Austin Abbruzzesi
Jacob McConomy
5-24-18

Final programming project

Setup:

Files included:

main.c, Makefile

Compile instructions:

Our code has been tested on the mlb.acad.ece.udel.edu, for best results, please run this project on that (or a similar) machine.
A makefile is included with our project. Running “make” will compile main.c and generate a “simulation.exe” executable. This program will prompt you for the filename of the desired input file.

Overview:

Basic functionality:

Instructions arrive, and based on the the type of instruction, the arguments are stripped off of the line and placed in an array for easier use. If the instruction is a job arrival, the job’s attributes are validated and then a job is created. Depending on the job’s attributes, it is placed in one of the several queues. Based on current time and quantum time, a job moves across queues until finally landing in the complete queue when the job is complete. If the instruction is a resource request or release, there is a check to see if the job that is making the request is running. If the job is running and all other requirements are met, then the job is allocated or releases those resources; if the job is not running then the request is ignored. If the instruction is a type “D” instruction, statistics are printed to the console, and a json is generated for the time at which the instruction arrived.

Parsing input files:

The input file is parsed one line at a time, the numbers are stripped out using array pointers and the atoi() function, then the arguments are put into an array and returned as an array of integers
The number of arguments, and what the arguments are used for depend on the first character in the line.

Data structures:

The main structures in our program are Jobs, Nodes, and Queues.

Jobs have the following attributes: job_num, memory_needed, max_device, runtime_left, priority, quantum_left, devices_allocated, arrival_time, completion_time.

Nodes have a Job pointer and the pointer to the next node.

Queues have a pointer to the front and the rear Nodes. Queues are implemented as linked lists with an enqueue and dequeue function, as well as a print_queue function. Our implementation of linked lists was adapted from the GeeksForGeeks implementation linked at the end of this document.

There are also create_job() and create_queue() functions which return pointers to the appropriate structures.

Job processing:

When a job is created, the memory_needed and priority is checked. If the available_memory > memory_needed, and the priority is one, the job is enqueued in HoldQueue_1. If the former is true and the priority is two, then the job is enqueued in HoldQueue_2. If there is enough available memory, the job is enqueued in the ready_queue.

Shortest-job-first:

When a Job arrival is initially processed, if it enters with a priority set to one then it gets put into the hold queue. This queue uses an insertion sort method to implement the shortest job first scheduler.

Handling time:

To handle time, two lines are read from the file at one time: the line we are currently processing, and the next line to be processed. The time between this instruction and the next instruction is used to calculate the movement of jobs between the different queues.

There are four cases for how jobs are moved in one time-step (the time between the current line, and the next line):

1. If we can finish our quantum by the time the next task comes in, and we can finish executing the entire job before the next task comes in
2. If we can finish the quantum before the next task comes in but our job won't finish executing in one quantum
3. If we can't finish our quantum by the time the next task comes in, and we can finish executing the entire job before the next task comes in
4. If we can't finish our quantum by the time the next task comes in, but our job won't finish executing in one quantum

Banker's Algorithm://todo

When devices are requested, we check to make sure fulfill the request is able to be met. If there is not then we put the job into the wait queue. If you can fulfill the request the quantum is finished normally and devices are allocated to the job.

Releasing devices:

There is a global job_released integer that is changed to 1 whenever an instruction with the first character "L" is processed, or whenever a job moves to the complete queue, thus releasing its resources.

When the job_released variable is changed to 1, the hold_queues and the wait queue are checked to see if any jobs can move from those queues back to the ready_queue.

Generating JSONS:

To generate the json files, we used the snprintf() function to return the contents of the queues in [1,2,3] format and used the fprintf function to print the appropriate fields and queues. Because the json for the final "D" command is formatted slightly differently, there is an integer global variable that is changed to 1 if the first argument of the D command is "9999." when this integer is 1, the format of the json generated by generatejson() is adjusted accordingly.

Sources:

Linked List Source Code: https://www.geeksforgeeks.org/linked-list-set-1-introduction/

Our Github Repo: https://github.com/auziabbruzzesi/Operating_Systems_Final_Proj