CSC 33200 (L) - Operating Systems – Fall 2022

Lab 3: Process Management System Calls
Date: 09/23/2022

Process

- 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 -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. Subsequentlychanging 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.

Example of fork() **system call structure**

Here a process *P* calls fork(), the operating system takes all the data associated with *P*, makes a brand-new copy of it in memory, and enters a new process *Q* into the list of current processes. Now both *P* and *Q* are on the list of processes, both about to return from the fork() call, and they continue.

The fork() system call returns *Q s* process ID to *P* and 0 to *Q*. This gives the two processes a way of doing different things. Generally, the code for a process looks something like the following.

```
int child = fork();
if(child == 0)
{
   //code specifying how the child process Q is to behave
}
else
{
   //code specifying how the parent process P is to behave
}
```

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

waitpid() **System Call**

- A parent process usually needs to synchronize its actions by waiting until the child process has either stopped or terminated its actions
- The waitpid() system call gives a process a way to wait for a process to stop. It's called as follows.

  pid = waitpid(child, &status, options);

  In this case, the operating system will block the calling process until the process with ID child ends
- The options parameter gives ways of modifying the behavior to, for example, not block the calling process. We can just use 0 for options here.
- When the process child ends, the operating system changes the int variable status to represent how child passed away (incorporating the exit code, should the calling process want that information), and it unblocks the calling process, returning the process ID of the process that just stopped.
- If the calling process does not have any child associated with it, wait will return immediately with avalue of -1

Pipe

pipe() creates a pipe, a unidirectional data channel that can be used for interprocess communication. The array *pipefd* is used to return two file descriptors referring to the ends of the pipe.

pipefd[0] refers to the read end of the pipe. *pipefd[1]* refers to the write end of the pipe. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe.

Lab 3. Marks: 30 **Submission Deadline: 10/06/2022**

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.

Marks: 5

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.**

Marks: 5

```
//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
```

3. Consider a series, S1 = 2 +4 +6+,..+10 and another series S2=1+3+5+..+9. and
Another series S3 = 1+2+3+...+10.

We know that S1+S2 = S3.

Now write a program,where a parent process creates 2 child process and computes S1 and S2. And Parent process computes S3. The input argument for program will be the end of series number for S3.

For example, if the execution file name is series.exe then, the argument will be
./series.exe 10

Child 1 will compute the series from 1 to upto 10 with difference 2. So, it would be 1+3+5+7+9
Child 2 will compute the series from 2 to upto 10 with difference 2. So, it would be 2+4+6+8+10

Parent will compute the series from 1 to 10 with difference 1.So, it would be,
1+2+3+4+5+6+7+8+9+10

Marks: 10

4. Open a file called readme.txt in the child process, read the contents and pass to the parent process. Parent process will write to readme.txt, "Parent is writing:" and write the contents it received from the child to the readme.txt file. (Hint: Use Pipe)

Marks: 10

Submission Instructions

- All the programs MUST be clearly indented and internally documented
- Make sure your programs compile and run without any errors
- Only include c files or txt files for submission. Do not include any executables.
- Save all your programs with meaningful names and zip into a single folder as: Lab3_[your last name here].zip (e.g., Lab3_Xyz.zip)
- Email your code with the subject line, "Lab3-CSC33200–Class#G-*lastname*" (e.g., Lab3 - CSC33200- Class#G-Xyz)
- Email: sdebnath@ccny.cuny.edu