HW 6

Mike Hanling

3 MAR 2017

Questions

1. (10 points) What is the difference between `_exit()` and `exit()` and `_Exit()`?

   `_exit()` is the system call that exits out of a process
   `exit()` is a library call that cleans up the current process and then calls `_exit()`
   `_Exit()` is a library call that is simply a wrapper for the sys call `_exit()` (no clean up)

2. (5 points) When a process successfully returns from `main()`, which of the three different exit calls is actually used? What exit value is typically used for the process when it returns from `main()` and why?

   A return from main calls `exit()` with an exit value of 0

3. (5 points) What is the difference between unbuffered, line buffered, and fully buffered with respect to output streams?

   Unbuffered: Every write call will immediately write to its dest Line Buffered: Writes will be commited once a new line character is supposed to be printed, or when the buffer is full
   Fully Buffered: The contents of the buffer will only be written once the buffer is completely filled

4. (20 points) Consider the following program snippets. What are the outputs of each? **Explain your answer!**

   (a)
```
int main(){
  fprintf(stdout, "Hello World!");
  return 0;
}
```

   ”Hello World!”

   (b)
```
int main(){
  fprintf(stdout, "Hello World!");
  exit(0);
}
```

   ”Hello World!”

   (c)
```
int main(){
  fprintf(stdout, "Hello World!");
  _Exit(0);
}
```

nothing

(d)
```
int main(){
  fprintf(stderr, "Hello World!");
  _exit(0);
}
```

nothing

5. (10 point) Why does the following code snippet properly check for a failed call to `execv()`?

```
int main(){
  char * ls_args[2] = { "/bin/ls", NULL} ;

  execv( ls_args[0], ls_args);
  perror("execve failed");

  exit(1); //failure
}
```

If the call to `execv` runs smoothly, then nothing sequentially after that call in the C program will execute because the processes are completely swapped out. Thus, having the call to `perror` afterwards will handle the possibility of the `ls` throwing an error.

6. (10 points) Consider setting up an **argv** array to be passed to execv() for the execution of following command:

```
ls l a /bin /usr/bin Fill in
```

Complete the **argv** deceleration in code

```
char * argv[] = { /* what goes here? */ } ;
```

`argv[1]` given that the program takes no other arguments

7. (5 points) The **fork()** system call is the only function that returns *twice*. Explain why this is?

It returns once as the parent and once as the parent

8. (5 points) If you were to compile and run the following program in the shell, which process'es **pid** would print to the screen? **Explain**

```
int main(){
  printf("Parent pid: %d\n", getppid());
}
```

It will return the pid of the kernel. The `getppid()` system call gets the pid of the current process's parent. The parent of the main is the kernel.

9. (5 points) The `wait()` system call will return when a child's status change of a child. What is the most typical status change that would make the system call return?

   Termination

10. (15 points) Using the manual page, provide a brief description of each of the status macros below:

    (a) `WIFEXITED()`

    Returns true if the child terminated normally (exit(3) or _exit(2) or return from main)

    (b) `WIFEXITSTATUS()`

    Does not exist. Notes and my man page says WEXITSTAUS(). Returns the exit status of the child. Should only be employed if WIFEXITED() returns true

    (c) `WIFSIGNALED()`

    Returns true id the child process was terminated by a signal

11. (10 points) Assume you were writing a program that checked if a file existed by using `ls`. (This is a silly way to do this, but just for the sake of argument)

    Recall that `ls` returns an exit status of 2 when the file does not exist and it cannot list it, and `ls` returns an exit status of 0 when the file does exist and can be listed. Complete the `wait()` portion of the program below. The output should be EXISTS! if the file specified in `argv[1]` exists and DOES NOT EXIST! If the file specified in `argv[1]` does not exist.

    (*hint: actually try and complete the program on your computer*)

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>
#include <sys/types.h>

int main(int argc, char * argv[]){
  pid_t cid;
  char * ls_args[] = {"ls", NULL, NULL};
  if(argc == 2){
    ls_args[1] = argv[1];
  }
  cid = fork();
  if( cid == 0 ){ /*child*/
    execvp(ls_args[0],ls_args);
    exit(1); /*error*/
  }

  /*parent*/
  int status;
  wait(&status);

  /* FINISH THIS PROGRA */

}
```

```
    if(WIFEXITED(status)) {
      if(WEXITSTATUS(status) == 0)
        printf("EXISTS!");
      if(WEXITSTATUS(status) == 2)
        printf("DOES NOT EXIST!");
    }

    return 0;
```