* 1. In theory, the program will print “hello” 2 times in each iteration. So, “hello” will be printed **2^60 = 1,152,921,504,606,846,976 times**.
  2. - If the processes run concurrently, it may lead to a higher rate of execution.

- Synchronization issues.

* 1. While the C program is executed, it takes control of the terminal and concurrently prints “hello” message 2^4=16 times. On the other hand, the shell is paused until the C program is completed, then the shell takes back control of the terminal and prints “prompt”. So, the timing of printing “prompt” is determined by the timing of finishing the C program execution not the timing of printing “hello” messages which are produced by the child processes of the C program.

1. 1. In the parent: a=1, c=2

In the child: a =40, c=20

In the grandchild: a=10, c=20

Since the child waits for the grandchild, there are 3 possible outputs:

* a=1, c=2

a=10, c=20

a =40, c=20

* a=10, c=20

a=1, c=2

a =40, c=20

* a=10, c=20

a =40, c=20

a=1, c=2

* 1. There are 3 unique processes.

The parent (P) forked one child (C1) then this child (C1) forked grandchild (C2) while the parent could not fork because the first condition (fork == 0) would be only true for C1.



1. - From Ready To Run: The scheduler selects a process from the ready queue to be executed.

- From Ready To Suspended: The OS moves a process temporarily because another high-priority process is ready.

- From Run To Ready: The time slice of the process is expired.

- From Run To Waiting: The process initiates an I/O operation like reading from a file.

- From Run To Suspended: The process is suspended by the user.

- From Waiting To Ready: The I/O operation that the process was waiting for finished.

- From Waiting To Suspended: The process is suspended by the OS to conserve resources.

- From Suspended To Ready: The process is resumed by the OS.



1. 1. - They both enable the processes to share resources, like CPU and memory, which improves the utilization of system resources.

- They both enable multiple programs to execute concurrently or in parallel on a single computer system.

- They both improve the responsiveness of a computer system. Multiprogramming allows the system to switch between programs when one is waiting for I/O, while time-sharing provides rapid user feedback by giving each user a small time slice for execution.

- They both include context switching, which is the process of saving and restoring the state of a process to achieve concurrency.

|  | **Multiprogramming** | **Time-Sharing** |
| --- | --- | --- |
| **Objective** | maximizing CPU utilization by keeping the CPU busy with some task at all times. | providing interactive and fair access to system resources among multiple users. |
| **User Interaction** | not focused on user interaction. | designed for interactive use. |
| **Fairness** | It may not provide fair access to resources. | It ensures that each user or task gets an equal share of CPU time |
| **Resource Allocation** | The CPU is shared among multiple programs simultaneously. | It allocates CPU time in small time slices for each task. |
| **Scheduling** | based on priority and available resources | It ensures that each task gets a fixed time slice before switching to the next. |

| **Scenario** | **Approach** | **Justification** |
| --- | --- | --- |
| Multi-user system like the CSE server | time-sharing | * It ensures that each user gets a fair share of the system resources. * It provides rapid response times to user inputs which makes it ideal for interactive use cases. |
| Batch Processing System (does not interface with users) | multiprogramming | * to maximize CPU utilization by keeping the CPU busy with tasks and execute them concurrently. * No need for rapid user interaction. |