2023101041 MP2 Report

System Calls

GOTTA COUNT ‘EM ALL

- In proc.h , modify the struct proc by adding an array syscall_counts of size 32
- In syscall.c, increment the value of a particular syscall in void syscall
- In user.h, define the function that can be called

```
int getsyscount(int mask);
```

- In user.pl, add the entry of the function to make the syscalls available for use.

```
entry("getsyscount");
```

- In syscall.h, define the syscall

```
#define SYS_getsyscount 23
```

- declare function as extern in syscall.c and add it to the array

```
extern uint64 sys_getsyscount(void);
[SYS_settickets] sys_settickets,
```

- implement your function in sysproc.c

WAKE ME UP WHEN MY TIMER ENDS

1. In proc.h , modify the struct proc by adding new fields mentioned below

```
uint64 clockcyclepassed;      //the address of the function wh
int alarmcount; // clock cycles after which handler must be ca
uint64 handleraddress; // stores handler address
struct trapframe *savedtrapframe;// savedtrapframe is essentia
int alaramflag; //  checks whether sigalarm is called or not
```

2. Initialise the fields in allocproc() of proc.c
3. In trap.c, if whichdev=2 and alarmflag is set, check if clockcyclepassed is more than alarmcount . if thats true, switch to handlerfunction.
4. After the handler function completes executing, we enter sigreturn. here, we switch back to the point where we left off.
5. Implement sigalarm and sigreturn systemcalls in sysproc.c and make neccesary changes for them as mentioned above

Scheduling

LBS

1. Create a seed based function called rand to generate random values in proc.c.
2. Add fields for storing tickets and start time to struct proc in proc.h.
3. Initialise these fields in alloc proc().
4. i created a global variable sudotime in dfs.h just the ticks that keeps track of number of processes that are created.
5. The entrytime is set to sudotime in fork since, all runnable processes go through this part.
6. Implement settickets in sysproc.c and make changes similar to the changes that were made for adding other functions like sigalaram, sigreturn and syscount.
7. In scheduler function of proc.c first create a variable totaltickets that stores the sum of tickets of all runnable processes. Now, use the randome function to generate a random number between 0 and totaltickets. find the process for

which prefixsum exceeds the random number generated. Probability of a process being generated is proportional to number of tickets the process has.

8. Now, among all the processes that the runnable select the process that arrived first using entrytime field of struct and context switch to the chosen process.

MLFQ

1. MLFQ is implemented without actually adding queue data structure, queue is simulated by adding fields to struct.
2. The following fields were added

```
int queue; // simulates queue
int entrytime; // stores entrytime
int tickstaken; // ticks used since entry
```

3. In scheduler function of proc.c first iterate over all the processes and select the process with least queue number.
4. Now, iterate over all processes again and find the process with least entrytime and context switch to the chosen process.
5. In trap.c , if ticks%48 is zero then boost all all the processes and set their queue value to 0.

```
#ifdef SCHED_MLFQ
 if(ticks%BOOSTINTERVAL==0)
 {
   boosted=one;
   for (struct proc *p = proc; p < &proc[NPROC]; p++)
   {
     // if(p->state==RUNNABLE)
     // {
       p->queue=0;
       p->tickstaken=0;
       p->entrytime=sudotime;
     // }
```

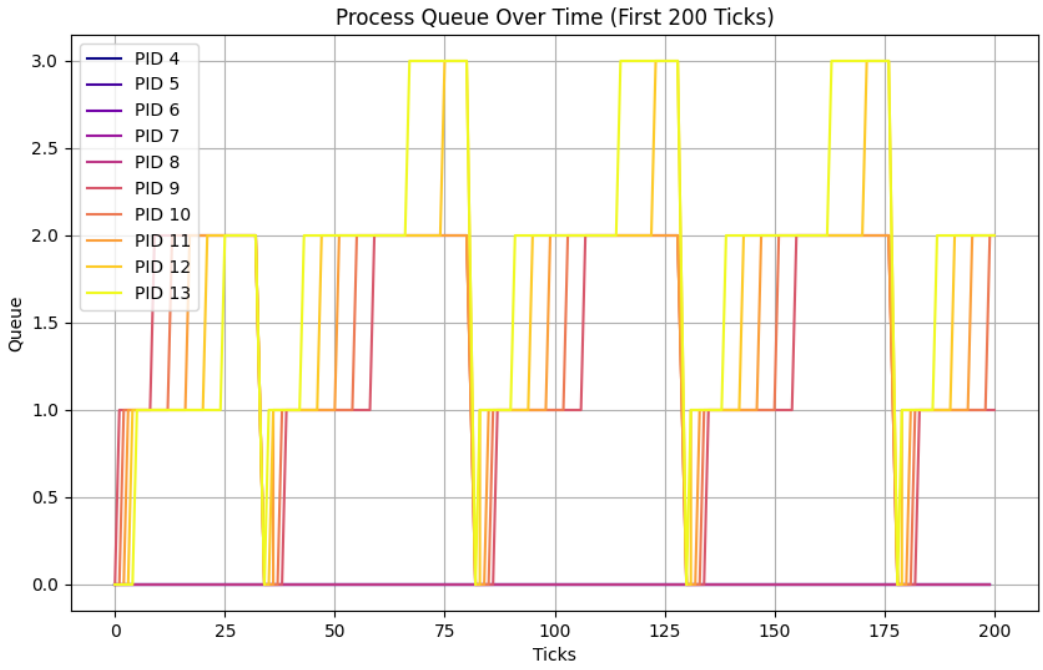
```
    }
    // totalticks=ticks;
    yield();
  }
  #endif
```

6. And if a process takes more ticks then what it has to take move it to the next queue.
7. I have introduced a variable called `sudotime` to schedule processes that arrive at the same clock interval. Previously, I was using `ticks` as a measure of entry time, but the order was not being preserved for processes arriving at the same clock tick.
8. if we have an I/O intrupt, we do not change tickstaken to zero to prevent gaming of processes, we just modify the entry time.

```
#ifdef SCHED_MLFQ
 else if(which_dev==1)
 {
   if(boosted!=one)
   {
     // p->tickstaken=0;
     sudotime++;
     p->entrytime=sudotime;
     if(p->queue!=3)
     {
       p->queue++;
     }
     p->state=RUNNABLE;

     yield();
   }
 }
 #endif
```

9. **MLFQ GRAPH**



PERFORMANCE COMPARISION

NFORK, IO	DEFAULT	MLFQ	LBS
10,5	rtime 12, wtime 151	rtime 13, wtime 146	rtime 13, wtime 126
20,5	rtime 19, wtime 320	rtime 19, wtime 259	rtime 19, wtime 189

What is the implication of adding the arrival time in the lottery based scheduling policy?

1. Helps prioritize earlier-arriving processes, preventing them from being starved by newer ones.
2. If all the processes have same number of tickets then the scheduler behaves similar to FCFS therefore things like convey effect will now come into picture.

```
The convoy effect refers to a situation where shorter
processes get stuck waiting for a long-running process to
complete, leading to poor performance.
```

3. IIf one of the processes that arrives first ends up in an infinite loop then the OS will never get control back, only possible way to get control back is to reboot the system.
4. Prioritizing arrival time might cause newly arriving processes to be delayed indefinitely, especially if early processes keep running long or are in an infinite loop.

EXAMPLE

Imagine three processes, A, B, and C:

Process A: Arrives at time 0, 10 tickets

Process B:Arrives at time 2, 10 tickets

Process C:Arrives at time 4, 10 tickets

without arrival time

- The lottery randomly selects one process based on ticket count.
- Even though Process C arrives last, it has an equal chance of being scheduled as Process A or B, as they all have the same number of tickets

with arrival time

- The scheduler may always choose Process A first because it arrived first, making the system act more like FCFS. Process B and C may get delayed, and if Process A runs for a long time, B and C could experience the **convoy effect**.

By considering arrival time, the system tends to favor processes that arrive earlier, but this can lead to fairness issues like convoy effect or delayed execution for newer processes.