CSSE 332 -- Operating Systems

Rose-Hulman Institute of Technology

Pipes

Name: Solution Key

Question	Points	Score
Question 1	5	
Question 2	20	
Question 3	5	
Question 4	5	
Question 5	10	
Total:	45	

Question 1. (5 points) In Unix, "unnamed" pipes are **bidirectional** means of communication that are managed by the kernel.

    A. True.

    B. False. Please remember that pipes are unidirectional.

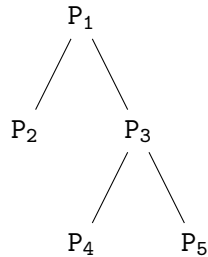**Question 2**. Consider the processes with the lineage relationship shown in Figure 1 below.

Figure 1: Lineage relationship for processes in **Question 2**

(a) Assume that before forking P_4 and P_5, P_3 creates a Unix pipe (let's call it σ) using the `pipe` system call.

    i. (5 points) P_4 and P_5 can communicate with each using σ.

        A. True.

        B. False.

    ii. (5 points) P_4 cannot use σ to communicate with P_3.

        A. True.

        B. False.

    iii. (5 points) P_4 can use σ to communicate with P_1.

        A. True.

        B. False.

(b) (5 points) Assume now that P_2 creates a pipe using the system call `pipe`, which of the below processes can P_2 communicate with using that pipe?

    A. P_1 B. P_3 C. P_4 D. P_5 **E. None of the above.**

Question 3. (5 points) A process that needs to **read** from a pipe must _____**close**_____ the **writing end** of that pipe. It can then use the _____**read**_____ system call to extract bytes from the pipe.

Question 4. (5 points) Briefly describe the events that happen when a process attempts to read from a pipe that has no more writers, but whose writing ends are still open.

> **Solution:** The process will hang forever since it assumes that more data might show up sometime in the future. It is essential to close writing ends when done to indicate to the reading processes that no further data will be added.

Question 5. Consider the following code snippet.

```
1  if(pipe(fd) < 0) {
2    perror("PANIC");
3    exit(EXIT_FAILURE);
4  }
5  int rc = fork();
6  if(rc == 0) {
7    // sleep for some 20 seconds, give parent time to write.
8    char buff[5];
9    int len;
10   while((len=read(fd[0], buff, 4))) {
11     buff[len] = 0;
12     printf("Read %s\n", buff);
13   }
14   close(fd[0]);
15 }
16 // close reading end
17 close(fd[0]);
18 write(fd[1], "hello world!", strlen("hello world!"));
19 write(fd[1], "nice try!", strlen("nice try!"));
20
21 // done
22 close(fd[1]);
23 // do other stuff and wait for child.
```

(a) (5 points) The code above contains a bug. Find it and suggest a way to fix it.

> **Solution:** The child process does not close the writing end of the pipe before starting to read. Add `close(fd[1]);` before the read operations.

(b) (5 points) Assume now that the bug has been fixed and that **all** of the parent's write operations finish before the child process reaches the `while` loop. What would be the output on the console when the child reads from the pipe?

> **Solution:** The child process will read the messages from the parent 4 bytes at a time, which means we will print them out as follows:
>
> ```
> hell
> o wo
> rld!
> nice
> try
> !
> ```