Due: Thursday Oct 22, 11:59PM

Students enrolled in CSCI 423 should complete this project with your partner. Students enrolled in CSCI 501 must complete this project independently. Each student/group must complete this project without the aid of any other student/group or any other source of Xinu code. Discussing high-level OS and Xinu concepts with other members of other groups is allowed provided that the specifics of how to code the project is not discussed.

Always follow the submission directions at the end the this document. Following directions is part of your grade. Any variance from the stated directions may result in zero credit for the project. Note that you can re-submit any project after the due date/time for a 30% penalty.

Objectives: After completing this assignment students will:

- understand the properties of operating system process scheduling.
- understand the problem of process starvation.
- be able to implement a process scheduling algorithm that avoids process starvation.

Preparation:
First, make a fresh copy of your work thus far.

    cp -R xinu-proj3 xinu-proj4

Untar the new project files on top of this directory:

    tar xvzf /home/ruth/pub/xinu-proj4.tgz

You should now see the new project files in with your old files. Be certain to make clean before compiling for the first time.
Priority Scheduling:

Your task for this assignment is to modify your Xinu kernel by adding priorities to each process and scheduling the use of the cpu based on the priority. To complete this assignment, you will have to make changes to several header files in the include/ directory, and several .c files in the system/ directory.

First, add priority scheduling to your operating system. This can be done in 5 easy steps:

- Add a priority field into PCB structure defined in include/proc.h.
- Add an effective priority field key into the process queue structure defined in include/queue.h. This effective priority field will be used as a key to sort the queue.
- Add priority parameter to create(), and properly initialize the priority field in each new PCB created.
- Build a new function in system/prioritize.c to implement a priority sorted ready queue. This does not require making any modifications to the primary queue maintenance functions already in system/queue.c. Add your new source file in compile/Makefile.
- Modify the ready() and resched() functions to properly use your new priority scheduler.

New testcases are in system/main.c, which should demonstrate priority-order execution once your new scheduler is operational. You will need to create others to fully test your implementation.

Starvation and Aging:

One of the chief drawbacks to simple priority scheduling is that low priority processes may be starved by high priority processes, that is, they may never get to run at all. One remedy for this is to implement aging, a scheme in which the effective priority of a process increases the longer it sits in the ready list without running.

Add a kernel configuration parameter AGING into include/kernel.h that when TRUE causes the effective priority of each process in the ready list to increase by one every time resched() is called.
Construct a test case that demonstrates process starvation when AGING is FALSE, but demonstrates aging when AGING is set to TRUE. Call your test case function main-starve, and put it in your main.c.

Preemption:

The new files system/clock* (as well as some adjustments to various loader configuration files) will provide you with basic preemption, as discussed in class. Take time to familiarize yourself with the contents of these files, as you will be responsible for understanding how these components of the operating system work.

Activate preemption by changing the PREEMPT constant in include/kernel.h to TRUE. How can you test that preemption is working in your system? Create a main program called main-preempt that demonstrates preemptive scheduling. Add it to main.c.

Submission:

Before submitting your project please type make clean from your compile directory. This removes the compiled files leaving only the source files. This is necessary to save space on the server.

To submit your files ON-TIME use the following command:

    turnin -c csci423 -p proj4 xinu-proj4

To submit your files LATE use the following command:

    turnin -c csci423 -p proj4-late xinu-proj4

Any variance from the following requirements may result in zero credit for the project.

- Your project MUST be submitted using the turnin command on xinu.cs.olemiss.edu.
- You MUST include the entire directory xinu-proj4.
- Your project MUST compile and run on xinu.cs.olemiss.edu using all of the original files with the exception of your create.c and ctxsw.S files.