Operating Systems

Lab 01

Goals:

Learn to work with Linux system calls related to process creation and control.

Policies:

It should go without saying that all the work that you will turn in for this lab will be yours. Do not surf the web to get inspiration for this assignment, do not include code that was not written by you. You should try your best to debug your code on your own, but it's fine to get help from a colleague as long as that means getting assistance to identify the problem and doesn't go as far as receiving source code to fix it (in writing or orally).

Set up:

- Creates a text file on Desktop.
- Write your C code to the text file.
- Start the Linux terminal
- Locate your text file in terminal:
  cd /mnt
  ls
  cd hgfs/
  ls
  cd Desktop/
  ls

Contents:

1. fork()

In this lab, we start experimenting with a few Unix system calls. The first one, we will look at is **fork**, which is used by a process to spawn an identical copy of itself. When the fork system call is executed, a new process is created which consists of a copy of the address space of the parent.

The fork call:

- Returns a value of type **pid_t** (essentially, an integer), and
- Does not take any input parameters, what is indicated by the formal parameter void.
- The return code for fork is **0** for the child process and **the process identifier** of child is returned to the parent process.
- On success, both processes continue execution at the instruction after the fork call.
- On failure, **-1** is returned to the parent process.

1) Example 1:

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

int main(){
    pid_t pid;
    pid = fork();

    if(pid==0){
        //Child process
        printf("As a Test in ChildProcess: pid=%d\n", pid);
        printf("Current process is CHILD process with pid=%d, my PARENT process's pid is %d!\n",
        getpid(),getppid());
    }else{
        //Parent process
        printf("As a Test in ParentProcess: pid=%d\n", pid);
        printf("current process is PARENT process with pid=%d, my CHILD process's pid is %d!\n",
        getpid(),pid);
    }
    return 0;
}
```

运行结果:

```
zaobohe@ubuntu:/mnt/hgfs/Desktop$ gcc fork1.c -o fork1
zaobohe@ubuntu:/mnt/hgfs/Desktop$ ./fork1
As a Test in ParentProcess: pid=23894
current process is PARENT process with pid=23893, my CHILD process's pid is 23894!
As a Test in ChildProcess: pid=0
Current process is CHILD process with pid=23894, my PARENT process's pid is 23893!
```

2) Example 2:

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

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

运行结果:

```
zaobohe@ubuntu:/mnt/hgfs/Desktop$ gcc fork2.c -o fork2
zaobohe@ubuntu:/mnt/hgfs/Desktop$ ./fork2
Hello
bye
bye
```

2. wait()

When a process creates a child process, sometimes it becomes necessary that the parent process should execute only after the child has finished. **wait()** system call does exactly this. It makes the parent process wait for child process to finish and then the parent continues its working from the statement after the wait().

To be exact wait makes the parent wait for the child to change state. The state change can be: the child terminated; the child was stopped by a signal; or the child was resumed by a signal.

wait() system call takes only one parameter which stores the status information of the process. Pass **NULL** as the value if you do not want to know the exit status of the child process and are simply concerned with making the parent wait for the child. On success, wait returns the **PID** of the terminated child process while on failure it returns **-1**.

1) Example 1:

```c
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
    pid_t cpid;
    if (fork()== 0)
        exit(0);
    else
        cpid = wait(NULL);

    printf("Parent pid = %d\n", getpid());
    printf("Child pid = %d\n", cpid);
    return 0;
}
```

运行结果:

```
zaobohe@ubuntu:/mnt/hgfs/Desktop$ gcc fork3.c -o fork3
zaobohe@ubuntu:/mnt/hgfs/Desktop$ ./fork3
Parent pid = 24082
Child pid = 24083
zaobohe@ubuntu:/mnt/hgfs/Desktop$
```

2) Example 2:

```c
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
    if (fork()== 0)
        printf("HC: hello from child\n");
    else {
        printf("HP: hello from parent\n");
        wait(NULL);
        printf("CT: child has terminated\n");
    }

    printf("Bye\n");
    return 0;
}
```

运行结果:

```
zaobohe@ubuntu:/mnt/hgfs/Desktop$ gcc fork4.c -o fork4
zaobohe@ubuntu:/mnt/hgfs/Desktop$ ./fork4
HP: hello from parent
HC: hello from child
Bye
CT: child has terminated
Bye
```

3) Example 3:

```c
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<stdlib.h>
int main()
{
    int pid,status;
    pid=fork();
    if(pid==-1)
    {
        printf("fork failed\n");
        exit(1);
    }

    if(pid==0)
    {
        /*child*/
        printf("CHILD HERE\n");
        int cont=0;
        while(cont<10)
        {
            printf("%d\n",cont++);
        }
        printf("child work done !!!!\n");
    }
    else
    {
        /*parent*/
        wait(&status);
        printf("\n PARENT HERE\n");
        printf("well done kid !!!!!\n");
    }
}
```

运行结果:

```
zaobohe@ubuntu:/mnt/hgfs/Desktop$ gcc fork5.c -o fork5
zaobohe@ubuntu:/mnt/hgfs/Desktop$ ./fork5
CHILD HERE
0
1
2
3
4
5
6
7
8
9
child work done !!!!

 PARENT HERE
well done kid !!!!!
```

3. What should you do in this lab?

Task 1:

Write a C program to create **4** processes where first process is the parent of the second one and the second process is the parent of the third one and the third process is the parent of the fourth one. Your program should be capable of:

- checking if the processes is forked with success,
- printing the pid and parent pid of each process,
- printing the parent pid of the first process.

Expected results:

```
zaobohe@ubuntu:/mnt/hgfs/Desktop$ gcc fork6.c -o fork6
zaobohe@ubuntu:/mnt/hgfs/Desktop$ ./fork6
I am a child.  My pid is:24253  my ppid is 24252
I am a child.  My pid is:24254  my ppid is 24253
I am a child.  My pid is:24255  my ppid is 24254
I am a child.  My pid is:24256  my ppid is 24255
I am a child.  My pid is:24257  my ppid is 24256
```

Task 2:

Write a C program to create a fan of processes. That is, process 1 is the parent of processes 2, 3, 4, 5, 6.

Expected results:

```
zaobohe@ubuntu:/mnt/hgfs/Desktop$ gcc fork7.c -o fork7
zaobohe@ubuntu:/mnt/hgfs/Desktop$ ./fork7
[son] pid 24276 from [parent] pid 24275
[son] pid 24278 from [parent] pid 24275
[son] pid 24279 from [parent] pid 24275
[son] pid 24280 from [parent] pid 24275
[son] pid 24277 from [parent] pid 24275
```

4. What should you submit:

1) **Please submit a zip file which includes the C code for these two tasks: with name: task1.c, task2.c**
2) **A short report to show: your idea, your code structure, and execution results.**
3) **Your zip file should be named as: "姓名_lab1"**
4) **Please submit your zip file by sending email to me as attachment: my email address: hzb564@jnu.edu.cn**