Fischer Davis, Jacob Montes, Justin Muskopf

CSCE 4600.001

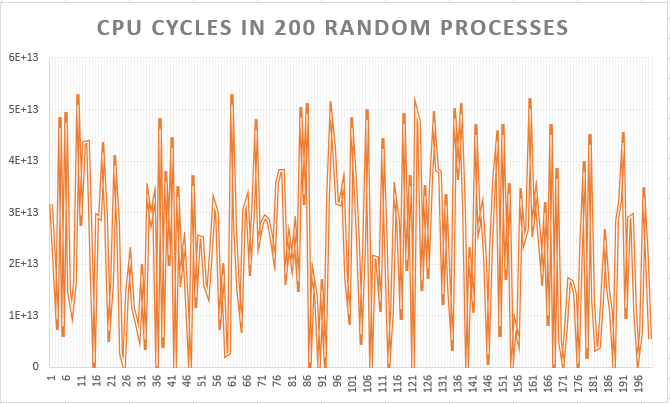
Due Date: 3/19/2019

OS Project 1- Process Scheduler

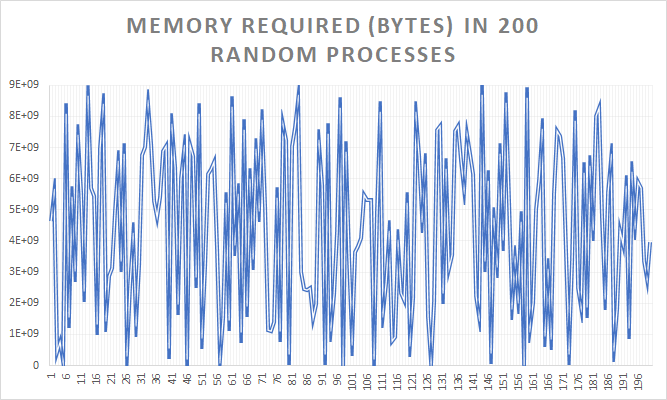
# Project Overview

Process schedulers are an essential resource within modern operating systems. A process scheduler assigns and coordinates the processes that are currently running on any one of a system’s processors (CPUs) at any given time using one - or several - different scheduling strategies. The task for this project was to develop a simulation of a processor scheduler, depicted by our *ProcessScheduler* class, that assigns processes (or jobs) to a set of simulated processors/processing nodes. We were to assume that our computing infrastructure had 5 processors available - each with its own speed and available memory; as simulated by our *Processor* class.

Processes were to be randomly generated, and would have a burst time (number of cycles to complete) and a memory requirement. A process’s burst-time was to be assigned at random, falling within the range of a minimum of 10\*106 cycles, and a maximum of 50\*1012 cycles. A process’s memory requirement was also to be assigned at random in the range between a minimum of 0.25MB, and a maximum requirement of 8GB. Therefore, before beginning on our scheduler, we were to develop a method that could generate a set of processes with random burst-times and memory requirements, and did so in our *ProcessGenerator* class. The processes were to be synthetically made, and not to be actually created. For problems 1-3, we were to assume that a set of 200 generated processes were known to the scheduler preemptively, and could therefore be analyzed before scheduling. For problem 4, the scheduler was to schedule the processes sequentially. To show the randomness of our generated processes, view the figures below:



*Figure 1: CPU requirements of 200 randomly generated processes*



*Figure 2: Memory requirements of 200 randomly generated processes*

# Assumptions

Each of our designed algorithms assumes that all 5 processors are capable of running in parallel with one-another, and that the only memory consumption on each processor is the process that is currently running on it.

# Scheduling Algorithm 1

Problem 1 required all 5 processors to be identical (i.e., same speed and memory). We were to develop and implement a scheduling algorithm that assigned a set of 200 randomly generated processes to the 5 processors such that the total turnaround time to complete the 200 processes was minimized. It was assumed that each processor was capable of a speed of 4GHz, and had a memory capacity of 8GB.

Since the solution allowed for us to use a preemptive approach, meaning that we knew the characteristics of every process in the set before we scheduled them, we decided to approach the algorithm’s development in the following manner:

1. Generate the set of 200 random processes, and provide it to the scheduler
2. The scheduler then orders the set of processes by the number of cycles it takes to execute each in ascending order
3. The scheduler then cycles through this sorted set of processes, and decides which processor to assign the process to by choosing the processor that has the least number of cycles assigned to it thus far
4. The total turnaround time is then determined by choosing the processor with the greatest total execution time, as shown by the following equation (EQ 1):

Since every processor is identical in this problem, each processor should optimally obtain 20% of the required cycles. An interesting analysis from this algorithm is that each processor, after *n = 20*trials, stayed within 0.38% of the optimal value, with averages of:

* PA = 19.62%
* PB = 19.80%
* PC = 19.99%
* PD = 20.21%
* PE = 20.38%

Keeping this in mind, the algorithm that we produced closely mimics the preemptive Shortest Job First (SJF) algorithm, which is provably optimal. Two text files were created that recorded the statistics for the problem. One file included the statistics of each process in the set of 200, while the other recorded the statistics of each processor.

# Scheduling Algorithm 2

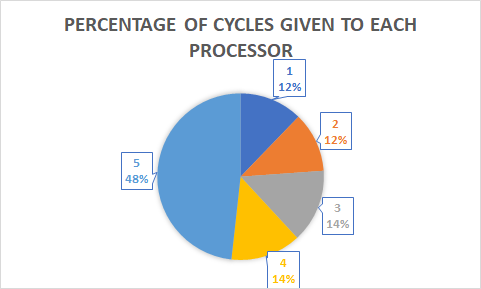
Problem 2 required us to assign processes to a specific processor such that the processor had sufficient memory available for the process’s memory requirements. We were to assume that the processing nodes were identical in speed but had the following memory availability:

* PA = PB = 2GB
* PC = PD = 4GB
* PE = 8GB

The limiting factor in this problem was the different memory availabilities of each processor. Since the solution allowed for us to use a preemptive approach, meaning that we knew the characteristics of every process in the set before we scheduled them, we decided to approach the algorithm’s development in the following manner:

1. Generate the set of 200 processes, and provide it to the scheduler
2. The scheduler then orders the set of processes by the number of cycles it takes to execute each in ascending order
3. The scheduler then cycles through this sorted list of processes, and for each process assigns it to the first processor that has both the least number of cycles and sufficient memory availability
4. The total turnaround time is then determined by choosing the processor with the greatest total execution time, as shown by EQ1 (shown above)

After *n = 20* trials, the average percentages of the generated load that each processor ended up with was:



*Figure 3: The percentage of load given to each processor in algorithm 2*

This distribution is to be expected since the only processor that has a memory capacity greater than 4GB is processor 5. Processors 1 and 2 have the lowest allocation, since their memory capacity of 2GB only contains 25% of the potential range of memory requirements, while processors 3 and 4 contain 50%, and processor 5 encompasses the entire memory range.

Like scheduling algorithm 1, this procedure most closely represents the preemptive SJF algorithm, but is slightly more realistic due to its considering memory constraints. After execution, two text files were created that recorded the statistics for the problem. One file included the statistics of each process in the set of 200, while the other recorded the statistics of each processor.

# Scheduling Algorithm 3

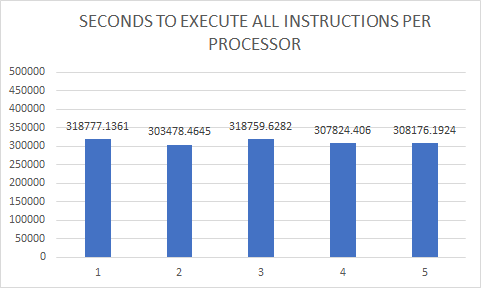
Problem 3 required that all of the processors had the same amount of memory, 8GB, but that they would differ in their speeds. The following are the speeds that the different processors were given:

* PA = PB = 2GHz
* PC = PD = 3GHz
* PE = 4GHz

The goal was to find the best assignment of the processes to the different processors. The solution for this was to modify our algorithm to check for the processor with the shortest current execution time. This solution also allowed for a preemptive approach in which we used this to help us schedule each process in a faster and more effective manner. We achieved this goal in the following manner:

1. Generate the 200 processes, and hand them to the scheduler one by one.
2. The scheduler then orders the set of processes by the number of cycles it takes to execute each in ascending order.
3. The scheduler then passes each process by each processor, adding the time it takes for the processor to execute the given process with the already existing total execution time of the processor. The scheduler then chooses and returns the processor with the the shortest accumulative execution time.
4. After determining the processor with the shortest execution time, the scheduler then assigns the process to that processor, minimizing the waiting time for each process.

We greatly enjoyed implementing this algorithm, as it not only assigned the processes to a processor based on their number of cycles, but also assigned each process to the processor that would have the shortest total execution time after the process’s addition to its queue. The distribution of the work allowed for a similar total execution time between all of the processors, with an average (*n = 20*) percent difference of 1.89%:



*Figure 4: Seconds to execute all instructions for each processor in algorithm 3*

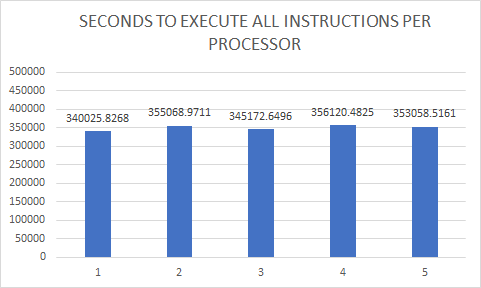
Two text files were created that recorded the statistics for the problem. One file included the statistics of each process in the set of 200, while the other recorded the statistics of each processor.

# Scheduling Algorithm 4

Problem 4 required the development of a scheduling methodology that could deal with the sequential arrival of the set of 200 processes. The scheduler could *not* inspect the entire set of processes, but was instead required to schedule them one-by-one in the order that they arrived in an attempt to achieve the best turnaround time possible. This means that, unlike problems 1-3, our algorithm did not allow us to use a preemptive approach. A non-preemptive approach means that we did not know the characteristics of every process in the set before we scheduled them, and therefore, we decided to approach the algorithm’s development in the following manner:

1. Randomly generate one process, and provide it to the scheduler.
2. The scheduler then cycles through each processor, adding the time it takes for the processor to execute the given process with the already existing total execution time of the processor, and returns the processor with the the shortest accumulative execution time.
3. After determining the processor with the shortest execution time, the scheduler then assigns the process to that processor, minimizing the waiting time for each process.
4. Repeat steps 1-3 for 200 processes.
5. The total turnaround time is then determined by choosing the processor with the greatest total execution time, as shown by EQ1 (shown above).

This algorithm most closely represents the reality of an actual process scheduler’s tasks. It is rare in modern computing that a process scheduler would know preemptively which processes were going to be given to it, and therefore, a scheduler must make its best attempt to schedule a process in real-time. Our scheduler, after n = 20 trials, was capable of creating a mostly optimized schedule, with the percent difference in total execution times between processors being 1.67%:



*Figure 5: Seconds to execute all instructions for each processor in algorithm 4*

Two text files were created that recorded the statistics for the problem. One file included the statistics of each process in the set of 200, while the other recorded the statistics of each processor.

# Conclusion

While not perfectly optimized, our scheduler was capable of staying within a calculated percent difference of 2% for all of our algorithms, and this margin of error is acceptable within any scientific community.