Name: Email:

Operating Systems CS 1217 Spring 2023

Quiz 1 - 15th February 2023

Solutions

Instructions

1. This quiz is **20 minutes, 20 points** and accounts for **3%** of your final grade.
2. Be precise in your answers; unnecessarily verbose answers will fetch negative marks
3. Write your answers in the space provided only.

Q1 Where are pipe objects located? Briefly describe how the `pipe()` system call is used to set up complex pipelines of bash commands, taking an example (similar to the one) discussed in class. (5 points)

Pipes are buffers located in memory. Pipes can provide a mechanism for one-way IPC between a parent and child processes, allowing for information transfer between the two.

The pipe system call creates an anonymous pipe object that can be addressed using two file handles: a read and a write handle.

In one example, the parent fork()s a child process, which needs the output of the parent as its input. A pipe object with both read and write descriptors is created by the parent, and the descriptors are replicated in the child. After the child is created, to create a one-way flow of information (from the parent to the child) the parent has to close the “read” end of the pipe, whereas the child has to close the write end of the pipe.

Q2 Consider a parent process P that has forked a child process C in the program below.

```
int a = 5;
int fd = open(...) //opening a file
int ret = fork();
if (ret > 0) {
  close(fd); //closing a file
  a = 6;
  ...
}
else if(ret==0) {
  printf("a=%d\n", a);
  read(fd, something);
}
```

After the new process is forked, assume that the parent is run first. Also assume that the child process is run after the parent completes its changes. (a) What is the value of the variable **a** as printed in the child process? (b) Will the attempt to read from the file descriptor succeed in the child? Explain your answers (10 points)

Prints 5, since the value is only changed in the parent.

Yes it will - the file is only closed in the parent. Rule of thumb - once a fork() happens successfully, the parent and child go their own separate ways. Copy-on-write (CoW), in this case, it just an optimization to fork(): the semantics of fork() don't change because of the optimization.

Q3. What would be the output of the **child** process in the code below. Why? (5 points)

```
int returnCode = fork();
//echo is a program that takes an input string
//and prints it out
char app[] = "/bin/echo";
char * const string_1[] = {app, "I am a string!!", NULL};
char * const string_2[] = {app, "I am also a string??", NULL};

if ( returnCode == 0 ) {
   exec(app, string_1);
   printf( "I am the Process A!\n" );
} else {
   exec(app, string_2);
   printf( "I am Process B!\n");
}
```

Ans: I am a string!! The child is exec'd with echo app, which prints **string_1**. The printf after the exec is never "reached".