PA1

CSC 501 Spring 2023

PA 1: Process Scheduling

Due: Monday, February 13th, 2023, 11:59 pm

1. Objective

The objectives of this lab are to get familiar with the concepts of process management like process priorities, scheduling and context switching.

2. Readings

The Xinu source code (in sys/), especially those related to process creation (create.c), scheduling (resched.c, resume.c suspend.c), termination (kill.c), priority change (chprio.c), as well as other related utility programs (e.g., ready.c) and system initialization code (initialize.c) etc.

3. What to do

You will also be using the csc501-lab0.tgz you have downloaded and compiled by following the lab setup guide. But you need to rename the whole directory to csc501-lab1. In this Lab you will implement two new scheduling policies that will avoid starvation between processes. At the end of this lab you will be able to explain the advantages and disadvantages of the two new scheduling policies. The default scheduler in Xinu will schedule the process with the higher priority. Starvation is produced in Xinu when there are two or more processes eligible for execution that have different priorities. The higher priority process gets to execute first which results in lower priority processes never getting any CPU time unless the higher priority process ends.

The two scheduling policies that you need to implement, as described below, should address this problem. Note that for each of them, you need to consider how to handle the Null process, so that this process is selected to run when and only when there are no other ready processes.

For both scheduling policies, a valid process priority value is an integer between 0 to 99, where 99 is the highest priority.

1. Aging Based Scheduler

The first scheduling policy is an Aging Based Scheduler. The basic idea of this scheduler is to gradually increase the priority of the processes waiting to be executed every time resched is called. In this scheduler you will have to increment the priorities of the processes waiting to be executed by 1 everytime resched is called. In the case of processes with equal priorities round robin would be used which is the default existing Xinu Policy.

For example, assume that there are three processes eligible for execution P1(1), P2(2), and P3(3) with priorities as mentioned. So if P3(3) is running and P1(1) and P2(2) are waiting to be executed. When resched is called we will have new priorities of processes waiting to be executed as follows: P1(1)-> P1(2) and P2(2)-> P2(3).

```
Initially P3(3) is running and P1(1) and P2(2) are waiting to be executed.

          Ready Queue
             after
          incrementing     Currently Running      Process Scheduled      New Ready Queue
1st Call: P1(2) P2(3)      P3(3) Running          P2(3) Scheduled        P1(2) P3(3)

2nd Call: P1(3) P3(4)      P2(3) Running          P3(4) Scheduled        P1(3) P2(3)
```

2. Linux-like Scheduler (based loosely on the 2.2 Linux kernel)

This scheduling algorithm tries to loosely emulate the Linux scheduler in 2.2 kernel. In this assignment, we consider all the processes “conventional processes” and uses the policies regarding SCHED_OTHER scheduling class within the 2.2 kernel. With this algorithm, the CPU time is divided into epochs. In each epoch, every process has a specified time quantum, whose duration is computed at the beginning of the epoch. An epoch will end when all the runnable processes have used up their quantum. If a process has used up its quantum, it will not be scheduled until the next epoch starts, but a process can be selected many times during the epoch if it has not used up its quantum.

When a new epoch starts, the scheduler will recalculate the time quantum of all processes (including blocked ones). This way, a blocked process will start in the epoch when it becomes runnable again. New processes created in the middle of an epoch will wait till the next epoch, however. For a process that has never executed or has exhausted its time quantum in the previous epoch, its new quantum value is set to its process priority (i.e., quantum = priority). A quantum of 10 allows a process to execute for 10 ticks (10 timer interrupts) within an epoch. For a process that did not get to use up its previously assigned quantum, we allow part of the unused quantum to be carried over to the new epoch. Suppose for each process, a variable counter describes how many ticks are left from its quantum, then at the beginning of the next epoch, quantum = floor(counter/2) + priority. For example, a counter of 5 and a priority of 10 will produce a new quantum value of 12. During each epoch, runnable processes are scheduled according to their goodness. For processes that have used up their quantum, their goodness value is 0. For other runnable processes, their goodness value is set considering both their priority and the amount of quantum allocation left: goodness = counter + priority. Note that round-robin is used among processes with equal goodness.

The priority can be changed by explicitly specifying the priority of the process during the create() system call or through the chprio() function. Priority changes made in the middle of an epoch, however, will only take effect in the next epoch. An example of how processes should be scheduled under this scheduler is as follows:

If there are processes P1, P2, P3 with time quantum 10,20,15 then the epoch would be equal to 10+20+15=45 and the possible schedule (with quantum duration specified in the braces) can be: P2(20), P3(15), P1(10), P2(20) ,P3(15), P1(10) but not: P2(20), P3(15), P2(20), P1(10).

Other Implementation Details:

1. void setschedclass(int sched_class)

This function should change the scheduling type to either of the supplied AGESCHED or LINUXSCHED

2. int getschedclass()

This function should return the scheduling class which should be either AGESCHED or LINUXSCHED

3. Each of the scheduling class should be defined as constants

```
#define AGESCHED 1
#define LINUXSCHED 2
```

4. Some of source files of interest are: create.c, resched.c, resume.c, suspend.c, ready.c, proc.h, kernel.h, etc.

5. Test files test1.c, (https://moodle-courses2223.wolfware.ncsu.edu/pluginfile.php/1237268/mod_assign/introattachment/0/test1.c?forcedownload=1) test2.c (https://moodle-courses2223.wolfware.ncsu.edu/pluginfile.php/1237268/mod_assign/introattachment/0/test2.c?forcedownload=1), test3.c (https://moodle-courses2223.wolfware.ncsu.edu/pluginfile.php/1237268/mod_assign/introattachment/0/test3.c?forcedownload=1) and test4.c (https://moodle-courses2223.wolfware.ncsu.edu/pluginfile.php/1237268/mod_assign/introattachment/0/test4.c?forcedownload=1) demonstrates how to create processes and call `chprio()` to change the priorities.

6. You can use testmain.c (https://moodle-courses2223.wolfware.ncsu.edu/pluginfile.php/1237268/mod_assign/introattachment/0/testmain.c?forcedownload=1) as your test case.

7. Expected results:

For Aging Based Scheduler: Three processes A, B and C are created with priorities 10, 20, 30. In each process, we keep increasing a variable, so the variable value is proportional to CPU time. The ratio should be close to 1 : 1 : 1 (this ratio is only with respect to this example). In general, the ratio might vary depending on the priorities taken. You are expected to get similar results as follows:

```
Start... B
Start... C
Start... A

Test Result: A = 110709, B = 106170, C = 110942
```

For Linux-like Scheduler: Three processes A, B and C are created with priorities 5, 50 and 90. In each process, we print a character ('A', 'B', or 'C') at a certain time interval for LOOP times (LOOP = 50). The main process also prints a character ('M') at the same time interval. You are expected to get similar results as follows:

```
MCCCCCCCCCCCCCBBBBBBBMMMACCCCCCCCCCCCCBB
BBBBBMMMACCCCCCCCCCCCBBBBBBBMMMACCCCCCCC
CCCCBBBBBBBMMABBBBBBBMMMABBBBBBBMMMABBBB
BBBMMMBMMAMMMAMMMMMAMMMAMMMAMMMMMMAMMAMM
MMMAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

4. Additional Questions

Write your answers to the following questions in a file named Lab1Answers.txt (in simple text). Please place this file in the sys/ directory and turn it in, along with the above programming assignment.

1. What are the advantages and disadvantages of each of the two scheduling policies? Also, give the advantages and disadvantages of the round robin scheduling policy originally implemented in Xinu.
2. Describe when each of the schedulers run the NULL process.
3. Give two suggestions how the above mentioned aging based scheduler could be changed to attain better fairness keeping the fundamental point of incrementing the priority to be the same in your new ideas.

Turn-in Instructions

Electronic turn-in instructions:

1. go to the `csc501-lab1/compile` directory and do `make clean`.
2. go to the directory of which your `csc501-lab1` directory is a subdirectory (NOTE: please do not rename `csc501-lab1`, or any of its subdirectories.)e.g., if `/home/uid/csc501-lab1` is your directory structure, goto `/home/uid/`
3. create a subdirectory TMP (under the directory `csc501-lab1`) and copy all the files you have modified/written, both .c files and .h files into the directory.
4. compress the `csc501-lab1` directory into a tgz file and submit on Moodle PA1. Please only upload one tgz file. `tar czf csc501-lab1.tgz csc501-lab1`

You can write code in main.c to test your procedures, but please note that when we test your programs we will replace the `main.c` file! Therefore, do not put any functionality in the `main.c` file.

Also, ALL debugging output **MUST** be turned off before you submit your code.

Grading Policy

- (10%) Source code can be compiled and the generated image is bootable. Please note that you will also get 0 point for the second part if your source code can not be compiled or can not generate a bootable image.
- (75%) Aging Based (30) Linux Like Scheduler (45).
- (15%) Each additional question earns five points.

Back to the CSC501 web page (https://wordpress-courses2223.wolfware.ncsu.edu/csc-501-002-sprg-2023/index/)

Search ... Search

Recent Comments

Archives

Categories

- No categories