Project 1: Inter Process Communication

ECE 434: Intro to Computer Systems -- Spring 2018

	Name (First Last)	NetID
Members:	Mahmoud Bachir	mb1516
	Rahul Gupta	rg720
	Gunbir Singh	gs664
	Jeremy Kritz	jsk208

Problem 0 [10]

File Size	Part A (seconds)	Part B (seconds)	Part C (seconds)
10	.326	.82	.002
100	.437	1.184	.007
1k	1.274	1.572	.022
10k	1.677	1.793	-------------------------
100k	2.371	3.041	-------------------------

Parts a and b both ran in similar times.
Part c went faster because of the divide and conquer approach.
Design choices.
0a was simple, it was a single process (which doesn't require a fork) that sums all the numbers and finds the max and min.
Part b was a bit harder, where each process spawned a process, but for this one I had a for loop for the total number of items in the list, then for each I spawned a process to deal add to the sum, and check whether the new number was the new max or min.
Part b was the most complex, and I went about it like a binary search, or a merge sort. I took the whole list and spawned two processes, one for the first half and one for the second. Each of those halved until there was a single number, which was added to the sum and checked against the max and min like in part b.
I learned about pipes, and how a divide and conquer approach with multiple processes can speed up tasks when compared to a single process.

Problem 1 [10]

1. If root process A is terminated prematurely, then all of its children processes become zombies, the task at hand is not completed, and resources are hogged from the system, without providing any benefit whatsoever.

2. I was very confused with this question as to what pid() is supposed to be used for. If you meant getppid() in this case instead. Displaying the tree with root getpid() is difference than getppid() bc issuing command getpid() gives you the process id of the current process, doing this at the root will give you its process. However, issuing command getppid() gives you the process id of the parent process. Issuing this at the root process would create an issue because the root process has no parent process.

3. The maximum number of trees that can be generated depends on the system that it is running on, mainly it depends on the amount of memory/resources that is available. More memory means there is more room for multiple processes to be running, and vice versa.

Problem 2 [20]

1. The order of start and termination messages is random, because the underlying processes print the message as per cpu time slot given by the cpu scheduler.

Problem 3 [20]

The first thing that needs to be stated is that this problem was implemented in C++. I was told by other students that you gave the go ahead to implement the solution in C++ for the more complex problems. The only reason we decided to do this is because the <vector> data structure is easier to manage than pointers, and since this project is focusing on teaching us interprocess communication, I believed it would be okay if we implemented the solution in C++.

1. When using the sleep() function, the programmer has absolutely no idea how long the process may take. By implementing the use of signals, the program is only executed when a signal is received/sent. Thereby, using signals instead of sleep() gives the programmer more control.

2. This function is tasked to wait for all the children processes of a parent to complete running before any parent process is killed. This ensure that no zombie children are created and system resources are not wasted.

Problem 4 [20]

1. The number of pipes necessary in this problem PER process varies depending on format style, givens, and restrictions on code. It would be possible to use only one pipe for all of the children processes granted an expression is broken into two pieces, with the second one being broken down as a child process for another numerical calculation. However, if we are writing a program that gives the user freedom to enter any numerical expression to be computed, one pipe must be used for every numerical operand.