Operating Systems Project

**Phase 2**

**Team 13**

# Members:

# Khalid Ali Abdulwahab

# Islam Ahmed Mohamed

# Ahmed Walid Fathy

# Abdallah Zaher Abu Sedu

## Data Structure:

1. **Queue** (FIFO) For Round Robin algorithm

2. **Min** **Heap** (priority queue) For High Priority First, and Short Remain Time algorithms

3. **Linked List** for storing empty blocks for memory allocation

## Algorithm:

# 1. Memory Allocation

1. create 9 linked lists for each block size 1, 2, 4, 8, …., 256.

2. get index of linked list with nearest ceil(power of two) for process allocating size.

3. check if size of linked list with index from step (2) is larger than zero allocate the process in the first block in the linked list

4. if condition of step(3) not fulfilled traverse up through the linked lists to get the block with smallest size of power of two larger than the allocating size IF ANY.

5. if no size found in step (4), then can’t allocate process.

6. if a block found in step(4), keep dividing block size by 2 and store the result until we get the minimum size that fits process size.

# 2. Memory Deallocation

1. get index of linked list with nearest ceil(power of two) for process size.

2. add process block to that linked list and keep merging empty blocks with their buddies if found.

## 3. 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

## 4. 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

8. check whether it can be allocated or not.

9.If there is enough memory space for this process, then memory is allocated and it is forked and scheduled.

10.If there is not enough memory space for this process, then it is pushed to a waiting list until another process finishes and its memory is deallocated from the system.

11. when a process is finished, then check the waiting list again if there is any process that needs to be allocated from queue

- repeat step (3)

12.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.

13. 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.

## 4. 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.

2. Arrival Time start with 0

3. No process with size < 1

# Workload Distribution

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

# Time for Tasks

Allocation and deallocation 6 hours.

Integration with phase one 1 Hour.

Debugging 2 Hours.