Operating systems can seem to multitask 2 or more running processes at once with a technique called Time Sharing; That is, “...running one process, then stopping it and running another, and so forth…” (Arpaci-Dusseau & Arpaci-Dusseau, 2012). In order to fully answer how the OS accomplishes this, a discussion of Limited Direct Execution will be employed. Following that, various scheduling methods will be summarized. Lastly, a specific operating system (Linux) will be reviewed in terms of the policy that it uses to perform context switching.

# Limited Direct Execution

In order for an operating system to implement Time Sharing policies, the OS has to both allow processes to run on the CPU, but also retain enough control so that it can switch from one process to another. Therefore, the OS imposes limitation on what a process can do without OS intervention, thereby retaining control. The OS does this “by first (during boot time) setting up the trap handlers and starting an interrupt timer, and then by only running processes in a restricted mode” (Arpaci-Dusseau & Arpaci-Dusseau, 2012). In this way, the OS can allow a process to efficiently use the CPU until a process makes a system call for a privileged operation or until the OS decides to cut off a process’s CPU usage in favor of another process.

# Scheduling Methods

Many scheduling policies have been designed with an eye toward efficient CPU usage and improving “waiting time, turnaround time, response time and context switch of processes” (Negi & Kalra, 2014). Below is a brief overview of various scheduling methods based on the metric/s which they attempt to improve (as discussed in Arpaci-Dusseau & Arpaci-Dusseau, 2012).

## Optimizing Turnaround Time

One group of scheduling methods is aimed at reducing turnaround time. Turnaround time is essentially the average time it takes to complete processes. The way in which many of these policies attempt to accomplish this goal is by running shorter processes ahead of longer processes. Examples of this group of policies are:

* First In, First Out (FIFO), aka First Come, First Served (FCFS)
* Shortest Job First (SJF)
* Shortest Time-to-Completion First (STCF), aka Premptive Shortest Job First (PSJF)

## Optimizing Response Time

Another type of scheduling method improves interactivity by minimizing response time. Round Robin (RR) does this by cyclically allowing processes to use the CPU for short lengths of time (quantums) and then switching to another process in the chain.

## Optimizing Both Turnaround and Response Time

The Multi-Level Feedback Queue attempts to balance the optimizations of both response time and turnaround time, which are by their very definition in opposition. It does this by incorporating the Round Robin technique within multiple priority levels of waiting queues; In addition, a process’s priority level is adjusted according to it’s recent behavior, with short jobs more often having higher priority, but not starving longer processes for CPU time.

## Proportional-Sharing of the CPU

Proportional-sharing schedulers (aka Fair-Share schedulers), ignore the Turnaround and Response Time metrics. Instead, they aim at dividing a CPU’s time among all competing processes in segments proportional to each process’s total use of the CPU. Examples of this group of policies are:

* Lottery Scheduling
* Stride Scheduling
* Completely Fair Scheduler (CFS)

# Linux OS

All of the schedulers discussed above can work well enough depending on system workload and uses. One way to learn about how context switching is implemented in modern operating systems is to delve into a specific operating system and review its unique scheduling policies. To this end, Linux was chosen because it makes up almost 60% of all Unix based websites, which themselves make up nearly 70% of all websites (“Usage statistics and market share of Unix for websites”, September 21, 2018).

Linux uses a mashup of FIFO and RR in a MLFQ (“4.2. CPU SCHEDULING”, n.d.). In the Linux scheme, Realtime processes have a priority queue ranging from 1 to 99, are time-critical, and must finish without interruptions; and as such they are always scheduled before other processes in a FIFO scheme unless they are blocked or preempted by a higher priority FIFO process. Normal processes are then run with other processes of the same priority level (also between 1 and 99) in a Round Robin fashion until they complete.

References

Arpaci-Dusseau, R. & Arpaci-Dusseau, A. (2012). Operating Systems: Three Easy Pieces. Madison, WI: University of Wisconsin-Madison. Retrieved from <http://pages.cs.wisc.edu/~remzi/OSTEP/>/

“4.2. CPU SCHEDULING” (n.d.). Product Documentation for Red Hat Enterprise Linux &. Retrieved from <https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/s-cpu-scheduler>

Negi, S. & Kalra, P. (2014). A Comparative Performance Analysis of Various Variants of Round Robin Scheduling Algorithm. International Journal of Information & Computation Technology, 4 (7), 765-772. Retrieved from <http://www.ripublication.com/irph/ijict_spl/ijictv4n7spl_18.pdf>

“Usage statistics and market share of Unix for websites” (September 21, 2018). W3Techs Web Technology Surverys. Retrieved from <https://w3techs.com/technologies/details/os-unix/all/all>