Operating Systems Project

Phase 1

Team

Members:

Khalid Ali

Islam Ahmed

Ahmed Walid

Abdallah Zaher

Data Structure:

1. **Queue** (FIFO) For Round Robin algorithm
2. **Min Heap** (priority queue) For High Priority First, and Short Remain Time algorithms

Algorithm:

1. Process Generator

1. Process generator reads input file
2. create scheduler and give it algorithm selected, Quantum of RR algorithm and Number of processes in input file as arguments.
3. create clock process
4. send process to scheduler when it's time of its arrival (uses message Queue)
4. wait for scheduler to exit to clear resources

2. Scheduler

1. create suitable data structure based on the algorithm
2. create shared memory that holds the quantum of the running process.
3. start infinite loop
4. receive all processes -sent from process generator - that have same arrival time at once and insert them to ready queue.
5. check if algorithm is SRTN (preemptive) and there's ready processes and it's remain time is less than the current running remain time (saved in shared memory)

6. if condition in (5) is fulfilled then raise signal SIGUSER1 to save current data of running process.
7. if there isn't a running process then create/resume next process from queue

- repeat step (3)

8.when step (6) save_state function is called and it checks

    1. check if running process has finished (remain time – in shared memory- is zero)
    2. if condition (8.1) is not fulfilled then update process remain time and state and send it a stop signal re-enqueue it to ready queue
    3. if is finished then save its data and check if is the last one in the input file
    4. if last process in input file save performance file then call DestroyClk and pass "TRUE" to kill all the group.

9. in step (7) create/resume process function

    1. retrieve the next processes from the ready queue
    2. determine the quantum for it to run based on the algorithm
    3. write the quantum time to the shared memory
    4. if process state is stopped, send CNT signal to continue with the new quantum
    5. if new processes fork it.
    6. update data of current running process.

3. Process

1. access shared memory created by scheduler to get its quantum
2. get its runtime as an argument from scheduler
3. doesn’t terminate unless summation of ran quantums is equal to its runtime.
4. run for the quantum and send a signal to scheduler to notify that it has finished its quantum so the scheduler would pause it if not finished and if finished the process will terminate itself
5. uses semaphore to prevent scheduler from stopping the running process while updating its data in shared memory to ensure that when the process is resumed it starts from beginning.

Assumptions

1. Assume processes in input file is sorted by arrival time.

Workload Distribution

Khalid Ali , Islam Ahmed were responsible for the whole phase.

Time for Tasks

The whole phase was done in about 4 days.

Process generator took us the less time.

The most time was spent on the scheduler logic and DEBUGGING.