OS Assignment 3 - SimpleScheduler

Group Details

Group - 103
Members : { Arjun Tandon <2022095>, Chahat Ahuja <2022138> }

Application Design

1. The program uses a shared file `a.txt` to communicate between the shell and the scheduler. It acts as a buffer which stores the executable name and their priorities that the scheduler has to run. Here the shell acts as a producer, the scheduler acts as the consumer
2. Proper locking is used to ensure concurrency is maintained and no race condition arises
3. The scheduler after each time slice checks for input from this shared file
4. It parses that input and creates a `process_node` structs and adds it to 2 queues, one is the `SQ (Scheduler Queue)`, the other being the `UQ (Universal Queue)`
5. A `status` value of 1 implies that the program has been terminated.
6. To manage `priority`, suppose the priority of a program is `3`, if the scheduler encounters it in a round robin iteration, it will make sure that it will include it in the process-batch `3` times before it sends it to the rear of the queue.
7. The statistic to see how priority handling benefits the high priority programs can be seen if we see the output of the scheduler in case the priorities of all programs were set as `1`
8. The scheduler first stores the `original rear` of the queue
9. It comes to know that it has completed one pass of the whole queue when in the last batch it processes, it encounters the `original_rear`
10. Scheduler is notified of the termination of the processes via `SIGCHLD` handler
11. It sets the status of those processes as 1 and just doesn't enqueue them
12. On exit, the cleanup occurs and all the `process_node`s and `process`es are freed

IPC Implementation

To implement a shared file IPC between the shell and the scheduler.
Here's the flow that occurs

1. Shell opens the file and creates it if it doesn't exist
2. Shell clears the file contents by using `ftruncate`
3. Shell locks the file for writing
4. Writes the executable names and priorities given to it by the user via `submit`
5. Unlocks the file for writing
6. Scheduler locks the file for reading purposes after each time slice
7. Reads the contents of the file and adds them to the process_queue
8. Just before unlocking the read-lock of the file, scheduler sends a `SIGSTOP` signal to it's parent, i.e., the shell
9. It quickly unlocks the file for reading, opens it again in write mode.
10. Locks it for writing and clears it by using `ftruncate`

11. It unlocks the file and sends a `SIGCONT` signal to the parent, i.e. the shell

Execution of the project

```
gcc main.c texts.c history.c custerrors.c filelocks.c

gcc -o ss simplesched.c -lm subsched.c filelocks.c filelocks.h texts.c

./a.out
```