CSCI 460 Operating Systems - Fall, 2015
Assignment 1: CPU Scheduling

Due Date

The assignment must be submitted by 8:00am on Tuesday, October 20. This assignment is worth 10% of the overall assessment in the course. Students must submit their assignments via the Canvas course web site. Assignment files are to be packed into a single zipped tar file named `WnnnnnnnnAssg1.tar.gz` (where `Wnnnnnnnn` is your student W-number).

Task

Your task is to implement two CPU scheduling algorithms which are simplified versions of the algorithms used in FreeBSD: the 4BSD scheduler and the ULE scheduler, and compare their performance under a simulated load. Both schedulers use a round-robin time-slice strategy whereby the process dispatched from the ready queue runs on the CPU until either:

- Its time-slice expires and it is returned to the ready queue;
- It waits for I/O; or
- It terminates.

The 4BSD Scheduler

This scheduler implements the ready queue as a 16-level priority feedback queue. Each process is assigned a priority in the range 0 through 15. The process's priority is varied during its execution, depending on the behavior of the process. When a process is created and first enters the ready queue, it is assigned a priority of 8. If the executing process uses its entire time-slice, its priority is decreased by 1 before it is returned to the ready queue. If the executing process waits for I/O, its priority is increased by 1 before it is returned to the ready queue, after the I/O operation is complete. The process selected by the scheduler for dispatch to the CPU for a context switch is the highest priority process that has been on the ready queue the longest.

The ULE Scheduler

This scheduler uses two queues, the *current* queue and the *next* queue. When a process is created, it is added to the *current* queue. If the executing process uses its entire time-slice, it is added to the end of the *next* queue. If the executing process waits for I/O, it is added to the end of the *current* queue after its I/O operation is complete. When the *current* queue becomes empty, the two queues are switched: the *next* queue becomes the *current* queue and the *current* queue becomes the *next* queue. The process selected by the scheduler for dispatch to the CPU for a context switch is the process at the head of the *current* queue.

Scheduler Performance

You are to measure the performance of the two schedulers under the simulated process load, for various values of the time-slice. The items to be measured are:

1. The average, minimum and maximum time that processes spend in the ready queue.
2. The average, minimum and maximum time that processes take to respond to an I/O operation. This is the time between the completion of an I/O operation and the time that the process is next dispatched to the CPU.

3. The proportion of time spent on scheduler overheads. This is the total run time minus the total time processes spend executing on the CPU.

Provided Software

The following software is provided for your use in the assignment.

SchedSim.h	Header file, defining the Simulate() function, provided by the simulator.
Dispatcher.h	Header file, defining the functions that you need to provide, for the simulator to call
SchedSim.o	Static library file for the 64-bit Linux environment, containing the simulator object code. This works with gcc.
Assg1.c	A demonstration program which implements the functions defined in Dispatcher.h and has a main() function which invokes the Simulate() function. You can compile and run this program with the provided header files and the static library file. Note that this program is for demonstration purposes only; it has limited functionality.

Software that you must write

You must write one or more C source files that implement all the functions defined in Dispatcher.h and provide a main() function, which invokes the Simulate() function defined in SchedSim.h.

What you must submit

- Source files, written in C, and a Makefile which can be used with the provided files to build an executable file under Linux with gcc, using the make utility.
- A short report of your findings on the performance of the two scheduling algorithms, for several values of the time-slice.

All your submitted files must be bundled into a single zipped tar file named `WnnnnnnnnAssg1.tar.gz` (where `Wnnnnnnnn` is your student W-number).

Grading

Your assignment grade will be determined by the following factors:

- Correctness of your implementation of the scheduler algorithm.
- Quality of the software that you write for the assignment, including choice of data structures, performance efficiency, conformance to the coding standards listed below, and the use of good programming practices.
- Completeness of your data collection and determination of the parameters of the scheduling algorithms.

Saving your files in a zipped tar file

You need to submit all your C files, header files, report file and Makefile. First you need to bundle them up into a single *tar* file. The term "tar" is an abbreviation of "tape archive" and goes back to the days when people would save a back-up copy of their files on magnetic tape. Nowadays, with the price of large disk drives so low, nobody uses tape anymore, but the concept of tar files survives.

1. Use the command:

   ```
   tar -cf WnnnnnnnnAssg1.tar *.c *.h Makefile docfile
   ```

   (where Wnnnnnnnn is your W-number and *docfile* is your report file).

The -cf specifies two options for the tar command: 'c' means create and 'f' means that the name of the resulting tar file comes next in the command.

Following the name of the tar file, we list the files to be included in the tar file. In this case, we want all the files in your current directory whose names end with `“.c” or “.h”,` the Makefile and the report document.

2. If you now use the command `ls` you should now see the file WnnnnnnnAssg1.tar in your directory.
3. If you use the command

   ```
   tar -tf WnnnnnnnAssg1.tar
   ```

   (where Wnnnnnnn is your W-number), it will list the files within the tar file.
4. Now compress the tar file using the gzip program:

   ```
   gzip WnnnnnnnAssg1.tar
   ```

5. By using the `ls` command again, you should see the file WnnnnnnnAssg1.tar.gz in your directory. This is the file that you need to submit through the **Assignment 1 Submission** link in the course web site.

Coding Standards

1. Use meaningful names that give the reader a clue as to the purpose of the thing being named.
2. Avoid the repeated use of numeric constants. For any numeric constants used in your program, use #define preprocessor directives to define a macro name and then use that name wherever the value is needed.
3. Use comments at the start of the program to identify the purpose of the program, the author and the date written.
4. Use functions to provide a solution to each sub-problem. Ensure that no function gets too large. Use 20 lines maximum as a guideline.
5. Use comments at the start of each function to describe the purpose of the function, the purpose of each parameter to the function, and the return value from the function.
6. Use comments at the start of each section of the program to explain what that part of the program does.
7. Use consistent indentation.