Report

Syscalls implemented:

1) waitx(int *wtime, int *rtime)
2) getpinfo(struct proc_stat *p)
3) setpriority(int priority)

waitx() is similar to the wait() syscall. waitx() assigns the wtime and rtime pointer to the waiting and running times of the process called.

getpinfo() assigns values to a proc_stat structure with data from the current process.

setpriority() assigns priority to the current process. If the newly assigned priority is lesser than the old priority, rescheduling takes place if PBS.

Scheduling Algorithms implemented:

1) FCFS: Schedules the processes with the lowest creation time(lowest pid) to be run first. Disables the yield() function inside trap.c, i.e. waits till the process changes its state from RUNNING.
2) PBS: Schedules the processes with lower priority to be run before the others. Runs the processes with same priority in a round-robin fashion.
3) MLFQ: Maintains 5 priority queues, runs the process at the head of the highest priority queue. Each queue has a seperate timeslice(1 tick for queue 0, 2 for 1, 4 for 2, 8 for 3, 16 for 4). If a process exhausts its timeslice, it is forced to yield, and is pushed to a lower priority queue. If a process is forced to wait for more than 30 ticks, it is pushed into a higher priority queue, and removed from the current queue.

FCFS on average performed worse than PBS and sometimes performed worse than Round Robin.
FCFS had a varied performance.
Our implementation of PBS doesn't grant it the effciency it can achieve.(priority based on cpu : i/o time)

The benchmarking consists of a parent process creating a number of forks and trying to run them simultaneously, each fork has varying sleeping(IO) and cpu burst subparts.

Point asked:
A program can run a few times and observe the amount of time it gets cpu access in each queue before being sent to the next queue.
Then on the basis of the queue it can organise its cpu and i/o(or any action that leads to it voluntarily relinquishing control) leading it to go back of the same queue, hence getting a lower prioirity and hence more preference to its execution leading to better throughput.