IT5002 Computer Systems and Applications
Tutorial 6
SOLUTIONS

1. Using Google or otherwise, research and show how bootstrapping works on Windows, LINUX or MacOS (pick ONE OS to talk about).

   Here we will cover MacOS (Specifically OS X). This is a quick summary, for the full list please see http://osxdaily.com/2007/01/22/what-happens-in-the-mac-os-x-boot-process/

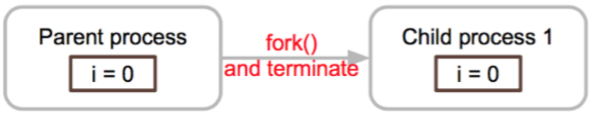
   - On power-on, the Mac starts executing code in the Unified Extensible Firmware Interface (EFI), which is in ROM. EFI is not unique to Macs and it replaces BIOS (Basic Input Output System) that was used in earlier computers.
   - EFI initializes all the hardware, and loads and hands control over to /System/Library/CoreServices/BootX, which is the OS X boot loader. BootX and Mach (see later) all run at a high kernel security level (kernel mode).
   - BootX loads the kernel, and starts loading up all the device drivers that are needed to run the screen, touchpad, WiFi interface, etc on your Mac.
   - The kernel init routine is started, and at this point the Mac's firmware hands control over to OS X. Various data structures are initialized.
   - I/O Kit is started, giving the OS access to devices on the Mac via their drivers.
   - The Mach service, an extensible microkernel communications service, is started. Mach provides various services like communication ports, remote procedure call (RPC) support, notifications, interprocess communications, etc.
   - The kernel switches to a standard user security level (user mode), and starts /sbin/init, which becomes the first process in the system (init is the master process of any UNIX system)
   - Init looks at rc.boot and rc.common to start up user-level scripts. These scripts do things like start up servers, or initialize software systems the user needs.
   - Init runs fsck, which determines if any full filesystem check is needed.
   - The main filesystem /dev/fd is mounted and is now accessible.
   - /etc/rc.cleanup is executed to do any filesystem cleaning, if needed.
   - Various network services are started.
   - The swap partition is mounted and the virtual memory system is started.
   - The OS X graphical shell is started.

2. How many processes are created in the following program? Include the original process created when the program is first run.

```
for(int i=0; i<10; i++) fork();
```

It is extremely tedious to work this out by hand. Fortunately there is an easier way. We start first with for(i=0; i<1; i++):

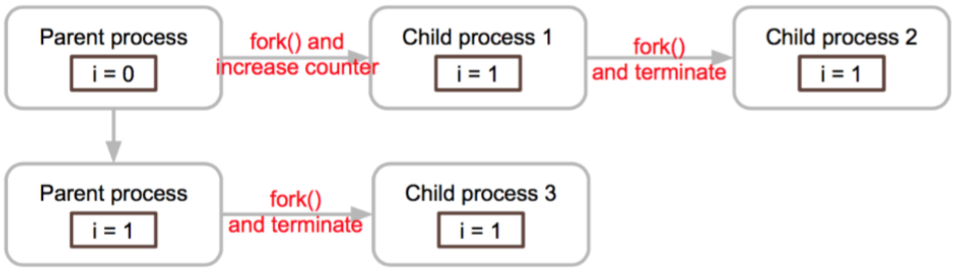
Analysis for (int i = 0; i < 1; i++):



Total number of processes: 2

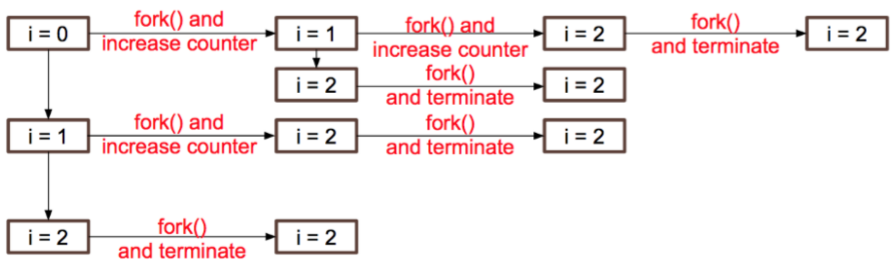
We do the same for for(i=0; i<2; i++):

Analysis for (int i = 0; i < 2; i++):



Total number of processes: 4

Analysis for (int i = 0; i < 3; i++):



Total number of processes: 8

```
We can see that the forks form essentially a binary
tree, and that for the general case n, we will have
2^n forks.
```

3. Explain, with an example, why the highest priority task can still be interrupted by the lowest priority interrupt, and how this affects ISR design. (Hint: Interrupts are implemented in hardware in the CPU itself).

This is because task priorities are seen only by the OS. As far as the CPU is concern, it is "just another program" that is running, with instructions being fetched and executed. On the other hand interrupt lines are checked at the end of each instruction execution cycle, so interrupts are always serviced regardless of their priority level (and of the priority level of the running task).

4. We know that to support multitasking we need to save the registers, program counter, stack pointers and machine status word (SREG in the lecture notes). What other pieces of information does the OS need to save about a process? Explain each piece.

   File handles / Open File Table: These are data structures that maintain information about files that are opened by the process, like the location that a process is inside the file, access rights to the file, file open modes, etc.
   Pending signals: A "signal" is an OS message to the process. For example SIGINT interrupts a process (pressing CTRL-C triggers a SIGINT), SIGTERM terminates a process (triggered using kill process_id), SIGKILL kills a process (triggered using kill -9 process_id), SIGSUSPEND suspends a process (triggered when you press CTRL-Z) etc. These signals either trigger default behaviors (e.g. breaking a process), or can be caught and handled by signal handlers in the program. More information: The difference between SIGTERM and SIGKILL is that SIGTERM allows a process to gracefully shut down its child processes, while SIGKILL kills the process immediately and can leave orphan child processes. Be kind. Always use SIGTERM and not SIGKILL. Don't leave orphans behind!
   Process Running State: Whether the process is suspending, ready, running, terminated, etc.
   Accounting Information: How much CPU time the process has used, how much disk space, network activity, etc.
   Process ID: Unique number identifying the process. Etc.

5. Given four batch jobs T1, T2, T3 and T4 with running times of 190 cycles, 300 cycles, 30 cycles and 130 cycles, find:

i) The average waiting time to run if the jobs are executed in order.

Task	Waiting time
T1	0
T2	0+190 = 190
T3	0 + 190 + 300 = 490
T4	0 + 190 + 300 + 30 = 520

Total = 0 + 190 + 490 + 520 = 1200 cycles
Average = 300 cycles waiting time.

ii) The average time to run if the jobs are executed SJF.

Task	Waiting time
T3	0
T4	0 + 30 = 30
T1	0 + 30 + 130 = 160
T2	0 + 30 + 130 + 190 = 350

Total waiting time = 0 + 30 + 160 + 350 = 540 cycles
Average = 135 cycles

From your answer and otherwise, explain the advantage and disadvantage of SJF. In particular, what if the number of jobs running is not fixed and new jobs can be added at any time?

SJF means that jobs on average wait less time to be executed, which means that if you submitted many jobs, you will get, on average, a faster response. This makes the system "feel better" because you don't have to wait too long to get a result.

SJF is prone to starvation if there's a long task, and someone keeps submitting short tasks.