CS & IT ENGINEERING

Operating System

Process Synchronization

Lecture -1

By- Vishvadeep Gothi sir

Recap of Previous Lecture

Topic Questions on Scheduling

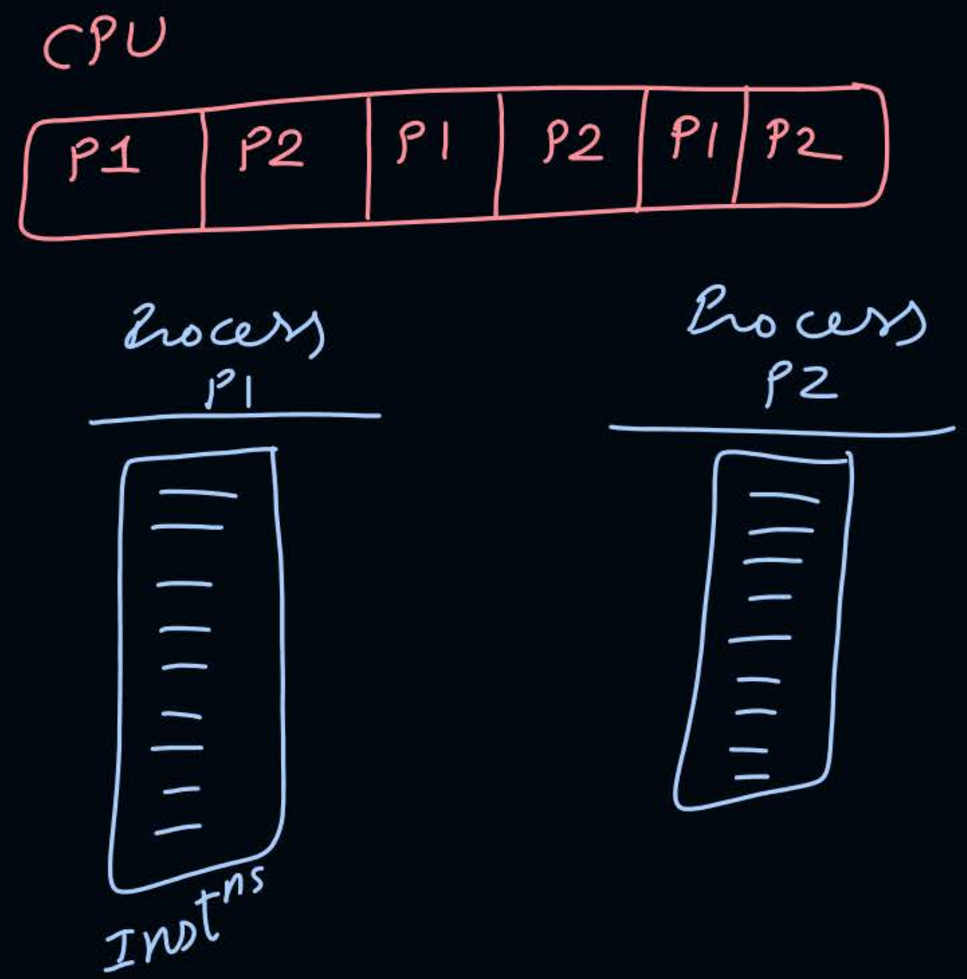
Topic Multithreading

Topics to be Covered

- **Topic** System Call: Fork()
- **Topic** Synchronization
- **Topic** Race Condition
- **Topic** Critical Section

Parallel execution (simultaneous)

Concurrent execution



Topic : Multithreading

Component of process

or

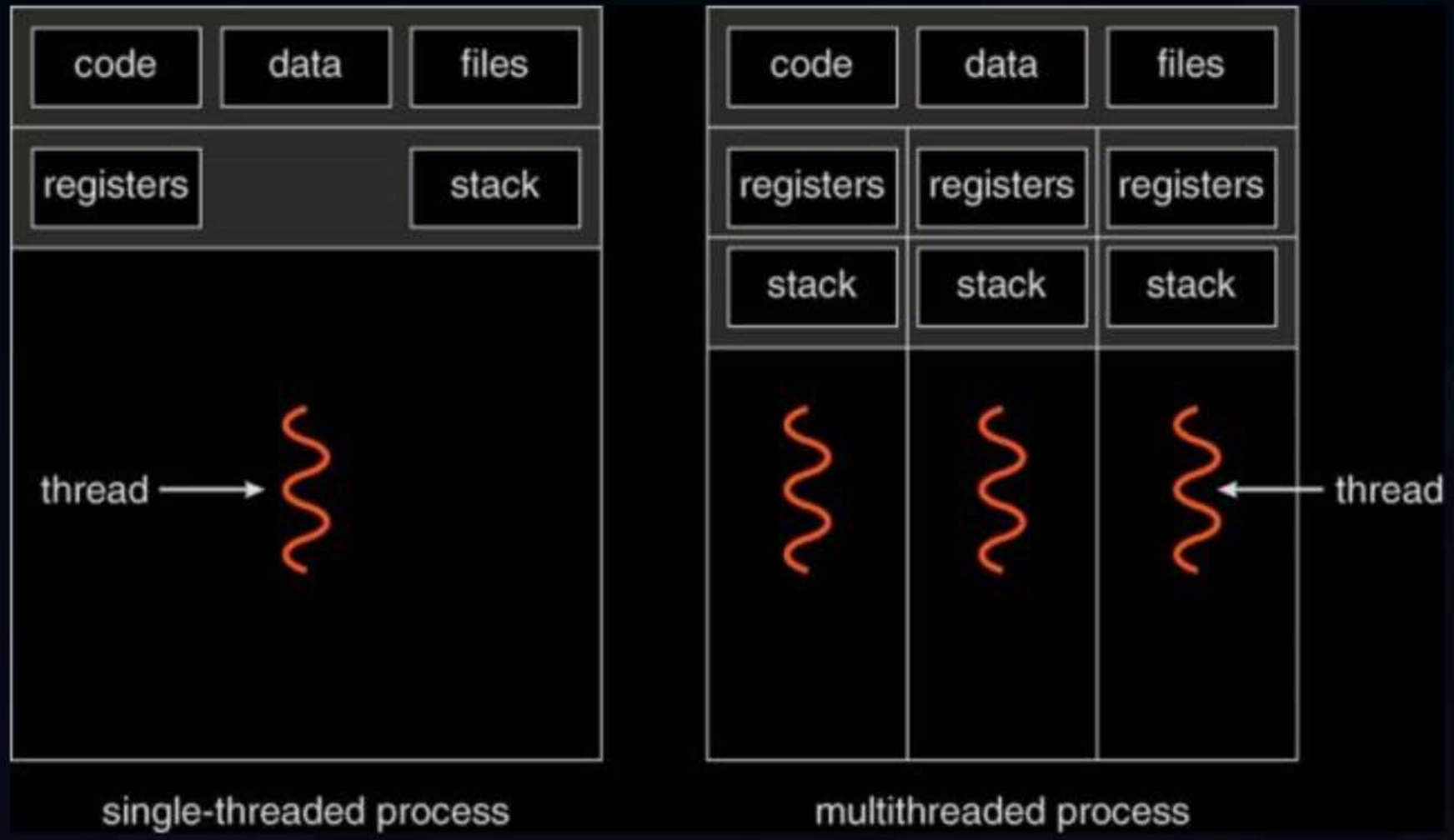
Lightweight Process

Provide a way to improve application performance through parallelism

Topic : Threads

Shared Among Threads	Unique For Each Thread
Code Section	Thread Id
Data Section	Register Set
OS Resources	Stack
Open Files & Signals	Program Counter
Heap	

Topic : Threads



Topic : Types of Threads

1. User Level Thread
2. Kernel Level Thread

Topic : Types of Threads

→ kernel (OS) does not have info^n about user-level threads.

User Threads	Kernel Thread
Multithreading in user process	Multithreading in kernel process
Created without kernel intervention	Kernel itself is multithreaded
Context switch is very fast	Context switch is slow
If one thread is blocked, OS blocks entire process	Individual thread can be blocked
Generic and can run on any OS	Specific to OS
Faster to create and manage	Slower to create and manage

Topic : System Call

Programmatic way in which a computer program requests a service from kernel

Topic : How System Call Works

Trap to OS

Request of process from User space → Interrupt request for kernel → Control Transfer to kernel → Kernel checks if operation can be done → Kernel performs operation → Kernel return the output to user process

kernel mode

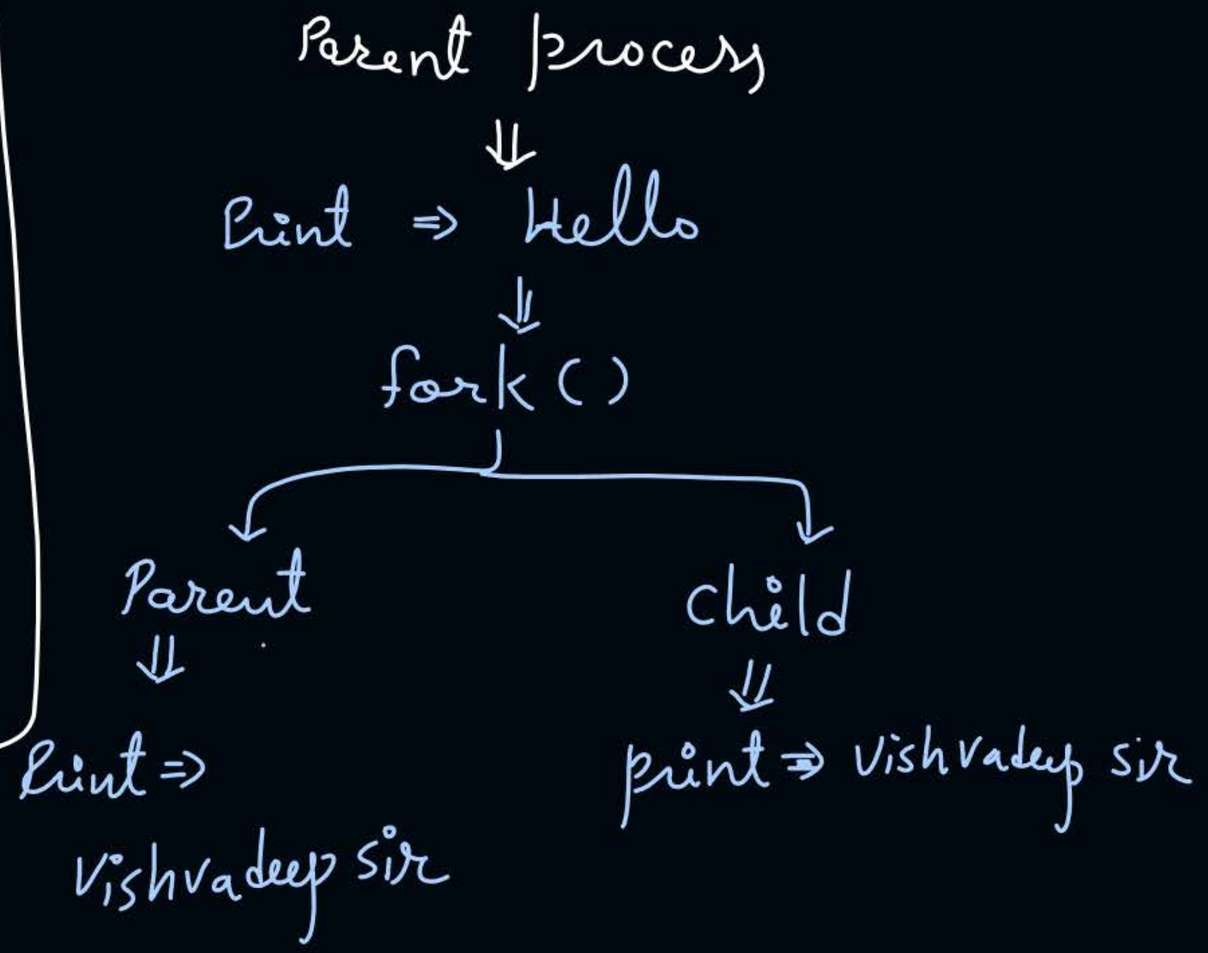
Topic : Fork() System Call

Fork system call is used for creating a new process, which is called child process.

Child process runs concurrently with the process that makes the fork() call (parent process).

→ child process starts executing after the fork() call which created it.

```c
#include<stdio.h>
void main(){
 printf("Hello ");
 fork();
 printf("Vishvadeep sir ");
}
```



output:-

Hello Vishvadeep sir Vishvadeep sir

Topic : Fork() System Call

It takes no parameters and returns an integer value

- Negative Value:
  Creation of a child process was unsuccessful

- Zero:
  Returned to the newly created child process

- Positive value:
  Returned to parent or caller. The value contains process ID of newly created child process

```
void main()
{
  if(fork())
  {
    printf("Hello");
  }
}
```

output:-

Hello

fork()

Parent

if(non-zero int which id pid of child)

⇓

Print ⇒ Hello

child

if (zero)

⇓

nothing printed

[MCQ]

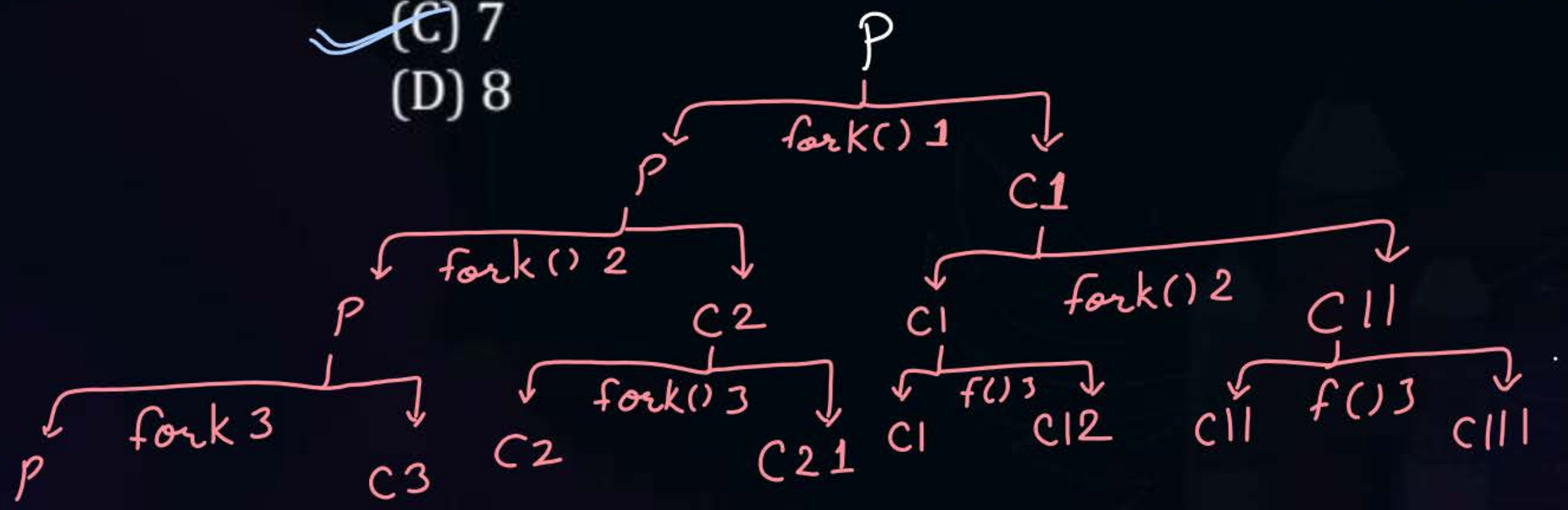
#Q. A process executes the code

```
fork();
fork();
fork();
```

The total number of child processes created is?

GATE-2008

(A) 3
(B) 4
(C) 7
(D) 8



[NAT]

#Q. A process executes the code

```
fork ();
:
fork ();
```

There are n such statements. The total number of child processes created is?

Total no. of processes $= 2^n$

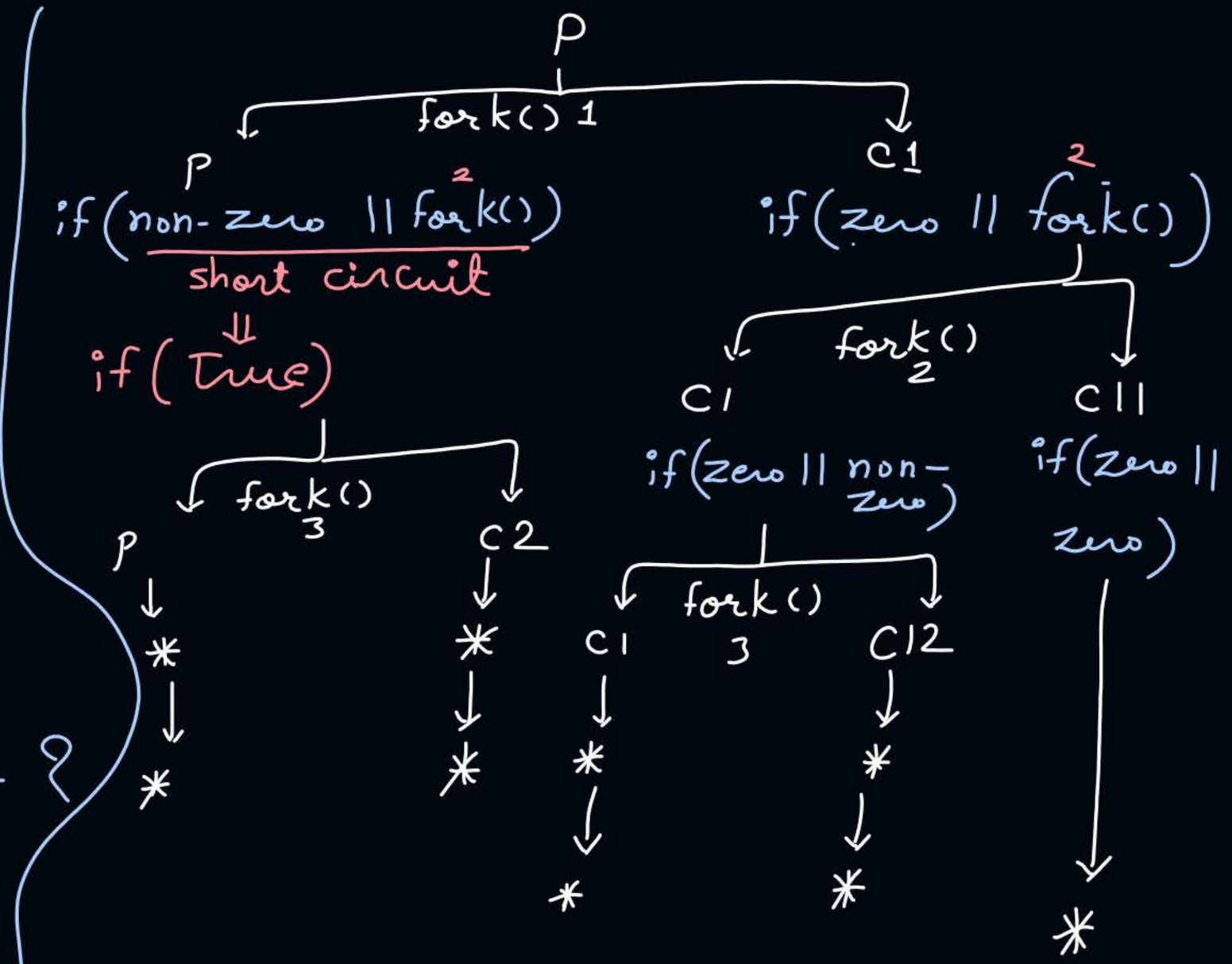
1 Parent

$2^n - 1$ Child

Ans $= 2^n - 1$

```
void main()
{
  if(fork() ||  fork())
  {
    fork();
    printf("*");
  }
printf("*");
}
```

no. of times '*' printed = 9 ?



Homework

```
void main()
{
  if (fork() && fork())
  {
    fork();
    printf("*");
  }
fork();
printf("*");
}
```

no. of times * printed —— ?

Topic : Wait() System Call

A call to wait() blocks the calling process until one of its child processes completes

After child process terminates, parent continues its execution after wait system call instruction

[NAT] Homework

#Q. Consider the following code snippet using the fork() and wait() system calls. Assume that the code compiles and runs correctly, and that the system calls run successfully without any errors.

```
int x = 3;
while(x > 0) {
fork();
printf("hello");
wait(NULL);
x--;
}
```

The total number of times the printf statement is executed is _____?

GATE-2024

Topic : Process Types

1. Independent → The processes which do not communicate with other processes

2. Cooperating/Coordinating/Communicating
   ⇓
   The processes which communicate with other process(s).

Communication b/w processes:- (IPC Interprocess Communication)

- → Pipe
- → Queue
- → shared variable ✓
- → message passing

Topic : Need Of Synchronization

→ Synchronizat^n is needed b/w communicating processes to get expected result out of their execution.

Topic : Problems Without Synchronization

Problems without Synchronization:

- Inconsistency
- Loss of Data
- Deadlock

Topic : Entire Process Requires Synchronization?

critical section

inst^ns of process which are involved in communication with other process.

These inst^ns will need synchronization

remainder section (does not need any synchronization)

Remaining are not involved in any communication.

Topic : Critical Section

The critical section is a code segment where the shared variables can be accessed.

2 mins Summary

Topic System Call: Fork()

Topic Synchronization

Topic Race Condition

Topic Critical Section

Happy Learning

THANK - YOU