CSSE 332 -- OPERATING SYSTEMS

Processes II

Name: SOLUTION KEY

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.

```c
int status;
int rc = fork();
if(rc == 0) {
  // some child code goes here...

  // done, leave
  if(success) {
    exit(0);
  } else {
    exit(5);
  }
}

// parent code here.
// let's check on the child.
wait(&status);
if(status == 0) {
  printf("My child completed successfully!\n");
} else {
  printf("My child was not successful!\n");
}
```

(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:

```
if(WIFEXITED(status)) {
  if(WEXITSTATUS(status) == 0) {
    printf("My child completed successfully!\n");
  } else {
    printf("My child was not successful!\n");
  }
} else {
  printf("My child crashed!!\n");
}
```

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.

```c
int rc = fork();
if(rc == 0) {
  // child process
  execlp("./buffalosay.bin", "./buffalosay.bin", arg_1, NULL);
  printf("Done with buffaloysay.bin, let's do other stuff!\n");

  // do something very important.

  // done, now can leave.
  exit(EXIT_SUCCESS);
}
```

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.

```c
int rc = fork();
if(rc == 0) {
  // child process
  execlp("./buffalosay.bin", "./buffalosay.bin", arg_1, NULL);
  perror("execlp has failed!");
  exit(EXIT_FAILURE);
}
wait(0);
printf("Done with buffaloysay.bin, let's do other stuff!\n");

// do something very important.

// done, now can leave.
exit(EXIT_SUCCESS);
```