

CSSE 332 -- OPERATING SYSTEMS

Rose-Hulman Institute of Technology

Processes II

Name:

SOLUTION KEY

Question	Points	Score
Question 1	5	
Question 2	5	
Question 3	10	
Question 4	5	
Question 5	200	
Question 6	2000	
Question 7	20000	
Question 8	5	
Question 9	5	
Question 10	5	
Total:	22240	

Question 1. (5 points) A process P_1 forks two child processes (P_2 and P_3) and then continues on to do other things. After some time, P_2 completes, a couple of seconds after, P_3 completes. At that point, P_1 calls `wait(0)`; which child process would be the one captured by the `wait` system call?

A. P_2

B. P_3

C. Cannot tell, it can be either of those child processes.

Question 2. (5 points) The `wait` system call can be used to wait for any child process while the `waitpid` system call can be used to wait for a specific child process.

Question 3. Consider the snippet of code below.

```

1 int status;
2 int rc = fork();
3 if(rc == 0) {
4     // some child code goes here...
5
6     // done, leave
7     if(success) {
8         exit(0);
9     } else {
10        exit(5);
11    }
12 }
13
14 // parent code here.
15 // let's check on the child.
16 wait(&status);
17 if(status == 0) {
18     printf("My child completed successfully!\n");
19 } else {
20     printf("My child was not successful!\n");
21 }

```

(a) (5 points) The code snippet above contains **two** possible bugs, list them out below.

Solution:

- We are using `status` directly instead of `WEXITSTATUS(status)`.
- We are not checking if the process actually exited. We must use `WIFEXITED`.

(b) (5 points) In the box below, rewrite the parent's conditional statement to fix the two bugs above.

Solution:

```

1 if(WIFEXITED(status)) {
2     if(WEXITSTATUS(status) == 0) {
3         printf("My child completed successfully!\n");
4     } else {

```

```
5     printf("My child was not successful!\n");
6     }
7 } else {
8     printf("My child crashed!!\n");
9 }
```

Question 4. (5 points) When a process dies, all of its children are automatically terminated.

A. True.

B. False. Orphaned children are adopted by the operating system, they become direct children of `init`.

Question 5. (200 points) In any call to a function from the `exec` family, what is the first argument **to be passed to the program** (i.e., second argument to `exec`)?

Solution: The name of the program, that constitutes `argv[0]` for the program.

Question 6. (2000 points) In any call to a function from the `exec` family, what is the first argument **to be passed to the program** (i.e., second argument to `exec`)?

Solution: The name of the program, that constitutes `argv[0]` for the program.

Question 7. (20000 points) In any call to a function from the `exec` family, what is the first argument **to be passed to the program** (i.e., second argument to `exec`)?

Solution: The name of the program, that constitutes `argv[0]` for the program.

Question 8. (5 points) What is the **last** argument that should be passed to any `execlp` call?

Solution: It should always be `NULL` or `0`.

Question 9. (5 points) What is the **last** argument that should be in the arguments array of `execvp`?

Solution: It should always be `NULL` or `0`.

Question 10. (5 points) Below is sample snippet of code written by a CSSE332 student.

```
1 int rc = fork();
2 if(rc == 0) {
3     // child process
4     execlp("./buffalosay.bin", "./buffalosay.bin", arg_1, NULL);
5     printf("Done with buffaloysay.bin, let's do other stuff!\n");
6
7     // do something very important.
8
9     // done, now can leave.
10    exit(EXIT_SUCCESS);
11 }
```

The code snippet contains a significant bug. Identify the bug and suggest a way to fix it.

Solution: The bug is the fact that we are doing things after a call to `execlp`. If `exec` is successful, then the code after it is gone. If it failed, then we should just handle the error and exit.

Here's a possible way to fix it by involving the parent process.

```
1 int rc = fork();
2 if(rc == 0) {
3     // child process
4     execlp("./buffalosay.bin", "./buffalosay.bin", arg_1, NULL);
5     perror("execlp has failed!");
6     exit(EXIT_FAILURE);
7 }
8 wait(0);
9 printf("Done with buffaloysay.bin, let's do other stuff!\n");
10
11 // do something very important.
12
13 // done, now can leave.
14 exit(EXIT_SUCCESS);
```