THE UNIVERSITY OF WAIKATO
DEPARTMENT OF COMPUTER SCIENCE

COMP301-13B — OPERATING SYSTEMS

ASSIGNMENT 2 : SCHEDULER

LECTURER	Anthony Blake
EMAIL	`ablake@waikato.ac.nz`
DUE	Friday 23 August 2013
WORTH	15% of final grade

INTRODUCTION

For this assignment, you will implement a system in userspace for scheduling tasks to be performed sometime in the future. I will refer to these scheduled tasks as "timers". Timer-based schedulers are very common in an operating system context: the cron program in Unix-style operating systems is an obvious example.

This assignment is designed to test your ability to manage both pointers and allocated memory effectively. You will need to be careful to avoid memory leaks and segmentation faults in your code. I would strongly recommend that you test your code using valgrind, which may help catch any obvious errors.

Be sure to read the assignment specifications carefully failure to use the correct data structures for your scheduler will cost you a lot of marks.

This assignment is worth 15% of your final grade for this course.

Note: This assignment is an individual assignment. Work on your own. You may discuss the assignment in general terms but do not share your code with anyone else. If you need help, either post on Moodle or contact the lecturers.

SPECIFICATION

Your goal is to write a simulated userspace scheduler using C. Your scheduler will take one argument on the command line: the name of a file containing a series of instructions that will act as input to your scheduler. Each line of this file will describe a single instruction and each instruction must be enacted in order of appearance in the input file.

To keep the assignment simple, there are some abstractions that you need to be aware of.

Time: Your scheduler will not need to operate in real-time. Instead, your scheduler will be periodically told what the current time is via an instruction in the input file; there is no need to worry about checking the system clock or anything like that. Time will be expressed as an unsigned 32 bit integer.

Tasks: To execute a task, simply print the current time and the task name to standard output. You do NOT need to execute any binaries, regardless of what the task is called.

There are six basic instruction types that you will need to implement for your scheduler. Some instructions will also specify arguments - these will follow the instruction type and will be separated by whitespace:

ADD: Schedule a non-repeating task. The first argument following ADD will be an unsigned integer describing the time when the task is to be executed. A second argument will give the name of the task. The name of the task will be a single word only.

If a task is scheduled for a time before or at the current time, it may be ignored.

ADDREP: Schedule a repeating task. An ADDREP instruction will have three arguments (in order): the first time that the task is to be executed, an integer describing how frequently the task should be repeated, and the name of the task (again, a single word only).

If a repeating task is scheduled for a time before or at the current time, all executions prior to and at the current time may be ignored but the task should be still be scheduled for the next appropriate time after the current time.

TIME: Tells the scheduler what the current time is. Only one argument follows the TIME instruction the current time as an integer. Following this instruction, you should execute any tasks which were scheduled to occur before the current time.

DEL: Removes all instances of a task from the scheduler. The argument after DEL will be the name of the task to be removed. If the task is a repeating task, then the removal also applies to all repeats of that task, i.e. if I DEL a repeating task, it should never occur again without specifically ADDing it again.

Deleting a task that does not exist in the list should have no effect.

LIST: Simply displays the entire schedule in chronological order. There are no arguments with a LIST command.

When a LIST instruction is encountered, your scheduler should write "Upcoming tasks:" to standard output, followed by a newline. Each pending task should then be written to standard output on a separate line. The line should contain the task name, then a space, then the time remaining until the task is due to execute. When writing out a repeating task, only the next scheduled execution of the task needs to be printed.

CLEAR: Clear the entire schedule list. No arguments are provided. All pending tasks should be removed from the list, including repeating tasks.

An example instruction file (and expected output) will be available on Moodle, which should demonstrate clearly what each instruction type will look like.

IMPLEMENTATION

Your scheduler should be implemented as a double linked list, sorted by the time that each task is due to execute. Each item in a double linked list will contain two pointers: one to the next item in the list (like a regular linked list) and one to the previous item in the list. This allows the list to be traversed in both forward and reverse order.

The reason for using a double linked list is that new scheduled tasks are more likely to be inserted at the end of the list rather than the front. As a result, it is more efficient to perform an insert operation starting at the end of the list and moving backwards. However, executing tasks still requires us to traverse the list in the conventional order so we have to be able to traverse the list in both directions.

To summarise, all insert operations should start at the tail of the list and work backwards through the list until the appropriate place in the list is found. All other operations should start from the head of the list and work forwards.

For this assignment, you must implement the double linked list yourself. You may not use any library implementations, e.g. glib. You may choose to implement your various list operations using either a recursive or iterative approach. An iterative solution is probably slightly easier, but you will not be able to rely as heavily on the example code given in class.

Assumptions

Some assumptions you can make that may make your life easier:

- No task name will be longer than 256 characters.
- No line in the input file will be longer than 1024 characters.
- All input files will begin with a TIME instruction that sets the current time.

Assessment

Your assignment must be submitted in electronic form using Moodle. You should submit an archive (e.g. made using tar) containing all of the relevant files.

Your submission must include:

- The source code for your scheduler
- A Makefile that can be used to build your scheduler, simply by typing 'make' on the command line.

Marks for the assignment will be distributed as follows:

- Designing Appropriate Data Structures: 10 marks
- Input Handling: 10 marks
- Double Linked List Implementation: 20 marks
- Producing Correct Output: 10 marks
- Memory Management: 15 marks
- Error Checking and Handling: 10 marks
- Coding Style and Documentation: 10 marks
- Makefile: 5 marks

The total number of marks for this assignment is 90 and the assignment is worth 15% of your final grade for this course.