Brian Duenas

CSE 460

Lab 3

20 points Total

1. Replacing a Process Image

Q: Modify test_exec so that the function execl is used instead of using execlp.

A:

```
//test_exec.cpp
#include <unistd.h>
#include <iostream>

using namespace std;

int main()
{
	cout << "Running ps with execlp\n";
	execl("/usr/bin/ps", "ps", "-ax", 0);

	cout << "Done!\n";

	return 0;
}
```

2. Duplicating a Process Image

Q: Try the "test_fork.cpp" program and explain what you see on the screen.

Output:

```
1 fork program starting
2 This is the parent
3 This is the child
4 This is the parent
5 This is the child
6 This is the child
7 This is the parent
8 This is the child
9 This is the child
```

A: The first line is the *cout* before the fork command. When the *fork* command is executed it creates a parent and child thread. Next the code run the *switch* statement, the child is told to print “This is the child” five times and the parent is told to print “This is the parent” three times. The print occur to whichever process is finished first.

3. Waiting for a Process

Q: Run the program and explain what you have seen on the screen.

Output:

```
1 fork program starting
2 This is the parent
3 This is the child
4 This is the parent
5 This is the child
6 This is the child
7 This is the parent
8 This is the child
9 This is the child
10Child finished : PID = 9691
11child exited with code 9
```

A: Same output as before now with lines ten and eleven. At line ten the parent process is waiting for the child process to finish then prints line ten, and then prints exit code on line eleven of the child.

Q: Modify the program so that the child process creates another child and wait for it. The grand child prints out the id's of itself, its parent and grandparent.

A:

```
//test_wait.cpp
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

int main()
{
        /*Same example code until code below*/
        for (int i = 0; i < n; ++i) {
                cout << message;
                sleep(1);
        }

        //print all ids
        if (pid != 0) {    //grandparent
                cout << "Parent process id: " << getpid() << endl;

                pid_t grandchild_pid;
                grandchild_pid = fork();

                if (grandchild_pid != 0) {    //parent
                        cout << "Child process id: " << getpid() << endl;
                }
                else    //child
                {
                        cout << "Grandchild process id: " << getpid() << endl;
                }
        }

        exit(exit_code);
}
```

Output:

```
fork program starting
This is the parent
This is the child
This is the parent
This is the child
This is the child
This is the parent
This is the child
This is the child
Parent process id : 9974
Child process id : 9974
Grandchild process id : 9976
```

4. Signals

Q: Run the program and hit ^C for a few times. What do you see? Why?

Output:

```
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
^COops!--I got a signal 2
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
^COops!--I got a signal 2
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
^COops!--I got a signal 2
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
^COops!--I got a signal 2
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
```

A: The program print the results and when *CTRL+C* is pressed it prints the message in the function. *CTRL+*C is an interrupt so every time it happens it runs the function *func* instead of shutting the program down.

Q: Run "test_alarm.cpp". What do you see? Why?

Output:

```
Alarm testing!
Waiting for alarm to go off!
Alarm has gone off
Done!
```

A: Alarm will wait five seconds before being set off by a signal. The signal is the child being killed.

Q: Modify your test_signal.cpp program above by using sigaction() to intercept SIGINT; replace the "for" loop with "while (1); you should be able to quit the program by entering "^\". (Need to intercept SIGQUIT.)

A:

```
//test_signal.cpp
#include <signal.h>
#include <unistd.h>
#include <iostream>

using namespace std;

void func(int sig)
{
	cout << "Oops! -- I got a signal " << sig << endl;
}

int main()
{
	(void)signal(SIGKILL, func); //SIGKILL

	while (1) {
		cout << "CSUSB CS 460 lab on signals" << endl;
		sleep(1);
	}
	return 0;
}
```

Output:

```
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
CSUSB CS 460 lab on signals
^\Quit(core dumped)
```

5. Study of XV6

Q: Compile and run xv6. Also run it in the debugger mode, disassemble the kernel in i386, and examine its code. Copy-and-paste some sample code in your report.

A:

Debugger Mode:

```
*** Now run 'gdb'.
qemu - system - i386 - nographic - drive file = fs.img, index = 1, media = disk, format =
raw - drive file = xv6.img, index = 0, media = disk, format = raw - smp 2 - m 512 - S -
gdb tcp::27050
xv6...
cpu1: starting 1
       cpu0 : starting 0
       sb : size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start
58
       init : starting sh
       $ ls
       .              1 1 512
       ..             1 1 512
       README         2 2 2290
       cat            2 3 13680
       echo           2 4 12688
       forktest       2 5 8124
       grep           2 6 15556
       init           2 7 13276
       kill           2 8 12740
       ln             2 9 12644
       ls             2 10 14828
       mkdir          2 11 12820
       rm             2 12 12804
       sh             2 13 23288
       stressfs       2 14 13468
       usertests      2 15 56404
       wc             2 16 14220
       zombie         2 17 12468
       console        3 18 0
```

GDB:

```
#1  0x801053a0 in sys_exec() at sysfile.c:420
420       return exec(path, argv);
(gdb)list
415           break;
416         }
417         if (fetchstr(uarg, &argv[i]) < 0)
       418           return -1;
419       }
420       return exec(path, argv);
421     }
422
423     int
424     sys_pipe(void)
(gdb) quit
A debugging session is active.

Inferior 1[Remote target] will be detached.

Quit anyway ? (y or n) y
Detaching from program : / home / csusb.edu / 005029683 / cse460 / temp / kernel, Remote target
Ending remote debugging.
```

Broken *ls -l* command:

```
cpu1: starting 1
       cpu0 : starting 0
       sb : size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start
58
       init : starting sh
       $ ls - l
       ls : cannot open - l
       $ ls - l
       ls : cannot open - l
```