Cairo University - Faculty of Engineering

Computer Engineering Department

Operating Systems (CMP303A)

# Project Report

## Team 8

| Name | Sec. | B.N |
|---|---|---|
| David Raafat | 1 | 24 |
| Abdallah Hussien | 2 | 2 |
| François Adham | 2 | 9 |
| Mahmoud Amr Nabil | 2 | 24 |

**Academic Year:**

**2020 – 2021**

# 1- Data Structure

We used one simple array to hold the values of WTA times to use it to calculate STD.
We have struct **Process** to hold all process data (PCB).
We have struct **Sector** which holds information about each sector in **Memory Queue** (Start address, end address, etc).
For scheduling we used queue with different priority method when pushing as following:

HPF → Priority.
SRTN → Remaining Time.
RR → Normal Queue.
Memory waiting queue (Couldn't be allocated) → Memory size

For Memory we used 1 struct **Memory** that contains 6 **Memory Queue** one for each size (8, 16, 32, 64, 128, 256).

Each **Memory Queue** consists of empty sectors that can be allocated.

All Implementations are linked list.

# 2- Algorithm

1- Process Generator
   a. Fork clk and scheduler processes and gives them any needed arguments (Algorithm to run).
   b. Send signal to inform the scheduler that he a new process arrived to the system and he sent its information by a message queue.
   c. Notifies the scheduler when last process was sent.

2- Scheduler
   a. Initialize **Memory** to have only 4 empty sectors with size 256 in the last **Memory Queue.**
   b. Declare two queues to hold **Ready** (Processes that allocated in the memory) and **Waiting** (Processes that arrives but couldn't be allocated in the memory)**.**
   c. When receive signal from process generator it goes and create new process with the sent info. and push it in **Ready** queue with the suitable method (this can happen any time while running).
   d. Check on algorithm to run the required algorithm.
      i. HPF
         1. Check if there any process in the head of the queue (Highest priorty) pop it and try to allocate it.
         2. Because HPF isn't preemptive I am sure that he will find space in the memory, so allocate it and fork it and send its initial remaining time to it.
         3. Update process data and remaining time and send the remaining time to the process every clock cycle by message queue.
         4. When the process finishes it send exit code to the scheduler to inform him that it finishes and leave the system.
         5. When the scheduler receives the exit code it deallocates the process and free its space and goes to pop again.
      ii. SRTN

1. Check if there any process in the head of the **Waiting** queue pop it and try to allocate it.
2. if there is no space push it again and pop from **Ready** queue (Shortest Remaining Time).
3. Check if it was stopped then send it **SIGCONT**, if it's new one try to allocate it.
4. If an empty suitable sector was found allocate it and fork it, if not push it in the **Waiting** queue and pop again from **Ready** queue.
5. After resuming or forking a process, the scheduler keeps track of the remaining time of the process, update its remaining time and send it to the process every clock cycle.
6. Every clock cycle the scheduler check if the remaining time of the head of **Ready** queue is less than current running process, it sends **SIGSTOP** to current process and push it into the **Ready** queue and repeats the above popping algorithm again.
7. When a process finishes its time, it sends exit code to the scheduler to inform him that it finishes.
8. When the scheduler receives the exit code it deallocates the process and free its space and goes to pop again.

iii. RR

1. Check if there any process in the head of the **Waiting** queue pop it and try to allocate it.
2. if there is no space push it again and pop from **Ready** queue.
3. Check if it was stopped then send it **SIGCONT**, if it's new one try to allocate it.
4. If an empty suitable sector was found allocate it and fork it, if not push it in the **Waiting** queue and pop again from **Ready** queue.
5. After resuming or forking a process, the scheduler keeps track of the remaining time of the process, update its remaining time and send it to the process every clock cycle.
6. Every clock cycle the scheduler check if the current process ran time equals to the quantum, it sends **SIGSTOP** to current process and push it into the **Ready** queue and repeats the above popping algorithm again.
7. When a process finishes its time, it sends exit code to the scheduler to inform him that it finishes.
8. When the scheduler receives the exit code it deallocates the process and free its space and goes to pop again.

e. It calculates performance summary and clears all resources and send exit code to process generator to inform him to finish the program and clear everything.

# 3- Assumptions

We assumed that the minimum memory size we can have is 8 bytes.

In memory we assumed that in the waiting queue biggest process with suitable size will be popped.

# 4- Work Load

| Team Member | Work load |
|---|---|
| **Abdullah Hussein** | Phase 1: scheduler algorithms and process<br>Phase 2: Memory algorithms |
| **David Raafat** | Phase 1: writing logs in files and calculate performance<br>Phase 2: writing logs<br>Phase 3: producer and consumer |
| **Francois Adham** | Phase 1: scheduler algorithms and process<br>Phase 2: Memory algorithms |
| **Mahmoud Amr** | Phase 1: process generator |

# 5- Time Table

| Phase | From date | To Date | Total Time |
|---|---|---|---|
| **Phase 1** | 25/12/2020 | 31/12/2020 | 6 days (2 days work) |
| **Phase 2** | 6/1/2021 | 10 / 1/2021 | 4 days (2 days work) |
| **Phase 3** | 13/1/2021 | 15/1/2021 | 2 days (1 day work) |