Advanced Operating Systems CS428

Lecture Two

Semester I, 2009-10

Graham Ellis
NUI Galway, Ireland

Review of fork, wait, exec..., pipe

In Unix, a filter is a program that reads its input from stdin, and writes its output to stdout. A pipeline of these commands can be strung together by a shell to create new commands.

For example, the following will print a count of files ending in ".cpp" found in the current directory and any subdirectories:

```
$ find .  -name "*.cpp" -print | wc -l
```

The shell forks itself, and uses pipes to tie the output of the `find` command to the input of the `wc` command. Two child processes are created, one for each command (`find` and `wc`). These child processes are overlaid with the code associated to the programs they are intended to execute using the exec(3) family of system calls.

In the above example, `find` will overlay the first child process, and `wc` will overlay the second child process, and the shell will use pipes to tie the output of `find` with the input of `wc`.

The shell forks itself, and uses pipes to tie the output of the `find` command to the input of the `wc` command. Two child processes are created, one for each command (`find` and `wc`). These child processes are overlaid with the code associated to the programs they are intended to execute using the exec(3) family of system calls.

In the above example, `find` will overlay the first child process, and `wc` will overlay the second child process, and the shell will use pipes to tie the output of `find` with the input of `wc`.

Forking is performed by the shell each time a user issues a command. A child process is created and overlaid using the exec family of commands.

The shell forks itself, and uses pipes to tie the output of the `find` command to the input of the `wc` command. Two child processes are created, one for each command (`find` and `wc`). These child processes are overlaid with the code associated to the programs they are intended to execute using the exec(3) family of system calls.

In the above example, `find` will overlay the first child process, and `wc` will overlay the second child process, and the shell will use pipes to tie the output of `find` with the input of `wc`.

Forking is performed by the shell each time a user issues a command. A child process is created and overlaid using the exec family of commands.

Let's make sure we understand the brown terms!

`fork()`

The `fork()` system call spawns a new child process which is an identical process to the parent except that it has a new system process ID.

`fork()`

The `fork()` system call spawns a new child process which is an identical process to the parent except that it has a new system process ID.

The process is copied in memory from the parent and a new process structure is assigned by the kernel.

`fork()`

The `fork()` system call spawns a new child process which is an identical process to the parent except that it has a new system process ID.

The process is copied in memory from the parent and a new process structure is assigned by the kernel.

`fork()` returns 0 to the child process, and it returns the child's process ID to the parent.

`fork()`

The `fork()` system call spawns a new child process which is an identical process to the parent except that it has a new system process ID.

The process is copied in memory from the parent and a new process structure is assigned by the kernel.

`fork()` returns 0 to the child process, and it returns the child's process ID to the parent.

The environment, resource limits, umask, controlling terminal, current working directory, root directory, signal masks and other process resources are also duplicated from the parent in the forked child process.

A first example using `fork()`.

A second example using `fork()`.

A third example using `fork()`.

`wait()`

The `wait()` call tells a process to wait for one of its children to end before performing its own task.

`wait()`

The `wait()` call tells a process to wait for one of its children to end before performing its own task.

`wait()` takes the address of an int. It puts the exit status of the child it waited for into this address.

An example using `wait()`.

Overlaying a process: the `exec...()` family

From within a program, the current process, meaning the program itself, can be overlaid with another process. To do this, the programmer calls a member of the `exec...` family.

The following example executes the ls command, specifying the path name of the executable (/bin/ls) and using arguments supplied directly to the command to produce single-column output.

`pipe()`

In UNIX the programmer can use `pipe()` to communicate between concurrent processes. The function prototype is given by

```
int pip(int pd[2])
```

`pipe()`

In UNIX the programmer can use `pipe()` to communicate between concurrent processes. The function prototype is given by

```
int pip(int pd[2])
```

The function call `pipe(pd)` creates an input/output mechanism called a pipe. After a pipe has been created, the system assumes that two or more cooperating processes created by subsequent calls to `fork()` will use `read()` and `write()` to pass data through the pipe. One descriptor, `pd[0]` is read from, and the other, `pd[1]` is written to.

`pipe()`

In UNIX the programmer can use `pipe()` to communicate between concurrent processes. The function prototype is given by

```
int pip(int pd[2])
```

The function call `pipe(pd)` creates an input/output mechanism called a pipe. After a pipe has been created, the system assumes that two or more cooperating processes created by subsequent calls to `fork()` will use `read()` and `write()` to pass data through the pipe. One descriptor, `pd[0]` is read from, and the other, `pd[1]` is written to.

`close()` can be used to explicitly close `pd[0]` and `pd[1]`.

An example using `pipe()`.

What exactly does the example do?

Second Homework

Write your own Linux shell in which

- the command `ls` prints an error message

```
Error:  Please type the Irish version of the command
```

- the command `loistaigh` produces a listing of the contents of the current directory.
- the command `cd` changes your directory.

E-mail me your shell as a file YourNameShell.c . Also, e-mail me a short description, in pdf format, of how you have implemented the shell.

Second Homework

Write your own Linux shell in which

- the command `ls` prints an error message

```
Error:  Please type the Irish version of the command
```

- the command `loistaigh` produces a listing of the contents of the current directory.
- the command `cd` changes your directory.

E-mail me your shell as a file YourNameShell.c . Also, e-mail me a short description, in pdf format, of how you have implemented the shell.

The following article from the Linux Gazette gives a great explanation of how to go about this.