Project 3: Scheduler Implementation

System Environment

Operating System: XV6

Compiler/Development Environment: GCC (GNU Compiler Collection).

Languages Used: C

Overview

In this assignment, I have implemented two scheduling algorithms: First-come first-serve (FCFS) and priority-based scheduling. I have reported the average wait and turnaround times for all my processes. Assuming uniq (user) and uniq (kernel) are two different processes, similar to my prior assignment, I created a new testsched.c (FCFS) and testschedpri.c (PRIORITY) that called both programs and reported these statistics. Similarly, I repeated the procedure for different processes.

Code Structure

1. **Function scheduler(void)**:
    - This function is the core of the scheduler and is responsible for selecting and running processes based on two scheduling policies: First-Come-First-Serve (FCFS) and Priority-Based Scheduling.
2. **Initialization**:
    - **struct proc *p, *p1;** declares pointers to process structures for iteration.
    - **struct cpu *c = cpu;** initializes a pointer to the CPU structure for the current CPU core.
    - **c->proc = 0;** sets the currently running process for this CPU core to NULL.
3. **Infinite Loop**:
    - **for(;;)** represents an infinite loop, meaning the scheduler will run continuously.
4. **Enabling Interrupts**:
    - **sti();** enables interrupts on the processor, allowing the operating system to respond to hardware and software interrupts.
5. **Lock Acquisition**:
    - **acquire(&ptable.lock);** acquires the process table lock to ensure exclusive access to process-related data structures, preventing race conditions.
6. **Policy-Based Selection**:

- The code checks the value of **policyy** to determine the scheduling policy to use.

7. **FCFS Policy (policyy == 0)**:
    - It selects the process with the earliest arrival time, and if the process hasn't started yet, it records the start time.
    - The code switches to the chosen process, updates its state to **RUNNING**, and performs a context switch.
    - The process's state is updated, and the end time (**etime**) is recorded.
    - The process table lock is released.
8. **Priority-Based Scheduling (policyy == 1)**:
    - It selects the process with the highest priority. Processes with lower priority values have higher priority.
    - The code iterates over the process table to find the process with the highest priority.
    - After selecting the process with the highest priority, it switches to that process, updates its state to **RUNNING**, and performs a context switch.
    - The process's state is updated, and the end time (**etime**) is recorded.
    - The process table lock is released.
9. **Release Lock**:
    - **release(&ptable.lock);** releases the process table lock, allowing other threads or processors to access process-related data structures.
10. This scheduler code is used in conjunction with **testsched.c** and **testschedpri.c** to test and evaluate the effectiveness of the scheduling policies in an xv6-based operating system.

Approach

Problem Understanding:

Begin by thoroughly understanding the problem. Need to implement two scheduling algorithms, First-Come-First-Serve (FCFS) and Priority-Based Scheduling, in the xv6 operating system. The goal is to report average wait and turnaround times for processes, specifically for "uniq (user)" and "uniq (kernel)"; "head (user)" and "head (kernel)" as separate processes.

Code Structure Design:

Plan the overall structure of the code to implement the scheduling algorithms. Consider creating a **scheduler** function. This function will handle process scheduling based on the selected policy (FCFS or Priority-Based).

Data Structure Enhancement:

Modify the **struct proc** in the xv6 source code (e.g., **proc.h**) to include any additional fields or attributes that are necessary for your scheduling algorithms. This might include fields like **arrival_time**, **start_time**, **priority**, etc., depending on the requirements of the scheduling policies.

Policy Selection:

Implement a mechanism to select the scheduling policy. This can be done using a variable like **policyy**. A value of 0 can represent FCFS, and a 1 can represent Priority-Based Scheduling.

FCFS Scheduling (policyy == 0):

If the selected policy is FCFS, need to implement the FCFS scheduling logic. This logic involves iterating through the process table to find the process with the earliest arrival time and scheduling it to run.

Priority-Based Scheduling (policyy == 1):

If the selected policy is Priority-Based Scheduling, implement the logic for selecting the process with the highest priority. This may require additional data structures or logic to manage and update priorities.

Process State Management:

Ensure proper management of process states. Should transition processes to the "RUNNING" state when they are scheduled to run and update their states accordingly.

Time Tracking:

Implement mechanisms to track time-related information such as start time, end time, and other relevant timestamps for processes. Update these timestamps as processes progress.

Locking and Synchronization:

Use locks and synchronization mechanisms (e.g., **acquire** and **release**) to ensure exclusive access to process-related data structures like the process table.

Report Generation:

After each process has completed its execution, calculate and store relevant statistics, such as wait times and turnaround times, for further reporting.

Testing and Evaluation:

Create test programs, such as **testsched.c** and **testschedpri.c**, to evaluate the performance of your scheduling algorithms. These test programs should create a variety of processes with different characteristics to thoroughly test the scheduler.

Steps to Run

Compile the program

Use the Xv6 build system to compile the scheduling utilities.

```
venkatarahulc@RAHUL:~/xv6$ make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*.o *.d *.asm *.sym vectors.S bootblock entryother \
initcode initcode.out kernel xv6.img fs.img kernelmemfs mkfs \
.gdbinit \
_test _cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _wc _zombie _head _uniq _head_user _uniq_user _ps _testsched _testschedpri _chpr
venkatarahulc@RAHUL:~/xv6$
```

```
venkatarahulc@RAHUL:~/xv6$ make
make: 'xv6.img' is up to date.
venkatarahulc@RAHUL:~/xv6$
```

Access the Xv6 Environment

Boot or launch the Xv6 operating system on the system or in an emulator, such as QEMU

```
venkatarahulc@RAHUL:~/xv6$ make qemu-nox
```

```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00



Booting from Hard Disk..xv6...
cpu0: starting
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ ls
.              1 1 512
..             1 1 512
README         2 2 2487
test           2 3 14840
cat            2 4 15612
echo           2 5 14500
forktest       2 6 8972
grep           2 7 18472
init           2 8 15124
kill           2 9 14604
ln             2 10 14488
ls             2 11 17044
mkdir          2 12 14612
rm             2 13 14592
sh             2 14 28664
stressfs       2 15 15508
usertests      2 16 63012
wc             2 17 16064
zombie         2 18 14188
head           2 19 15752
uniq           2 20 15884
head_user      2 21 16548
uniq_user      2 22 20068
ps             2 23 14584
testsched      2 24 15400
testschedpri   2 25 16004
chpr           2 26 14732
test_uniq.txt  2 27 161
test_head_stat 2 28 177
test_head_city 2 29 347
console        3 30 0
$
```

Run the Program

First Come First Serve Scheduling

→testsched uniq_user uniq_kernel (or) vice versa

```
$ testsched uniq_user test_uniq.txt uniq test_uniq.txt
Testing FCFS Scheduling
Uniq command is getting executed in user mode.
I understand the Operating system.
I love to work on OS.
Thanks xv6.
Executing the custom uniq command in kernel mode.
I understand the Operating system.
I love to work on OS.
Thanks xv6.
2 0 1
26650 2 2
26652 26650 249
Average turnaround time: 8884
Average wait time: 8800
$
```

→testsched head_kernel head_user(or) vice versa

```
$ testsched head test_head_city.txt head_user test_head_state.txt
Testing FCFS Scheduling
--- test_head_city.txt ---
Executing the custom head command in kernel mode.
Line 1: Connecticut - Hartford
Line 2: Delaware - Dover
Line 3: Georgia - Atlanta
Line 4: Maryland - Annapolis
Line 5: Massachusetts - Boston
Line 6: New Hampshire - Concord
Line 7: New Jersey - Trenton
Line 8: New York - Albany
Line 9: North Carolina - Raleigh
Line 10: Pennsylvania - Harrisburg
Line 11: Rhode Island - Providence
Line 12: South Carolina - Columbia
Line 13: Virginia - Richmond
Line 14: Florida - Tallahassee
Executing the custom head command in user mode.
--- test_head_state.txt ---
Line 1: Connecticut
Line 2: Delaware
Line 3: Georgia
Line 4: Maryland
Line 5: Massachusetts
Line 6: New Hampshire
Line 7: New Jersey
Line 8: New York
Line 9: North Carolina
Line 10: Pennsylvania
Line 11: Rhode Island
Line 12: South Carolina
Line 13: Virginia
Line 14: Florida
2 0 1
137050 2 2
137052 137050 249
Average turnaround time: 45684
Average wait time: 45600
$
```

→testsched head_user uniq_kernel (or) uniq_user head_kernel in any order

```
$ testsched head_user test_head_city.txt uniq test_uniq.txt
Testing FCFS Scheduling
Executing the custom head command in user mode.
--- test_head_city.txt ---
Line 1: Connecticut - Hartford
Line 2: Delaware - Dover
Line 3: Georgia - Atlanta
Line 4: Maryland - Annapolis
Line 5: Massachusetts - Boston
Line 6: New Hampshire - Concord
Line 7: New Jersey - Trenton
Line 8: New York - Albany
Line 9: North Carolina - Raleigh
Line 10: Pennsylvania - Harrisburg
Line 11: Rhode Island - Providence
Line 12: South Carolina - Columbia
Line 13: Virginia - Richmond
Line 14: Florida - Tallahassee
Executing the custom uniq command in kernel mode.
I understand the Operating system.
I love to work on OS.
Thanks xv6.
2 0 1
99285 2 2
99288 99285 249
Average turnaround time: 33096
Average wait time: 33012
$
```

```
$ testsched uniq_user test_uniq.txt head test_head_state.txt
Testing FCFS Scheduling
Uniq command is getting executed in user mode.
I understand the Operating system.
I love to work on OS.
Thanks xv6.
--- test_head_state.txt ---
Executing the custom head command in kernel mode.
Line 1: Connecticut
Line 2: Delaware
Line 3: Georgia
Line 4: Maryland
Line 5: Massachusetts
Line 6: New Hampshire
Line 7: New Jersey
Line 8: New York
Line 9: North Carolina
Line 10: Pennsylvania
Line 11: Rhode Island
Line 12: South Carolina
Line 13: Virginia
Line 14: Florida
2 0 1
109428 2 2
109429 109428 249
Average turnaround time: 36476
Average wait time: 36392
$
```

Priority Based Scheduling

→When the User process has the highest priority

```
$ testschedpri head_user test_head_city.txt 2 head test_head_state.txt 8
Testing Priority-Based Scheduling
Executing the custom head command in user mode.
--- test_head_city.txt ---
Line 1: Connecticut - Hartford
Line 2: Delaware - Dover
Line 3: Georgia - Atlanta
Line 4: Maryland - Annapolis
Line 5: Massachusetts - Boston
Line 6: New Hampshire - Concord
Line 7: New Jersey - Trenton
Line 8: New York - Albany
Line 9: North Carolina - Raleigh
Line 10: Pennsylvania - Harrisburg
Line 11: Rhode Island - Providence
Line 12: South Carolina - Columbia
Line 13: Virginia - Richmond
Line 14: Florida - Tallahassee
--- test_head_state.txt ---
Executing the custom head command in kernel mode.
Line 1: Connecticut
Line 2: Delaware
Line 3: Georgia
Line 4: Maryland
Line 5: Massachusetts
Line 6: New Hampshire
Line 7: New Jersey
Line 8: New York
Line 9: North Carolina
Line 10: Pennsylvania
Line 11: Rhode Island
Line 12: South Carolina
Line 13: Virginia
Line 14: Florida
2 0 1
159615 2 2
159623 159615 249
Average turnaround time: 53207
Average wait time: 53123
$
```

here the lower the priority number the higher the priority

So **`head_user`** is assigned 2 whereas **`head_kernel`** is assigned 8 which makes the schedular execute the user first followed by the kernel and report the statistics.

Works the same for **`uniq`** and **`uniq_user`.**

→When the Kernel process has the highest priority

```
$ testschedpri uniq test_uniq.txt 3 uniq_user test_uniq.txt 9
Testing Priority-Based Scheduling
Executing the custom uniq command in kernel mode.
I understand the Operating system.
I love to work on OS.
Thanks xv6.
Uniq command is getting executed in user mode.
I understand the Operating system.
I love to work on OS.
Thanks xv6.
2 0 1
207906 2 2
207907 207906 249
Average turnaround time: 69302
Average wait time: 69218
$
```

So **`uniq_user`** is assigned 9 whereas **`uniq_kernel`** is assigned 3 which makes the schedular execute the kernel first followed by the user and report the statistics.

Works the same for **`head`** and **`head_user`.**

→Now in default cases where no priority is passed always the kernel processes have higher priority when compared to user processes.

```
$ testschedpri uniq_user test_uniq.txt head test_head_city.txt
Testing Priority-Based Scheduling
Uniq command is getting executed in user mode.
I understand the Operating system.
I love to work on OS.
Thanks xv6.
--- test_head_city.txt ---
Executing the custom head command in kernel mode.
Line 1: Connecticut - Hartford
Line 2: Delaware - Dover
Line 3: Georgia - Atlanta
Line 4: Maryland - Annapolis
Line 5: Massachusetts - Boston
Line 6: New Hampshire - Concord
Line 7: New Jersey - Trenton
Line 8: New York - Albany
Line 9: North Carolina - Raleigh
Line 10: Pennsylvania - Harrisburg
Line 11: Rhode Island - Providence
Line 12: South Carolina - Columbia
Line 13: Virginia - Richmond
Line 14: Florida - Tallahassee
2 0 1
3341 2 2
3343 3341 244
Average turnaround time: 1114
Average wait time: 1032
$
```

In the above example as there are no priority numbers specified **`head_kernel`** is executed first followed by **`uniq_user`**.

Resources Used

https://medium.com/@harshalshree03/xv6-implementing-ps-nice-system-calls-and-priority-scheduling-b12fa10494e4

https://www.youtube.com/watch?v=DZ0-GMtOtEc&t=691s

https://www.youtube.com/watch?v=hIXRrv-cBA4&t=330s