# Homework 3 — due in class on Tuesday 26 September

Total number of points available on this homework is 105. Full credit is equivalent to 100 points.

1. (10 pts.) Stallings Section 3.5 states that Unix is unsuitable for real-time applications because a process executing in kernel mode may not be preempted. Elaborate.

2. (15 pts.) The Unix kernel will dynamicaly grow a process's stack in virtual memory as needed, but it will never try to shrink it. Consider the case in which a program calls a C function that allocates a local array on the stack that consumes 10KB. The kernel will expand the stack segment to accommodate it. When the subroutine returns, the stack pointer is adjusted and this space could be released by the kernel, but it is not released. Explain why it would be possible to shrink the stack at this point, and why the Unix kernel does not shrink it.

3. (15 pts.) Linux provides a system call named `clone`. It creates a new process identifier but allows the programmer to decide whether or not the new process should share memory with the old process and whether the new process should share open files with the old process (and a few other things). The result is four ways to create a new process. Identify the meaning and one potential use of each combination of sharing.

4. (15 pts.) On Unix, the `top` command allows you to observe some information about running processes. Run

```
top d 30
```

and observe the columns labelled CPU and COMMAND. (On systems other than Linux the command arguments could be slightly different.) CPU shows the percentage of the CPU's time that the process is getting. COMMAND identifies the user command being run in the process. The display will be updated every 30 seconds.

While top is running, run Emacs (or your favorite text editor) and type random text until the top display is updated twice. (Don't type words. Keep your eye on the top display.) What percentage of the CPU time does Emacs get while you are typing? How about top? Now, with Emacs running, but without typing into Emacs, observe top again through two display updates. What percentage CPU time does Emacs have now? How about top now? Are they the same? If so, why? If not, why not?

Now compile the following C program:

```
main ( ) {
  while ( 1 ) {
    ;
  }
}
```

Watch top through at least two display updates. What do you observe this time? How do you explain any differences that you observe? Please be sure to kill this program by typing control-C as soon as you are done.

5. (10 pts.) What hardware and OS resources does a thread use? How do the resources a thread uses differ from those a process uses? Name an application that benefits from threads, and one that does not.

6. (10 pts.) For kernel-level threads, what actions must the kernel take on a switch between threads in the same process? Between processes? Compare with corresponding actions for user-level threads.

7. (15 pts.) In the description of Solaris user-level threads (Stallings 4.5), it is stated that a user-level thread may yield to another thread of the same priority. Isn't it possible that there will be a runnable thread of higher priority and that therefore the yield function should result in yielding to a thread of the same or higher priority?

8. (15 pts.) Given the following mix of jobs, job lengths, and arrival times, assume a time slice of 15 and compute the completion and average response time of each job for the FIFO (first-come first-served), RR (round-robin), and SRTF (job with shortest remaining time first, with preemption) algorithms. Please use the following table format for your solution.

| | | | Scheduling Algorithms | | |
|---|---|---|---|---|---|
| Job | length | arrival time | FIFO | RR | SRTF |
| 0 | 75 | 0 | | | |
| 1 | 40 | 10 | | | |
| 2 | 25 | 10 | | | |
| 3 | 20 | 80 | | | |
| 4 | 45 | 85 | | | |
| | | Avg. RT | | | |