CS 537: Intro to Operating Systems (Fall 2017)
Worksheet 1 - Intro & Processes

Due: Sep 13th 2017 (Wed) in-class OR email to Zhewen before 5:30 pm

1. **Intro to Operating Systems**

   (a) **Virtualization**
   Consider the two C code snippets shown below. You may assume that this code compiles and executes without any errors.

Program 1

```
int main(int argc, char *argv[]){
  int *ptr = malloc(sizeof(int));
  *ptr = 1;
  printf("pid:%d; addr: %p; value:%d\n", getpid(), ptr, *ptr);
  sleep(10);
  printf("pid:%d; addr: %p; value:%d\n", getpid(), ptr, *ptr);
}
```

Program 2

```
int main(int argc, char *argv[]){
  int *ptr = malloc(sizeof(int));
  printf("pid:%d; addr: %p; value:%d\n", getpid(), ptr, *ptr);
  sleep(10);
  *ptr = 2;
  printf("pid:%d; addr: %p; value:%d\n", getpid(), ptr, *ptr);
}
```

Assuming that Address Space Layout Randomization (ASLR) is disabled, what are the outputs of the following two scenarios? You may assume that the pids for process 1 and process 2 are 1 and 2 respectively and the address of the malloc'd memory in heap is 0xA0B0C0.

Scenario i. Run program 1, after 5 seconds, run program 2.

Scenario ii. Run program 2, after 5 seconds, run program 1.

If the above 2 code snippets are run on a machine with a single CPU, and a main memory of size 1 GB, what are the hardware resources that are being virtualized? Only CPU OR only memory OR both? Explain your answer.

(b) **Concurrency**
Assume that two threads are running the following assembly code (that increments a shared variable) concurrently on a uniprocessor computer.

```
mov 0x2000, %eax  # get the value at the address
add $1, %eax      # increment it
mov %eax, 0x2000  # store it back
```

Assume that the initial value in the memory address 0x2000 is zero (0). Now the two threads get interleaved as follows.

Thread 0	Thread 1

`mov 0x2000, %eax`
`add $1, %eax`	

	`mov 0x2000, %eax`
`mov %eax, 0x2000`	

	`add $1, %eax`
	`mov %eax, 0x2000`

i. What is the value stored in address 0x2000 after all the above instructions are executed?

ii. Assume there is a loop outside these three lines of assembly code and each thread runs this loop for 100 times. What is the **minimum** and the **maximum** possible final value in address 0x2000 after both these threads execute this assembly code 100 times? The assembly instructions for the two threads may get interleaved in any order during execution.

iii. What is the problem here and how can it be solved?

(c) **Persistence**
Consider the C code snippet shown below:

```
int main(int argc, char *argv[]){
  int fd = open("/tmp/file",  O_WRONLY | O_CREAT | O_TRUNC, S_IRWXU);
  int buf[3] = {100, 200, 300};
  write(fd, buf, 3*sizeof(int));
}
```

i. After executing this program, file `/tmp/file` was created and some bytes were written. What type of file is it?

ii. What will we see if we open this file in a text editor like vim? Can you explain why you see these contents?

iii. If we open this file with a program like **xxd**, the following contents are displayed. Can you explain what this content means and why is it formatted in this way? Can you also comment on the **endianness** of the machine in which this code was run? You may assume that 0x64 is stored at byte 0.

```
6400 0000 c800 0000 2c01 0000
```

iv. What is the **size** of the file (in bytes) `/tmp/file` at the end of the `write()` system call? You may assume that the size of an integer is 4 bytes.

v. What does the option `S_IRWXU` in `open()` stand for?

2. **Processes**
   Assume you have a system with three processes (A, B, and C) and a single CPU. Processes can be in one of five states: (a) RUNNING, (b) READY, (c) BLOCKED, (d) NOT YET CREATED or (e) TERMINATED

   Given the following cumulative timeline of process behavior, indicate the **state** the specified process is in AFTER that step, and all preceding steps, have taken place.

   Step 1: Process A is loaded into memory and begins; it is the only user-level process in the system.
   Process A is in which state?

   Step 2: Process B is created and is scheduled to run.
   Process A is in which state?

   Process B is in which state?

   Step 3: The running process issues an I/O request to the disk.
   Process A is in which state?

   Process B is in which state?

   Step 4: Process C is created but it is not yet scheduled.
   Process A is in which state?

   Process B is in which state?

   Process C is in which state?

   Step 5: The time-slice (i.e., time for which a process can use the CPU) of the running process expires. Process C is scheduled.
   Process A is in which state?

   Process B is in which state?

   Process C is in which state?

   Step 6: The previously issued I/O request completes; the process that issued that I/O request is scheduled.
   Process A is in which state?

   Process B is in which state?

   Process C is in which state?