Logo

Description automatically generated Logo, company name

Description automatically generated

**OS-Project-Report**

**Team 9**

***Team Members***

|  |  |  |
| --- | --- | --- |
| Name | BN | Section |
| Abdelrahman Hamdy | 36 | 1 |
| Abdelrahman Noaman | 2 | 2 |
| Adham Ali Abdelaal | 12 | 1 |
| Eslam Ashraf Ibrahim | 13 | 1 |

5/30/2022

Data Structures Used (found in headers.h)

1. Queue
2. Priority Queue
3. Process Data

* Arrival time
* Priority
* Run time
* ID
* Memsize

1. Memory Node

* Size
* Start
* Pointer to the next memory node

1. PCB for each process

* ID
* PID
* Arrival time
* Burst time
* Finish time
* Running time
* Stop time
* Priority
* Start time
* Start Address in memory
* Memory Size

Algorithms Explanation

1. HPF

* The main loop checks if the processes are all finished or not. In an inner loop, we get all the processes arrived at this particular second and enqueue them in the ready queue
* If there is no current process and the ready queue isn’t empty, we directly pop from the ready queue as it’s already sorted with the minimum priority, then start the process, fork it, and execl Process.c
* Last check: If the remaining time of the running process is zero, then it is finished and the running process is set to NULL. The finished processes counter is incremented as well and the loop continues.

1. SRTN

- Receiving the processes from the process generator

- Checking if the coming process's burst time is smaller than the remaining time of the running process

- If it's smaller than the running process will be added to the ready queue and saved in stop resuming queue and the smaller one will be the running process

- Then every time we a process starts then we check if this process was stopped before “existing in stop resuming queue" or not to know if it started or resumed to resume the stopped process

1. RR

* Check if there are arriving processes, receive them in ready queue
* Check if there is a processes finished or the time slot ended, change the state of the process from running to finished or ready respectively
* Check if there are processes in ready queue and no process is running so pick up one of them and run it

Workload Distribution

|  |  |
| --- | --- |
| Abdelrahman Hamdy | * Process Generator * HPF * Memory calls * Memory log file |
| Abdelrahman Noaman | * SRTN * Data Structures * Process.c * Scheduler perf file |
| Adham Ali | * HPF * RR * Main scheduler functions * Memory allocation |
| Eslam Ashraf | * HPF * RR * Scheduler log file * Memory deallocation |

Assumptions

* In the memory waiting queue, it is implemented as a priority queue based on the algorithm so if it’s a SRTN, the priority is the remaining time while for RR, it is based on the memory size where the smaller one gets put into the ready queue first. As for the HPF, there is no need since the running process is the only one put into the ready queue.
* We made an array of arrivals in the process generator as a shared memory with the scheduler in order to make sure that any process arriving at a specific time is read by the scheduler and not skipped.
* We synchronize between the stopped process and the arrived one if they come in the same second so that the stopped process is put into the queue before the arrived process.

Time taken for each task

|  |  |
| --- | --- |
| Task | Time Taken |
| Process Generator | 8 hours |
| HPF | 2 hours |
| SRTN | 6 hours |
| RR | 6 hours |
| Scheduler | 7 hours |
| Data Structures | 1-2 hours |
| Scheduler log | 1 hour |
| Scheduler perf | 1 hour |
| Memory log | 3 hours |
| Memory Allocation | 5 hours |
| Memory Deallocation | 6 hours |