CprE 308 Homework 1

Department of Electrical and Computer Engineering
Iowa State University

Student Name: Gurumanie Singh Dhiman

University-ID: 911192340

Problem 1 (10 points)
Answer true (T) or false (F) for each following statements:

(T) 1. OS is responsible for managing the hardware resources in a computer.
(F) 2. Linux is one of the most widely used Microkernel OS.
(F) 3. The heap region of a process' address space is automatically allocated and deallocated.
(F) 4. System call is used by the hardware to communicate with the OS.
(T) 5. Context-switch time is overhead.
(T) 6. OS controls the execution of programs to prevent errors and improper use of the computer.
(T) 7. Monolithic kernel may have higher performance than Microkernel in practice because it incurs less communication overhead among modules.
(F) 8. Type 1 hypervisor requires support from an underlying OS.
(F) 9. The malloc() library call in C dynamically allocates memory in the stack region.

Problem 2 (5 points)
What is a process? What is a "Zombie" process?

A process is a program in execution (also considered as active, whereas program is considered passive), and is an instance of a running program. A process can be in one of three basic states, namely, running, ready, and blocked (however practical OSes typically have more than three states.
"Zombie" processes are a special process state, it is basically a process that has completed execution but the entry is still visible in the process table. This allows the parent to read the child's exit status, and once the exit status is read, then the zombie process is removed (reaped). The downside to having too many of these types of "Zombie" processes is that they still use resources such as memory.

Problem 3 (5 points)
Compare the advantage & disadvantage of Monolithic kernel and Microkernel.

	Monolithic Kernel	Microkernel
Advantages	Faster, less seperated so the communication is faster in comparision to a Microkernel	Provides isolation, more secure in comparision to a Monolithic Kernel

Disadvantages	Less secure and less reliable compared to Microkernel. A bug in one service can crash the entire kernel, and it is harder to maintain.	Slower due to extra overhead of inter process communications (IPC) between the kernel and user-space services.

Problem 4 (5 points)
What is the maximum depth of the stack for the following code in terms of number of stack frames (one stack frame per function call)? Be sure to count the first call to main.

```
int main(){
    f (12);
    return 1;
}

int f (int n){
    if (n <= 0)
        return 0;
    else
        return f (n – 1) + 2 * f (n – 4);
}
```

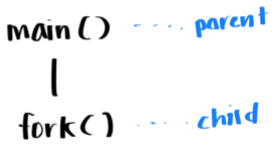
The maximum depth of the stack in terms of number of stack frames is 14. This is because we have 1 call to main, then we call the function f 12 times until it hits the base case, and we call it 1 more time after. So the maximum depth of the stack will be a total of 1 + 12 + 1 = 14.

Problem 5 (5 points)
(1) How many processes does the following code create? (2) Draw the process tree of the program.

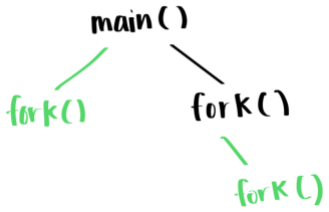
```
int main() {
        int i;
        for (i=1; i<3; i++)
                fork();
        return 1;
}
```

The following code above creates 4 processes.

for i = 1 :



for i = 2 :



Problem 6 (5 points)
Please write all possible outputs from the following piece of code:

```
int main(void) {
pid_t pid = fork();
if (pid > 0) {
        printf("I am
        the parent\n");
 } else if (pid == 0){
        printf("I am the child\n");
 }
        else printf("ERROR!\n");
       return 0;
}
```

Solution 1:
I am the parent
I am the child

Solution 2:
I am the child
I am the parent

Solution 3:
ERROR!

Problem 7 (5 points)
Consider the following code. Assume all system calls return successfully and the actual process IDs of the parent and child during the execution are 2600 and 2603, respectively. What are the values of pid/pid1 at lines A, B, C, D?

```
int main() { pid_t
        pid, pid1;
        pid =
fork();          if
(pid < 0) {
              fprintf(stderr, "Fork Failed");
                return 1;
      } else if (pid == 0){
                pid1 = getpid();
              printf("child: pid = %d", pid); /* A */
              printf("child: pid1 = %d", pid1); /* B */
       }
       } else {
                pid1 = getpid();
              printf("parent: pid = %d", pid); /* C */
              printf("parent: pid1 = %d", pid1); /* D */
                wait(NULL);
       }
      return 0;
}
```

Line A: 0
Line B: 2603
Line C: 2603
Line D: 2600