and will be allotted one time quantum.

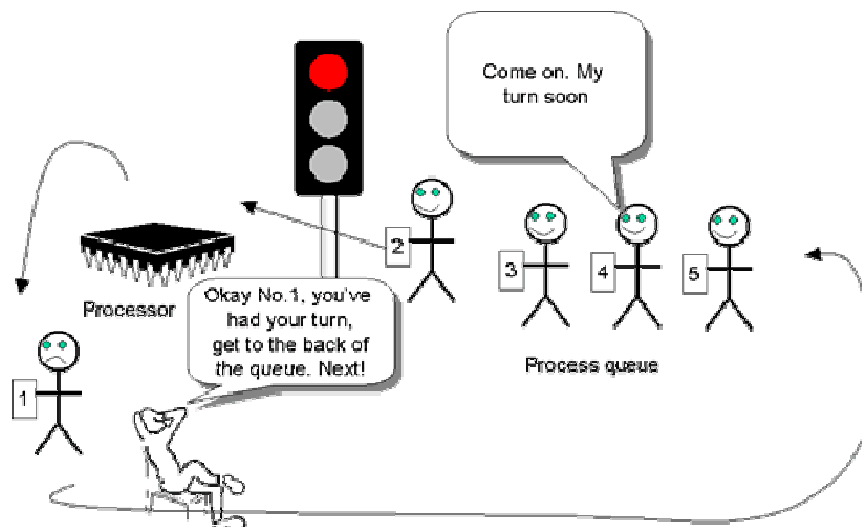


Fig. Round Robin Scheduling Scheme

In the Shortest Remaining Time First (SRTF) algorithm, the process that is running is compared to the processes in the ready queue. If a

process in the ready queue is shorter than the process running, then the running task is preempted and the CPU is given to the shorter process until it is finished. This algorithm is similar to the SJF algorithm except that preemption is added.

The operating system has grown very powerful throughout the decades. It has gone from handling I/O operations to managing all the resources of the computer. It has become the middle man between the user and the hardware. Providing us with a beautiful interface and allowing us to run programs with a single click of a button.

Time sharing between processes would be impossible without the operating system. Because of OS like Linux and Windows XP we are able to run multiple programs simultaneously without having to worry about conflict. We are able to browse the web and chat with our friends over seas while writing our papers. The operating system has become an essential part of our everyday lives. We see more and more devices come loaded with an OS. And I predict that one day even our refrigerators will come equipped with an OS.

Bibliography

Silberschatz, Peter Galvin, Greg Gagne, 2005, *Operating System Concepts*, John Wiley & Sons, Inc, Hoboken, NJ, pp 81-125 and 153-185

Images:

SJF Scheduling Scheme

<http://www.cs.jhu.edu/~yairamir/cs418/os2/img024.gif>

Figure Round Robin Scheduling Scheme

http://www.dcs.napier.ac.uk/~bill/pics_round_robin.gif

Figure Process States

Image Not Available Anymore

Figure FCFS Scheduling Scheme

Image Not Available Anymore