CSSE 332 - Operating Systems

Exam 1 Review

Name: Solution Key

Question 1. Consider the following snippet of code, take a moment and think carefully about what it is trying to do.

```
1  for(int i = 0; i < 2; i++) {
2    int pid = fork();
3    if(pid == 0) {
4      int my_pid = getpid();
5      printf("I am the middle child, my pid is %d!\n", my_pid);
6      pid = fork();
7      if(pid == 0) {
8        printf("I am the grandchild, my pid is %d\n", my_pid);
9      } else {
10       printf("Middle child going away forever!\n");
11       exit(0);
12     }
13   }
14   printf("I am the parent and I just gave birth to my first child!\n");
15 }
```

(a) (5 points) What will print on the screen when this code executes?

> **Solution:** Will be a mangle of things as the grandchildren will not exit, so they will continue the loop, printing as if they were parents, and then going on to spawn children and grandchildren of their own.

(b) (5 points) The code snippet contains crucial bug, what is it and suggest a way to fix it?

> **Solution:** Grandchildren need to call `exit`.

Question 2. (5 points) Circle the correct answer each time, suggest a fix if the answer is *false*.

- On the exam, I should open the `man` page for `fork` and `wait`, lookup functions or macros we haven't used in class and use those to solve the problem (e.g., `WNOHANG`).

  A. True

  B. False

- To wait for a specific child, you can use the `wait` system call but you pass it an integer pointer (e.g., `wait(&status);`).

  A. True

  B. False

- I would like to execute the binary `buffalosay.bin` with the input arguments `1` and some integer `x`. To do so, I would use

  ```
  execlp("./buffalosay.bin", "1", (char*)x, 0);
  ```

  A. True

  B. False

- There are signals that cannot be overridden or ignored.

  A. True

  B. False

- Using `alarm(3);` guarantees that `SIGALRM` will be delivered **exactly** 3 seconds after it has been set.

  A. True

  B. False

- You can use the same pipe to communicate in both directions between a parent and a child.

  A. True

  B. False

Question 3. (5 points) In the box below, write down the pattern used to wait **for a specific** child and read its exit status.

Solution:

```c
waitpid(childpid, &status, 0);
if(WIFEXITED(status)) {
  int ec = WEXITSTATUS(status);
  printf("Child %d exited with code %d\n", childpid, ec);
}
```

Question 4. (5 points) In the box below, write down the general structure of creating a pipe between a parent and a child where the parent is the writer and the child is the reader.

Solution:

```c
int fd[2];

int rc = pipe(fd[2]);
if(rc < 0) {
  perror("PANIC: pipe failed!");
  exit(EXIT_FAILURE);
}

if(fork() == 0) {
  // child is reader
  clode(fd[1]);

  read(fd[0], ...);

  close(fd[0]);

  exit(0);
} else  {
  // parent is writer
  close(fd[0]);
  write(fd[1], ...);
  close(fd[1]);
  exit(0);
}
```

Question 5. (5 points) Consider a scenario with three processes: a parent and two children. The parent has created a pipe between itself and its two children. It then wants to send `hello` to its first child and `bye` to its second child. It is crucial that each child receives the message intended for it.

Mohammad suggests to implement this using the following code snippet in each process (assume all pipes and ends are set up correctly).

```c
// parent

// Send "hello" to child 1
write(fd[1], "hello", 5);

// Send "bye" to child 2
write(fd[1], "bye", 3);
```

```c
// child 1
char buff[6];

// I want to read the
   hello from my parent
read(fd[0], buff, 5);
buff[5] = '\0';
```

```c
// child 2
char buff[4];

// I want to read the
   bye from my parent
read(fd[0], buff, 3);
buff[4] = '\0';
```

(a) (5 points) Describe possible ways in which the above code snippet can go wrong!

Solution: We don't know who will read first and consume things from the pipe. Child 2 might get a partial message of that of child 1, and then will child 1 might get the rest.

(b) (5 points) Describe a possible way to fix the above issues.

Solution: Use two separate pipes.