This week’s discussion is focused on how an operating system virtualizes the CPU, thereby creating the illusion of multiple CPUs. An important point to make before delving into *how* virtualizing the CPU is accomplished needs to be that hardware devices can have more than one physical CPU (Arpaci-Dusseau & Arpaci-Dusseau, 2012). For the sake of discussion, however, let us limit the discussion to hardware devices with only one physical CPU to be virtualized. With this in mind, the point of this writing is to describe how the OS can multitask several processes (running programs) *seemingly* at once.

Operating systems feign running several processes on the CPU at once through a technique called *Time Sharing*, which amounts to “...running one process, then stopping it and running another, and so forth…” (Arpaci-Dusseau & Arpaci-Dusseau, 2012). Through time-managed sharing, multiple processes can use the CPU and other device resources, with the only downfall being that the speed and performance of individual processes necessarily decreases as more and more processes vie for the use of the CPU.

Perhaps the most practical question regarding virtualizing the CPU is determining how the OS decides which process among many will access system resources at any given point in time, based on the process state that the processes are in. According to Arpaci-Dusseau & Arpaci-Dusseau (2012), the processes can transition between the following states: running, ready, and blocked. In addition, this context switch is determined by a scheduling policy based on a variety of factors (including performance metrics, workload knowledge, and historical information) to make this decision.

One article that I read (Negi & Kalra, 2014) discusses different algorithms that an operating system can use to schedule processes while multitasking programs. The point of experimenting with different scheduling policies is to make the most efficient use of the CPU by reducing “waiting time, turnaround time, response time and context switch of processes” (Negi & Kalra, 2014). According to this same article, most operating systems use the “Round Robin” scheduling policy for CPU time sharing. This policy will go through the various processes cyclically, giving them each a fixed amount of time to run and then switching to the next one in the cycle, with the just performed process going to the bottom of the ready list. Apparently, the Round Robin algorithm can be made more efficient, by coming up with better and better algorithms to determining how long each process should have access to the CPU before switching to the next process. Round Robin is already more efficient than other policies like First Come First Serve or Smallest Job First, but it can be improved further by optimizing the amount of time each process has access to the CPU in the Round Robin (Negi & Kalra, 2014).

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/>/

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>