OS Lab Report – Week 3

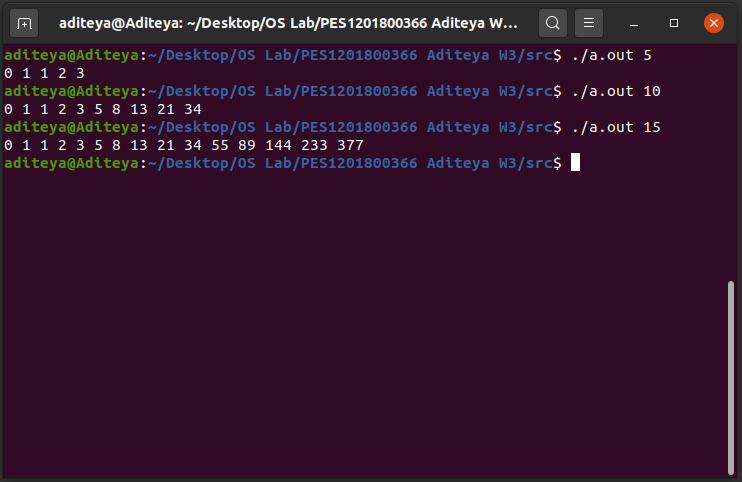
PES1201800366

Aditeya Baral

1. Question 1

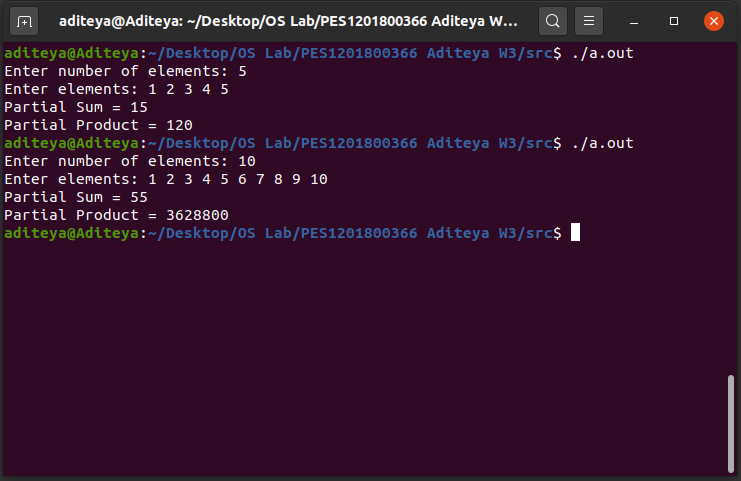
The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, ....

Write a C program using the fork() system call that generates the Fibonacci sequence in the child process. The number of the sequence should be provided in the command line. For example, if 5 is provided as an input to the program, the first five numbers in the Fibonacci sequence should be output by the child process.



2. Question 2

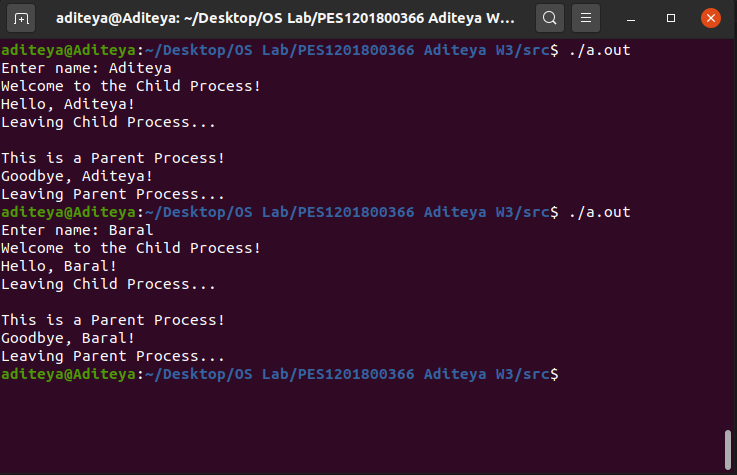
Write a C program that uses the child to compute partial sums and parent to compute the partial products of an array of integers. Both child and parent should print the respective total sum and product values. Use an array with a minimum of 5 elements.



3. Question 3

Write a C program to demonstrate the use of fork(), exec() and wait() all in one program. Use any one of the family of exec system calls such as execl() or execvp().

Example: If the input/argument to the program is one of these: ls, ls –l, find, <executable\_program> - then your program should display the output of the same command (like the output of “ls –l” command) or the executable\_program that was passed as an argument.



4. Questions

1. ***What is the role of the init process on UNIX and Linux systems regarding process termination?***

*Answer* – When a child process terminates, its process still exists in the PID table and hence continues to take up space. This space is released when the parent process calls wait() and releases the child process’ entry. If the parent process however does not call wait(), this leaves the child process in the orphan state.

On UNIX and Linux systems, the init process is assigned as the new parent to orphan processes. The init process periodically calls wait() to obtain the exit status of all orphans and releases their process table entries.

2. ***What is a subreaper process?***

*Answer* – A subreaper process is the closest living ancestor of a child process that has become orphaned. Instead of assigning the init process to be the new parent, the subreaper process becomes the parent. It can now use wait() to reap the child process’ exit status and free its process table entry.

3. ***What causes a defunct process on the Linux system and how can you avoid it?***

*Answer* – Defunct or Zombie Processes, are processes that have not terminated properly, without their parent process having called wait() and hence their process continues to exist in the process table. We can avoid Defunct Processes by

1. Using wait() in the parent process to reap the child process’ exit status and free its process table entry.
2. Processing the signal received from SIGCHILD using the signal (SIGCHILD, func) call. The function func includes a wait() call that reaps the child’s exit status and frees the child’s process table entry.

4. ***How can you identify zombie processes on the Linux system?***

*Answer* – The Process Status command line utility (accessed using the ps command) can be used to view the status and information related to processes running in a Linux system. The output contains a STAT column, which holds the status, and an entry of Z under this column indicates a zombie process.

5. ***What does child process inherit from its parent?***

*Answer* – The child process will inherit most of its parent’s attributes such as file descriptors, privileges scheduling attributes, open message queue descriptors and resources such as files opened. However these are inherited as a copy so modifying them does not modify the parent’s copies.