Report

The report consists of all the edits and implementations of the specifications mentioned in the assignment.

Spec 1: System Calls

Addition of System calls

Process of adding system calls:

1. Define the function in the syscalls.h and declarations syscalls.c
2. Write the function implementation in the sysproc.c
3. Make the valid changes that were required in the remaining files for the implementation
4. Changes were made in `proc.h`, `proc.c->fork()`,`proc.c->allocproc()`, `trap.c->usertrap()`.
5. They are nothing but the initializations and updating them.

Added System Calls

Trace

1. Added `strace.c` in user for testing the syscall.
2. Made required changes in `MAKEFILE` to make this file executable.

Sigalarm and Sigreturn

1. Added `alarmtest.c` in user for testing the syscall.
2. Made required changes in `MAKEFILE` to make this file executable.

Spec 2: Scheduling

The following are implemented

FCFS

1. The basic concept is to run the process that is first created
2. We add a new variable to the `struct proc` in `proc.h` (creation time is noted)
3. Initializing the creation time to 0 when the process is allocated in allocproc
4. The `scheduler()` function is modified by choosing the function with the minimum creation time
5. Disabling the preemption (as it is FCFS) in `kerneltrap` and `usertrap` from the `trap.c` file.

PBS

1. Made a non-preemtive method PBS
2. Added some variables to the struct like sched_time which says number of times it is called,sleep_time, priority, and niceness are implemented
3. Made a new scheduling logic for PBS in the `scheduler()` which schedules according to the rules of dynamic priority.
4. The added variables are found by using `update_time` function
5. Made syscall `set_priority` which changes static priority of the process accordingly.
6. The basic logic is to run the process with the highest priority

MLFQ

1. The requirements of this are Queue Implementation
2. New variables like which queue, ticks used, time spent in each queue
3. These are initialized accordingly
4. Using 5 queues
5. Aging is implemented in the code at acts preemptively
6. The process in the highest priority is implemented
7. Modifications are made accordingly in the following functions `allocproc` , `scheduler()`, `clockintr`, `kerneltrap`, `usertrap`.

8. It is being tested on only 1 CPU.

LBS

1. Things which are being used in this are inbuilt `rand`
2. The following codes are implemented `set_tickets`
3. `tickets` is used in struct proc
4. sys_call `set_tickets` is used
5. The time slices are given according to the probability proportional to tickets in each process

SCHEDULER Analysis

We have tested the timings, wait time, and run time.
`waitx` is used which is provided in the tut.

```
Benchmarks


CPUS = 1

  |         Scheduler          | <rtime> | <wtime> |
  | :-----------------------: | :-----: | :-----: |
  |            RR              |   12    |   150   |
  |           FCFS             |   26    |   119   |
  |            PBS             |    13   |   127   |
  |           MLFQ             |    13   |   142   |
  |            LBS             |    12   |   147   |

CPUS = 3

  |         Scheduler          | <rtime> | <wtime> |
  | :-----------------------: | :-----: | :-----: |
  |            RR              |   11    |   111   |
  |           FCFS             |   29    |   34    |
  |            PBS             |   14    |   105   |
  |            LBS             |   11    |   111   |
```

Spec 3: Copy-on-write fork

Idea :

1. When a parent process creates a child process then both processes initially will share the same pages in memory.
2. These `shared pages will be marked as copy-on-write`.
3. If any of these processes will try to modify the shared pages then copy of these pages will be created
4. The `modifications will be done on the copy of pages` by that process.
5. Thus not affecting the other process.

Modifications made :

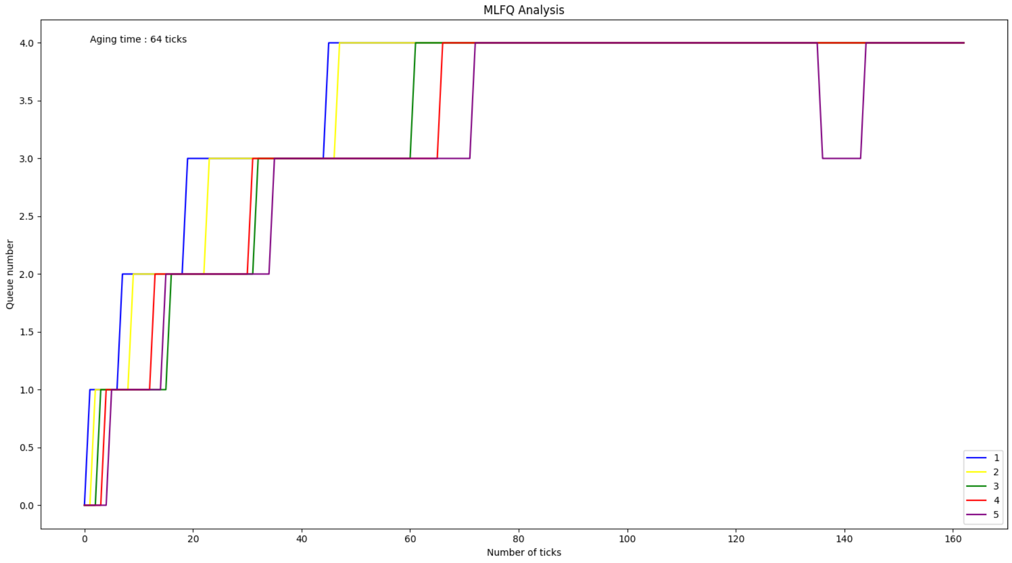
1. Updated the `uvmuncopy()`, `copyout()` functions in vm.c
2. Declared and implemented the function `page_fault_handler()` in `trap.c`
3. Modified `trapinit()` in `trap.c` with some conditiong on `r_scause()`
4. Declared and implemented the functions `init_page_ref()`, `dec_page_ref()`, `inc_page_ref()`, `get_page_ref()` in `kalloc.c`
5. Used the above functions in `kfree()` and `kinit()` in `kalloc.c`

Cowtest

1. Added `cowtest.c` in user for testing the syscall.
2. Made required changes in `MAKEFILE` to make this file executable.
3. For the implementation of COW fork, we used the online sources git repos of MIT students.

MLFQ Analysis:

Timeline graphs for processes that are being managed by MLFQ Scheduler.



Question in the Assignment:

If a process voluntarily relinquishes control of the CPU (example: For doing I/O), it leaves
the queuing network, and when the process becomes ready again after the I/O, it is inserted at the tail of the same queue, from which it is relinquished earlier.

Solution: The above case can be used by process when it yields its CPU time before (Allocated time as in for I/O). Now when it is back in the list we have added it tot the back of the queue with same queue number

Teammates:

1. Sai Madhusudan Gunda(2021111028)
2. Devisetti Sai Asrith(2021111022)