Name: Leela Sumanth Manduva

Assignment 3: Xv6 Kernel: generate random numbers and basic schedulers

Xv6 version used: xv6-riscv
To execute:

- The xv6 folder has all the xv6-riscv files.
- Replace the individual files present in rr, fcfs, and random with the files in the xv6 folder and then execute make clean; make; make qemu to test the random number generator, modified round robin, and fcfs schedulers.

1. Implementing random number generator on xv6:

References:

1. Creating a System call - https://medium.com/@harshalshree03/xv6-implementing-ps-nice-system-calls-and-priority-scheduling-b12fa10494e4
2. Generate a value - https://stackoverflow.com/questions/7602919/how-do-i-generate-random-numbers-without-rand-function

Explanation:

Creating system call:

- A system call number is assigned to a new system call (SYS_getunique) in syscall.h.
- A pointer to the system call function is added to the newly created system call in syscall.c.
- The system call function prototype is added to syscall.c.
- The system call function is implemented in sysproc.c.
- Then function name “getunique” is placed in user.h. When the user calls this function the system call number 22 is triggered which further calls the function implemented in sysproc.c.

The logic for generating random numbers:

- First, a seed value is taken. Then by using a modified version of the congruential generator method a random number is generated in the specified range. In my case (99-999).
- This method generates random numbers but once the xv6 is restarted it is printing the random numbers in the same order.
- So, I have acquired ticks from the OS and added it to the seed so that it won’t print numbers in the same order once xv6 is restarted.

2. Implementing basic Scheduler in xv6

References:

1. For Performance Analysis- https://medium.com/@harshalshree03/xv6-implementing-ps-nice-system-calls-and-priority-scheduling-b12fa10494e4
2. https://xiayingp.gitbook.io/build_a_os/scheduling/xv6-cpu-scheduling

Explanation:

1. Modified Round Robin:

- Xv6 uses a round-robin scheduler by default.
- The time quantum is defined in start.c file. The interval is set to (1000000)1/10th second.
- Once this interval is completed the OS calls the yield function which gives the CPU to the next RUNNABLE process.
- I have changed this interval value to (100000)1/100th second.
- The modified round-robin switches more often than the default scheduler as the time quantum is less.

Default Scheduler:

- Dummy process:

  No of switches: 194. Total time: 19.2 sec

```
$ dd
Final: 192
************************************************************
name     pid     state           noOfSwitch      TotalTime
init     1       SLEEPING        24              0
sh       2       SLEEPING        13              -1
dd       3       SLEEPING        7               -10
dd       4       RUNNING         194             192
************************************************************
```

- Tail:

  No of switches: 10 Total time: 0.02 sec

```
************************************************************
name     pid     state           noOfSwitch      TotalTime
init     1       SLEEPING        24              0
sh       2       SLEEPING        19              -1
tail     9       RUNNING         10              2
************************************************************
```

- Head:

No of switches: 7 Total time: 0.01 sec

```
************************************************************
name      pid      state            noOfSwitch       TotalTime
init      1        SLEEPING         24               0
sh        2        SLEEPING         21               -1
head      10       RUNNING          7                1
************************************************************
```

Round Robin with time slice of 1/100th second:

- Dummy process:

  No of Switches: 2002 Total time: 19.9 sec

```
$ dd
Final: 1990
************************************************************
name      pid      state            noOfSwitch       TotalTime
init      1        SLEEPING         28               0
sh        2        SLEEPING         15               -6
dd        3        SLEEPING         7                -894
dd        4        RUNNING          2002             1991
************************************************************
```

- Tail:

  No of Switches: 24 Total time: 0.15 sec

```
************************************************************
name      pid      state            noOfSwitch       TotalTime
init      1        SLEEPING         29               0
sh        2        SLEEPING         15               -5
tail      3        RUNNING          24               15
************************************************************
```

- Head:

  No of Switches: 16 Total time: 0.09sec

```
************************************************************
name      pid      state            noOfSwitch       TotalTime
init      1        SLEEPING         29               0
sh        2        SLEEPING         17               -5
head      4        RUNNING          16               9
************************************************************
```

Performance Analysis Default RR vs Modified RR:

Process Name	No of Switches (Default RR)	No of Switches (Modified RR)	Total Time (Default RR)	Total Time (Modified RR)
Dummy Process	**194**	**2002**	**19.2**	**19.9**
Tail	**10**	**24**	**0.02**	**0.15**
Head	**7**	**16**	**0.01**	**0.09**

2. FCFS:

- Xv6 uses an FCFS with a round-robin scheduler by default and the time quantum is defined in start.c file.
- Once this interval is completed the OS calls the yield function which gives the CPU to the next RUNNABLE process.
- I have added a stopYield flag in trap.c file. When this flag is set to 1 it will stop the OS from calling the yield function.
- As the yield function is not called once the interval is completed, the scheduler now works as a FCFS.
- From the below screenshots, you can see that the no of switches value is 1. This indicates that the scheduler is not preempting the process.

Default Scheduler:

- Dummy process:

  No of switches: 194. Total time: 19.2 sec

```
$ dd
Final: 192
*************************************************************
name     pid     state           noOfSwitch      TotalTime
init     1       SLEEPING        24              0
sh       2       SLEEPING        13              -1
dd       3       SLEEPING        7               -10
dd       4       RUNNING         194             192
*************************************************************
```

- Tail:

  No of switches: 10 Total time: 0.02 sec

```
**********************************************************
name     pid     state           noOfSwitch      TotalTime
init     1       SLEEPING        24              0
sh       2       SLEEPING        19              -1
tail     9       RUNNING         10              2
**********************************************************
```

- Head:

  No of switches: 7 Total time: 0.01 sec

```
**********************************************************
name     pid     state           noOfSwitch      TotalTime
init     1       SLEEPING        24              0
sh       2       SLEEPING        21              -1
head     10      RUNNING         7               1
**********************************************************
```

FCFS Scheduler:

- Dummy process:

  No of Switches: 1 Total time: 18.6 sec

```
$ dd
Final: 186
**********************************************************
name     pid     state           noOfSwitch      TotalTime
init     1       SLEEPING        24              0
sh       2       SLEEPING        13              -1
dd       3       SLEEPING        6               -26
dd       4       RUNNING         1               186
**********************************************************
```

- Tail:

  No of Switches: 1 Total time: 0.02 sec

```
**********************************************************
name     pid     state           noOfSwitch      TotalTime
init     1       SLEEPING        24              0
sh       2       SLEEPING        15              -2
tail     4       RUNNING         1               2
**********************************************************
```

- Head:

No of Switches: 16 Total time: 0.09sec

```
*************************************************************
name      pid     state           noOfSwitch      TotalTime
init      1       SLEEPING        24              0
sh        2       SLEEPING        17              -1
head      5       RUNNING         1               1
*************************************************************
```

Performance Analysis Default vs FCFS:

Process Name	No of Switches (Default)	No of Switches (FCFS)	Total Time (Default)	Total Time (FCFS)
Dummy Process	**194**	**1**	**19.2**	**18.6**
Tail	**10**	**1**	**0.02**	**0.02**
Head	**7**	**1**	**0.01**	**0.01**

3. Performance Analysis Modified RR vs FCFS:

Process Name	No of Switches (Modified RR)	No of Switches (FCFS)	Total Time (Modified RR)	Total Time (FCFS)
Dummy Process	**2002**	**1**	**19.9**	**18.6**
Tail	**24**	**1**	**0.15**	**0.02**
Head	**16**	**1**	**0.09**	**0.01**

Performance Analysis Explanation:

1. No of Switches Calculation:
   - I have added a new variable noOfSwitch in the process structure.
   - Then I declared it to 0 in allocproc() function in proc.c.
   - This value is updated in the scheduler() function before the context switch happens.

2. Total Time Calculation:

- I have declared two variables startTime and endTime in the process structure.
- Then I assigned ticks to startTime and endTime to 0 in allocproc() function.
- Once a user program call exit() function. The endTime variable is updated to ticks.
- This will enable us to save the start time and end time of a process.

3. state(): (This is printed for every process)
   - state() function is call in exit() function.
   - This will just display the details of the processes like name, pid, state, the number of times context switches happened, and the total time of a process.
   - The total time will be a positive value only if the process exits and changes its state to ZOMBIE.
   - If the process is still running the total time column will just display the starting time of a process in negative value.

4. d() and dd() Dummy programs:
   - These are just two dummy programs that consume CPU time.
   - d() will run for infinity.
   - dd() will run for a limited time.