Problem 2 (Process Monitoring) Under the Linux OS, `ps` is a command that displays information about all processes currently running in your system. Read the man page of `ps` command. Enter the following commands: (1) `ps –ef | more` and (2) `ps –aux | more`. Both of these will result in displaying a long list of processes. Identify what processes are started when the system is booted, and what processes are started later on. For each process, find out who owns it, what code it is running, and how much CPU/memory it has used.

Problem 1 Process Creation:

The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, …., which can formally be expressed as:

\[
\begin{align*}
\text{fib}_0 &= 0 \\
\text{fib}_1 &= 1 \\
\text{fib}_n &= \text{fib}_{n-1} + \text{fib}_{n-2}
\end{align*}
\]

Write a C-Program using the `fork()` system call that generates the Fibonacci sequence in the child process. The number of sequence will be provided in command line. For example if 5 is provided, the first five numbers in the Fibonacci sequence will be output by the child process. Because of the parent and the child have their own copies of the data, it will be necessary for the child to output the sequence. Have the parent invoke a `wait()` call to wait for the child process to complete before exiting the program. Perform necessary error checking to ensure that a non-negative number is passed on the command line.

Problem 3 (IPC using shared memory):

The file `fibonacci_shmt.c` (on the homework page) implements the Fibonacci series using message passing. Study the code and explain the communication process. Fill the comments where they were omitted in the code.

Problem 4 (IPC using Message Passing):

The files `message_send.c` and `message_rec.c` on the homework webpage implement a sender a receiver process after the message passing paradigm. Study and understand the code. Compile the two programs separately. Execute them in different sequences and in different shells.

1. **Document your observations.**
2. **Modify the code such that the sender and the receiver provide the following output:**
   1. (Process 1) Sends the message "Are you hearing me?"
   2. (Process 2) Receives the message and replies "Loud and Clear".
   3. (Process 1) Receives the reply and then says "I can hear you too".