HW 7

Joshua Thompson

4 March 2017

Questions

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

   `_exit() is a system call while the others are library calls`

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?

   The program typically uses the exit() function and returns with the value 0. Which shows that the program ran without any errors.

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

   The difference is when the information is released, whether it be input information or output. For unbuffered information it is released immediately which is good for error messages. For line buffered, the information is released when a newline character is read. For fully buffered, the information is released when the buffer size is 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” is the output

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

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

There is no output

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

there is no output

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
}
```

It checks correctly because of the perror and the exit(1). They are only called when the execv fails

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? */ } ;
```

”/bin/ls”, ”-l”, ”-a”,”/bin”,”/usr/bin”,NULL;

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

This is because it runs two instances of the program where the function is called. In order to end the two programs, it must return from each of them.

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());
}
```

The pid of the bash shell would be returned because that is the parent pid of the program's pid.

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?

The termination of the process

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

    (a) `WIFEXITED()`

    The function returns true if the child process terminates normally.

    (b) `WIFEXITSTATUS()`

    Returns the exit status of the child which consists of the 8 least significant digits of the status.

    (c) `WIFSIGNALED()`

    write your answer here

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 */

}
```

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

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

  if( cid == 0)
  {
    execv(ls_args[0], ls_args);
  }
  else if(cid > 0)
  {
    int status, t;
    t = wait(&status);

    if(WEXITSTATUS(status) == 0)
      printf("EXISTS!\n");
    else if(WEXITSTATUS(status) == 2)
      printf("DOES NOT EXIST!\n");
  }
  return 0;
}
```