CS69001 Computing Laboratory – I

Practice Exercises: Set 4

1. Write a program to do the following. The user specifies a set of URLs in the command line. Your program opens a tab (or window) of the web-browser to display each URL supplied. The mozilla firefox web-browser can be run by executing **`firefox URL`**. Here is a sample run of your program.

```
$ ./a.out http://cse.iitkgp.ac.in/ https://www.facebook.com/ file:/// about:license
```

2. Deduce how many lines (as a function of n) are printed by the call $f(n)$, where n is a positive integer. Prove your formula mathematically.

```
void f ( unsigned int n )
{
   unsigned int i;

   for (i=0; i<n; ++i) {
      fork();
      printf("i = %d\n", i);
   }
}
```

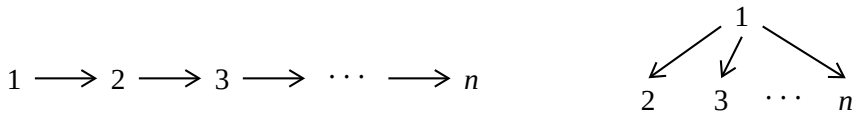
3. **(a)** Embed the above function in a program that reads n from the command line. Run the code for small values of n (like $n \leqslant 6$) to verify your formula. Your program should not print anything other than the lines printed by **`f(n)`**.

(b) Run the executable and redirect the output to a pipe or a file. Some samples runs are given below.

```
$ ./a.out | wc
$ ./a.out | wc -l
$ ./a.out | less
$ ./a.out > foutput.txt
```

How many lines are printed by your code in these runs? Explain the anomaly.

4. The user supplies a small positive integer n. A total of n processes are to be created. Each process is given an integer tag in the range $[1,n]$. This tag is a user-specified process ID—not the system-given PID. If a process with tag T creates a process with tag T', we denote this by $T \longrightarrow T'$. The following figure specifies two ways of creating the n processes. The process you launch by running the executable of your program is called the *initial process*. This is at the root of the process tree that your program creates. The initial process gets the first tag 1. Other processes get their tags from their respective parent processes.



(a) (b)

(a) In this part, the process with tag T creates the process with tag $T+1$. Your run of the program launches process 1 (the initial process), which forks process 2, which in turn forks process 3, and so on.

(b) In this part, your initial process with tag 1 creates the $n-1$ other processes with tags $2,3,4,\ldots,n$.

For each of these two process trees, write a program to do the following. The same program is run by all the n processes. Write a function to create the processes as specified. After the function returns, each process simulates work by sleeping for a random duration in the range 1–5 seconds (use **`sleep()`** or **`usleep()`**). After this *work*, each process waits for the child process/processes (if any) created by it to terminate. After all child processes terminate, the process itself terminates.

Each process must know its tag, and should print the tags and the PIDs of the child processes created by it. Finally, each process must print each child tag after the child terminates.

5. Write a program **`avg.c`** that reads, from the user, a positive integer n followed by n integers, and computes and prints the (floating-point) average of the input integers. Now, suppose that n and the n integers are stored in a data file. Write a second program that reads the name of the data file from the user. It then runs the program for computing the average, after redirecting its **`stdin`** to read from the data file. Use **`dup()`**.

6. Write a program to do the following. The user specifies a command (the name of an executable) and a set of arguments for the command. Your program runs the command with the specified inputs, and collects the output (**`stdout`** only) produced by the run in a string. Use **`freopen()`** and **`setbuf()`**.

7. Augment the programs of Exercise 4 as follows. The initial process (the process with tag 1) reads a string from the user. It then broadcasts the string to all other processes. Each process, after receiving the string, prints it along with its own tag. In Part (a), process i sends the string to process $i+1$ for all $i = 1,2,3,\ldots,n-1$, whereas in Part (b), process 1 sends the string to all other processes. Use pipes to perform all parent-to-child communications.

8. Repeat Exercise 6 with the difference that here process n reads the string from the user. The broadcast process now involves some child-to-parent communications too.