1. What is the difference between a thread and a process?

2. Which process has pid #1?

3. Give one example of a signal we have seen in class.

4. Why can fork() fail?

5. How many processes are created (not including the process associated with main) by the following code fragment? You can assume fork calls will always be successful.

   ```c
   int main() {
     fork();
     fork();
     return 0;
   }
   ```

6. Implement the process function below. The function creates a child process that computes and prints the square of the child_value parameter. The parent process performs the same computation, but using the parent_value parameter. For example, running the program with input values 4 and 8 will generate the following results:

   4 8
   Child: 64
   Parent: 16
   Processing Done

   - You may not modify the main function.
   - The child will print the square value by using the message "Child: " followed by the result.
   - The parent will print the square value by using the message "Parent: " followed by the result.
   - We don’t care about the order in which the output appears. For example, ”Processing Done” can appear before the child output.

   ```c
   int main() {
     int parent_value, child_value;

     scanf("%d%d", &parent_value, &child_value);
     process(parent_value, child_value);
     printf("Processing Done\n");

     return 0;
   }

   void process(int parent_value, int child_value) {
   ```

7. Why do we need to reap processes?

8. Suppose a parent process creates three children and executes a single call to wait(). Which process will be reaped?

9. Suppose a child process has finished, but init cannot reap it. What can you say about the status of the parent process?

10. Suppose you don’t know how many children a parent process created. What code snippet do you need to make sure all child processes are reaped?
11. What will happen to the stack and heap of a process when an exec* system call (e.g., execvp) is executed? What will happen to the process id?

12. Implement a C program that creates two child processes. One child will execute the "who" Unix command and the other the "cal 5 2015" command. The parent process will print the message "Done" after both children have finished. Make sure the parent reaps its children.