Alex Blair

Mckenna Galle

Julia Abbott

Operating Systems

Program Specs.

[Jabbott19@my.whitworth.edu](mailto:Jabbott19@my.whitworth.edu)

[Mgalle19@my.whitworth.edu](mailto:Mgalle19@my.whitworth.edu)

Scheduling Algorithm Simulation

Requirements:

* Multi-core processor
* Windows operating system
* Java JDK and JRE
* Useable input and output devices such as a keyboard and mouse.
* Main memory
* Disk memory
* Data structures such as (Vectors, Queues, Array Lists, Priority Queue)

Language:

Our group has chosen to use Java so that we can easily separate each algorithm into a class and

In this simulation we aim to simulate multiple different scheduling algorithms for different processes that a PC may have running. We will keep track of a number of statistics for these algorithms. These include, program ID, CPU burst time in milliseconds, IO burst in milliseconds, and CPU utilization, average wait times, average execution times, throughput, average turnaround time, and average response time. We will design our own method of creating a PCB or Program control block to control all running processes. We will create classes for each algorithm and use these objects in conjunction with the PCB to begin the simulation. If possible we would like to separate each algorithm into its own thread so that they can run concurrently and then compare results at the end. As far as the data, we will write the data to a file and this file will be our results for presentation.

Assumptions:

* For this project we will assume that a process has two states, ready and running.
* We will also assume that the processor is a multicore processor

PCB Structure:

* Variables include: PID (Process ID), State, Program counter, Memory info
* IO status

PCB Methods:

* Some methods include: Get state, get counter number, get Memory info, get IO status,

Each CPU register will be its own class and inherit from the parent PCB class. The register contents will be saved to a file as a type of progress report during execution of each algorithm.

The algorithms for scheduling will all be separate classes as well and inherit from the parent PCB class to have access to the PCB functionality.

Multiprocessor Approach:

We will use the load sharing algorithm approach. This will be done through a couple of critical sections and a ready queue as well as a set of Boolean statements to check if each core is under the maximum usage. Each process will be assigned a thread and a process id and when the process leaves the ready queue we will use a critical section to secure this access. Then there will be a critical section for when then threads run the processes. The process will be separate functions that must run for a “x” amount of time each. Each time will be static for the separate functions. We will record the results for CPU usage, response time, average wait time, into a file.

Uniprocessor approach:

The first come first served will be done with a critical section that only allows one thread through at a time. The processes simulated through different functions that take “x” amount of cycles of the processor to run. There will only be one thread that runs and when it finishes it will move on to the next process. The process will also return its results of how long it had to wait and this will be written into an I/O file.

The round robin scheduler will have one thread running and each process that will be executed is in the form of a series of functions that do some simple calculations and write the results to a file. Each function will be run for the specified quantum time between 20 and 100. We will experiment with numbers between 20 and 100 to find the time that has the least amount of wait time for each process. We will switch which function is executing if the time quantum is reached. Each process will write out to an I/O file to say how long it waited, and the CPU usage and response time.

If we can we may also attempt to implement a shortest process next algorithm. For this we will use a series of functions as well as the number of threads needed to execute the process to determine which function should execute first. These will then be placed into a ready queue and then when the process is ready to execute, it will go through a check to see how many threads are needed to execute and the least number of threads will take priority. Then we will record the results in an output file.

For multilevel feedback queues we will have three queues. The first process there will be put in queue zero, and the second will be put in queue two. The first queue will run for “x” amount of time. The second queue will run for 2x while the thread will run for 3x. If a process doesn’t finish in the first queue it will be preempted and moved to the second queue. Then the next process will be moved into the first queue. This will keep happening until the process is completed.

Team Schedule

Saturday: FCFS and RR

Sunday: Output to files

Monday: FCFS and RR done (Testing)

Tuesday: Multilevel feedback (Testing)

Wednesday: Multiprocessor

Thursday: Testing

UML:

Our UML is in our GitHub Repository.