BITS Pilani - Hyderabad Campus
Operating Systems (CS F372)
Tutorial - 3

This tutorial will introduce you to system calls and commands related to process creation and management in the Linux OS.

1. What are system calls? How are they different from normal function calls?

What is *fork* system call used for?
Let’s try some programs to understand these system calls.

```
#include<stdio.h>
#include<unistd.h>
int main()
{
fork();
printf(“hello \n”); /* NOTE: The code following the fork() runs in two
                       processes – the original process(parent), and the
                       child process. The child gets a copy of all the
                       local variables from parent */
return 0;
}
```

Parent normally waits for the child process. Only after the child finishes its execution, the parent resumes. This is done by the **wait**() or **waitpid**() system call.
getpid() returns PID of the current process, **getppid**() returns PID of its parent.

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
int main()
{
        pid_t p;
        int status;
        p = fork();
        if(p < 0) {
                perror("fork");
        } else if (p == 0) { /* child process */
                printf("This is child process and the PID is %d\n", getpid());
                sleep(5);
                printf("child done\n");
        } else {   /* parent process */
                printf("This is parent process and PID of child is %d\n", p);
                wait(&status);
                printf("parent done\n");
        }
        return 0;
}
```

2. How can you trace system calls in Linux?
Use *strace* to trace all the system calls called by a program as follows:
Assuming you have compiled executable file a.out:

```
$ strace ./a.out
```

To follow the child process in the trace, give –f option to strace:

```
$ strace –f ./a.out
```

It will follow the child as well (i.e. it will print system calls executed in the child process also).

3. Write a program to create a zombie/defunct process. (What is a zombie process?)
Zombie/defunct process : it has finished execution but the parent has not yet reaped it.. (what is reaping?)
Use sleep() system call to help create the above processes.

You can see that the child process is defunct by running the following command:

```
$ ps –ef | grep a.out
```

Or, by finding out the pid of the child, and grep-ing the *State* of the process in *status* file corresponding to its entry in */proc* . For example, if the PID is 12273:

```
$ grep State /proc/12273/status
```

shows the status as:

```
State:    Z (zombie)
```

4. How can we run a new program in a process: exec() family of calls.
Fork system call is normally followed by one of the exec family of functions in the child. For example:

```
int s;
if(fork() == 0)
    execlp("ls", "ls", "-la", NULL);
else
    wait(&s);
```

Homework exercise: Create your own shell called myshell; it should be able to run: cat, ls, and pwd commands.
The following are the functionalities your shell should have:
When you run myshell it displays a prompt $ and wait for one of the commands that it can run.
You can type one of the following commands: cat, ls, pwd
The shell forks and uses exec to run the command in child process. After the command output is displayed, the shell prints $ in a new line and waits for your next command.
If you enter a command that is different from the above, it should print error: “command not found”, display prompt $ and waits for your command. The shell terminates only when you type *exit* or press Ctrl-C.