# CSc 332 - Operating Systems

Lab – Spring 2016

Instructor: Arun Adiththan, email: arun.cuny@gmail.com

## Process Management System Calls

February 19, 2016

- A process is basically a single running program
- Each running process has a unique number - a process identifier, `pid` (an integer value)
- Each process has its own address space
- Processes are organized hierarchically. Each process has a **parent** process which explicitly arranged to create it. The processes created by a given parent are called its **child** processes
- The C function `getpid()` will return the `pid` of process
- A child inherits *many* of its attributes from the parent process
- The UNIX command `ps` will list all current processes running on your machine and will list the `pid`
- **Remark:** When UNIX is first started, there is only one visible process in the system. This process is called `init` with `pid 1`. The only way to create a new process in UNIX is to duplicate an existing process, so `init` is the ancestor of all subsequent processes

**Process Creation**

- Each process is named by a process ID number
- A unique process ID is allocated to each process when it is created
- Processes are created with the `fork()` system call (the operation of creating a new process is sometimes called forking a process)
- The `fork()` system call does not take an argument
- If the `fork()` system call fails, it will return a -1
- If the `fork()` system call is successful, the process ID of the child process is returned in the parent process and a 0 is returned in the child process
- When a `fork()` system call is made, the operating system generates a copy of the parent process which becomes the child process
- Some of the specific attributes of the child process that differ from the parent process are:
  - The child process has its own unique process ID
  - The parent process ID of the child process is the process ID of its parent process

  - The child process gets its own copies of the parent process's open file descriptors. Subsequently changing attributes of the file descriptors in the parent process won't affect the file descriptors in the child, and vice versa

- When the lifetime of a process ends, its termination is reported to its parent and all the resources including the PID is freed.

A simple example of `fork()`

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

int main(void)
{
   printf("Hello\n");
   fork();
   printf("Bye\n");
   return 0;
}
```

**Comments**

- *Hello* - printed once by parent process
- *Bye* - printed twice, once by the parent and once by the child
- After the `fork()` system call, both the parent and child processes are running and continue their execution at the next statement in the parent process
- The child process begins execution at the statement immediately after the fork, not at the beginning of the program
- There is no guarantee which process will print *Bye* first
- **Type and run the above code to better understand the above points**

**Process Identification**

- You can get the process ID of a process by calling `getpid()`
- The function `getppid()` returns the process ID of the parent of the current process
- Your program should include the header files `unistd.h` and `sys/types.h` to use these functions

`wait()` **System Call**

- A parent process usually needs to synchronize its actions by waiting until the child process has either stopped or terminated its actions
- If you want your program to wait for a child process to finish executing before continuing, you must do this explicitly after the fork operation, by using `wait()` system call.

- The `wait()` system call accepts a single argument, which is a pointer to an integer
- If the calling process does not have any child associated with it, wait will return immediately with a value of -1

**TASK 2**

**DUE: Feb. 25, 2015, 11:59 PM – 15 Points**

**Part 1** Write a program `children.c`, and let the parent process produce two child processes. One prints out *"I am child one, my pid is: " PID*, and the other prints out *"I am child two, my pid is: " PID*. Guarantee that the parent terminates after the children terminate (Note, you need to wait for two child processes here). Use the `getpid()` function to retrieve the PID. Refer `man wait` for more details on wait system call.

**Part 2** Consider the parent process as *P*. The program consists of *fork()* system call statements placed at different points in the code to create new processes *Q* and *R*. The program also shows three variables: *a*, *b*, and *pid* - with the print out of these variables occurring from various processes. Show the values of *pid*, *a*, and *b* printed by the processes *P*, *Q*, and *R*.

```
//parent P
int a=10, b=25, fq=0, fr=0
fq=fork()    // fork a child - call it Process Q
if(fq==0)    // Child successfully forked
   a=a+b
   print values of a, b, and process_id
   fr=fork() // fork another child - call it Process R
   if(fr!=0)
       b=b+20
       print values of a, b, and process_id
   else
       a=(a*b)+30
       print values of a, b, and process_id
else
   b=a+b-5;
   print values of a, b, and process_id
```

**Submission Instructions: Part 1**: Save your program with given name. **Part 2**: Type and run the code to get the values of a, b, and process id. Briefly write your interpretation (in a text file) on working of the code to explain how your code arrive at the values you printed out. Note, run your program for Part 2 multiple times and see whether there are any changes in the order of execution before writing your report.

Save your responses to part 1 and 2 in a single folder and zip as: task2_*lastname*.zip. Make sure your program compile and run without any errors. Email your code with the subject line, "Task 2 - CSc 332 - *lastname*".

**Next Class**: Having several processes run the same program is only occasionally useful. But the child can execute another program using one of the *exec* functions/system calls. We'll see various `exec` system calls next week. Also, there may be instances where a parent dies before its child. We shall see how such cases are handled.

******