George Suarez

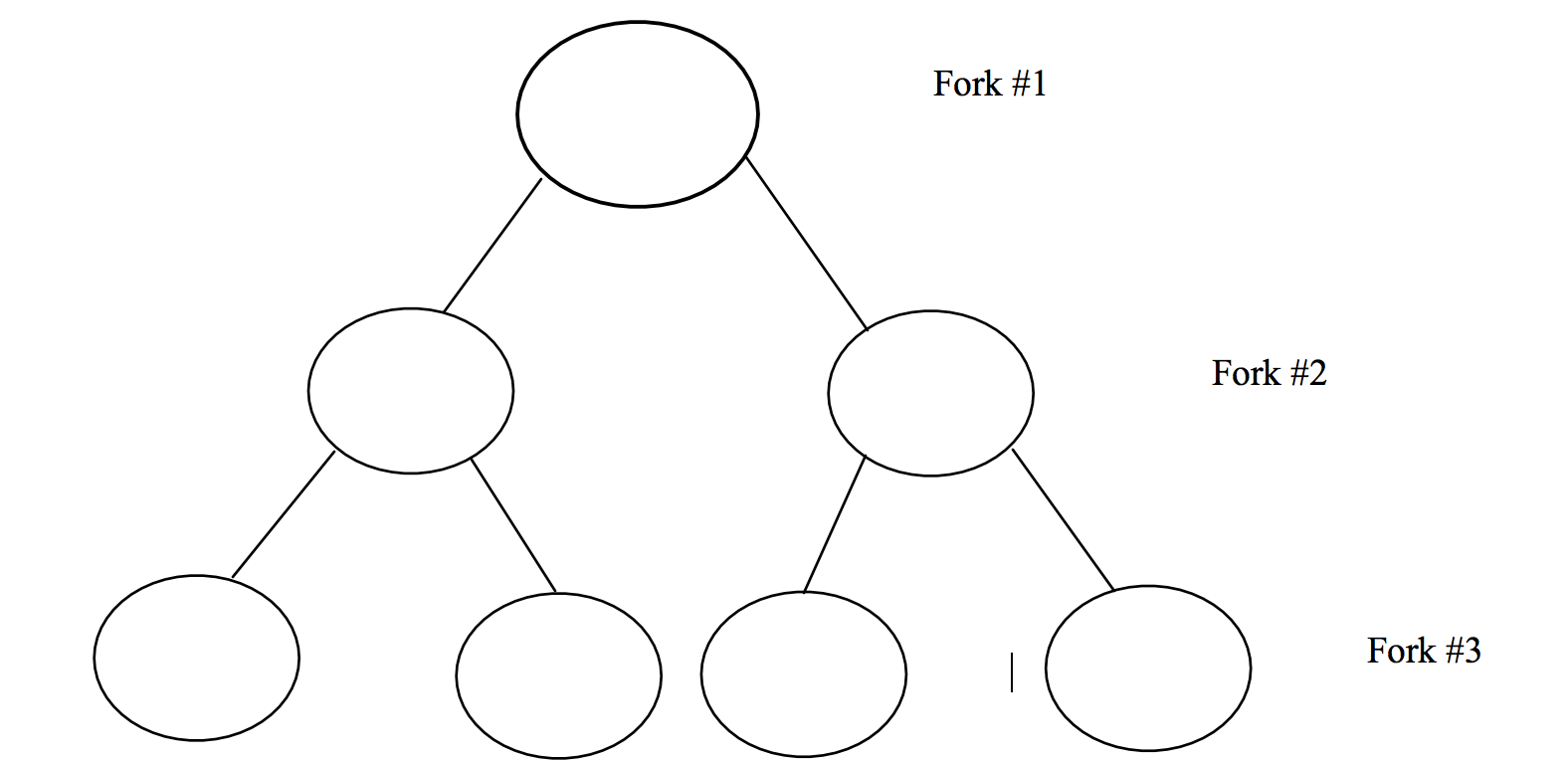
CSE 460

Homework 1

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

int main()  
{   
 fork();  
 fork();  
 fork();  
 return 0;  
}

* The following code creates 8 processes because each fork that is being called doubles the amount of the current process including the original process. So, we can say that each fork call creates processes where equals to the number of times fork is being called. The following tree represents this behavior.



1. 1. Write a C/C++-program that creates a chain of 10 processes and prints out their process ids and relationships. For example, process 1 is the parent of process 2, process 2 is the parent of process 3, process 3 is the parent of 4 and so on. Each child has to print out all her ancestors identified by the process ids.

*chain\_processes.cpp*:

#include <iostream>

#include <unistd.h>

using namespace std;

int main()

{

pid\_t pid = fork();

for (int i = 0; i < 10; ++i)

{

if (pid == 0)

{

cout << "This is a child with PID: " << getpid() << ". My Parent PID is: " << getppid() << endl;

pid = fork();

}

else

{

wait(0);

}

}

return 0;

}

Output:

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Homework/Homework 1 on master\***

**$** ./chain\_processes

This is a child with PID: 45122. My Parent PID is: 45122

This is a child with PID: 45123. My Parent PID is: 45123

This is a child with PID: 45124. My Parent PID is: 45124

This is a child with PID: 45125. My Parent PID is: 45125

This is a child with PID: 45126. My Parent PID is: 45126

This is a child with PID: 45127. My Parent PID is: 45127

This is a child with PID: 45128. My Parent PID is: 45128

This is a child with PID: 45129. My Parent PID is: 45129

This is a child with PID: 45130. My Parent PID is: 45130

This is a child with PID: 45131. My Parent PID is: 45131

* 1. Write a C/C++-program that creates a fan of 10 processes. That is, process 1 is the parent of processes 2, 3, 4, 5, 6 and so on.

*fan\_processes.cpp:*

#include <iostream>

#include <unistd.h>

using namespace std;

int main()

{

pid\_t parent\_pid = getpid();

cout << "Parent PID: " << parent\_pid << endl;

pid\_t pid = fork();

for (int i = 0; i < 10; i++)

{

if (pid > 0) // Parent process

{

pid = fork();

if (pid == 0)

{

cout << "I am child process and my PID is: " << getpid() << " and my parent PID is: " << getppid() << endl;

}

else

{

wait(0);

}

}

}

return 0;

}

~

Output:

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Homework/Homework 1 on master\***

**$** ./fan\_processes

Parent PID: 45187

I am child process and my PID is: 45189 and my parent PID is: 45187

I am child process and my PID is: 45190 and my parent PID is: 45187

I am child process and my PID is: 45191 and my parent PID is: 45187

I am child process and my PID is: 45192 and my parent PID is: 45187

I am child process and my PID is: 45193 and my parent PID is: 45187

I am child process and my PID is: 45194 and my parent PID is: 45187

I am child process and my PID is: 45195 and my parent PID is: 45187

I am child process and my PID is: 45196 and my parent PID is: 45187

I am child process and my PID is: 45197 and my parent PID is: 45187

I am child process and my PID is: 45198 and my parent PID is: 45187

3.

a. Write a simple program named **test1.cpp**, which contains an infinite **while** loop. Compile the program to an executable named **test1** and run it in the background.

*test1.cpp:*

#include <iostream>

using namespace std;

int main()

{

int x = 0;

while (1)

{

++x;

}

return 0;

}

Output:

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Homework/Homework 1 on master\***

**$** ./test1 &

[1] 45266

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Homework/Homework 1 on master\***

**$** ./test1 &

[2] 45277

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Homework/Homework 1 on master\***

**$** ./test1 &

[3] 45288

b. Write a shell script that searches for whether the process **test1** is in the system. If it is not, your script displays the message 'Process test1 not running!'. If it is running, your script kills the process, and displays the message 'Process test1 killed!'.

*terminateProcess:*

for pid in $(ps -e -f | grep $1 | grep -v grep | grep -v $0 | awk '{print $2}')

do

kill $pid

done

if [ "$pid" == "" ];

then

echo "Process $1 is not running!"

else

echo "Process $1 killed!"

fi

Output:

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Homework/Homework 1 on master\***

**$** ./terminateProcess test1

Process test1 killed!

[1] Terminated: 15 ./test1

[2]- Terminated: 15 ./test1

[3]+ Terminated: 15 ./test1

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Homework/Homework 1 on master\***

**$** ./terminateProcess test1

Process test1 is not running!