CSc 332 (L) - Operating Systems

Lab – Spring 2018

Discuss in Lab: February 23 2018

This handout describes the **exec** family of functions, for executing a comand. You can use these functions to make a child process execute a new program after it has been forked. The functions in this family differ in how you specify the arguments, but otherwise they all do the same thing. They are declared in the header file , `unistd.h`, .

execv (char *filename, char *const argv[])

The `execv` function executes the file named by filename as a new process image.

The `argv` argument is an array of null-terminated strings that is used to provide a value for the argv argument to the main function of the program to be executed. The last element of this array *must* be a null pointer.

execvp (char *filename, char *const argv[])

The `execvp` function is similar to `execv`, except that it searches the directories listed in the PATH environment variable to find the full file name of a file from filename if filename does not contain a slash.

This function is useful for executing system utility programs, because it looks for them in the places that the user has chosen. **Shells use** `execvp` **to run the commands** that users type.

execl (char *filename, const char *arg0, ...)

This is similar to `execv`, but the `argv` strings are specified individually instead of as an array. A null pointer must be passed as the last such argument.

execlp (char *filename, const char *arg0, ...)

This function is like `execl`, except that it performs the same file name searching as the `execvp` function.

Example 1: Using execv(...) command

Note: This version will not search the path, so the full name of the executable file must be given. Parameters to main() are passed in a single array of character pointers.

```
#include <stdio.h>
#include <unistd.h>

int main (int argc, char *argv[])
{
   execv ("/bin/echo", &argv[0]);
   printf ("EXECV Failed\n");
   /* The above line will be printed only on error and not otherwise */
}
```

Sample Output

```
$ gcc execv_exl.c -o execv_exl
$ ./execv_exl Hello World!
Hello World!
```

Example 2: Using execvp(...) command

Note: This version searches the path, so the full name of the executable need not be given. Parameters to main() are passed in a single array of character pointers. *This is the form used inside a shell!*

```
#include <stdio.h>
#include <unistd.h>

int main (int argc, char *argv[])
{
   execvp ("echo", &argv[0]);
   printf ("EXECVP Failed\n");
   /* The above line will be printed only on error and not otherwise */
}
```

Sample Output

```
$ gcc execvp_ex2.c -o execvp_ex2
$ ./execvp_ex2 Hello World!
Hello World!
```

Instructions

- Read man page of `exec` system call: `man exec`, to know the syntax of all the four variants in detail
- Compile and execute the examples 1 and 2 to get a feel on how these system call works before you start working on task 3.
- You may also be interested to look at the following link to gain understanding of the process control concepts. `http://www.cs.uregina.ca/Links/class-info/330/Fork/fork.html`

TASK 3

DUE: March 2, 11:59 PM – 25 Points

If late– 15 Points

Part 1 Write a program where a child is created to execute command that tells you the date and time in Unix.

Use `execl(...)`.

Note, you need to specify the full path of the file name that gives you date and time information.

Announce the successful forking of child process by displaying its PID.

Part 2 Write a program where a child is created to execute a command that shows all files (including hidden files) in a directory with information such as permissions, owner, size, and when last modified.

Use `execvp(...)`. I suggest that you list /usr/bin or something in /etc so I can easily run your code without modifying it.

For the command to list directory contents with various options, refer the handout on Unix file system sent to you in the first class. Which version of ls shows hidden files?

Announce the successful forking of child process by displaying its PID.

Part 3

[Step 1] Prcs_P1.c: Create two files namely, `destination1.txt` and `destination2.txt` with read, write, and execute permissions.

[Step 2] Prcs_P2.c: Copy the contents of `source.txt`[1] into `destination1.txt` and `destination2.txt` as per the following procedure.

1. Read the next 100 characters from `source.txt`, and among characters read, replace each character '1' with character 'A' and all characters are then written in `destination1.txt`
2. Then the next 50 characters are read from `source.txt`, and among characters read, replace each character '2' with character 'B' and all characters are then written in `destination2.txt`.
3. The previous steps are repeated until the end of file `source.txt`. The last read may not have 100 or 50 characters.

Once you're done with successful creation of executables for the above two steps do the following.

Write a C program and call it `Parent_Prcs.c`. Execute the files as per the following procedure using `execv` system call. Use `sleep` system calls to introduce delays.

[Step 3] Fork a child process, say `Child 1` and execute `Prcs_P1`. This will create two destination files according to Step 1.

[Step 4] After `Child 1` finishes its execution, fork another child process, say `Child 2` and execute `Prcs_P2` that accomplishes the procedure described in Step 2.

Submission Instructions:

Save your responses to part 1, 2, and 3[2] in a single folder and zip as: task3_*lastname*.zip. Make sure your program compile and run without any errors. Email your zip file with the subject line, "Task 3 - CSc 332G - *lastname*". (I modified this slightly because parens in your filenename cause me a little problem.)

***

[1]Use the same source.txt given for Task 2

[2]Note: Your response to Part 3 should contain 3 C files, namely, Prcs_P1.c, Prcs_P2.c, and Parent_Prcs.c