George Suarez

CSE 460

Lab 3 – Processes, Signals, and Study of XV6

1. Replacing a Process Image

*test\_exec.cpp:*

//test\_exec.cpp

#include <unistd.h>

#include <iostream>

using namespace std;

int main()

{

cout << "Running ps with execl\n" ;

execl( "ps", "ps", "-ax", 0 );

cout << "Done!\n";

return 0;

}

*Output:*

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 3 on master\***

**$** ./test\_exec

Running ps with execl

Done!

1. Duplicating a Process Image

*test\_fork.cpp:*

// test\_fork.cpp

#include <sys/types.h>

#include <unistd.h>

#include <iostream>

using namespace std;

int main()

{

pid\_t pid; //process id

char \*message;

int n;

cout << "fork program starting\n";

pid = fork();

switch (pid)

{

case -1:

cout << "Fork failure!\n";

return 1;

case 0:

message = "This is the child\n";

n = 5;

break;

default:

message = "This is the parent\n";

n = 3;

break;

}

for (int i = 0; i < n; ++i)

{

cout << message;

sleep(1);

}

return 0;

}

*Output:*

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 3 on master\***

**$** ./test\_fork

fork program starting

This is the parent

This is the child

This is the child

This is the parent

This is the child

This is the parent

This is the child

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 3 on master\***

**$** This is the child

**Explanation**:

The program first outputs the string “fork program starting”. Next, the program then outputs “This is the parent” because the value that is returned after using *fork()* returns a value greater than 1 which means it is a parent process and it assigning the variable *n* to 3, but it gets interesting because right after it outputted “This is the child” and assigns *n* to 5. This makes sense since child processes run concurrently with its’ parent process. The parent process does not wait for the children process to finish. That is why it outputs “This is the child” for a second time as it is trying to output that message five times; meanwhile, the parent process is running normally which is trying to output “This is the parent” three times. The parent process finishes before the child process which is why it outputs the “This is the child” in the next line followed by a newline.

1. Waiting for a Process

*test\_wait.cpp*:

//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()

{

pid\_t pid; //process id

char \*message;

int n;

int exit\_code;

cout << "fork program starting\n";

pid = fork();

switch (pid)

{

case -1:

cout << "Fork failure!\n";

return 1;

case 0:

message = "This is the child\n";

n = 5;

exit\_code = 9;

break;

default:

message = "This is the parent\n";

n = 3;

exit\_code = 0;

break;

}

for (int i = 0; i < n; ++i)

{

cout << message;

sleep(1);

}

//waiting for child to finish

if (pid != 0)

{ //parent

int stat\_val;

pid\_t child\_pid;

child\_pid = wait(&stat\_val); //wait for child

cout << "Child finished: PID = " << child\_pid << endl;

if (WIFEXITED(stat\_val))

cout << "child exited with code " << WEXITSTATUS(stat\_val) << endl;

else

cout << "child terminated abnormally!" << endl;

}

exit(exit\_code);

}

*Output*:

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 3 on master\***

**$** ./test\_wait

fork program starting

This is the parent

This is the child

This is the child

This is the parent

This is the child

This is the parent

This is the child

This is the child

Child finished: PID = 55784

child exited with code 9

**Explanation:**

This program almost does the same thing as *test\_fork.cpp,* except that the parent process waits for the child process to terminate before the program ends.

*test\_wait.cpp (modified)*:

//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()

{

pid\_t pid; //process id

pid\_t grandparent = getpid();

char \*message;

int n;

int exit\_code;

cout << "fork program starting\n";

pid = fork();

switch (pid)

{

case -1:

cout << "Fork failure!\n";

return 1;

case 0:

pid = fork();

switch (pid)

{

case -1:

cout << "Fork failure\n";

return 1;

case 0:

cout << "This is the grandchild PID = " << getpid() << "\nThis is the parent PID = " << getppid() << "\nThis is the grandparent PID = " << grandparent << endl;

break;

default:

break;

}

message = "This is the child\n";

n = 5;

exit\_code = 9;

break;

default:

message = "This is the parent\n";

n = 3;

exit\_code = 0;

break;

}

for (int i = 0; i < n; ++i)

{

cout << message;

sleep(1);

}

//waiting for child to finish

if (pid != 0)

{ //parent

int stat\_val;

pid\_t child\_pid;

child\_pid = wait(&stat\_val); //wait for child

cout << "Child finished: PID = " << child\_pid << endl;

if (WIFEXITED(stat\_val))

cout << "child exited with code " << WEXITSTATUS(stat\_val) << endl;

else

cout << "child terminated abnormally!" << endl;

}

exit(exit\_code);

}

*Output:*

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 3 on master\***

**$** ./test\_wait

fork program starting

This is the parent

This is the child

This is the grandchild PID = 13736

This is the parent PID = 13735

This is the grandparent PID = 13734

This is the child

This is the child

This is the child

This is the parent

This is the child

This is the parent

This is the child

This is the child

This is the child

This is the child

This is the child

Child finished: PID = 13736

child exited with code 9

Child finished: PID = 13735

child exited with code 9

1. Signals

*test\_signal.cpp:*

//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(SIGINT, func); //catch terminal interrupts

for (int i = 0; i < 20; ++i)

{

cout << "CSUSB CS 460 lab on signals" << endl;

sleep(1);

}

return 0;

}

*Output:*

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 3 on master\***

**$** ./test\_signal

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

^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

^COops! -- I got a signal 2

CSUSB CS 460 lab on signals

**Explanation:**

This program outputs “CSUSB CS 460 lab on signals” twenty times for every 1 second. When *Ctrl-C* is pressed a few times, it outputs “Oops! – I got a signal 2 “because it is sending a signal to the process to interrupt it and the *SIGINT* is the second option that you can send in the *signal* function which is why it is outputting 2.

*test\_alarm.cpp:*

//test\_alarm.cpp

#include <signal.h>

#include <unistd.h>

#include <iostream>

using namespace std;

//simulates an alarm clock

void ding(int sig)

{

cout << "Alarm has gone off " << endl;

}

//tell child process to wait for 5 seconds before sending

//a SIGALRM signal to its parent.

int main()

{

int pid;

cout << "Alarm testing!" << endl;

if ((pid = fork()) == 0)

{ //child

sleep(5);

/\*

Get parent process id, send SIGALARM signal to it.

\*/

kill(getppid(), SIGALRM);

return 1;

}

//parent process arranges to catch SIGALRM with a call

//to signal and then waits for the inevitable.

cout << "Waiting for alarm to go off!" << endl;

(void)signal(SIGALRM, ding);

pause(); //process suspended, waiting for signals to wake up

cout << "Done!" << endl;

return 1;

}

*Output:*

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 3 on master\***

**$** ./test\_alarm

Alarm testing!

Waiting for alarm to go off!

Alarm has gone off

Done!

**Explanation:**

This program starts a new process by using *fork()* which then makes the child process sleep for five seconds. Soon after that, a *SIGALARM* is being sent to the parent process and then it exits. The parent process catches *SIGALARM* with a call by *signal()*, and then pauses until a signal has been received. The *pause()* function suspends the execution of the program until a signal occurs. The *kill()* functiondoes not terminate the process right away, but it instead sends a specified signal to the specified process so that it can be terminated. If no signal has been received, then a *TERM* signal is sent instead which will kill processes that do not catch this signal.

*Test\_signal.cpp (modified):*

//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()

{

struct sigaction act;

(void)sigaction(SIGINT, &act, NULL); //catch terminal interrupts

while ( 1 );

return 0;

}

*Output:*

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 3 on master\***

**$** ./test\_signal

^\Quit: 3

1. Study of XV6

*Sample code of XV6:*

Dump of assembler code for function acquire:

0x801042e0 <+0>: push ebp

0x801042e1 <+1>: mov ebp,esp

0x801042e3 <+3>: push esi

0x801042e4 <+4>: push ebx

0x801042e5 <+5>: call 0x801042a0 <pushcli>

0x801042ea <+10>: mov ebx,DWORD PTR [ebp+0x8]

0x801042ed <+13>: mov eax,DWORD PTR [ebx]

0x801042ef <+15>: test eax,eax

0x801042f1 <+17>: jne 0x80104378 <acquire+152>

0x801042f7 <+23>: mov edx,0x1

0x801042fc <+28>: jmp 0x80104303 <acquire+35>

0x801042fe <+30>: xchg ax,ax

=> 0x80104300 <+32>: mov ebx,DWORD PTR [ebp+0x8]

0x80104303 <+35>: mov eax,edx

0x80104305 <+37>: lock xchg DWORD PTR [ebx],eax

0x80104308 <+40>: test eax,eax

0x8010430a <+42>: jne 0x80104300 <acquire+32>

0x8010430c <+44>: lock or DWORD PTR [esp],0x0

0x80104311 <+49>: mov ebx,DWORD PTR [ebp+0x8]

0x80104314 <+52>: call 0x80103750 <mycpu>

0x80104319 <+57>: xor edx,edx

0x8010431b <+59>: lea ecx,[ebx+0xc]

0x8010431e <+62>: mov DWORD PTR [ebx+0x8],eax

0x80104321 <+65>: mov eax,ebp

0x80104323 <+67>: nop

0x80104324 <+68>: lea esi,[esi+eiz\*1+0x0]

0x80104328 <+72>: lea ebx,[eax-0x80000000]

0x8010432e <+78>: cmp ebx,0x7ffffffe

0x80104334 <+84>: ja 0x80104350 <acquire+112>

0x80104336 <+86>: mov ebx,DWORD PTR [eax+0x4]

0x80104339 <+89>: mov DWORD PTR [ecx+edx\*4],ebx

0x8010433c <+92>: add edx,0x1

0x8010433f <+95>: mov eax,DWORD PTR [eax]

0x80104341 <+97>: cmp edx,0xa

0x80104344 <+100>: jne 0x80104328 <acquire+72>

0x80104346 <+102>: lea esp,[ebp-0x8]

0x80104349 <+105>: pop ebx

0x8010434a <+106>: pop esi

0x8010434b <+107>: pop ebp

0x8010434c <+108>: ret

0x8010434d <+109>: lea esi,[esi+0x0]

0x80104350 <+112>: lea eax,[ecx+edx\*4]