

**Operating Systems**

**CMPS303**

Project Report Phase 1

TEAM 4

|  |  |
| --- | --- |
| **Team Member** | **ID** |
| Sarraa Muhammad Tareq | 1200029 |
| Ahmed Waleed Wafeek | 1200107 |
| Amr Ashraf Attia | 1200042 |
| Karim Magdy Mounir | 1210070 |

Table of Contents

[Data Structure 3](#_Toc165635410)

[Algorithm Explanation 5](#_Toc165635411)

[HPF (Highest Priority First) Algorithm 5](#_Toc165635412)

[SRTN (Shortest Remaining Time Next) Algorithm 5](#_Toc165635413)

[Round Robin Scheduling Algorithm 6](#_Toc165635414)

[Responsibility of each file 7](#_Toc165635415)

[Results 9](#_Toc165635416)

[processes.txt (Input File): 9](#_Toc165635417)

[Output files: 9](#_Toc165635418)

[HPF algorithm (The schedular.log) 9](#_Toc165635419)

[HPF algorithm (The schedular.perf) 10](#_Toc165635420)

[SRTN algorithm (The schedular.log) 10](#_Toc165635421)

[SRTN algorithm (The schedular.perf) 10](#_Toc165635422)

[RR algorithm at quantum=3 (The schedular.log) 11](#_Toc165635423)

[RR algorithm at quantum=3 (The schedular.perf) 11](#_Toc165635424)

[Assumptions 12](#_Toc165635425)

[Workload Distribution 14](#_Toc165635426)

[Time Taken for Each Task 15](#_Toc165635427)

# Data Structure

1. **struct processInputData**: This structure is used to represent the input data for each process, including the process ID, arrival time, run time, and priority.
2. **struct processData**: This structure represents a processed version of the input data. It includes additional fields such as the process ID, process ID (PID) of the process when it's running, remaining time, turnaround time, finished time, waiting time, weighted turnaround time, current state (e.g., ready, started, finished), and a quantum value (for the Round Robin scheduler).
3. **struct Node**: This structure is used to create linked list nodes. Each node contains a pointer to a process data structure and a pointer to the next node in the list.
4. **struct List**: This structure represents a linked list. It contains a pointer to the head of the list, the size of the list, and a flag indicating whether the list should prioritize processes based on remaining time (for the Shortest Remaining Time Next scheduler) or priority (for other schedulers) or if it is a normal linked list.

**For RR**: a queue data structure. Specifically, a regular queue to manage the processes and their execution order. This queue is being used to store the processes and determine the order in which they will be executed based on the RR algorithm.

1. **struct msgbuff:** This structure is used for message passing between the process generator and the scheduler. It contains an integer message type (mType) and a processData structure representing information about a process.
2. **struct msgBuff1:** This structure is used for message passing between the scheduler and individual processes. It contains an integer message type (mType) indicating the PID of the process to which the message is sent, and an integer decrement value used in decrementing the remaining time of a process.
3. **struct processData:** This structure is used to store information about each process that needs to be scheduled. It includes fields such as id (process ID), arrivalTime (time at which the process arrives), runTime (total runtime of the process), priority (priority of the process), remainingTime (time remaining for the process to complete), finishedTime (time at which the process finishes execution), turnAroundTime (total time taken for the process to complete), waitingTime (time spent waiting in the ready queue), weightedTurnAroundTime (turnaround time normalized by the total runtime), and state (current state of the process, e.g., ready, running, or finished). This structure helps manage and track the state and progress of each process in the scheduling system.

# Algorithm Explanation

## HPF (Highest Priority First) Algorithm

1. **Process Arrival:** While there are messages in the message queue, the algorithm dequeues them and processes the incoming processes. Each new process is allocated memory, its state is set to "ready", and it is enqueued based on its priority.
2. **Running Process:** If there is a process currently running (**isRunning** is true), the algorithm sends a message to the running process to decrement its remaining time. If the process finishes (remaining time reaches 0), it waits for the process to finish, updates its state to "finished", records the finish time, and increments the **finishedProcesses** counter.
3. **Selecting a New Process:** If there is no process currently running, the algorithm dequeues the highest priority process from the **processQueue** and forks a new process to execute it. The process state is set to "started", and the process ID is updated. If forking fails, an error message is printed.

## SRTN (Shortest Remaining Time Next) Algorithm

1. **Process Arrival:** Similar to HPF, the SRTN algorithm processes incoming processes from the message queue. Each new process is allocated memory, its state is set to "ready", and it is enqueued based on its remaining time.
2. **Running Process:** If there is a process currently running (**isRunning** is true), the algorithm sends a message to the running process to decrement its remaining time. If the process finishes (remaining time reaches 0), it waits for the process to finish, updates its state to "finished", records the finish time, and increments the **finishedProcesses** counter.
3. **Selecting a New Process:** If there is no process currently running, the algorithm selects the process with the shortest remaining time from the **processQueue** to run next. If there are processes in the **processStoppedQueue** (previously stopped processes), and the shortest remaining time in the stopped queue is less than or equal to the shortest remaining time in the ready queue, a process from the stopped queue is resumed. Otherwise, the algorithm dequeues the process with the shortest remaining time from the ready queue and starts it.

## Round Robin Scheduling Algorithm

1. **Process Arrival:** Similar to the other algorithms, the Round Robin algorithm processes incoming processes from the message queue. Each new process is allocated memory, its state is set to "ready", and it is enqueued in the **processQueue**.
2. **Running Process:** If there is a process currently running (**isRunning** is true), the algorithm sends a message to the running process to decrement its remaining time and quantum. If the process finishes (remaining time reaches 0), it waits for the process to finish, updates its state to "finished", records the finish time, and increments the **finishedProcesses** counter. If the quantum for the running process reaches 0, the process is preempted, its state is set to "ready", and it is reinserted into the **processQueue**.
3. **Selecting a New Process:** If there is no process currently running, the algorithm dequeues the next process from the **processQueue** and starts it. The process state is set to "started", and the process ID is updated.

## Responsibility of each file

**Makefile:** Has the commands needed to run the whole project.

**Clk.c**: Added a semaphore in order to synchronize it with the scheduler.c: This semaphore

was used in order that each scheduler cycle takes exactly one clock cycle.

**process.c:** Runs when the schedular gives it a sign to start and when it finishes (remaining

time is equal to zero) it is responsible for telling the scheduler that the process ended.

**scheduler.c:** This file was forked by the process\_generator.c and it is mainly responsible

for forking each process based on the input algorithm that was sent from the process

generator. It communicates with 3 other files each with a certain way.

* It communicates with the process\_generator.c along with a message queue using

(struct msgbuff) to receive the processes that were read earlier by the

process\_generator from the processes.txt.

* It communicates with the process.c with another message queue through the

message buffer (struct msgBuff1) in order to inform the currently running process

to work for 1 seconds. This helps the scheduler to keep track of the running process

and the remaining time for each process.

* It communicates with the clk.c through a semaphore sem\_sync that was used to

synchronise the scheduler cycles along with the clock cycle as explained before.

**process\_generator.c:** The file reads the input file , then asks the user for the chosen

scheduling algorithm and its parameters, if there are any. It initiates and create the

scheduler and clock processes. It provides each process with its parameters. It sends the

information to the scheduler at the appropriate time (when a process arrives), so that it

will be put it in its turn. At the end, it clears IPC resources.

**headers.h:** This file was included in many other .c files, we used it to include our List

& Queue and node implementation along with their functionalities that were designed to

satisfy all algorithms. In addition to this, we included the structs created in this file in

order to be used among other files without the need to re-write it again in each file. For

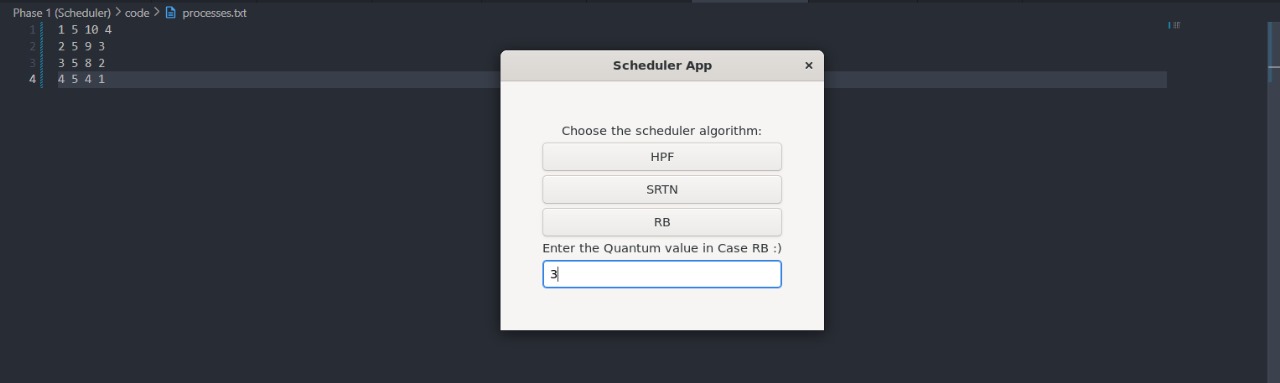
example: struct processData, struct processInputData, struct msgbuff, struct

msgBuff1.

**test\_generator.c:** creates processes.txt ( which is the input file)

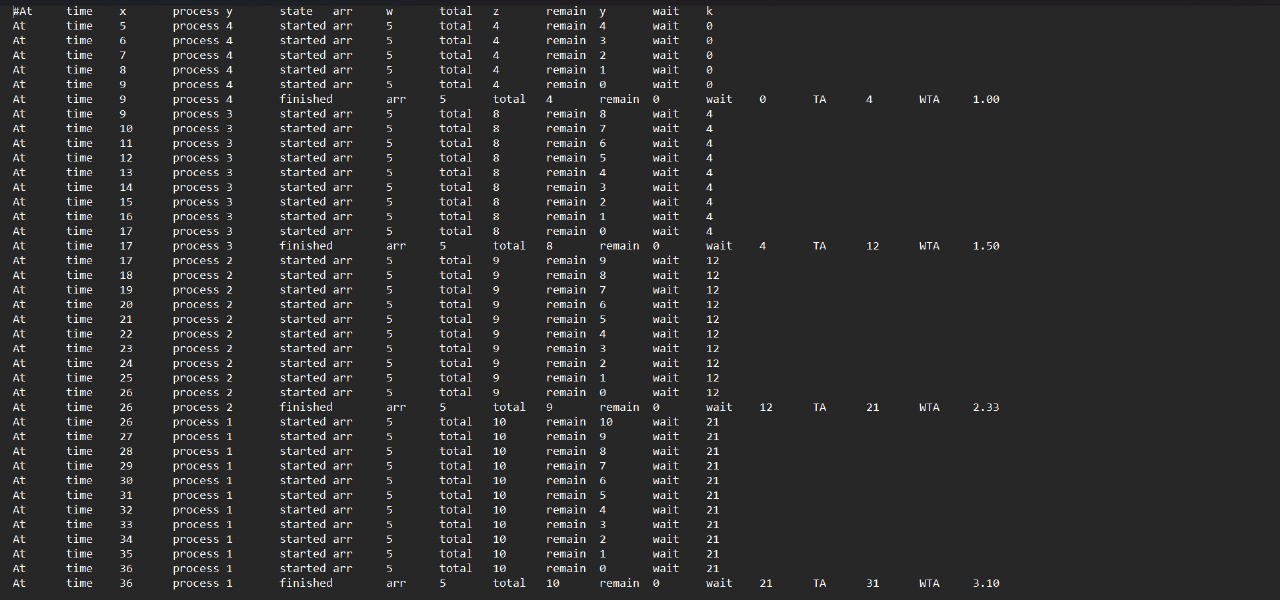
# Results

## processes.txt (Input File):

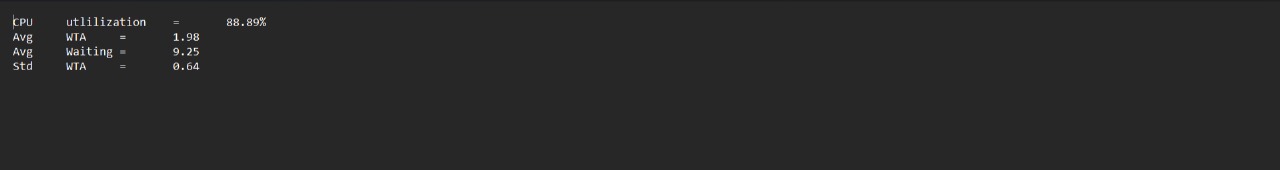


## Output files:

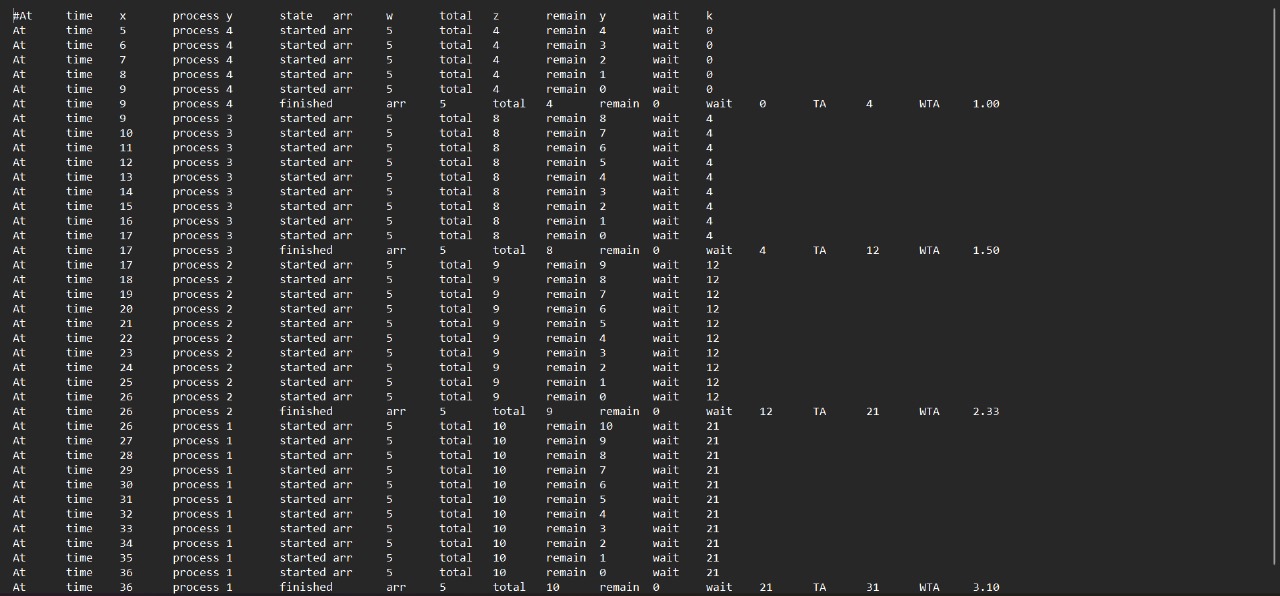
### HPF algorithm (The schedular.log)



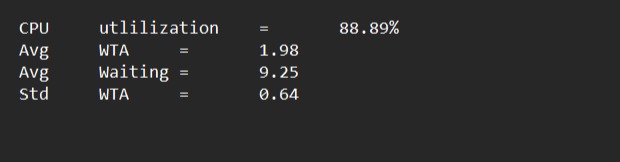
### HPF algorithm (The schedular.perf)



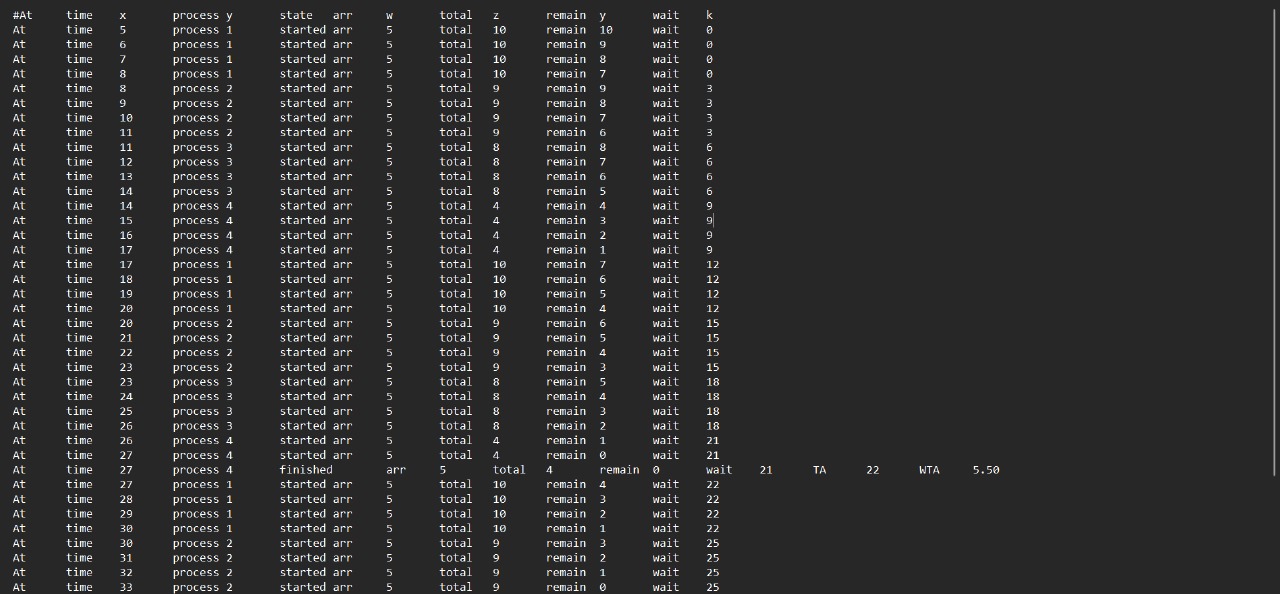
### SRTN algorithm (The schedular.log)



### SRTN algorithm (The schedular.perf)



### RR algorithm at quantum=3 (The schedular.log)



### RR algorithm at quantum=3 (The schedular.perf)



# Assumptions

**General assumptions:** The least arrival time is at t=1

**HPF assumptions:** If two processes arrive at the same time and are both ready to be scheduled, the algorithm will select the process that was enqueued first.

**SRTN assumptions:** if two processes arrive at the same time, the behavior depends on the remaining time of the processes in the system.

1. **Both Processes Have Equal Remaining Time:**
   * If both processes have equal remaining time, the algorithm will select the process that was enqueued first.
2. **One Process Has Less Remaining Time:**
   * If one process has less remaining time than the other, the algorithm will select the process with less remaining time.
3. **Both Processes Have Equal Remaining Time and One is Running:**
   * If one of the processes is already running and another process arrives with equal remaining time, the running process will continue to execute until its remaining time becomes less than the new process, or until it completes execution.
4. **Both Processes Have Equal Remaining Time and None is Running:**
   * If neither process is running and both have equal remaining time, the algorithm will select the process that was enqueued first.

**Round Robin assumptions:**   
If two processes arrive at the same time, they will both be added to the process queue. The round-robin scheduler will then schedule the processes in the order they were added to the queue, giving each process a quantum of time to run. If the processes' execution times are short enough, they may both complete their execution without any issues. However, if the processes' execution times are longer than the quantum, they will be preempted and added back to the end of the queue to await their next turn to execute. This will continue until all processes have completed their execution.

# Workload Distribution

|  |  |
| --- | --- |
| **Team Member** | **Tasks** |
| Sarraa Muhammad Tareq | RR Algorithm  headers.h  Creating Test cases & testing communcation between process generator & schedular |
| Ahmed Waleed Wafeek | SRTN Algorithm  synchronization between schedular and clk  communication between schedular & process  SRTN Algorithm Debugging & OS Design of the schedular  GUI |
| Amr Ashraf Attia | clk.c (creating semaphore)  Calculations (log and perf)  Report  HPF Algorithm |
| Karim Magdy Mounir | process.c  process\_generator.c |

# Time Taken for Each Task

|  |  |
| --- | --- |
| Task | Time Taken |
| Headers File | 1 hour |
| Synchronization | 1 day |
| Calculations | 2 hours |
| Creating Testcases | 1 day |
| Testing | 2 days |
| Process Generator | 4 hours |
| Process File | 3 hours |
| HPF | 1 day |
| STRN | 5 hours |
| RR | 2 days |