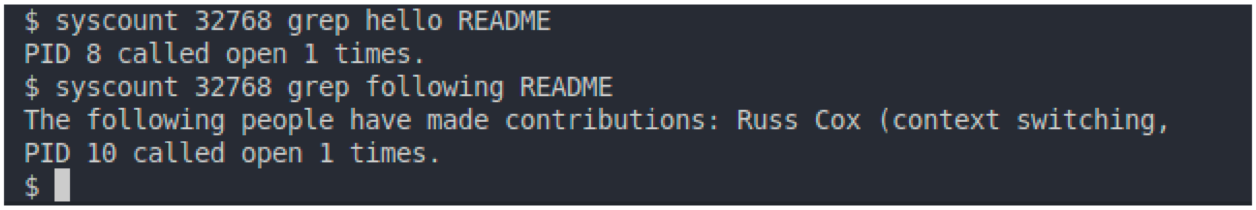
OSN Mini Project - 2

XV6

System Calls

1. Gotta count’em all

- Added the new system call getsyscount to syscall.c and syscall.h
- Changed process structure to array of count of syscalls.Initialized this array to be complete zero in allocproc. In syscall.c in syscall function this count was increased.
- As syscount calls the process in child therefore changes were made in wait so that parent also get updated value of syscall Count array.
- Made a user program for this.

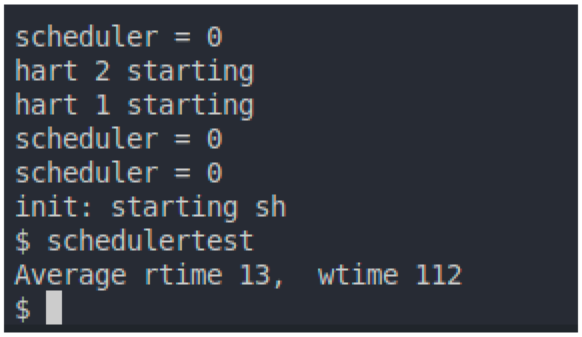


```
$ syscount 32768 grep hello README
PID 8 called open 1 times.
$ syscount 32768 grep following README
The following people have made contributions: Russ Cox (context switching,
PID 10 called open 1 times.
$ 
```

2.Wake me up when my time ends

- Similarly, added syscall sigalarm and sigreturn to syscall.c and syscall.h.
- Added new parameters to process structure in proc.h.
- In sigalram system call updated interval and handler with scanned values. Also set alarm_set flag to zero.
- In sigreturn alarm_set again set to zero. This is called when handler function return the control again to the process.
- Main part is implemented in trap.c where i have used alarm_set flag to know whether we have to handle or not. There i have seen if ticks(obviously once increase here also) is greater than interval if so then alarm_set flag turned1 and the saved_alarm_tf is moved to trapframe and its epc is updated with handler.

Scheduling



```
scheduler = 0
hart 2 starting
hart 1 starting
scheduler = 0
scheduler = 0
init: starting sh
$ schedulertest
Average rtime 13,  wtime 112
$ 
```

(Default Round Robin)

1. LBS

- Added new variables(tickets and arrivalTime) to process structure. Done changes in fork so that child get the same tickets as parent.
- Logic is first I calculated total tickets then randomly a winning ticket is selected. Then I iterate through all process to find the winning process.The reiterate through the process list to find a runnable process with same number of tickets but less arrival time. Then just switch the processes.

Implications of Arrival Time:

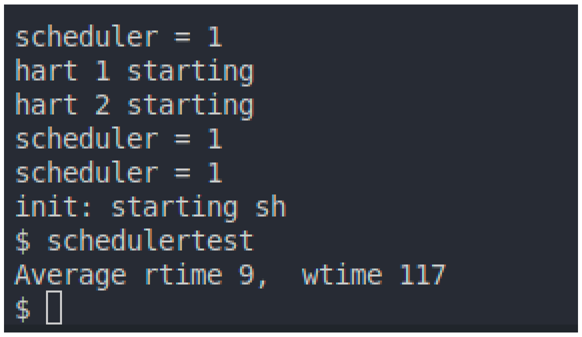
The inclusion of arrival time ensures **fairness** by prioritizing processes that arrive earlier. This prevents long waiting times for early-arriving processes, promoting a fairer CPU allocation.

Pitfalls:

One potential issue is that **new processes** with a **high number of tickets** may keep arriving, preventing lower-ticket processes from accessing the CPU. As a result, the system may repeatedly select a process with the highest number of tickets, creating an opportunity for **misuse**. An application could exploit this by continuously setting a high number of tickets for itself, monopolizing CPU time and starving other processes.

When All Processes Have the Same Number of Tickets:

If all processes have **identical ticket counts**, the system effectively behaves like **First-Come-First-Serve (FCFS)** scheduling. In such a scenario, the process that arrived earliest will always get CPU access, as the system selects the first matching process among the runnable ones. This eliminates the randomness or fairness typically expected from a lottery-based system.



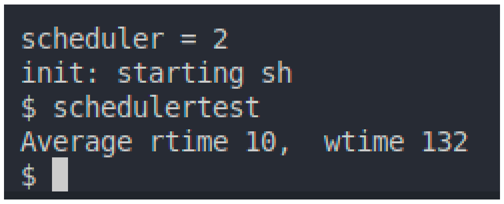
```
scheduler = 1
hart 1 starting
hart 2 starting
scheduler = 1
scheduler = 1
init: starting sh
$ schedulertest
Average rtime 9,  wtime 117
$ 
```

(With LBS)

2.MLFQ

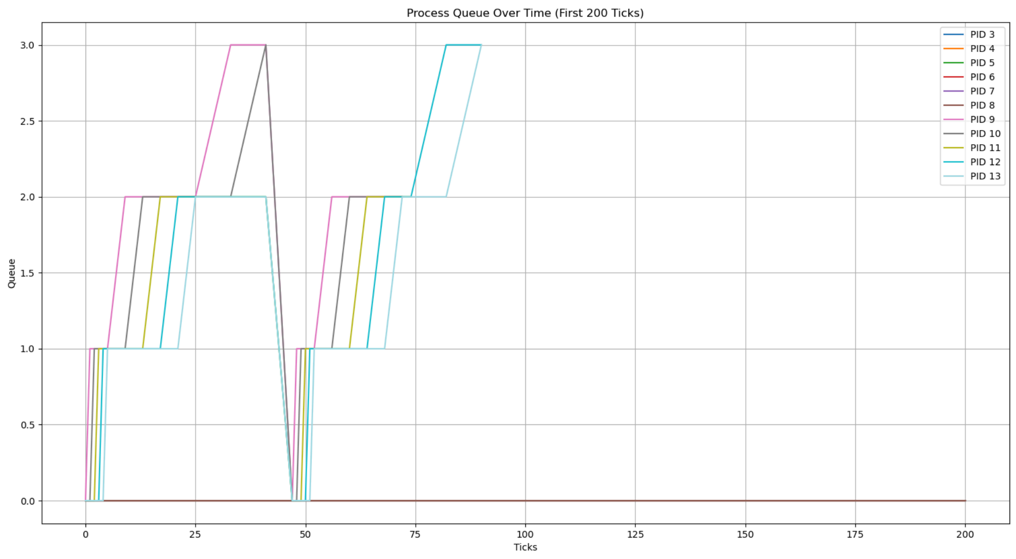
- Added more variables to process structure to know the priority, ticks used and total ticks used.
- I have made 4 arrays which i am using as queue and a array queue count which will save the queue count of each queue.
- I have defined custom functions add_to_queue , remove_from_queue , select_next_process.
- Here both above functions do what it name suggests and select_next_process selects the next runnable process according to the priority.
- Now in the scheduler first i selects the next process remove it from queue and if after usings its ticks if it is still runnable then i will again add it to the queue according to checking the priority.

- I have implemented priority boost logic in clock interrupt in this if ticks have exceeded the BOOST_TICKS that is defined to be 48 then will increase every process priority to be zero.
- In trap file i have done yield logic on increasing which is pretty standard.



```
scheduler = 2
init: starting sh
$ schedulertest
Average rtime 10,  wtime 132
$ 
```

(With MLFQ)



The MLFQ GRAPH