COEN 346

OPERATING SYSTEMS

PROGRAMMING PROJECT #3

Process Scheduling

Section: YK-X

Name: Jeffrey Tang ID: 40037656

Name: Hong Yi Wang ID: 40034150

Course Mentor:

Lab Instructor:

Date Performed: 02 March 2018

Date Submitted: 23 March 2018

"We certify that this submission is my original work and meets the Faculty's Expectations of Originality.

1. OBJECTIVE

The goal of the project is to implement and simulate a priority based scheduler in a system of a single processor. The algorithm to be implemented is a simplified version of the O(1) scheduler used in versions of Linux.

2. PROCEDURE/DISCUSSION

- TECHNICAL/FUNCTIONAL DESCRIPTION
    - We have to create a class Process in order to create a Process object in the main. This way, we can update their unique traits after each burst time in the thread. Moreover, we need to create member functions in order to update their status after each object has been processed.
    - We need a scheduler function to control the flow and processing of each process and update their priority, wait time and status. Moreover, it keeps track of the 2 queues to see which is active and inactive. It is like the control center of the operations.
    - Within the scheduler, there is a need for mutex lock as it will lock the thread when there is a process inside getting processed. The mutex lock is of type mutex coming directly from the C++ library.
    - We have to create an add function to add each process of a vector into the queues. The add function will be controlled by the scheduler function. The vector will contain all processes that need to be processed.
    - We will need a sorting function in order to sort the queue according to the processes' priority.
- PROGRAM FLOW
    - In the main, the code will read from the file "pbs_inputs.txt" and identify each characteristic of 1 process and, using these characteristic, will create a Process object. The Process object will be stored in a vector.
    - Once all processes have been read, it will get the arrival time of the process stored in index 0 of the vector and save it in variable "time". It will update the number of processes that the code has read.
    - Then it goes into the while loop to see if the current index of the vector and smaller than the total number of processes and if the time is smaller than 33000. If it does, it will repeat the while loop.
    - In the while loop, the scheduler is called.
    - The scheduler will is the main control of the program. The while loop is there to repeat the process that the scheduler goes through. The loop will go on until there will be no process left. The scheduler will control

the addition of processes from the vector that saved every process from the text file with their arrival time. It will manipulate their priority and burst time as well as switching flag from the queues. The whole implementation of the scheduler will be explained in the below section.

- IMPORATNT FUNCTIONS / TEMINOLOGY

numberOfProcess is a global variable tracking the total number of processes.
currentVectorPosition is a global variable tracking the current index.
mutex mtx is a global variable of type mutex for the mutex lock.

addToQueue: The function takes in parameter processVector of type Process, the expiredQueue of type Process and the current time. The method will check in the if statement if the arrival time of the current index of the processVector is equal to the currentTime. If it does, it will push the process at that index into the expired queue. Then, it will use the sort function to sort the queue. Once sorted, the currentVectorPosition will increment and point to the next index.

```
void addToQueue(vector<Process> processVector, vector <Process> expiredQueue, int currentTime) {

    //processVector is already in order of arrival time. So, it process will be moved to the queues in order.

    if (processVector[currentVectorPosition].getArrival() == currentTime) { // check if arrival time equals currentTime
        expiredQueue.push_back(processVector[currentVectorPosition]); //insert the process in the expired queue
        sortingFunction(expiredQueue); //sort the queue by checking priority everytime a process is inserted
        currentVectorPosition++;
    }
}
```

Figure 1: addToQueue method

Scheduler:
It takes in parameter Q1 and Q2 of vector of type Process, flag1 and flag2 of type bool, currentTime and processVector vector of type Process. It checks first if the numberOfProcess is equal to 0, it it does the methods ends. Else if the flag1 is false, it will execute Q2 since Q1 will be inactive.In a for loop until it reaches the end of Q2, if at index i the status is equal to “Started”, first lock the process using mtx.lock(). Then the process will print out the Start message at the currentTime. Then it will check whether the burstTime has reached 0, if it does, it terminates the program and it will decrement the numberOfProcess. Else, the pause message will come out and pause at the currentTime.

If at index i the status is equal to “Arrived”, first lock the process using mtx.lock(). Then call the addToQueue method to add the process at the index of

processVector in Q1. Q2 will still run any process in it and if the burst time reaches 0, it terminates it and decrement the numberOfProcess.

If at index i the status is equal to “Resumed”, first lock the process using mtx.lock(). Then it print resumed and checks if the process’ burst time has reach 0, it it has, terminate it and decrement the numberOfProcess. If it didn’t, it’ll check whether Granted has reached 2, if it did, update its priority by calling the priorityUpdate function. If it didn’t, pause it and update the current time.

```
void scheduler(vector <Process> Q1, vector <Process> Q2, bool flag1, bool flag2, int currentTime, vector <Process> processV

    if (numberOfProcess == 0) {
        return;
    }

    else if (!flag1) { //execute Q2 since Q1 is expired
        for (int i = 0; i < Q2.size - 1; i++)
        {
            if (Q2[i].getStatus == "Started") {

                mtx.lock();
                Q2[i].start(currentTime);
                if (Q2[i].getBurst() == 0) {
                    Q2[i].terminated(currentTime, Q2); //delete the object from Q2 vector
                    numberOfProcess--;

                }
                else
                    Q2[i].paused(currentTime);

                mtx.unlock();
                currentVectorPosition++;
            }
            else if (Q2[i].getStatus == "Arrived") {
                mtx.lock();
                addToQueue(processVector, Q1, currentTime);
                if (Q2[i].getBurst() == 0) {
```

Figure 2: Snippet of Scheduler function

3. RESULTS (MAY INCLUDE PICTURES)

THE CODE COMPILES BUT DOESN’T SHOW ANYHTHING. WE WERE UNABLE TO SHOW ANY RESULTS, HENCE THIS PORTION IS EMPTY

4. CONCLUSION (MAY INCLUDE BELOW POINTS)

- LEARNING ASPECT
  - We learned how a scheduler works. How it controls every little tasks and verification that needed to be check before proceeding to the next task. Without a good scheduler, some process may go on starvation and never get executed by the CPU.
- CHALLENGES FACED

- Getting the whole structure of the program planned and having a coherent use of functions was very hard. Our code did not show anything, it might be because of this reason that it didn't work.
- The structure of the scheduler was very hard to do. It required many conditions that needed to be checked in order to execute a tasks.