CS 450 Exam 1 — Mon. 2/29/2016

Name: _____

Rules and Hints

- You may use one handwritten 8.5 × 11″ cheat sheet (front and back). This is the only additional resource you may consult during this exam. *No calculators.*
- Include step-by-step explanations and comments in your answers, and show as much of your work as possible, in order to maximize your partial credit.

Grade

	Your Score	**Max Score**
Problem 1: Hardware support		16
Problem 2: Mode transfer		12
Problem 3: Syscalls		16
Problem 4: Child processes		16
Problem 5: Pipes		20
Problem 6: Processes and threads		20
Total		**100**

Problem 1: Hardware support (16 points)

The first three sub-questions refer to hardware-supported kernel vs. user modes.

Part A

Why are privileged ISA instructions necessary? Give an example or a scenario that requires them.

Part B

Why is hardware support for memory management necessary? Give an example or scenario that requires it.

Part C

Why are interrupts necessary? Give an example or scenario that requires them.

Part D

State at least one attribute of an ISA or hardware that is helpful for virtualization, and explain.

Problem 2: Mode transfer (12 points)

Give an example of each of the following types of mode transfer:

Part A

Externally triggered, user to kernel

Part B

Implicitly triggered by the program, user to kernel

Part C

Explicitly triggered by the program, user to kernel

Problem 3: Syscalls (16 points)

Part A: Programmer

When a programmer wants to make a system call in x86, what steps must his/her assembly code take?

Part B: Hardware

What does the hardware need to do to execute a single machine-code syscall instruction?

Part C: Syscall handler

What does the syscall handler need to do (besides the actual open(), write(), or whatever)?

Part D: Hardware, part 2

What does the hardware need to do to execute a return-from-syscall instruction?

Problem 4: Child processes (16 points)

Show a possible output for the following program (assuming that all necessary headers have been included). Answers may vary – you will get full credit if your answer is internally consistent.

```
int main(void) {
    pid_t x = fork();
    pid_t y = fork();

    printf("Process %d (parent %d) forked %d and %d\n",
              getpid(), getppid(), x, y);

    return 0;
}
```

Problem 5: Pipes (20 points)

Part A: Code (12 points)

Add to the skeleton code on the next page so that the three child processes combine to execute the following command: `ps aux | grep root | more`. Assume that all necessary headers have been included. You do not need to worry about error handling, but you should close all unused file descriptors.
Note: you will get partial credit for just running these 3 commands, so don't leave this blank!

Part B: Visualization (8 points)

Draw the final state of the two pipes and the three child processes' file descriptor tables.

Intentionally left blank to make sure that the code and instructions for Problem 5 are separated.

```
int main(void) {    // Skeleton code for Problem 5
    int pfd1[2], pfd2[2];
    pipe(pfd1);
    pipe(pfd2);

    if ( fork() == 0 ) {









    }
    if ( fork() == 0 ) {









    }
    if ( fork() == 0 ) {









    }




    while ( wait(NULL) != -1) ;
}
```

Problem 6: Processes and threads (20 points)

Part A: True/False (12 points)

Circle "True" or "False" to answer each question. If you circle an answer and then change your mind, write out *your entire answer* next to the row.

Note: If two processes (or threads) share an address space, then Address X refers to the same actual memory location (and thus the same data) for each.

Parent and child **processes** share the same file descriptor table by default.	True	False
Parent and child **processes** share the same global variable area in memory.	True	False
Parent and child **processes** share the same stack.	True	False
Parent and child **processes** share the same heap.	True	False
Parent and child **processes** share the same code segment in memory.	True	False
Parent and child **processes** share the same address space.	True	False
Parent and child **threads** share the same file descriptor table by default.	True	False
Parent and child **threads** share the same global variable area in memory.	True	False
Parent and child **threads** share the same stack.	True	False
Parent and child **threads** share the same heap.	True	False
Parent and child **threads** share the same code segment in memory.	True	False
Parent and child **threads** share the same address space.	True	False

Part B: Communication (8 points)

Name a mechanism that allows a child **process** to communicate data to its parent.

Name a mechanism that allows a child **thread** to communicate data to its parent.

Is it possible for a parent process to exit while its child continues executing? What about a parent thread?