**Advanced Operating System**

**Ch5 Process API homework**

**Cheng Lin Hsu**

**Questions1:**Writeaprogramthatcallsfork().Beforecallingfork(),havethe main process access a variable (e.g., x) and set its value to some- thing (e.g., 100). What value is the variable in the child process? What happens to the variable when both the child and parent change the value of x?

**Code**: **q1.c**

一張含有 文字 的圖片

自動產生的描述

**Execution** **Result for q1.c :**

一張含有 文字 的圖片

自動產生的描述

**Solution**: In Line 9, we have declared a variable x with value 100. As we can see that in Lines 15-20 and 21-28 show the code for the child process and the parent process, respectively. In Lines 18 and 25, we print out the value of x for the child and the parent process, which both are 100 (as shown in lines 4 and 7 of the execution results). It is because the child process is created with the same content of the parent -- including the data segment as well as the stack. Later, in Lines 19 and 26, we changed the values of x in the child and the parent to 111 and 222, respectively. We also print out the changed values in Lines 20 and 27, which are 111 and 222. This can be shown that the forked child process has the same contents of its parent at the time it was created. However, after that, the child and the parent are two independent processes.

**Questions** **2**:Write a program that opens a file (with the open() system call) and then calls fork() to create a new process. Can both the child and parent access the file descriptor returned by open()? What happens when they are writing to the file concurrently, i.e., at the same time?

**Code**: **q2.c**

**一張含有 文字 的圖片

自動產生的描述**

**Execution** **Result for q2.c :**

一張含有 文字 的圖片

自動產生的描述

**Solution**: In Line 11 we have declared a variable file with open(). As we can see that in Lines 17-20 and 21-27 show the code for the child process and the parent process, respectively. In Lines 19 and 24, we declared a pointer to a character \*s with some character ,for the child and the parent process, respectively. In Lines 20 and 25,we used write() to write what is s stored to q2.txt (as shown in lines 3 and 4 of the execution results).So for the question mark1 my answer is both child and parent can access the file descriptor open using open().and the for the question mark2 my answer is both are able to write to the file but the order in which they do is un-deterministic(if we don’t use wait()).

**Questions** **3** :Write another program using fork(). The child process should print “hello”; the parent process should print “goodbye”. You should try to ensure that the child process always prints first; can you do this *without* calling **wait()** in the parent?

**Code**: **q3.c**

**一張含有 文字 的圖片

自動產生的描述Execution** **Result for q3.c :**

**一張含有 文字 的圖片

自動產生的描述**

**Solution:** First In Line 12 we declared a variable wc to fork(),

And then we can see that in Lines 17-20 and 21-24 show the code for the child process and the parent process, In Lines 19 we print out hello for the child process ,And then In Lines 23 we print out goodbye for parent process (as shown in Line 2 and 3 of the execution result ,To make sure that the child process always print first and without calling wait() in parent ,In Lines 22 ,We used waitpid() to ensure that child always print first.( waitpid() is a function which is similar to wait()

**Questions** **4:** Write a program that calls fork() and then calls some form of exec() to run the program /bin/ls. See if you can try all of the variants of exec(), including (on Linux) execl(), execle(), execlp(), execv(), execvp(), and execvpe(). Why do you think there are so many variants of the same basic call?

**Code q4.c**:

一張含有 文字 的圖片

自動產生的描述

**Execution** **Result for q4.c :**

**一張含有 文字 的圖片

自動產生的描述**

**Solution:** In Line 8 we declared a variable wc to call the fork() system call , And then in Lines 11-24 show the code with child process. Respectively. In Lines 13 and 14 we declared a character pointer argv and envp to set parameter for exec(),then in lines 17-23 we can try what we want to use for exec() function, If you want to use another exec() ,Just need to delete the comment what we want to use ,and then comment what you don’t use.(as shown the execution result in Lines 3.For the question mark1 my answer is how the program is found, how the arguments are specified, and where the environment comes from.

* The calls with v in the name take an array parameter to specify the argv[] array of the new program. The end of the arguments is indicated by an array element containing NULL.
* The calls with l in the name take the arguments of the new program as a variable-length argument list to the function itself. The end of the arguments is indicated by a (char \*)NULL argument. You should always include the type cast, because NULL is allowed to be an integer constant, and default argument conversions when calling a variadic function won't convert that to a pointer.
* The calls with e in the name take an extra argument (or arguments in the l case) to provide the environment of the new program; otherwise, the program inherits the current process's environment. This is provided in the same way as the argv array: an array for execve(), separate arguments for execle().
* The calls with p in the name search the PATH environment variable to find the program if it doesn't have a directory in it (i.e. it doesn't contain a / character). Otherwise, the program name is always treated as a path to the executable.

**Questions5:**Now write a program that uses wait() to wait for the child process to finish in the parent. What does wait() return? What happens if you use wait() in the child?

**Code q5.c**:

一張含有 文字 的圖片

自動產生的描述

**Execution** **Result for q5.c :**

**一張含有 文字 的圖片

自動產生的描述**

**Solution:** In Line 9, we have declared a variable wc with wait(). As we can see that in Lines 14-16 and 17-20 show the code for the child process and the parent process, respectively. In Lines 16 and 19, we print out who I am for the child and the parent process , and then In Line 21-22 we print out pid and wc for child and parent (as shown in lines 2-3 and 4-5 of the execution results).so if we used wait() in parent it will return child’s pid and then if we used wait() in child it will return -1

**Questions** **6:** Write a slight modification of the previous program, this time using waitpid() instead of wait(). When would waitpid()be useful?

**Code q6.c**:

一張含有 文字 的圖片

自動產生的描述

**Execution** **Result for q6.c :**

**一張含有 文字 的圖片

自動產生的描述**

**Solution:** I In Line 9, we have declared a variable wc with waitpid(). As we can see that in Lines 14-16 and 17-20 show the code for the child process and the parent process, respectively. In Lines 16 and 19, we print out who I am for the child and the parent process , and then In Line 21-22 we print out pid and wc for child and parent (as shown in lines 2-3 and 4-5 of the execution results),

So for the question mark2 my answer is if you have many child process and you don’t want to wait for all child process complete you can us waitpid() instead of wait(),because use waitpid() just need to wait a specific pid what you choose.

**Questions** **7:** Write a program that creates a child process, and then in the child closes standard output (STDOUT FILENO). What happens if the child calls printf() to print some output after closing the descriptor?

**Code q7.c**:

一張含有 文字 的圖片

自動產生的描述

**Execution** **Result for q7.c:**

**一張含有 文字 的圖片

自動產生的描述**

**Solution:** In Line 9, we have declared a variable rc with fork() As we can see that in Lines 14-17 and 18-22 show the code for the child process and the parent process, In Line 16 we close the standard output , and then in line 17 and 20 we tried to print who am I for child process and parent process .but in execution result we can see only parent print who am I,because of that we can know if we close the STDOUT before print we can’t print anything.

**Questions 8:** Write a program that creates two children, and connects the stan- dard output of one to the standard input of the other, using the pipe() system call.

**Code q8.c**:

一張含有 文字 的圖片

自動產生的描述

**Execution** **Result for q8.c :**

**一張含有 文字 的圖片

自動產生的描述**

**Solution:** We create a pipe with the pipe() system call in Line 14 .We create the first child with a fork() in Line 16,which we call rc1, then, in the section of code running for the parent, we make a second fork() call to make the second child, which we call rc2 in Line 33. child for rc1 close STDOUT and p[1],After that we uses dup to make STDOUT point to the write end of the pipe and passes a string to the write end of the pipe in Line 26-29. Then child for rc2 close STDIN and p[0] ,After that we uses dup to make STDIN point to the read end of the pipe and use a buff to reads from the read end of the pipe. This connects the STDOUT of rc1 to the stdin of rc2 In Line 43-48.(as shown in execution result line 4.