OSN Assignment-4 Report

The report mentions all the details and implementations of the specifications as required in the assignment

Specification 1: Addition of System Calls

A system call is way for programs to interact with operating system. A computer program makes system call when it makes request to operating system's kernel. System calls are used for hardware services, to create or execute process, and for communicating with kernel services, including application and process scheduling.

In order to define your own system call in xv6, you need to make changes to 5 files. Namely, these files are as follows.

- `kernel/syscall.h`
- `kernel/syscall.c`
- `kernel/sysproc.c`
- `user/usys.pl`
- `user/user.h`

System calls added:

Trace:

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

Sigalarm and Sigreturn:

- Added `alarm.c` in user for testing the syscall.
- Made required changes in `MAKEFILE` to make this file executable.

Specification 2: Scheduling

- RoundRobin is followed as the default scheduling algorithm for the xv-6 given to us.
- We have added the following scheduling algorithms.
    - FCFS - First Come First Serve

    - PBS - Priority based Scheduling
    - LBS - Lottery based Scheduling
- We need to add the changes / the codes for various scheduling algorithms in the `scheduler()` in the `proc.c` file.

FCFS:

- The idea of FCFS is to allocate CPU to the processes based on their increasing order of creation time.
- First, we added a new variable called `start_time` in the process control block of the process i.e., `struct proc()`.
- Now, If the scheduler is set as FCFS, Now we traverse through all the processes and select a process with the least `start_time` and allocate CPU to it by following the acquire and release locks.
- Thus, FCFS is realised through this algorithm.

PBS:

- The idea of FCFS is to allocate CPU to the processes based on their priorities (while lowest priority value having the highest priority).
- First, we added new variables called,
    - `rtime,` - the run time of the process till then.
    - `stime,` - the sleep time of the process till then.
    - `spriority,` - the priority of the process (default it is set to 60).
    - `calls,` - the number of the the process has been allocated CPU.
    - `nice` - the niceness of the process (default is set to 5) which decides the dynamic priority of the process.

  which are the in the process control block of the process i.e., `struct proc()`.
- Added a system call `set_priority` - using same steps as specified in spec1.
- The `set_priority` system call is used to update the priority of a process and returns the previous priority of the process.
- The `set_priority` calls the function `set_spriority` which does its work.
    - `int set_spriority(int new_priority, int pid)`
- Functions added in the `proc.c`

- `update_time()` - updates the rtime and stime for the processes.
- `int set_spriority(int new_priority, int pid)` - assigns new priority to the process.

- The niceness is calculated as follows:

```
uint64 nice_factor;
if (p->calls) {
    if(p->rtime + p->stime ==0)
        nice_factor=5;
    else
        nice_factor = (p->stime/(p->stime+p->rtime))*10;
}
else {
    nice_factor = 5;
}
p->nice = nice_factor;
uint64 DP = max(0, min(p->spriority - p->nice + 5, 100));
```

- The meaning of niceness values are:
  - 5 is neutral
  - 10 helps priority by 5
  - 0 hurts priority by 5
- In this way, the priority gets updated and the CPU is allocated to the processes with highest priority thus, realising the priority based scheduling.

LBS:

- Each process is given of tickets which may have the golden ticket, now we will run a random generator for the golden ticket and allocate CPU to the process having the golden ticket.
- First, we added a new variable called `tickets` in the process control block of the process i.e., `struct proc()`.
- Added a system call `set_tickets` - using same steps as specified in spec1.
- Functions added in the `proc.c`
  - `random_gen` - to generate the golden ticket.
- Now, we iterate through the processes and check which process has the golden ticket and allocate the CPU to the particular process.
- In this way we have realised the lottery based scheduling.

Scheduler Analysis:

- We have used the `waitx` file provided in the tutorial and added `scheduler.c` to analyse the various scheduling algorithms implemented.

- Some of the analysis is shown here,

  3 CPUs

Scheduler	ROUNDROBIN	FCFS	PBS
avg rtime	109	34	105
avg wtime	14	28	14

  1 CPU

Scheduler	ROUNDROBIN	FCFS	PBS	LBS
avg rtime	147	126	126	147
avg wtime	13	13	13	13

Specification 3: Copy-on-write fork

- Basically when we fork a process, parent and child processes are created the default xv6 copies the page table of the parent and assigns it to the child process, which inculcates data redundancy.
- The idea of this specification is to create copy of the page table of the parent only if required (when child process involves writing into the page table) else it just reads the page table and do not cultivate any changes to it.
- So, we create a `shared memory of the page table and assign it to the child process` whenever possible.
- This helps in increase of the available memory as compared to the previous case.

Implementation:

- Updated the `uvmcopy(), copyout()` functions in vm.c.
- Implemented the function `page_fault_handler()` in `trap.c`.
- Modified `trapinit()` in `trap.c` with else if condition on `r_scause()`.
- Declared and implemented the functions,
  - `init_page_ref()`
  - `dec_page_ref()` and `inc_page_ref()` in `kalloc.c`.

Cow-test:

- Added `cowtest.c` in user for testing the syscall.

- Made necessary changes in `MAKEFILE` to make file executable.