REPORT

Requirement-1: System Call

- For implementing the trace system call,and strace userprogram I changed the following files:

1. kernel/syscall.c:

- This file generally contains all the function pointers to all the system call functions so, addition of any system call need to have a pointer.

```
[SYS_trace]    sys_trace
```

2. kernel/syscall.h:

- In this file all the indices to the array of function pointers in syscall.c are defined, For this I have given a new index to my trace system call.

```
#define SYS_trace 22
```

3. kernel/sysproc.c:

- In this file the implementation of system call is written.Any type of arguments like mask is read here with the help of argint() function written in syscall.c and passed to function written in kernel/proc.c.

4. kernel/proc.c:

- The actual logic of all systemcalls is written here.

5. system/defs.h:

- Any type of function added in the proc.c should contain a header in defs.h. So I added definition of function here also.

6. user/usys.pl

- It is generally written in perl language which does the conversion to Assembly language which acts as an interface between kernel and userprograms.
- So for the userprogram to interact with trace system call, I have added an entry in this file.

```
entry("trace")
```

7. user/user.h

- Any type of user function's prototype need to be added in user.h.So I added a prototype

```
int trace(int);
```

8. Makefile

- Any user program added need to have an entry in the makefile -> u_progs. So I added a entry here.

    _strace\

Actual Implementation of trace:

- Our job is to trace every system_call invoked according to mask given.
- In general any system call comes to syscall.c before executing it's function So I have written a print statement to display systemcall name, systemcall arguments invoked according to the mask given.
- And to get the tracemask everytime I have added a field called tracemask in struct proc in kernel/proc.h.(which generally holds all the information of the process).

strace.c

This is a userprogram which accepts command line arguments from the user and calls the trace system call accordingly.

Requirement-2: Scheduler

In general the scheduler used in xv6 OS is Round Robin.

- It is generally written in kernel/proc.c in a function named scheduler.
- So I have written all the three scheduling policies in this function
- So to select the scheduler I used #ifdef and #endif everytime and modified accordingly in makefile to get the type of scheduling policy from the user.

1. FCFS

As FCFS generally schedules all the process based on the creation time of the process, so I defined a field called ctime in struct proc in system/proc.c which keeps track of the creation time of every process .

So whenever I need the creation time of the process I used (&proc->ctime).

I will initialize ctime with 0 in allocproc function in syscall.c.

Actual Implementation of FCFS

I just iterated through all the process available and just schedule the process based on the lowest creation of all the available process.

So whenever I find my lowest process I will just swap (&proc->context)with the (&cpu->context) which handles the context switching process.

2. PBS

PBS generally schedules process based on the priority of the process, so I have created a field called static_priority in struct proc in system/proc.c which keeps track of priority of each process.

So whenever I need the static_priority of the process I used (&proc->static_priority).

So to get the niceness and dynamic priority of each process we require how much time the process is in sleep mode. So to keep track of sleep time I defined two fields named s_time(process sleeping time since it is last scheduled) and last_scheduled_time(time when the process is last scheduled).

I will initialize rtime with 0,s_time with 0,last_scheduled_time with 0 in allocproc function in syscall.c.

And to keep track of s_time I placed in s_time with the current ticks in sleep() function and last_scheduled_time with the current ticks in wake() function.

Actual Implementation of PBS

I just iterated through all the runnable process available and calculate dynamic priority of each process by the given formula and I will findout the highest priority process.

To calculate sleep time,sleep time = (p->s_time – p->last_scheduled_time).

So whenever I find my highest priority process I will just swap (&highest_priority_proc->context)with the (&cpu->context) which handles the context switching process.

For static priority it is set to default (60) and can be changed with setpriority command whose syntax is setpriority [priority] [pid].

Implementaion of setpriority system call:

I just changed all the files accordingly as mentioned above for the strace function.

This system call generally takes pid,priority as input to the system call and returns a old_static priority.

In this system call it changes the priority and modifies the niceness and dynamic priority accordingly.

3.MLFQ

To keep track of entry time of each process I defined field called entry_time in struct_proc in kernel/proc.c

IMPLEMENTATION OF MLFQ

- The processes are relocated to other queues based on CPU bursts in this preemptive scheduling.
- There are no implemented queues, but each process has a variable for its current queue in struct proc.

- Every queue has a varied number of time slices, such as a queue of 0 to 1 ticks. 1–2 tick line a 2-tick queue 4 - 16 tick queue, 3 - 8 tick queue
- Aging is used to prevent processes from waiting in queues indefinitely. If a process's entrance time in the current queue is greater than 16 ticks, it is promoted to a higher priority queue.
- Demotion occurs if the time spent in the current queue is greater than two times the current queue number, whenever a process has finished its time slice, or whenever there is an interruption.
- Entry_time determines the process's position in the queue.
  when it is scheduled, it is reset to the current time, which eliminates the wait time in the queue by storing the entrance time in the current queue.

REQUIREMENT -3

Procdump

In response to the control + p signal, it outputs a list of processes.

IMPLEMENTATION OF PROCDUMP

The procdump function in the proc.c file has undergone all the changes, and it produces the following information regarding all the active processes: ID priority (only pbs and mlfq) rtime (running time) state waiting time (wtime) nrun (the quantity of times the scheduler ran the process) Qi (just mlfq).

REPORT

- Round robin schdeduling (default) Average running time : 18 Average waiting time : 119
- First come first serve Average running time : 24 Average waiting time : 102
- Priority Based Scheduling Average running time : 15 Average waiting time : 110
- Multi Level Feedback Queue Average running time : 13 Average waiting time : 112

  Thus I conclude that fcfs runs well as it has less waiting time.