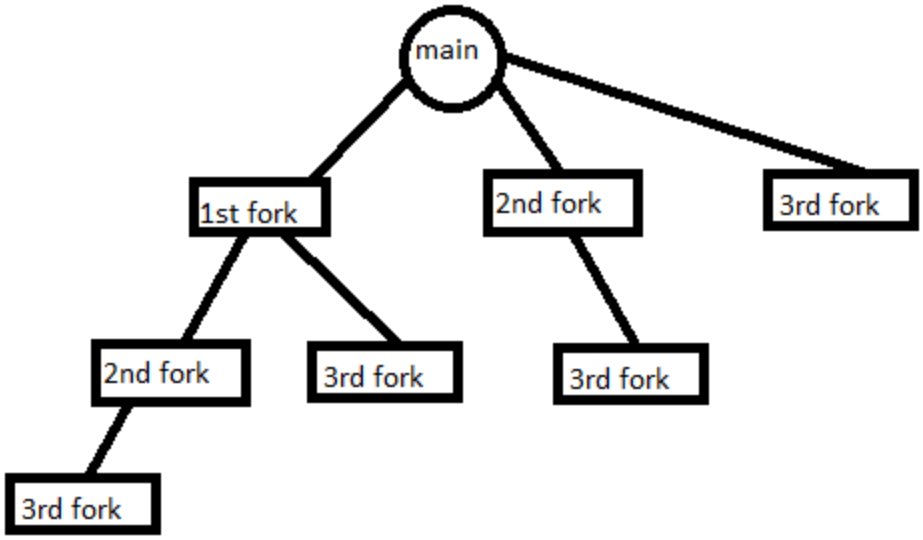
Eric Villanueva
CSE 330

Homework #1

I believed I earned a perfect score 40/40 points. I put 110% effort into this one. All my work is completed, correct, readable.

Question 1)

Answer: In total, the main program creates 7 processes. 8 if you include the main process.



Explanation: After the first fork, the main process and the 1st fork both have 2 forks remaining. Main will fork, and so will the 1st fork; this leaves 1 main, one 1st fork, two 2nd forks. Each of these will fork into a 3rd fork, leaving one main, one 1st fork, two 2nd forks, and four 3rd forks.
This leaves 8 processes, including main.
Side Note: An interesting pattern is that this forking tree follows the same pattern as Pascal's triangle, where the number of processes at a certain level will equal C(depth-1, level); The pattern of the above tree is 1,3,3,1.

Question 2)
Part A)
Chaining Processes
//chain.cpp

```
1.  /*chaining processes*/
2.
3.  #include <iostream>     //output
4.  #include <unistd.h>     //fork
5.  #include <stack>        //stack
6.  #include <stdlib.h>     //wait
7.  #include <sys/types.h>
8.  #include <sys/wait.h>
9.
10.
11. /*
12.         To create a chain of processes, this means that every process
13.         can fork only once, meaning a process will fork, wait, and exit
14. */
15.
16. using namespace std;
17.
18. void chainfork();
19. void printancestors(stack<int> a);
20.
21. void chainfork()
22. {
23.         stack<int> ancestors;
24.         int pid = 0;
25.
26.         for (int level = 0; level < 10; ++level)
27.         {
28.                 pid = fork();
29.                 if (pid == 0)//if child
30.                 {
31.                         ancestors.push(getppid());
32.                 }//endif child
33.
34.                 else //if parent
35.                 {
36.                         printancestors(ancestors);
37.                         wait(NULL);//waits for child to finish
38.                         exit(0);//program goes byebye
39.                 }//endif parent
40.         }
41. }
42.
43. void printancestors(stack<int> a)
44. {
45.         cout << "Hello, my pid is " << getpid() << endl;
46.         if (a.empty())//if parent
47.                 cout << "I'm the Parent Process";
48.         else //if not parent
49.         {
50.                 cout << "My ancestor processes are " << a.top();//lists first ancestor
51.                 a.pop();
52.
53.                 while (!a.empty()) //lists the rest of the ancestors
54.                 {
55.                         cout << ", " << a.top();
```

```
56.                         a.pop();
57.                 }
58.         }//endif not parent
59.                 cout << ".\n-------------------------\n";
60. }
61.
62.
63. int main()
64. {
65.         chainfork();
66.         return 0;
67. }
```

//Chaining Processes Output

```
Captain@Falcon:~/hw1 $ ./a.out
Hello, my pid is 35512
I'm the Parent Process.
-------------------------
Hello, my pid is 35513
My ancestor processes are 35512.
-------------------------
Hello, my pid is 35514
My ancestor processes are 35513, 35512.
-------------------------
Hello, my pid is 35515
My ancestor processes are 35514, 35513, 35512.
-------------------------
Hello, my pid is 35516
My ancestor processes are 35515, 35514, 35513, 35512.
-------------------------
Hello, my pid is 35517
My ancestor processes are 35516, 35515, 35514, 35513, 35512.
-------------------------
Hello, my pid is 35518
My ancestor processes are 35517, 35516, 35515, 35514, 35513, 35512.
-------------------------
Hello, my pid is 35519
My ancestor processes are 35518, 35517, 35516, 35515, 35514, 35513, 35512.
-------------------------
Hello, my pid is 35520
My ancestor processes are 35519, 35518, 35517, 35516, 35515, 35514, 35513, 35512.
-------------------------
Hello, my pid is 35521
My ancestor processes are 35520, 35519, 35518, 35517, 35516, 35515, 35514, 35513,
35512.
-------------------------
Captain@Falcon:~/hw1 $
```

Part B)
//Fanning processes
//fan.cpp

```
1.  /*Fanning processes*/
2.
3.  #include <iostream>     //output
4.  #include <unistd.h>     //fork
5.  #include <stack>        //stack
6.  #include <stdlib.h>     //wait
7.  #include <sys/types.h>
8.  #include <sys/wait.h>
9.
10. /*
11.         To create a fan of processes, this means that only the parent process
12.         may fork, and child processes exit before they can fork
13.
14.         This file repurposes some code from my "Chaining Processes" file
15. */
16.
17. using namespace std;
18.
19. void fanfork();
20. void printancestors(stack<int> a);
21.
22. void fanfork()
23. {
24.         stack<int> ancestors;
25.         int pid = getpid();//CHANGE
26.
27.         for (int level = 0; level < 10; ++level)
28.         {
29.
30.                 if (pid == 0)//if child
31.                 {
32.                         ancestors.push(getppid());
33.                         printancestors(ancestors);
34.                         exit(0);//program goes byebye
35.                 }//endif child
36.
37.                 else //if parent
38.                 {
39.                         pid = fork();
40.                         wait(NULL);//waits for child to finish
41.                 }//endif parent
42.         }
43. }
44.
45. void printancestors(stack<int> a)
46. {
47.         cout << "Hello, my pid is " << getpid() << endl;
48.         if (a.empty())//if parent
49.                 cout << "I'm the Parent Process";
50.         else //if not parent
51.         {
52.                 cout << "My ancestor processes are " << a.top();//lists first ancestor
53.                 a.pop();
54.
55.                 while (!a.empty()) //lists the rest of the ancestors
```

```
56.                 {
57.                         cout << ", " << a.top();
58.                         a.pop();
59.                 }
60.         }//endif not parent
61.                 cout << ".\n-------------------------\n";
62. }
63.
64.
65. int main()
66. {
67.         fanfork();
68.         return 0;
69. }
```

//Fanning Processes Sample output

```
Captain@Falcon:~/hw1 $ ./a.out
Hello, my pid is 36473
My ancestor processes are 36472.
-------------------------
Hello, my pid is 36474
My ancestor processes are 36472.
-------------------------
Hello, my pid is 36475
My ancestor processes are 36472.
-------------------------
Hello, my pid is 36476
My ancestor processes are 36472.
-------------------------
Hello, my pid is 36477
My ancestor processes are 36472.
-------------------------
Hello, my pid is 36478
My ancestor processes are 36472.
-------------------------
Hello, my pid is 36479
My ancestor processes are 36472.
-------------------------
Hello, my pid is 36480
My ancestor processes are 36472.
-------------------------
Hello, my pid is 36481
My ancestor processes are 36472.
-------------------------
Captain@Falcon:~/hw1 $
```

Question 3)

Part A)

```
1.  //test1.cpp
2.  int main()
3.  {
4.          while(1){}
5.
6.          return 0;
7.  }
8.  //that's all
```

Part B)

//The bash script, called destroy

```
Captain@Falcon:~/hw1 $ cat destroy
#Finds test1 instances and kills them all,
#no mercy
#
#PRCS is the string returned from all the pipes, "the killing list"
PRCS=$(ps auxw | grep "test1" | grep -v grep | awk '{print $2}')
#the pipe "grep -v grep" will remove the grep process from the killing list
#
#Will only kill processes if it can
#note that -z is a boolean function. returns if string is empty
if [ -z $PRCS ]; then echo; echo "Process test1 does not exist";echo;
else kill $PRCS; echo "Process test1 killed!"; echo;
fi
Captain@Falcon:~/hw1 $
```

//Testing the bash script

```
Captain@Falcon:~/hw1 $ ./test1 &
[1] 32672
Captain@Falcon:~/hw1 $ ./test1 &
[2] 32673
Captain@Falcon:~/hw1 $ ./test1 &
[3] 32674
Captain@Falcon:~/hw1 $ ./test1 &
[4] 32675
Captain@Falcon:~/hw1 $ ./destroy
./destroy: line 8: [: too many arguments
[1]   Terminated              ./test1
[2]   Terminated              ./test1
[3]-  Terminated              ./test1
[4]+  Terminated              ./test1
Process test1 killed!

Captain@Falcon:~/hw1 $ ./destroy

Process test1 does not exist

Captain@Falcon:~/hw1 $
```

Closing Notes:

- For a C++ syntax highlighter, I used http://www.planetb.ca/projects/syntaxHighlighter/ , it converts ASCII into a copy/paste-able format. Try it yourself!
-