Project 2 Readme

Group Members:

Prati Jain

Aditya Khetarpal

Rajdeep Savani

System Environment:

OS: Ubuntu 24.04 LTS

Compiler: 13.2.0

QEMU version: 8.2.2

Part 1: Adding uniq and find calls

1 uniq

The uniq command in Linux is a utility that reports or filters out repeated lines in a file. It detects and deletes adjacent duplicate lines. The command reads input from a file and writes filtered data to standard output.

Implementation Steps

1. **Basic uniq functionality (no flags)**

Implement the basic functionality of uniq to filter out adjacent matching lines from the input file.

Command: uniq os.txt

2 **uniq -c filename**

Implement the -c flag to prefix each line of output with the number of occurrences of the line.

3. **uniq -u filename**

Implement the -u flag to only print unique lines from the file.

4. **uniq -w [N] filename**

Implement the -w [N] flag to compare only the first N characters of each line when determining uniqueness.

Command: uniq -w 1 os.txt

5. **cat filename | uniq**

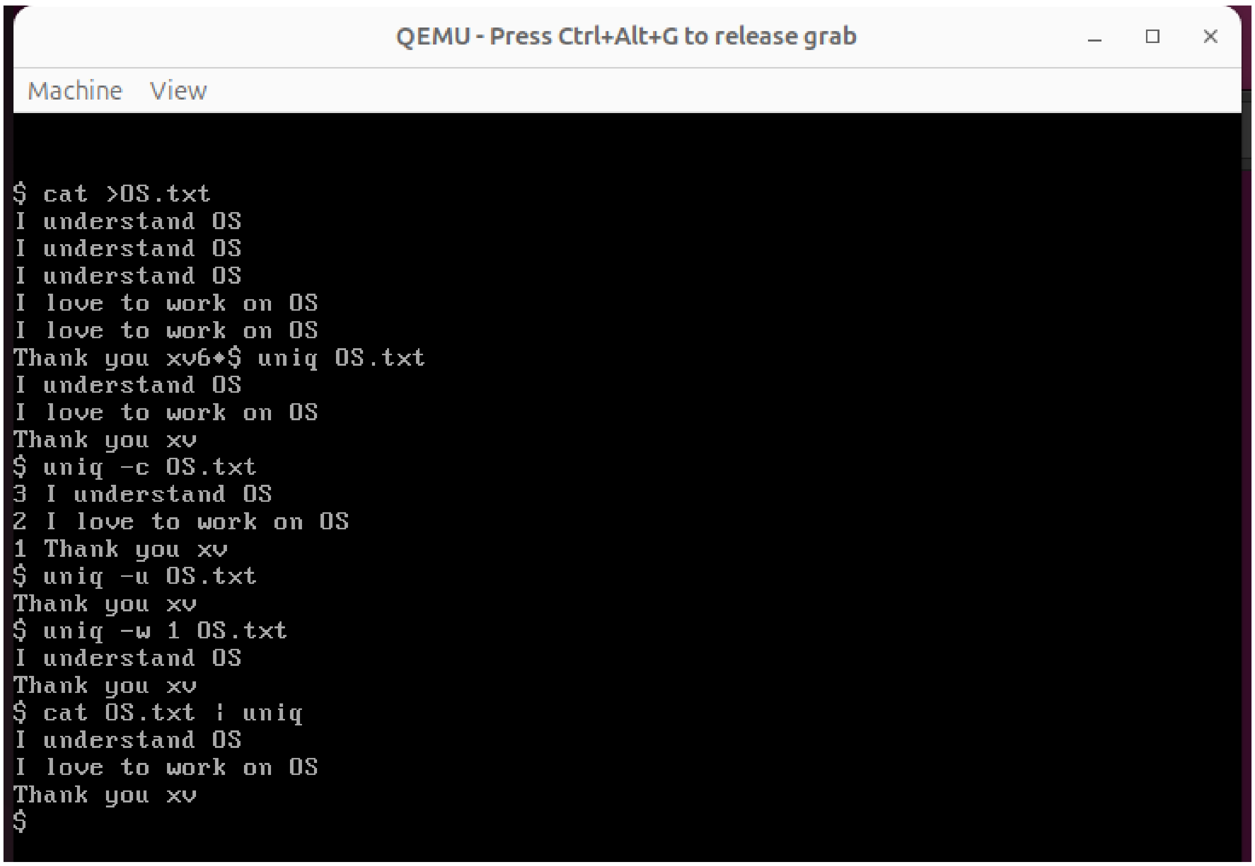
Ensure that uniq can read from standard input

Modification to Makefile

Add _uniq to the UPROGS list in the Makefile to ensure its compiled and included in the xv6 image.

Compile and Run

1. Run make clean to ensure a fresh build.
2. Run make qemu to compile the entire xv6 system, including the new uniq command, and start the xv6 emulation in QEMU.



2. find

The find command is used to search for files in a directory tree with a specific name.

Implementation Steps

1. Default find functionality: find <folder> -name <name>

Implement basic find functionality to search for files with a specific name in a folder.

```
QEMU
Machine   View
SeaBIOS (version 1.16.3-debian-1.16.3-2)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1EFCB050+1EF0B050 CA00



Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ echo > b
$ mkdir a
$ echo > a/b
$ mkdir a/aa
$ echo > a/aa/b
$ find . -name b
./b
./a/b
./a/aa/b
$
```

2. find <folder> -name <name> -type f

Implement the -type flag with the f option to only find files.

```
./a/aa/b
$ find . -name b -type f
./b
./a/b
./a/aa/b
$
```

3. **find <folder> -name <name> -type d**

Implement the -type flag with the d option to only find directories.

```
$ find . -name b -type d
$
```

4. **find <folder> -name <name> -inum <number of bytes><inode number>**

Implement the -inum flag to search for files by inode number.

```
$ find . -name b -inum 25
./b
$
```

5. **find <folder> -name <name> -inum +<number of bytes><inode number>**

Implement the -inum flag with a + prefix to search for files with an inode number greater than the specified value.

```
$ find . -name b -inum +25
./a/b
./a/aa/b
$
```

6. **find <folder> -name <name> -inum -<number of bytes><inode number>**

Implement the -inum flag with a - prefix to search for files with an inode number less than the specified value.

```
$ find . -name b -inum -25
$
```

7. **find <folder> -name <name> -printi**

```
$ find . -name b -printi
25 ./b
27 ./a/b
29 ./a/aa/b
```

Modification to Makefile

Add _find to the UPROGS list in the Makefile to ensure it's compiled and included in the xv6 image.

Compile and Run

1 Run make clean to ensure a fresh build.

2 Run make qemu to compile the xv6 system, including the new find command, and start the QEMU emulator.

Part 2: Ticks Running

Currently, xv6 has no way of checking how long a process has been in the RUNNING state. In this part we added functionality to track how many ticks each process has been scheduled in xv6. We implemented a system call, ticks_running(pid), that returns this value for the given process ID. The syscall should return 0 if the process exists but hasn't been scheduled yet, and -1 if there is no process with the supplied PID in the process table.

Implementation Steps

1. **Add ticks_running system call**
    - Modify necessary kernel files to implement the ticks_running() system call:
        - **proc.c**: Add functionality to track the number of ticks each process has been running.
        - **sysproc.c**: Implement the sys_ticks_running() function that will return the number of ticks for the specified process.
        - **syscall.h**: Define a new system call number for SYS_ticks_running.

- **syscall.c**: Add an external reference to sys_ticks_running() and update the syscalls array.
- **user.h**: Declare the user-space function int ticks_running(int pid); to allow user programs to call this syscall.
- **usys.S**: Create a system call stub for ticks_running().
- **ticks_running_test.c:** Program to test the ticks_running() system call.

Modification to Makefile

- **Add _ticks_running_test to the UPROGS list** in the Makefile to ensure it is compiled and included in the xv6 image.

Compile and Run

1. **Run make clean** to ensure a fresh build.
2. **Run make qemu** to compile the entire xv6 system, including the new ticks_running() system call, and start the xv6 emulation in QEMU.
3. **Test the syscall** by writing a simple user program (ticks_running_test) to call ticks_running() for different process IDs and validate the returned values.

```
$ ticks_running_test 1
Process 1 has ticks: 27
$ ticks_running_test 2
Process 2 has ticks: 21
$ ticks_running_test 999
Failed to get ticks for process 999 (process does not exist)
$
```

```
Running stress test with ticks monitoring...
Stress test starting (Parent PID: 4)
Child process created (PID: 7)
PID 7: Initial ticks: 1
PID 7: Starting I/O operations
PID 7: Final tiChild process created (PIcks: 9 (Total: 8)
D: 8)
PID 8: Initial ticks: 1
PID 8: Starting I/O operations
PID 4: Initial ticks: 32
PID 4: Starting I/O operatPID 8: Final tickions
s: 8 (Total: 7)
PID 4: Final ticks: 37 (Total: 5)
Child process 7 completed
```

Part 3: Implementing a Simple Scheduler

We implemented the scheduler to select processes based on their predicted job length.

Default - RoundRobin

Implementation Steps

1. **Add SJF Scheduler Logic**
   - Modify the scheduler in proc.c to implement the Shortest Job First (SJF) scheduling algorithm. Assign a predicted job length to each process when it enters the scheduling queue. We used rand() for random prediction or compute an exponential average.
2. **Modify Makefile**
   - Update the Makefile to support switching between the default and SJF scheduler. Add a compile-time option (SCHEDULER=SJF) to select the SJF scheduler.
3. **Add System Call**
   - Implement the sjf_job_length(pid) system call in sysproc.c.
   - Update the system call table in syscall.h and syscall.c to include the new system call.
   - Declare the user-space function int sjf_job_length(int pid); in user.h to allow user programs to call this syscall.

Compile and Run

- Run make clean to ensure a fresh build.
- Use make qemu SCHEDULER=SJF to compile and run xv6 with the SJF scheduler.

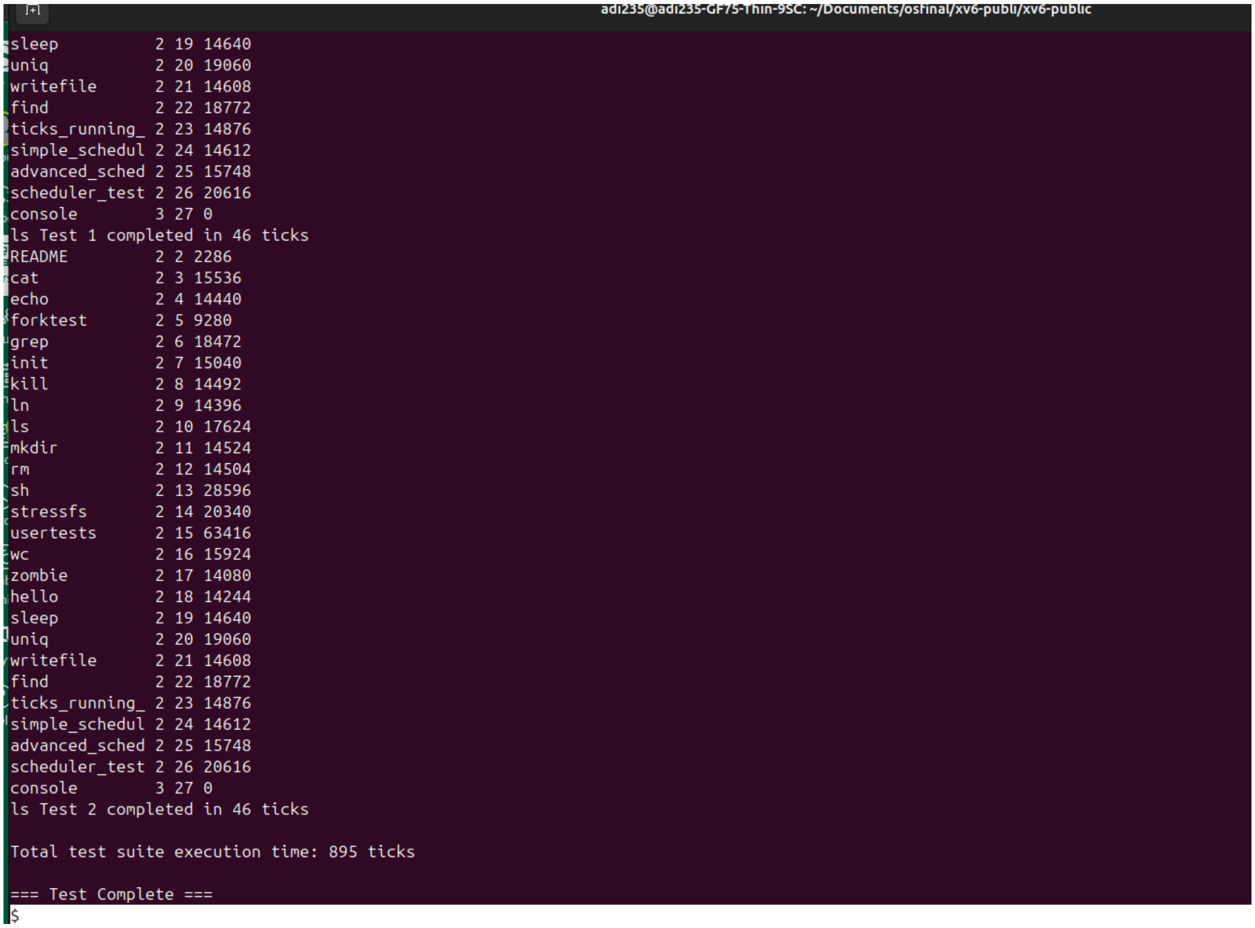
Challenges Faced

Scheduler Integration: Integrating the SJF scheduler into the existing xv6 scheduling system required a deep understanding of the scheduler's interactions with process states and interrupts.

DEFAULT

adi235@adi235-GF75-Thin-9SC: ~/Documents/osfinal/xv6-publi/xv6-public

```
console        3 27 0
$ scheduler_test

=== Scheduler Performance Test ===

Test 1: CPU-bound processes
CPU Test 0 completed in 196 ticks
CPU Test 1 completed in 199 ticks
CPU Test 2 completed in 200 ticks

Test 2: I/O operations
I/O Test 0 completed in 8 ticks
I/O Test 1 completed in 7 ticks
I/O Test 2 completed in 7 ticks

Test 3: Pipe operations
Pipe Test 0 completed in 2 ticks
Pipe Test 1 completed in 1 ticks
Pipe Test 2 completed in 1 ticks

Real-world test: ls command
README         2 2 2286
cat            2 3 15536
echo           2 4 14440
forktest       2 5 9280
grep           2 6 18472
init           2 7 15040
kill           2 8 14492
ln             2 9 14396
ls             2 10 17624
mkdir          2 11 14524
rm             2 12 14504
sh             2 13 28596
stressfs       2 14 20340
usertests      2 15 63416
wc             2 16 15924
zombie         2 17 14080
hello          2 18 14244
sleep          2 19 14640
uniq           2 20 19060
writefile      2 21 14608
find           2 22 18772
```



```
adi235@adi235-GF75-Thin-9SC: ~/Documents/osfinal/xv6-publi/xv6-public
sleep          2 19 14640
uniq           2 20 19060
writefile      2 21 14608
find           2 22 18772
ticks_running_ 2 23 14876
simple_schedul 2 24 14612
advanced_sched 2 25 15748
scheduler_test 2 26 20616
console        3 27 0
ls Test 1 completed in 46 ticks
README         2 2 2286
cat            2 3 15536
echo           2 4 14440
forktest       2 5 9280
grep           2 6 18472
init           2 7 15040
kill           2 8 14492
ln             2 9 14396
ls             2 10 17624
mkdir          2 11 14524
rm             2 12 14504
sh             2 13 28596
stressfs       2 14 20340
usertests      2 15 63416
wc             2 16 15924
zombie         2 17 14080
hello          2 18 14244
sleep          2 19 14640
uniq           2 20 19060
writefile      2 21 14608
find           2 22 18772
ticks_running_ 2 23 14876
simple_schedul 2 24 14612
advanced_sched 2 25 15748
scheduler_test 2 26 20616
console        3 27 0
ls Test 2 completed in 46 ticks

Total test suite execution time: 895 ticks

=== Test Complete ===
$
```

SJF

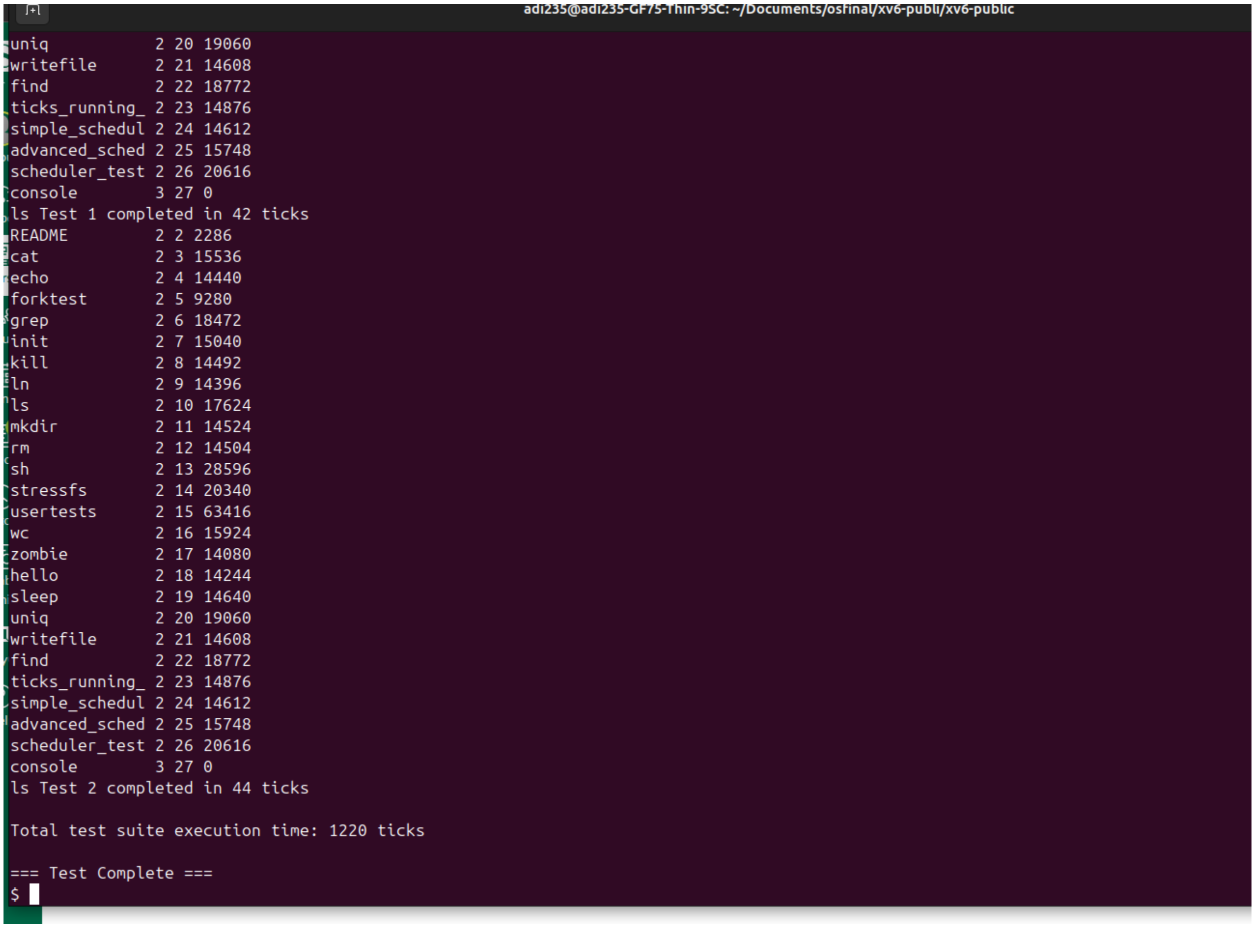
adi235@adi235-GF75-Thin-9SC: ~/Documents/osfinal/xv6-publi/xv6-public

```
sleep          2 19 14640
uniq           2 20 19060
writefile      2 21 14608
find           2 22 18772
ticks_running_ 2 23 14876
simple_schedul 2 24 14612
advanced_sched 2 25 15748
scheduler_test 2 26 20616
console        3 27 0
ls Test 1 completed in 42 ticks
README         2 2 2286
cat            2 3 15536
echo           2 4 14440
forktest       2 5 9280
grep           2 6 18472
init           2 7 15040
kill           2 8 14492
ln             2 9 14396
ls             2 10 17624
mkdir          2 11 14524
rm             2 12 14504
sh             2 13 28596
stressfs       2 14 20340
usertests      2 15 63416
wc             2 16 15924
zombie         2 17 14080
hello          2 18 14244
sleep          2 19 14640
uniq           2 20 19060
writefile      2 21 14608
find           2 22 18772
ticks_running_ 2 23 14876
simple_schedul 2 24 14612
advanced_sched 2 25 15748
scheduler_test 2 26 20616
console        3 27 0
ls Test 2 completed in 42 ticks

Total test suite execution time: 933 ticks

=== Test Complete ===
$
```

adi235@adi235-GF75-Thin-9SC: ~/Documents/osfinal/xv6-publi/xv6-public

```
find           2 22 18772
ticks_running_ 2 23 14876
simple_schedul 2 24 14612
advanced_sched 2 25 15748
scheduler_test 2 26 20616
console        3 27 0
$ scheduler_test

=== Scheduler Performance Test ===

Test 1: CPU-bound processes
CPU Test 0 completed in 216 ticks
CPU Test 1 completed in 209 ticks
CPU Test 2 completed in 210 ticks

Test 2: I/O operations
I/O Test 0 completed in 8 ticks
I/O Test 1 completed in 8 ticks
I/O Test 2 completed in 7 ticks

Test 3: Pipe operations
Pipe Test 0 completed in 2 ticks
Pipe Test 1 completed in 1 ticks
Pipe Test 2 completed in 1 ticks

Real-world test: ls command
README         2 2 2286
cat            2 3 15536
echo           2 4 14440
forktest       2 5 9280
grep           2 6 18472
init           2 7 15040
kill           2 8 14492
ln             2 9 14396
ls             2 10 17624
mkdir          2 11 14524
rm             2 12 14504
sh             2 13 28596
stressfs       2 14 20340
usertests      2 15 63416
wc             2 16 15924
zombie         2 17 14080
```

LOTTERY



```
adi235@adi235-GF75-Thin-9SC: ~/Documents/osfinal/xv6-publi/xv6-public
uniq           2 20 19060
writefile      2 21 14608
find           2 22 18772
ticks_running_ 2 23 14876
simple_schedul 2 24 14612
advanced_sched 2 25 15748
scheduler_test 2 26 20616
console        3 27 0
ls Test 1 completed in 42 ticks
README         2 2 2286
cat            2 3 15536
echo           2 4 14440
forktest       2 5 9280
grep           2 6 18472
init           2 7 15040
kill           2 8 14492
ln             2 9 14396
ls             2 10 17624
mkdir          2 11 14524
rm             2 12 14504
sh             2 13 28596
stressfs       2 14 20340
usertests      2 15 63416
wc             2 16 15924
zombie         2 17 14080
hello          2 18 14244
sleep          2 19 14640
uniq           2 20 19060
writefile      2 21 14608
find           2 22 18772
ticks_running_ 2 23 14876
simple_schedul 2 24 14612
advanced_sched 2 25 15748
scheduler_test 2 26 20616
console        3 27 0
ls Test 2 completed in 44 ticks

Total test suite execution time: 1220 ticks

=== Test Complete ===
$
```

```
ticks_running_ 2 23 14876
simple_schedul 2 24 14612
advanced_sched 2 25 15748
scheduler_test 2 26 20616
console        3 27 0
$ scheduler_test

=== Scheduler Performance Test ===

Test 1: CPU-bound processes
CPU Test 0 completed in 302 ticks
CPU Test 1 completed in 318 ticks
CPU Test 2 completed in 298 ticks

Test 2: I/O operations
I/O Test 0 completed in 8 ticks
I/O Test 1 completed in 7 ticks
I/O Test 2 completed in 8 ticks

Test 3: Pipe operations
Pipe Test 0 completed in 2 ticks
Pipe Test 1 completed in 1 ticks
Pipe Test 2 completed in 1 ticks

Real-world test: ls command
README         2 2 2286
cat            2 3 15536
echo           2 4 14440
forktest       2 5 9280
grep           2 6 18472
init           2 7 15040
kill           2 8 14492
ln             2 9 14396
ls             2 10 17624
mkdir          2 11 14524
rm             2 12 14504
sh             2 13 28596
stressfs       2 14 20340
usertests      2 15 63416
wc             2 16 15924
zombie         2 17 14080
```

Part 4: A (More) Advanced Scheduler

It includes modifying the scheduler to accommodate the chosen scheduling strategy, as well as writing system calls to manage and query scheduler parameters.

Implementation Steps

1. **Add Lottery Scheduler Logic**
    - Modify the scheduler in proc.c to implement the Lottery Scheduling algorithm. Use the get_random(min, max) function to randomly select a process based on the number of tickets held.
    - Define a global constant for the default number of tickets that each process starts with.
2. **Add System Calls**
    - Implement the set_lottery_tickets(tickets) and get_lottery_tickets(pid) system calls in sysproc.c.

- Update the system call table in syscall.h and syscall.c to include the new system calls.
- Declare the user-space functions int set_lottery_tickets(int tickets); and int get_lottery_tickets(int pid); in user.h to allow user programs to call these syscalls.

3. **Modify Makefile**
    - Update the Makefile to support switching between the default and Lottery scheduler. Add a compile-time option (SCHEDULER=LOTTERY) to select the Lottery scheduler.
4. **Compile and Run**
    - Run make clean to ensure a fresh build.
    - Use make qemu SCHEDULER=LOTTERY to compile and run xv6 with the Lottery scheduler.