CS3523 Project1: Implement Priority and Earliest-Deadline-First Schedulers in Minix3

Autumn 2016

Deadline: 5th November 2016, 9:00 pm

Goal:

As studied in the class, scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated the CPU. The main goal for this project is to modify the MINIX 3 scheduler to implement new scheduling algorithm. You must implement the following schedulers:

- Non-preemptive Priority Scheduler
- Earliest-Deadline-First Scheduler (EDF)

Description

The goal of this assignment is to implement the Non-preemptive Priority and Earliest-Deadline-First Schedulers. Through this project, you will learn about the implementation of schedulers in minix.

Non-Preemptive Priority Scheduler

Here you have to implement the non-preemptive version of EDF covered in the book. The priorities are set based on the deadlines of the program to execute. Closer the deadline, higher the priority like in EDF.

Processes get added to the ready-queue as they enter the system. Once the CPU becomes free, the process with the highest priority, i.e., earliest deadline get scheduled. Incoming process does not preempt the process currently executing in the CPU if it has higher priority.

If the deadline of a process is crossed before it can be scheduled, then it is not scheduled. This information is noted down for computing statistics.

Earliest-Deadline-First Scheduler

Here you have to implement the EDF algorithm discussed in the book. The priorities are set based on the deadlines of the program to execute. Closer the deadline, higher the priority like in EDF.

Processes get added to the ready-queue as they enter the system. If a process Pi with closer deadline i.e., higher priority comes into the system while the process currently running on the CPU, Pj has lower priority then Pi preempts Pj and starts running in the CPU (until it gets preempted or waits for IO or completes). Pj then gets added to the ready-queue.

Like in the case of non-preemptive schedulers, if the deadline of a process is crossed before it can be scheduled, then it is not scheduled. This information is noted down for computing statistics.

Deadlines

As discussed, each process has a deadline. You should develop APIs to set the deadline of each process.

Expected Output

The output must show when a process Pi gets added to the ready-queue, gets scheduled on the CPU, moves to the waiting-queue etc. Essentially, the output must show the complete log of all the activities of a process.

Assume that four processes, proc1, proc2, proc3, proc4 are scheduled at the same time. Let their deadlines be 20, 21, 22 and 23 respectively. All these four processes could be executing the same program such as longrun. The output should be as follows:

```
/* First show the times when these processes have been added to the ready-queue */
proc1 with deadline 20 has been added to the ready queue
proc2 with deadline 21 has been added to the ready queue
proc3 with deadline 22 has been added to the ready queue
proc4 with deadline 23 has been added to the ready queue
        .........
        .........

/* As the processes execute show when they are scheduled on the CPU
proc1 with deadline 20 has been scheduled
        .........
proc2 with deadline 21 has been scheduled
        .........
proc3 with deadline 22 has been scheduled
        .........
```

The output must demonstrate that the scheduler is indeed executing the processes by their deadlines.

Expected Input

The input to the program can be taken from a file in the following format. Here let each process to execute be “longrun.c” which is attached. With this program, the duration of execution can be controlled by loopcount.

```
n       // Total number of processes

/* ID           Arrival Time        Deadline      Loopcount */
P1              10:00               10:03               50
P2              10:01               10:04               51
P3              10:00               10:02               60
P4              10:10               10:12               65
                  .
                  .
                  .
Pn              10:01               10:30               30
```

Assume that all the entries are ordered by their arrival times. The processes arrival time is exponentially distributed with an average of λ ms. The time between arrival and the execution time is again exponentially distributed with an average μ1 ms. Based on this time, you can decide the value of Loopcount. Then, the time between execution time and the deadline is exponentially distributed with an average μ2 ms.

Let us illustrate these values with examples. Suppose Pi arrives at time 11:00 and Pi+1 arrives at time 11:00 + x1. Then the average of the difference x1 is exponentially distributed with λ ms. Let Pi execute for x2 ms. Then the average of various x2 is exponentially distributed with μ1 ms. Let the value of loopcount be x2 * C where C is a constant > 1. For instance, C could be 10. Suppose the deadline of Pi is 11:00 + x3. Then the average of various x3 is exponentially distributed with μ2 ms. It is clear that μ1 < μ2.

Given a set of n, λ, μ1, μ2 the input file can be generated. You can develop some simple scripts to generate such an input file.

Performance Analysis of different schedulers for a given user workload

To analyze the results of this project, you have to plot two graphs. One graph capturing the average turn-around and waiting times. The other graph capturing the number of processes missing their deadlines.

Graph1: Comparison of Turnaround & Waiting Times

For this analysis, have n, μ1, μ2 to be fixed. Then vary λ to measure turn-around & waiting times for each algorithm: Non-preemptive Scheduler and EDF scheduler. For instance have n, μ1, μ2 to be fixed at 30, 20, 30 and vary λ from 10 to 50 in the increments of 10.

As discussed above, have loopcount to be based on the computed execution time, x2. Let it be C times x2 for some constant C.

Thus the x-axis of the graph will be inter-arrival time average λ. The y-axis will be time. As mentioned above, the graph should measure the turn-around & waiting times for each algorithm: Non-preemptive Scheduler and EDF scheduler. Thus the graph will have four curves.

As mentioned above, you can use longrun.c to for all the test-cases. Since you do not have control over system jobs that are in system ready queue while user jobs are running in your user ready queue(s), repeat same test case for at least 5 times (i.e., 5 re-runs of same expt) and do average over those 5 runs to compute statistics of turn-around and waiting Times.

In this analysis, you can ignore the processes that are not scheduled because their deadline was crossed.

Graph2: Comparison of Processes that have missed their Deadlines

In this case, have n, λ , μ1 to be fixed. Then vary μ2. Measure the number of processes that miss their deadlines under both these schedulers. For instance have n, λ, μ1, μ2 to be fixed at 30, 20ms, 20 ms and vary μ2 from 30 to 70 in the increments of 10.

Similar to the above cases, have loopcount to be based on the computed execution time, x2. Let it be C times x2 for some constant C.

Thus the x-axis of the graph will be average deadline time μ2. The y-axis will be number of processes missing their deadlines. The graphs should measure this metric for both algorithms: Non-preemptive Scheduler and EDF scheduler.

Feel free to change these number in case you observe that too many processes are missing their deadlines or all the processes are able to complete their execution within the deadline under both the schedulers. Similar to graph1 take average of each point over those 5 experiments to compute the statistics.

Deliverables

You have to submit the following by 5th November 2016 on google classroom:

- A report consisting of analysis stated above
- The source program named as <rollno.>-sched.c
- a readme file

Create a zipped archive of all your files and name it as <rollno.>-project.zip. Upload it on google classroom. Strictly follow this naming convention. Otherwise, your assignment will not be evaluated.

Your CODE and other associated files must be in a single directory; the TA will copy them to his MINIX installation and compile and run them there. Do not submit object files, assembler files, or executables. Make a list of source and test files in your README file.

Your report document should describe the design of your assignment in enough detail that a knowledgeable programmer could duplicate your work. This includes descriptions of the data structures you use, all non-trivial algorithms. The report should contain a description of each function including its purpose, inputs, outputs, and assumptions it makes about the inputs or outputs. Also the report should contain performance analysis of multiple schedulers in this design document.

Design Document should include:

1. Problem Definition
2. Introduction: Explain Minix architecture in terms of kernel functionalities and server functionalities in perspective of the blocks (eg PM, vFS) used in implementing the code.
3. Explain your design with flow chart to show data and control flow of your code. Details of the changes made to the code along with their files information is to be added as subsections
4. Testing the functionality of the code, write proper test cases for testing the scheduling algorithm.
5. Each test case should contain:
   - Testcase
   - Description of Test case
   - Input
   - Expected Output
   - Sample outputs, screenshots (if any).

Evaluation Criteria:(Total 120 Marks)
1.Execution with Expected output: 50 marks.
2.Design Document & Viva: 50 Marks.
3.Coding Style (Naming conventions, Comments, Indentation, Error Handling) 20 Marks

The TAs will conduct Viva to understand your design better.

The deadline for submission is 5th November 2016, 9:00 pm on google classroom.

Help & Reference:

Refer existing scheduler implementation in minix3.2.1 source code. You may refer the following files for understanding Minix scheduler and kernel source code also needs to be referred.
You should understand the total flow of existing scheduler in Minix like:
1.When new user process enters in which queue it will be placed?

2.When Quantum expires what scheduler will do?
3.How it is prioritizing the processes?
4.How it is balancing the processes among all queues to execute them?

/usr/src/servers/sched/schedproc.h (To include your scheduling criteria parameters)
/usr/src/servers/sched/proto.h (define prototype of schudeuler function)
/usr/src/servers/sched/schedule.c (To define your scheduler code)
/usr/src/include/minix/config.h (To Introduce new Queues)

Reading Materials and other Useful Links

1. Minix3 Wiki: http://wiki.minix3.org/en/
2. Minix Installation: http://wiki.minix3.org/en/UsersGuide/DoingInstallation
3. Overview of Minix: http://www.minix3.org/docs/jorrit-herder/osr-jul06.pdf and http://www.minix3.org/docs/login-2010.pdf
4. Minix Documentation: http://www.minix3.org/documentation/index.html
5. Refer other URLs given on Moodle

Note: This project statement is similar to the project given last year as a part of OS course. So, feel free to discuss with your seniors.