CS 3031

OPERATING SYSTEMS LAB

MINIX SCHEDULERS

GROUP ASSIGNMENT 2

AMAN VERMA (CS11B002)
BHUPESH RAJ S (CS11B006)
RITVIK JAISWAL (CS11B031)

PROBLEM DEFINITION

To modify the existing Minix scheduler to implement Multilevel Feedback Queue scheduling and Lottery Scheduling.

Multilevel Feedback Queue Scheduler

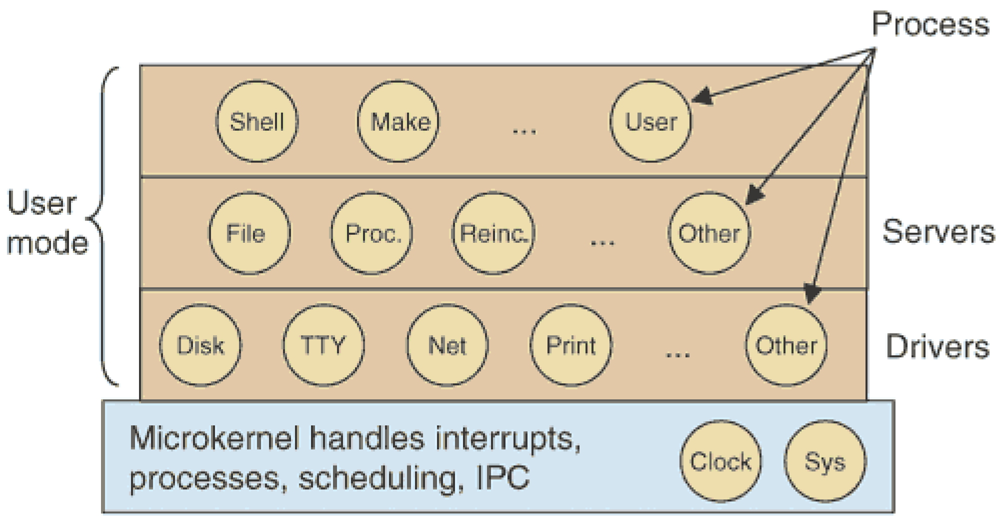
Every new process is placed at the tail of the topmost queue. As the time quantum of the process in a queue expires, it is enqueued to the tail of the next queue. The scheduler runs the processes from the head of the topmost queue. Processes in a queue are scheduled to run only if there are no process waiting in any of the upper queues.

Lottery Scheduler

Each process is allotted a random number of tickets. The scheduler selects a ticket at random. The process holding that ticket is the process which gets to run in the CPU.

INTRODUCTION

Minix 3 has been constructed as a series of layers.



At the bottom is the microkernel, running in kernel mode. The rest of the code runs in user mode.

The kernel is responsible for low level and privileged operations such as programming the CPU and MMU, interrupt handling and inter process communication, and contains two tasks (SYS and CLOCK) to support the user mode parts of the operating system.

Above the microkernel are the device drivers. Above this layer is the Servers layer. The simplest servers provide virtual file system (VFS), process management (PM), and memory management (MM) functionality. The reincarnation server (RS) keeps track of all servers and drivers and can transparently repair the system when certain failures occur.

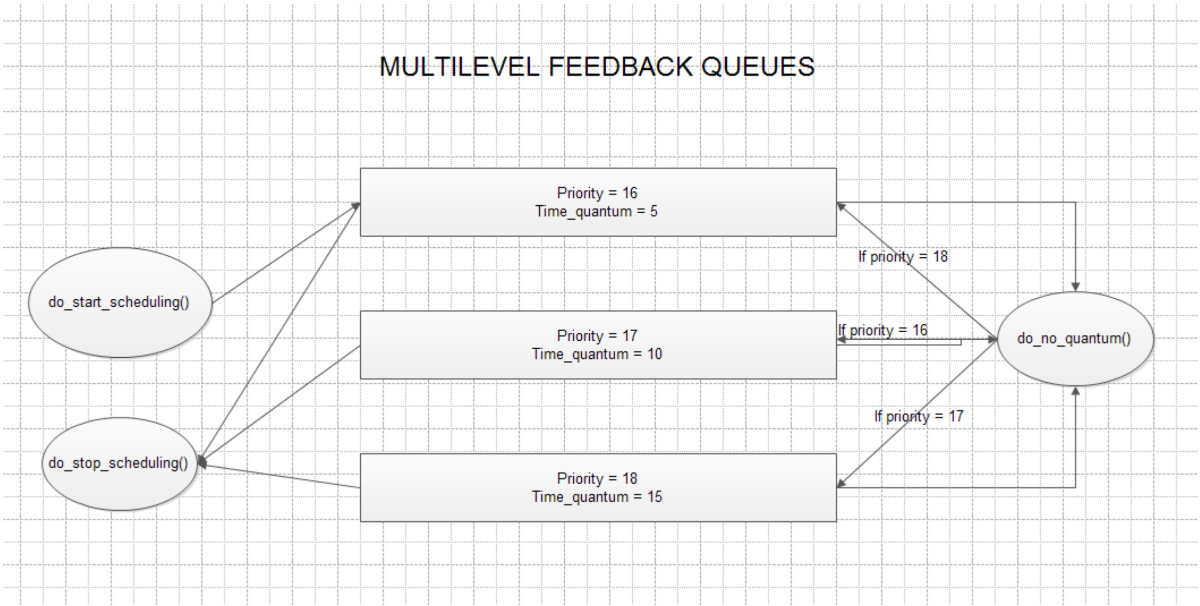
PM is responsible for process management such as creating and removuing processes, assigning process IDs and priorities and controlling the flow of execution. It also maintains relations between processes. It is responsible for POSIX signal handling.

VFS manages the file system. It is an ordinary file server that handles standard POSIX calls such as open(), read() and write(). VFS supports multiple, different file servers. Both VFS and each file server run as isolated, user-mode processes. The file system underneath each mount point servers by a separate file server so that a file server failure can affect only a subtree of the virtual file system.

IMPLEMENTATION

MULTILEVEL FEEDBACK QUEUES

FLOWCHART FOR MULTILEVEL FEEDBACK QUEUES



Changes made in config.h

- NR_SCHED_QUEUES is changed to 19 from 16 so we added extra three queues
- MAX_USER_Q is set to 16 so the maximum priority for user processes is 16
- USER_Q is also set to 16 so the user process will start with the maximum priority

Changes made in schedproc.h

- Introduced a new variable time_spent which keeps track of the time spent by the process in CPU

Changes made in schedule.c

We changed the DEFAULT_USER_TIME_SLICE from 200 to 5

In do_start_scheduling() function :

- We set the rmp->priority to be MAX_USER_Q which is 16
- We set the rmp->time_slice to be the default user time slice
- We set the rmp->time_spent to be zero i.e. we start the process so no time is spent on the CPU

In do_noquantum() function:

- We check if the process priority is between 16 to 17 then set the
    - time_slice = (priority - 14) * DEFAULT_USER_TIME_SLICE
    - priority += 1
    - so if priority is 16 time_slice = 10 and the priority is set to 17

- If priority = 18 then
    - set time_slice = DEFAULT_TIME_SLICE and
    - rmp->priority = 16 i.e. it is moved to the topmost queue

- If priority is between 0 to 14:
    - rmp->priority += 1

    - In this function we also print the process ID, rmp->time_slice and rmp->priority

In do_stop_scheduling() function :

- In this function the process is finished so we print the process ID, rmp->time_spent ie how much time the process spent on the CPU which will give the running time.

In balance_queues() function:

- For every five seconds balance queue function is called. If the priority is between 16 to 19

  rmp-priority = 16

  rmp->time_slice = DEFAULT_USER_TIME_SLICE

  so all the process are moved to the topmost queue after some time T.

TEST CASES

We used 3 test files longrun1.c , longrun2.c and longrun3.c. These files has 3 inputs

1. Id of the program

2. Maximum value of the loop

3. No of such loops

So the test cases are like 2 nested loops. We created mytest.c for multilevel feedback queue by creating 10 child process and in each child process we executed the instances of longrun1 longrun2 and longrun3. For Multilevel Feedback queue we gave least value of maximum value of the loop and no of loops since the time quantum is less compared to the lottery scheduling.

EXPECTED INPUT

Let the mytest contain 3 fork() and executes longrun1 , longrun2 and longrun3. We can define 3 process with id 1 , 2 , 3 and we can give any value of maximum number > 100 and no of loops > 100. If the process with id 1 , 2 , 3 has pid = 100 ,101 , 102 we can expect the following output.

EXPECTED OUTPUT

* Process 100 consumed time quantum 5 with priority 16
* Process 101 consumed time quantum 5 with priority 16
* Process 102 consumed time quantum 5 with priority 16
* Process 100 consumed time quantum 10 with priority 17
* Process 101 consumed time quantum 10 with priority 17
* Process 102 consumed time quantum 10 with priority 17
* Process 100 consumed time quantum 15 with priority 18
* Process 100 consumed time quantum 5 with priority 16
* Process 100 consumed time quantum 10 with priority 17
* Process 101 consumed time quantum 15 with priority 18
* Process 101 consumed time quantum 5 with priority 16
* Process 101 consumed time quantum 10 with priority 17
* Process 102 consumed time quantum 15 with priority 18
* Process 102 consumed time quantum 5 with priority 16
* Process 102 consumed time quantum 10 with priority 17
* Balance Queues
* Process 100 consumed time quantum 5 with priority 16
* Process 101 consumed time quantum 5 with priority 16
* Process 102 consumed time quantum 5 with priority 16
* Process 100 consumed time quantum 10 with priority 17
* Process 101 consumed time quantum 10 with priority 17

SAMPLE OUTPUT AND SCREENSHOT

MULTILEVEL FEEDBACK QUEUE

OUTPUT

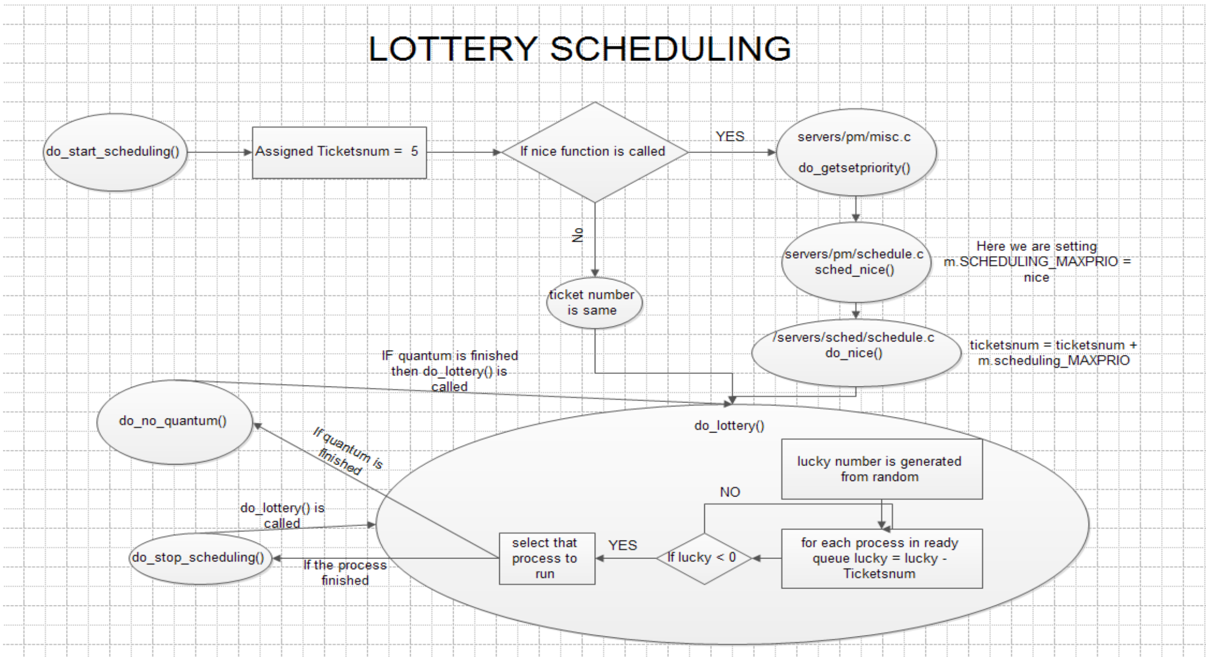
```
 The Process 292683 has consumed quantum 15 with priority 18
 The Process 292683 has consumed quantum 5 with priority 16
 The Process 292683 has consumed quantum 10 with priority 17
 The Process 292686 has consumed quantum 15 with priority 18
Balance Queues to the topmost Queues
 The Process 292686 has consumed quantum 5 with priority 16
 The Process 292682 has consumed quantum 5 with priority 16
 The Process 292683 has consumed quantum 5 with priority 16
 The Process 292684 has consumed quantum 5 with priority 16
 The Process 292685 has consumed quantum 5 with priority 16
 The Process 292687 has consumed quantum 5 with priority 16
 The Process 292688 has consumed quantum 5 with priority 16
 The Process 292689 has consumed quantum 5 with priority 16
 The Process 292691 has consumed quantum 5 with priority 16
 The Process 292690 has consumed quantum 5 with priority 16
 The Process 292686 has consumed quantum 10 with priority 17
 The Process 292682 has consumed quantum 10 with priority 17
 The Process 292683 has consumed quantum 10 with priority 17
 The Process 292684 has consumed quantum 10 with priority 17
 The Process 292685 has consumed quantum 10 with priority 17
 The Process 292687 has consumed quantum 10 with priority 17
 The Process 292688 has consumed quantum 10 with priority 17
 The Process 292689 has consumed quantum 10 with priority 17
 The Process 292691 has consumed quantum 10 with priority 17
 The Process 292690 has consumed quantum 10 with priority 17
 The Process 292686 has consumed quantum 15 with priority 18
 The Process 292686 has consumed quantum 5 with priority 16
 The Process 292686 has consumed quantum 10 with priority 17
 The Process 292682 has consumed quantum 15 with priority 18
 The Process 292682 has consumed quantum 5 with priority 16
 The Process 292682 has consumed quantum 10 with priority 17
 The Process 292683 has consumed quantum 15 with priority 18
 The Process 292683 has consumed quantum 5 with priority 16
 The Process 292683 has consumed quantum 10 with priority 17
```

TAT

```
Time of 8 is 0.000000
Time of 1 is 3.000000
Time of 5 is 3.000000
Time of 9 is 2.000000
Time of 4 is 2.000000
Time of 6 is 11.000000
Time of 2 is 12.000000
Time of 10 is 11.000000
Time of 3 is 27.000000
Time of 7 is 27.000000
```

LOTTERY SCHEDULING

FLOWCHART FOR LOTTERY SCHEDULING



Changes made in config.h:

- We changed the value of MAX_USER_Q to be 12
- The value of USER_Q to be 13

Changes made in schedproc.h :

- introduced a new variable time_spent which keeps track of the time spent by the process in CPU
- introduced a new variable called ticketsNum which indicates the number of tickets the process holds

Changes made in servers/pm/schedule.c :

In sched_nice() function:

- We commented the lines which converts the given nice value to the priority and we set the value of m.SCHEDULING_MAXPRIO to be the nice value.

Changes made in schedule.c :

In do_start_scheduling() function:

- We initialize the number of tickets the process holds to be 5 so that each process has same priority
- We set the time_spent variable to zero as it starts the scheduling.
- The process default start at the USER_Q which is 13

In do_noquantum() function:

- If the process is in the user queue then we kept the process in the same queue else
- We lowered the priority by 1.
- Then we called the do_lottery() function

In do_stop_scheduling() function:

- If the process finished executing then we printed the time spent by the process in the CPU
- Then we called the do_lottery() function

In do_nice() function:

- If in the program nice() function is called the control goes to the servers/pm/misc.c do_getsetpriority()
- Which calls the function sched_nice() in servers/pm/schedule.c in which we set the value of m.SHEDULING_MAXPRIO to be nice value

- So in do_nice function made the value of rmp->ticketsNum to be rmp->ticketsNum + m.SCHEDULING_MAXPRIO. So the value of nice is directly added to the number of tickets and the value of the rmp->priority remains the same
- So the nice(60); for a process implies rmp->ticketsNum for that process is rmp->ticketsNum + 60;
- We also took care that the value of the ticket doesn't decrease below 5.
- Then do_lottery() function is called.

We introduced a function called do_lottery():

In do_lottery() function:

- We calculate the total number of tickets for process present in the USER_Q. A lucky number is generated between zero and total number of tickets
- Then we iterate through all the process in the USER_Q and we do lucky = lucky - rmp->ticketsNum ie. (r=r-t). If lucky becomes less than zero that
- Process is selected to run by increasing its priority from 13 to 12.
- So if the tickets value is high it is likely that process has more possibility to run.

TEST CASES

We used 3 test files longrun1.c, longrun2.c and longrun3.c. These files has 4 inputs

1. Id of the program
2. Maximum value of the loop
3. No of such loops
4. Nice value for the process

So the test cases are like 2 nested loops. We created mytest.c for Lottery scheduling by creating 10 child process and in each child process we executed the instances of longrun1 longrun2 and longrun3. For Lottery scheduling we gave same value of maximum value of the loop and no of loops for each instances which is slightly greater than MLFQ values and different nice values to see the delay in process with lesser tickets.

EXPECTED INPUT

Let the mytest contain 3 fork() and executes longrun1 , longrun2 and longrun3. We can define 3 process with id 1, 2, 3 and we can give any value of maximum number > 100 and no of loops > 100 and we gave nice value to be 60, 70 and 80. If the process with id 1, 2, 3 has pid = 100, 101, 102 we can expect the following output.

EXPECTED OUTPUT

If there are three process with pid = 100 , 101 , 102 are running with nice values 60 , 70 and 80 the expected output is :

* Process 102 consumed time quantum 200 with priority 13 and tickets 85
* Process 102 consumed time quantum 200 with priority 13 and tickets 85
* Process 102 consumed time quantum 200 with priority 13 and tickets 85
* Process 101 consumed time quantum 200 with priority 13 and tickets 75
* Process 100 consumed time quantum 200 with priority 13 and tickets 65
* Process 102 consumed time quantum 200 with priority 13 and tickets 85
* Process 101 consumed time quantum 200 with priority 13 and tickets 75
* Process 102 consumed time quantum 200 with priority 13 and tickets 85
* Process 101 consumed time quantum 200 with priority 13 and tickets 75
* Process 101 consumed time quantum 200 with priority 13 and tickets 75
* Process 100 consumed time quantum 200 with priority 13 and tickets 65
* Process 102 consumed time quantum 200 with priority 13 and tickets 85
* Process 100 consumed time quantum 200 with priority 13 and tickets 65
* Process 101 consumed time quantum 200 with priority 13 and tickets 75
* Process 101 consumed time quantum 200 with priority 13 and tickets 75

SAMPLE OUTPUT AND SCREENSHOT

LOTTERY SCHEDULING

OUTPUT

```
Process 73161 is executed whose ticket no is 89 with time quantum 200 with priority 13
Process 73163 is executed whose ticket no is 15 with time quantum 200 with priority 13
Process 73157 is executed whose ticket no is 65 with time quantum 200 with priority 13
Process 73162 is executed whose ticket no is 55 with time quantum 200 with priority 13
Process 73160 is executed whose ticket no is 75 with time quantum 200 with priority 13
Process 73161 is executed whose ticket no is 89 with time quantum 200 with priority 13
Process 73163 is executed whose ticket no is 15 with time quantum 200 with priority 13
Process 73162 is executed whose ticket no is 55 with time quantum 200 with priority 13
Process 73160 is executed whose ticket no is 75 with time quantum 200 with priority 13
Process 73161 is executed whose ticket no is 89 with time quantum 200 with priority 13
Process 73163 is executed whose ticket no is 15 with time quantum 200 with priority 13
Process 73162 is executed whose ticket no is 55 with time quantum 200 with priority 13
Process 73164 is executed whose ticket no is 35 with time quantum 200 with priority 13
Process 73160 is executed whose ticket no is 75 with time quantum 200 with priority 13
Process 73161 is executed whose ticket no is 89 with time quantum 200 with priority 13
Process 73163 is executed whose ticket no is 15 with time quantum 200 with priority 13
Process 73162 is executed whose ticket no is 55 with time quantum 200 with priority 13
Process 73164 is executed whose ticket no is 35 with time quantum 200 with priority 13
Process 73160 is executed whose ticket no is 75 with time quantum 200 with priority 13
Process 73161 is executed whose ticket no is 89 with time quantum 200 with priority 13
```

TAT

```
Time of 2 is 2.000000
Time of 3 is 2.000000
Time of 1 is 2.000000
Time of 5 is 24.000000
Time of 4 is 25.000000
Time of 7 is 24.000000
Time of 6 is 23.000000
Time of 8 is 24.000000
Time of 9 is 24.000000
Time of 10 is 25.000000
```

PERFORMANCE ANALYSIS

How we calculated the value of Turnaround Time and Waiting time

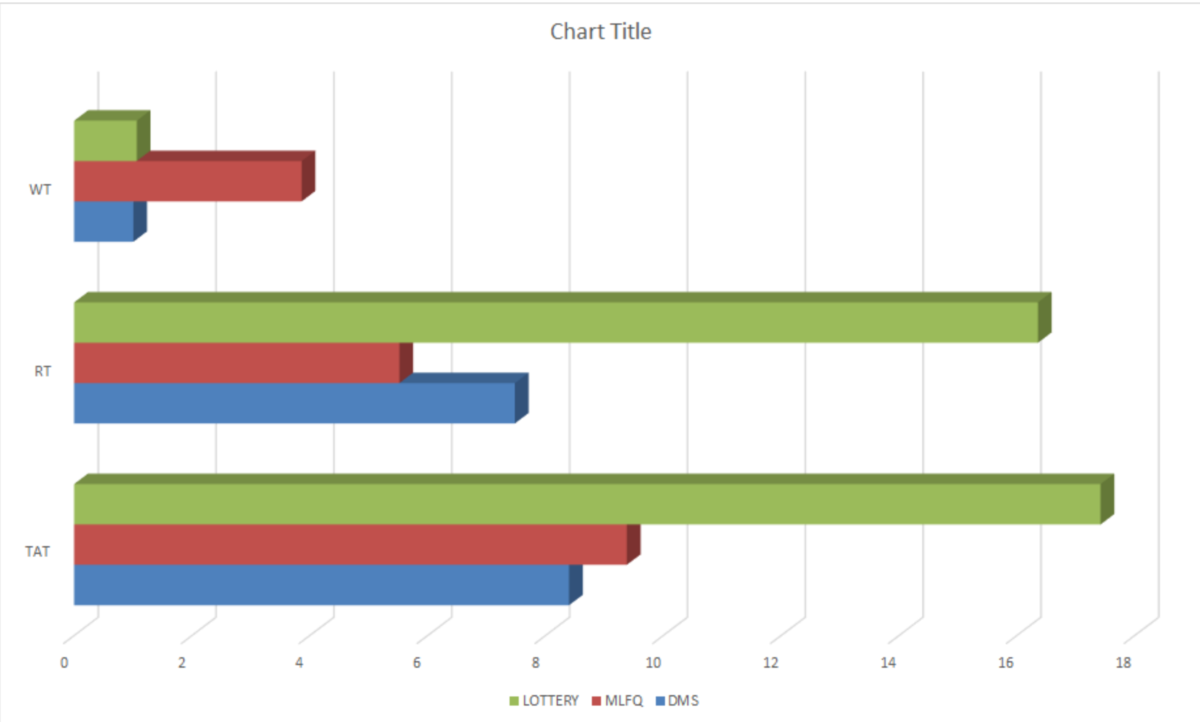
In each test case we calculated the turnaround time by starting the clock at the start of the program and stopping the clock at the end of the program. The difference in time gives the turnaround time. The id of the process and the turnaround time are written in a file.

For calculating the waiting time we need running time of the program which can be obtained from time_spent variable we are printing in the scheduler. We open the /var/log/messages in Minix and look for the corresponding endpoint id and the time_spent printed by that process in do_stop_scheduling(). The difference in Turnaround Time and Running time gives the value of the Waiting time. If Waiting time is less we can say the scheduler is better.

This is because typically utilization and throughput are traded off for better Response Time. So If the Response time is less we can say the scheduler is better.

	TAT	RT	WT
DMS	8.4	7.48	1.004
MLFQ	9.38	5.5195	3.86
LOTTERY	17.42	16.358	1.062

The graph was plotted based on the above table.



So based on the Graph we can say that the Lottery scheduler has less waiting time which means Throughput is higher for lottery scheduling.

CONCLUSION

The Minix scheduler was modified and the new schedulers were compared with the originals. The performance was analysed and plotted on a bar graph. Lottery scheduler turns out to be the best with respect to the waiting times for the processes but it may also lead to starvation.