Notes for A1 markers

Read the assignment. If there is something you don't understand please ask me to explain it.
For each student allocated to you: download the zipped submission (if they submitted). Unzip it. There should be Part1, Part2 and Part3 directories. If not please make them and put OSA1.c in Part1, OSA2.c in Part2 etc, along with either their littleThread.h or the original one. Put the original threads1.c in Part1, threads2.c in Part2 etc and the corresponding threads1alt.c, threads2alt.c, threads3alt.c.
Enter the marks in the rubric provided in the FastGrader section of Assignment 1 on Canvas. Get there from the Grades link.
DO NOT UNMUTE THE ASSIGNMENT. We will do this when all the marking is complete.

Marking the programs

Marking guide (28 marks = 7%)
Students name and UPI in every file.
If they haven't modified a file e.g. littleThread.h they don't have to have their name and UPI in it.
But they should in OSA1.1.c, OSA1.2.c, OSA1.3.c and A1answers.txt.
1 mark

Part 1 - OSA1.1.c
There is only the one call to the switcher function in the main function.
This was intended to stop people putting a loop in their code. Because no loop is required in scheduling. (Obviously a loop is ok when printing the thread states.) I should have said no loops for scheduling but they get the mark for this if there is only one call to switcher in the main function.
1 mark

There is a simple scheduler function which selects the next thread to run. The threads are maintained in a circular linked list.
Check there is a scheduler function which seems to select the next thread to run. They should be playing with at least the "->next" field in the struct thread. If they schedule from an array rather than a linked list take off 0.5 of a mark.
1 mark

The stack space for a thread is freed when the thread completes.
There should be a "free(thread->stackAddr)" somewhere after the "(thread->start)()". In my case it is in the scheduler function which is called from removeThread which is called from associateStack.
1 mark

There is a printThreadStates function which prints out the state of all threads in the order of creation of the threads. Even finished threads are shown.
This should be easy to see when you run the program.
2 marks

Works correctly using given threads1.c.
Mark in the Linux image in the lab (or possibly your own recent version of Ubuntu but if there are problems you really need to compare the result with the lab Linux image).

Add the files "threads1.c" and "threads1alt.c" to the student's Part1 directory. Also add "littleThread.h" if they didn't include it.
Compile using "gcc OSA1.c". Run with "./a.out".
The output should be as on page 3 of the handout. Do NOT worry about extra or fewer spaces/lines etc.
Focus on:
The states of the two threads at the beginning should be "ready".
First thread 0 is running, then it prints "hi" 5 times, then if changes to finished and thread 1 is running.
Thread 1 prints "bye" 5 times, then it finishes too. When a thread finishes the line "disposing n" should appear.
The program should stop (i.e. it shouldn't get stuck).
Full marks if all the above. Take off 0.5 of a mark for each thing which doesn't work properly.
Add a comment if marks are taken off.
2 marks

Works correctly using different threads1.c.
You will have to edit their OSA1.1.c code to "#include threads1alt.c" instead of "threads1.c".
Mark very much like the above but all 5 threads have to run.
2 marks

Part 2 - OSA1.2.c
There is a threadYield function which schedules the next thread to run.
1 mark

Works correctly using given threads2.c. The program terminates cleanly by returning to the main function when the threads have all finished.
Add the files "threads2.c" and "threads2alt.c" to the student's Part2 directory. Also add "littleThread.h" if they didn't include it.
Compile using "gcc OSA2.c". Run with "./a.out".
Mark very much like Part1. The output should show jumping between threads when they call threadYield as on page 4 of the handout. If the program doesn't finish cleanly (after all threads are finished) they get zero.
1 mark

Works correctly using different threads2.c.
Change the code to use "threads2alt.c" as in Part1.
Things to look for. All threads are finished at the end. Program terminates cleanly.
The last thread to finish must be threadID 3, the one to print "hi 4".
When there is only the one remaining thread they don't have to print the Thread States because they only HAVE to do it when switching to a new thread (and at the start and finish).
2 marks

Part 3 - OSA1.3.c
Using given threads3.c the program output shows that the threads are being pre-empted.
Add the files "threads3.c" and "threads3alt.c" to the student's Part3 directory. Also add "littleThread.h" if they didn't include it.
Compile using "gcc OSA3.c". Run with "./a.out".

First of all make sure that their setitimer code is setting the signal for every 20 milliseconds.

```
        timer.it_interval.tv_sec = 0;
        timer.it_interval.tv_usec = 20000;      // 20 milliseconds
        timer.it_value.tv_sec = 0;
        timer.it_value.tv_usec = 20000;
```

Some people claimed they always finished each thread to completion if using 20 milliseconds. I have my doubts about this, but if it looks like this you can drop the tv_usec values to 1000 (1 millisecond). If their solution is still finishing each thread without any preemption then I would presume it isn't working properly and they don't get any marks.
You should see a reasonable amount of output (or lots if you changed the code to 1 millisecond) with bits of output coming from each of the threads scattered throughout the Thread States output.
Give the two marks if it appears that threads are being preempted.
At the end all threads should be marked "finished". Otherwise take off a mark.
2 marks

The program terminates cleanly by returning to the main function when the other threads have all
finished.
1 mark

Works correctly using different threads3.c.
Make the change to their code to include "threads3alt.c" and try again.
Almost always the last thread to finish should be either ID 1 or ID 2. If not, run it again twice more. If none of these have one of those threads finishing last give only 1 mark. (There is a small probability this will not work but after three times I would be sceptical that their code works properly.)
2 marks

Marking the questions
Question 1
There is a comment saying "Wow!" in the switcher function in OSA1.c. This is to indicate something very bad is happening here. Explain what it is.
The problem is the code is throwing away the stack it is currently executing on (this is enough to get the 3 marks, if they don't mention it is the current stack they can only get 2 marks). We get away with it in this case because we immediately call longjmp but this is really a bad thing to do. In a real system the thread should really run to completion and its stack would be cleared up by another thread.
3 marks

Question 2
Why are the time consuming calculations in threads3.c required in order to demonstrate the effectiveness of the pre-emptive scheduler?
So that we can see that preemption is really happening. If the threads finish before the time slice is up we won't see this.
2 marks

Question 3
In threads3.c there is some code around the call to rand() to block signals and then allow them again. Explain what can happen if this is not done. Also give an explanation as to why this can happen.
Without stopping the timer signals during the rand function the program can get stuck (self deadlock) - 2 marks
This is because rand is not thread or signal or async safe. - 1 mark for something like this.
In reality rand sets a lock which a thread acquires as it executes, then if preemption occurs at this point the next thread will eventually call rand and find the lock taken, hence it blocks. As far as the OS is concerned there is only one thread in the process and so the process is stopped. The last mark for something like this.
4 marks

Take off marks for late submissions.
If submitted on the 22nd of August take off 5% of their earned mark. e.g. If they got 20/28 then take off 5% of 20 = 1. So the final mark is 19/28.
If submitted on the 23rd of August take off 10% of their earned mark.
Enter a comment if penalised for late submission.