Michael Smith

CSE 460

Homework 1

23 January 2018

1. **How many processes does the following piece of code create? Why?**

   This code will create a total of 7 processes for the fork function, and a total of 8 when the main function is included. The main function calls the first fork which creates the first branch. Then main calls the second fork function and that creates a second branch. Lastly the main calls the third function creating a final branch which is finished after the third fork finishes. However; the first branch will call a second fork which then calls a third fork, and the first fork also calls a third fork. Making the first branch have four processes. Then the second branch, the second fork will call a third fork. Making the second branch have 2 processes.

2. **Part a) Chaining processes**

   Chain.cpp

```cpp
/*      This is the chain process program for question number 2 part a
        Build: g++ -o chain chain.cpp
        Run: ./chain
*/

#include <iostream>
#include <unistd.h>
#include <stack>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>

using namespace std;

void chain_fork();
void printparents(stack<int> par);

void chain_fork()
{
        stack<int> parents;
        int pid = 0;

        for(int i = 0; i < 10; ++i)
        {
                pid = fork();
                if(pid == 0) //Case a: child
                        parents.push(getppid());
                else //Case b: parent
                {
                        printparents(parents);
                        wait(NULL); // This is needed to wait for the child to finish
                        exit(0);
                }
        }

}

void printparents(stack<int> par)
{

        cout << "My pid is: " << getpid() << endl;
        if (par.empty()) //Parent case
                cout << "I am the parent process.";
        else //Child case
        {
                cout << "My parents are " << par.top();
                par.pop(); //pop the parents from the stack

                while(!par.empty()) // to list all the parents
                {
                        cout << "," << par.top();
                        par.pop(); //continue to pop from the stack
                }
        }
        cout << endl;

}

int main()
{
        chain_fork();
        return 0;
}
~
~
```

Chain's Output

```
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$ vi chain.cpp
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$ g++ -o chain chain.cpp
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$ ./chain
My pid is: 3995
I am the parent process.
My pid is: 3996
My parents are 3995
My pid is: 3997
My parents are 3996,3995
My pid is: 3998
My parents are 3997,3996,3995
My pid is: 3999
My parents are 3998,3997,3996,3995
My pid is: 4000
My parents are 3999,3998,3997,3996,3995
My pid is: 4001
My parents are 4000,3999,3998,3997,3996,3995
My pid is: 4002
My parents are 4001,4000,3999,3998,3997,3996,3995
My pid is: 4003
My parents are 4002,4001,4000,3999,3998,3997,3996,3995
My pid is: 4004
My parents are 4003,4002,4001,4000,3999,3998,3997,3996,3995
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$
```

Part B

Fanout.cpp

```cpp
/*      This is the fan process program for question number 2 part b
        Build: g++ -o fan fanout.cpp
        Run: ./fan
*/

#include <iostream>
#include <unistd.h>
#include <stack>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>

using namespace std;

void fan_fork();
void printparents(stack<int> par);

void fan_fork()
{
        stack<int> parents;
        int pid = getpid();

        for(int i = 0; i < 10; ++i)
        {
                if(pid == 0) //Case a: child
                {       parents.push(getppid());
                        printparents(parents);
                        exit(0);
                }
                else //Case b: parent
                {
                        pid = fork();
                        wait(NULL);
                }
        }

}

void printparents(stack<int> par)
{

        cout << "My pid is: " << getpid() << endl;
        if (par.empty()) //Parent case
                cout << "I am the parent process.";
        else //Child case
        {
                cout << "My parents are " << par.top();
                par.pop(); //pop the parents from the stack

                while(!par.empty()) // to list all the parents
                {
                        cout << "," << par.top();
                        par.pop(); //continue to pop from the stack
                }
        }
        cout << endl;

}
```

Fanout Output

```
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$ vi fanout.cpp
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$ g++ -o fan fanout.cpp
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$ ./fan
My pid is: 4042
My parents are 4041
My pid is: 4043
My parents are 4041
My pid is: 4044
My parents are 4041
My pid is: 4045
My parents are 4041
My pid is: 4046
My parents are 4041
My pid is: 4047
My parents are 4041
My pid is: 4048
My parents are 4041
My pid is: 4049
My parents are 4041
My pid is: 4050
My parents are 4041
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$
```

3. **Test1.cpp**

```
//Test1.cpp for the infinite loop

int main()
{
        while(1){}
        return 0;
}
~
~
```

KillTest Shell Script

```
#Finds all instances of the test1 app and kills them.
PROC=$(ps auxw | grep "test1" | grep -v grep | awk '{print $2}')

if [ -z $PROC ]; then echo; echo "Process test1 is not found"; echo;
else kill $PROC; echo "Test1 was killed"; echo;
fi
~
```

KillTest Output

```
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$ ./test1 &
[1] 4094
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$ ./test1 &
[2] 4095
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$ ./test1 &
[3] 4096
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$ ./test1 &
[4] 4097
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$ ./KillTest
./KillTest: line 4: [: too many arguments
Test1 was killed

[1]   Terminated              ./test1
[2]   Terminated              ./test1
[3]-  Terminated              ./test1
[4]+  Terminated              ./test1
mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$ ./KillTest

Process test1 is not found

mikesmith@DESKTOP-SOKJJBR:~/cse460/hmwk1$
```

Score: 40/40 I achieved each part of the assignment with no errors. Also had captured perfect outputs for each of the steps of the assignments.