

# OS Assignment-3 Design Document

**Topic:** SimpleScheduler: A Process Scheduler in C from Scratch

**Team Members:** Medha Kashyap (2022292)  
Himanshu Raj (2022216)

**Group Number:** 10

**GitHub Repository Link:** <https://github.com/rahi-senpai/OS>

## **Members Contribution:**

- Medha Kashyap: Starter code, documentation, design doc, error handling
- Himanshu Raj: Round robin implementation and advanced functionalities

**References:** API man pages (SHM, semaphore, signal, kill APIs)  
Lecture slides

## **Explanation:**

We have used shared memory to communicate between shell and scheduler processes. Scheduler is launched when you launch the shell. We have shared the history array (contains everything related to a process) between processes and used the kill API to send SIGCONT and SIGSTOP signals to processes with their PIDs after a time quantum (which is taken as input in milliseconds). Ready queue is a priority queue and running queue is a normal queue. For scheduling policy we have implemented a simple (naive) version of linux CFS, where we run a process from the ready queue till the specified tslice. We considered vruntime to be the comparing attribute and extract the processes with minimum vruntime to enqueue in the running queue and number of maximum processes in the running queue is also taken as input.

Execution time is the CPU burst time of a process. We have used semaphores every time we access shm so it can affect time due to sem\_wait API.

We have not used dummy\_main.h in our implementation.

## Statistics (Conclusions):

```
rahi@rahi:/mnt/d/OS/assignment3$ ./shell 2 1000
Initializing simple scheduler...
Initializing simple shell...
os@shell:~$ submit ./fib 1
os@shell:~$ submit ls 4
os@shell:~$ Makefile dummy_main.h      fib fib.c question.pdf scheduler shell simpleScheduler.c simpleShell.c
701408733
^C
Caught SIGINT signal for termination
Terminating simple scheduler
Exiting simple shell...

Command PID      Execution_time Waiting_time
submit ./fib 1    304         5289ms    0ms
submit ls 4       305          8ms   1019ms
rahi@rahi:/mnt/d/OS/assignment3$
```

This screenshot shows that a short job with least priority is executed before a long job with highest priority implying fair scheduling policy (no starvation).

```
^C
Caught SIGINT signal for termination
Terminating simple scheduler...
Exiting simple shell...

Command      PID      Execution_time  Waiting_time
submit ./fib 1    135693      4188ms       717ms
submit ./fib 2    135918      4234ms       290ms
submit ./fib 3    136060      4319ms       434ms
submit ./fib 4    136131      4309ms       518ms
```

This screenshot shows that resources are shared very fairly across processes in the ready queue as the wait time for the same processes with different priorities is almost the same, and execution time varies a little due to context switch overheads.