CS & IT ENGINEERING

Operating System

Multithreading

(One Shot)

Recap of Previous Lecture

Topic Multilevel Queue Scheduling

Topic Multilevel Feedback Queue Scheduling

Topic Questions on Scheduling

Topics to be Covered

Topic Multithreading

Topic System Call: Fork()

Topic : Multilevel Queue Scheduling

Queue 1: RR with Q=3
Queue 2: FCFS

Process	Arrival Time	Burst Time	Queue
P1	0	3	1
P2	0	3	1
P3	2	8	2
P4	10	4	1
P5	11	6	2
P6	11	3	1
P7	19	2	1
P8	13	5	2

[MCQ]

#Q. Consider three processes, all arriving at time zero, with total execution time of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The operating system uses a shortest remaining compute time first scheduling algorithm and schedules a new process either when the running process gets blocked on I/O or when the running process finishes its compute burst. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle?

GATE - 2006

A 0%

B 10.6%

C 30.0%

D 89.4%

[MCQ]

#Q. The arrival time, priority and duration of the CPU and I/O bursts for each of three processes P1, P2 and P3 are given in the table below. Each process has a CPU burst followed by an I.O burst followed by another CPU burst. Assume that each process has its own I/O resource.

Process	AT	Priority	BT (CPU)	BT(I/O)	BT(CPU)
P1	0	2	1	5	3
P2	2	3(lowest)	3	3	1
P3	3	1(highest)	2	3	1

The multi – programmed operating system uses preemptive priority scheduling. What are the finish times of the process P1, P2 and P3 ? **GATE - 2006**

A 11, 15, 9

B 10, 15, 9

C 11, 16, 10

D 12, 17, 11

Topic : Multithreading

Thread :-

Component of process

or

Lightweight Process

ex:-
downloading a file
&
showing progress on UI

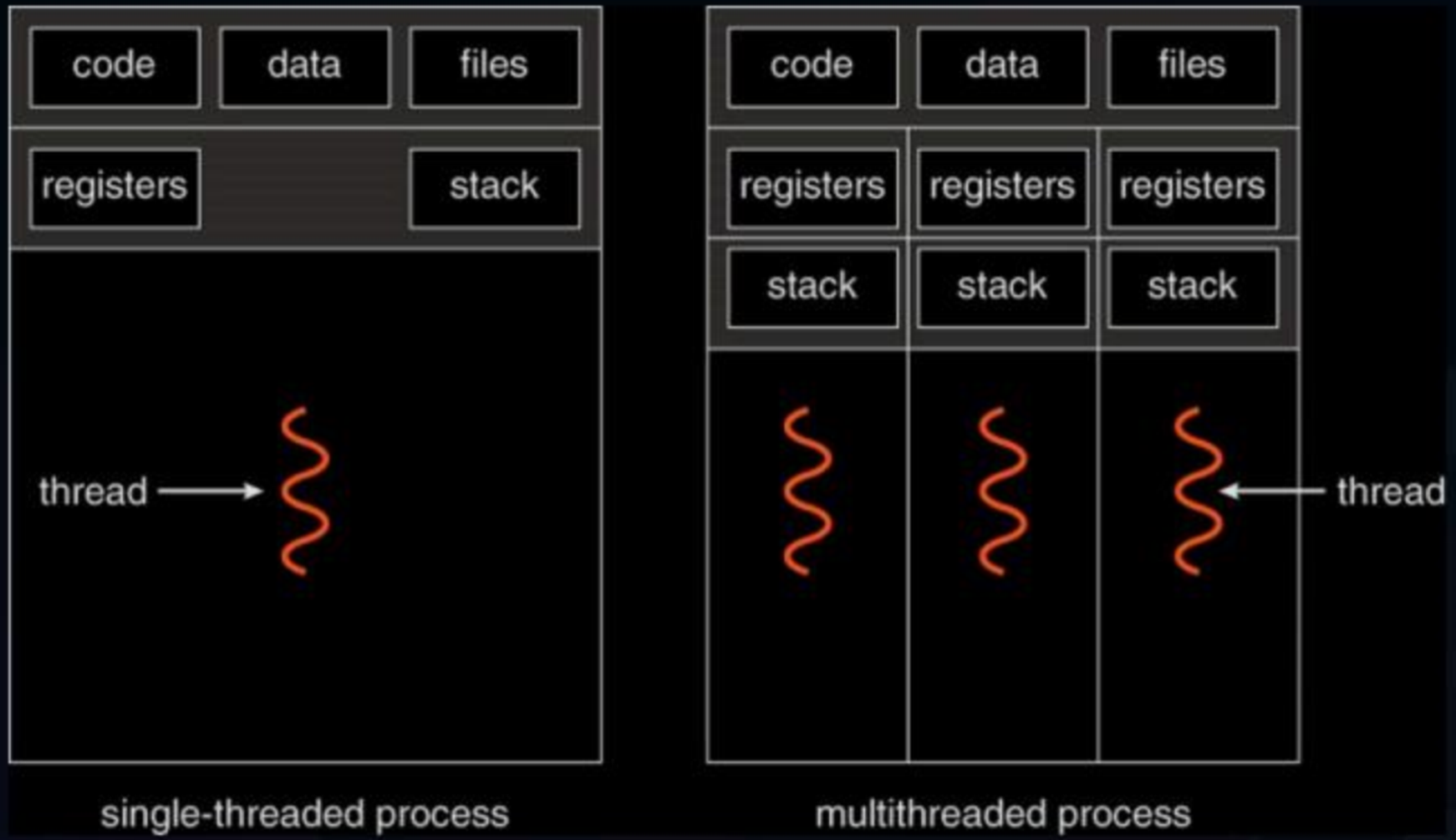
Provide a way to improve application performance through parallelism

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	

Thread control block

Topic : Threads



Topic : Advantages of Multithreading

- Responsiveness
- Faster Context Switch
- Resource Sharing
- Economy
- Communication
- Utilization of Multiprocessor Architecture

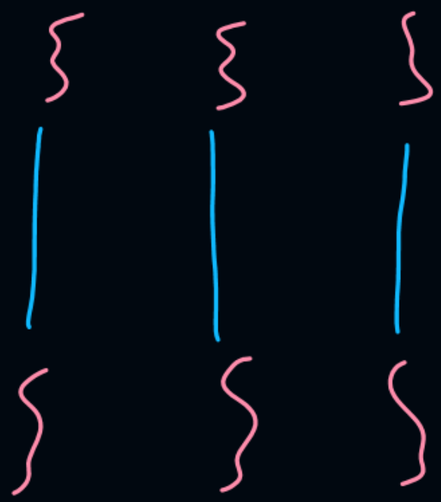
Topic : Types of Threads

1. User Level Thread → multithreading done in user process
2. Kernel Level Thread → —11— OS —11—

Topic : Types of 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

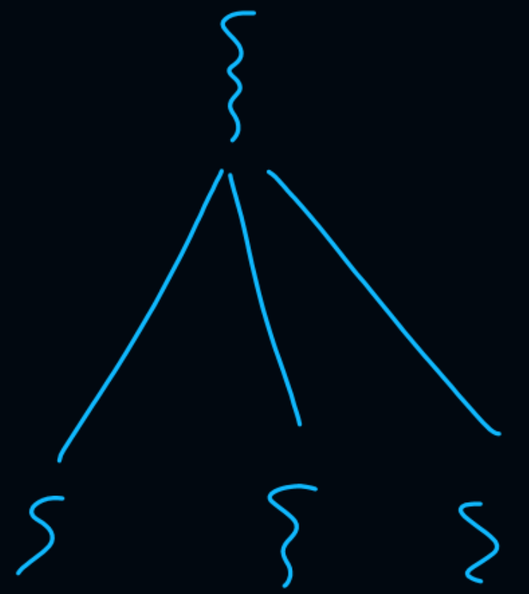
one-to-one mapping



user thread

kernel thread

one-to-many



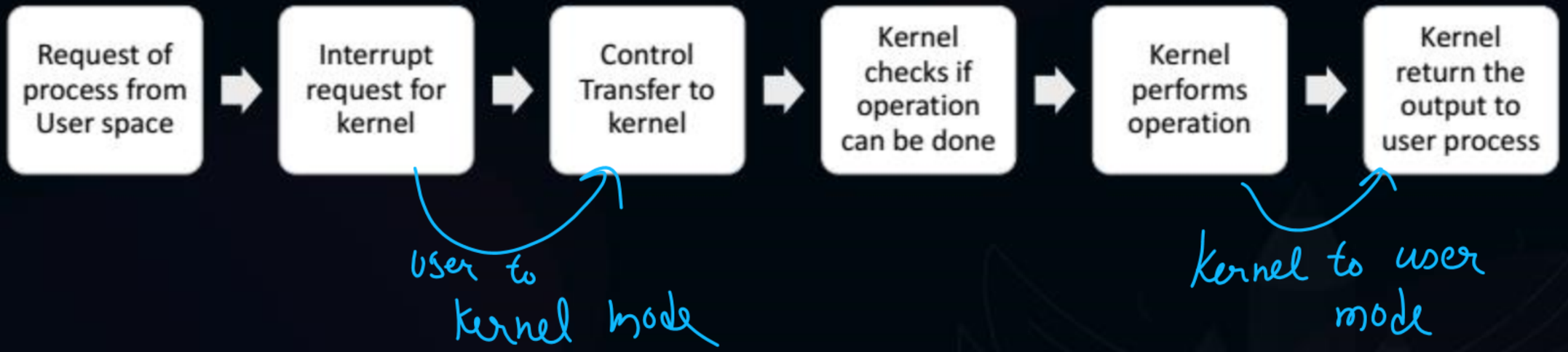
many-to-many

Topic : System Call

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

malloc() ⇒ does not need system call

Topic : How System Call Works



System Call Category

Process control

1. create process (for example, fork on Unix-like systems, or NtCreateProcess in the Windows NT Native API)
2. terminate process
3. load, execute
4. get/set process attributes
5. wait for time, wait event, signal event
6. allocate and free memory

malloc ()

System Call Category

File Management

1. create file, delete file
2. open, close
3. read, write, reposition
4. get/set file attributes

System Call Category

Device Management

1. request device, release device
2. read, write, reposition
3. get/set device attributes
4. logically attach or detach devices

System Call Category

Information Maintenance

1. get/set total system information (including time, date, computer name, enterprise etc.)
2. get/set process, file, or device metadata (including author, opener, creation time and date, etc.)

System Call Category

Communication

1. create, delete communication connection
2. send, receive messages
3. transfer status information
4. attach or detach remote devices

System Call Category

Protection

1. get/set file permissions

System Call

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

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).

Topic : Fork() System Call

It takes no parameters and returns an integer value

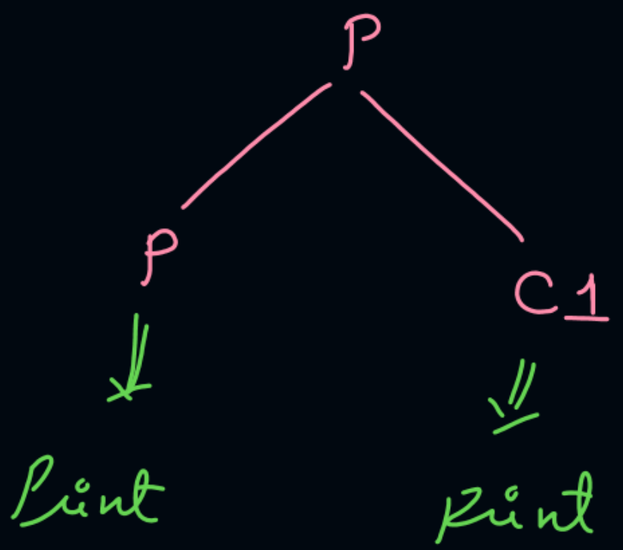
- Negative Value:
  Creation of a child process was unsuccessful

- 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

ex:-

```
main()
{
  fork();
  printf("I am a GATE topper\n");
}
```

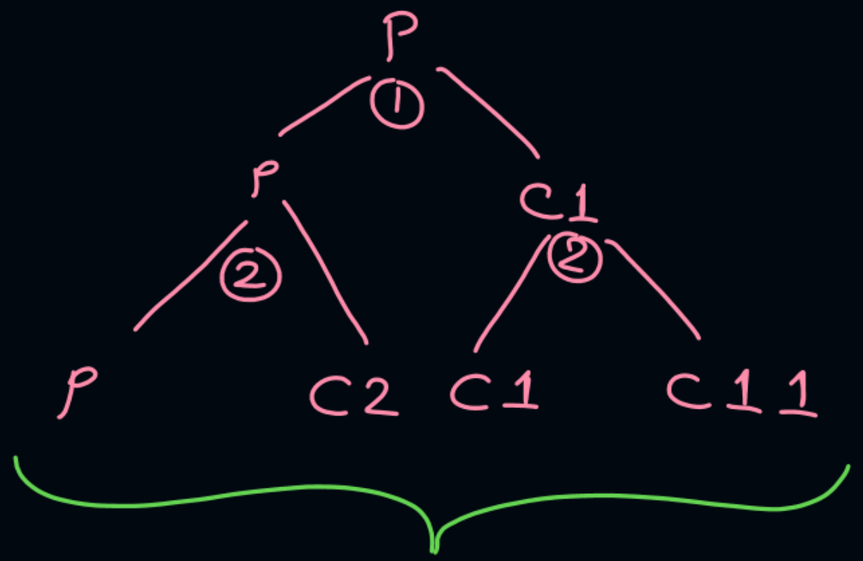


output:-

I am a GATE topper
I am a GATE topper

P

```
fork();  ①
fork();  ②
```



4 processes
- 1 Parent
- 3 child process

n fork()
- 1 parent
- $2^n - 1$ child processes

total = 2^n process

[MCQ]

#Q. A process executes the code

1. fork ();
2. fork ();
3. fork ();

The total number of child processes created is?

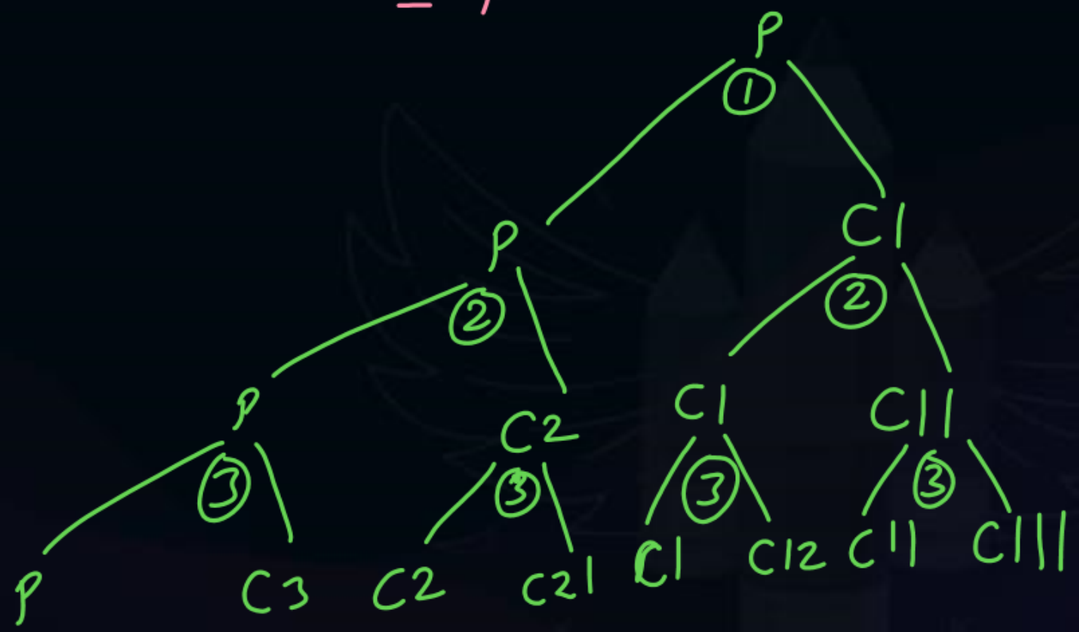
$= 2^3 - 1$

$= 7$

(A) 3 (C) 7

(B) 4 (D) 8

GATE-2008



[NAT]

#Q. A process executes the code

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

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

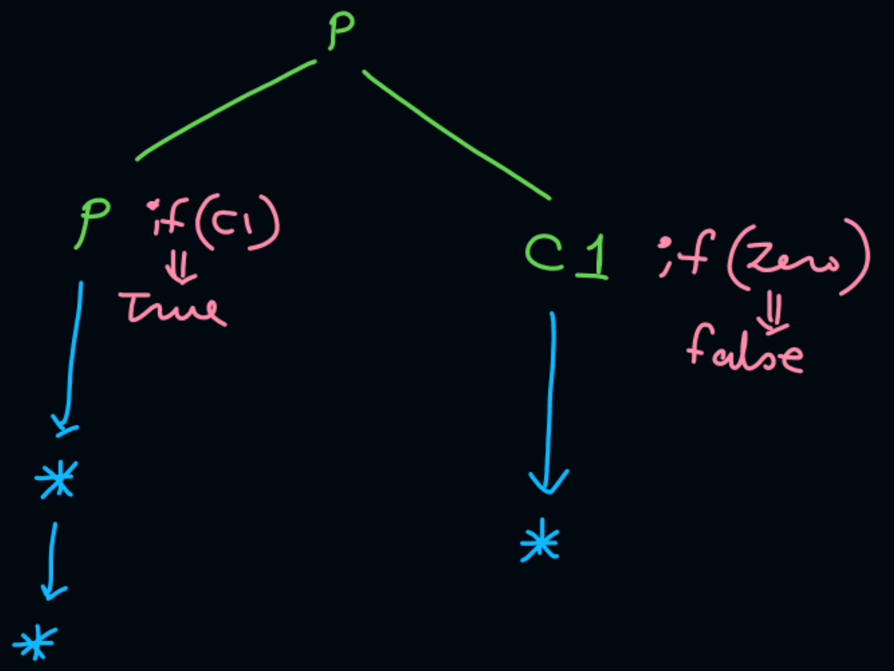
```
for (i = 0; i < n; i++)
{ fork();
}
```

$2^n - 1$

ques)

```
if (fork())
{
   printf(" * ");
}
printf("*");
```

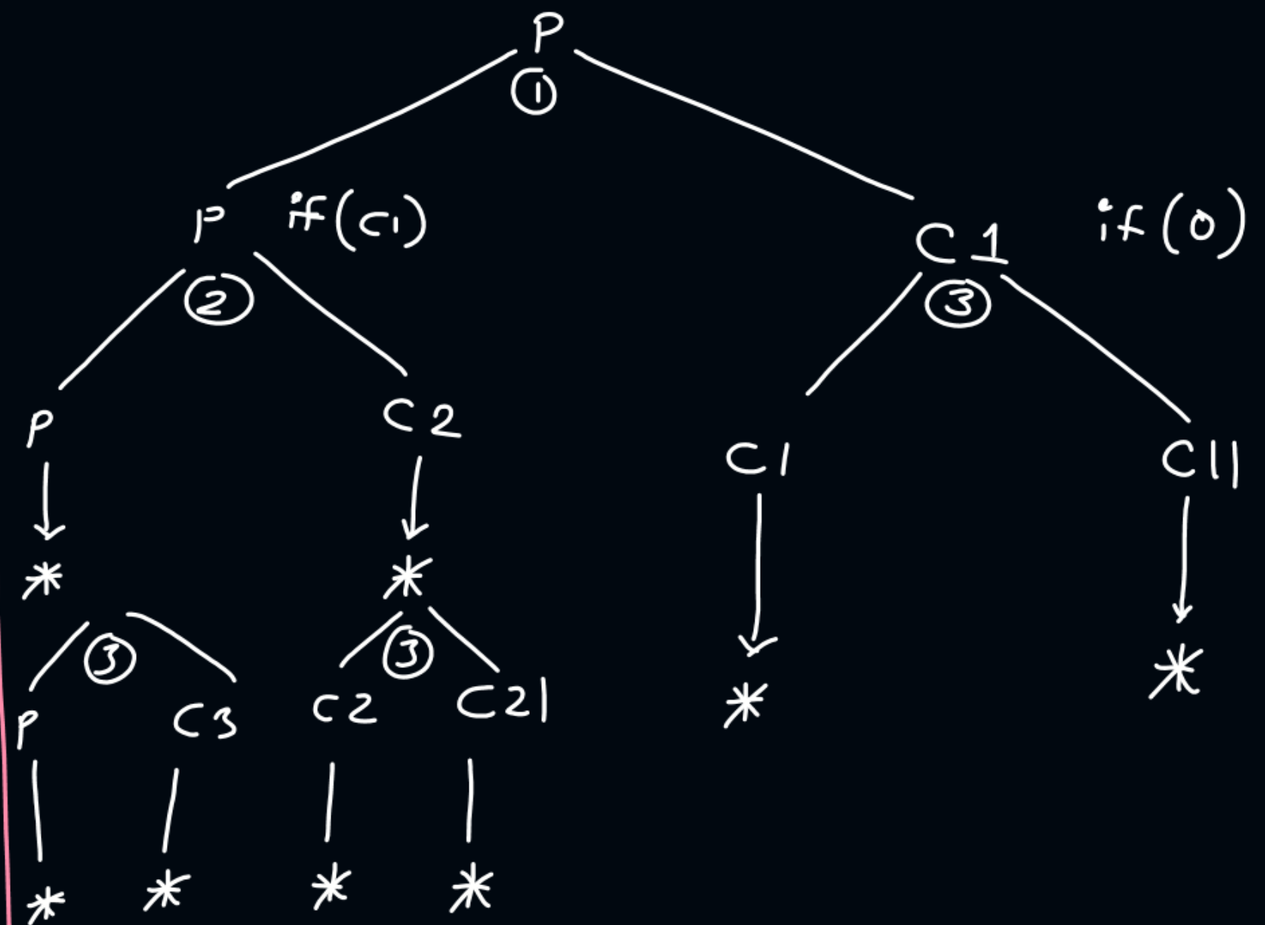
no. of times * is printed 3 &



Ques

```
if ( fork() )        // ①
{
    fork();          // ②
    printf("*");
}
fork();              // ③
printf("*");
```

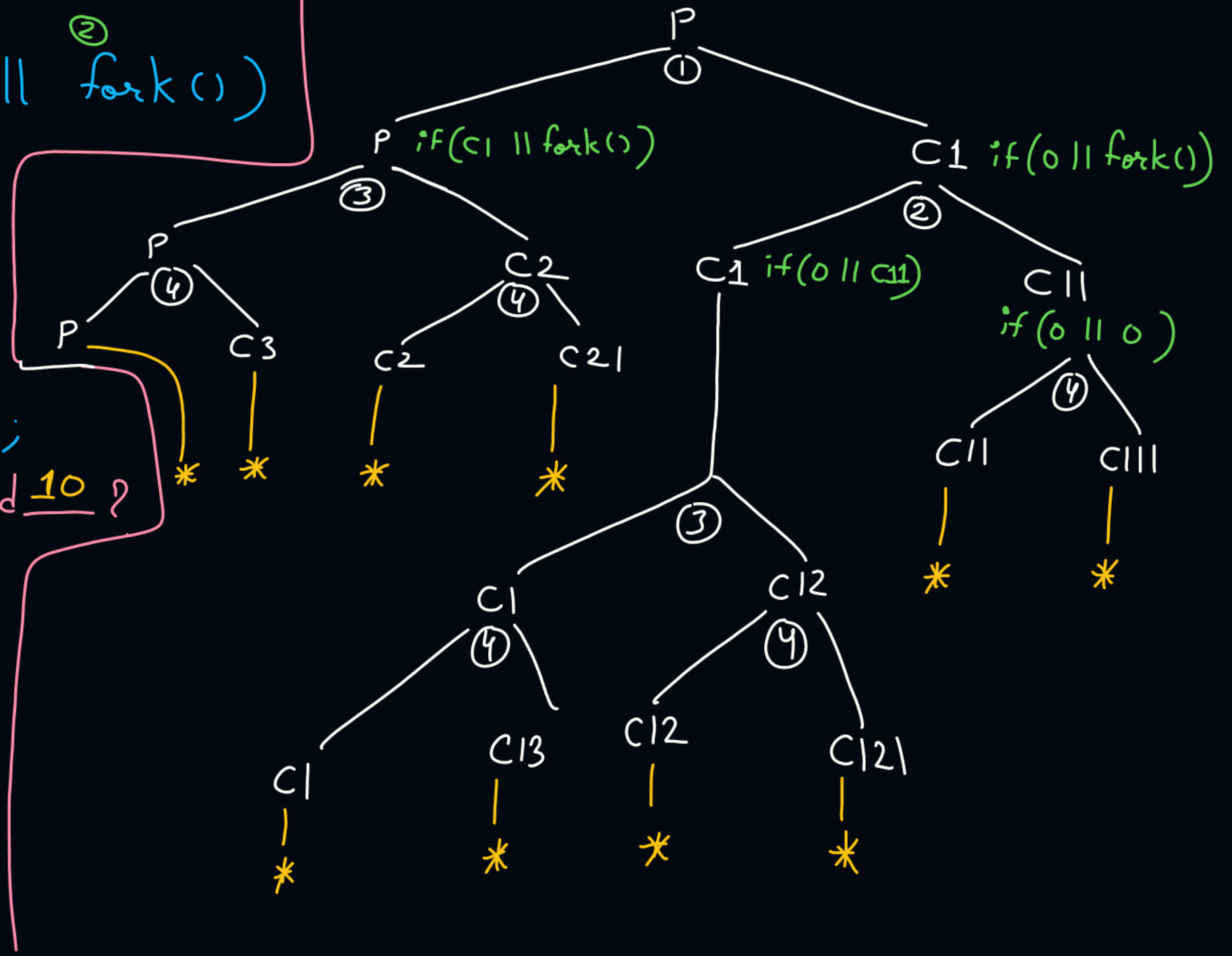
no. of times * is printed — 8 ?



Ques)

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

no. of times * is printed 10 ?



H.W.

Ques)

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

no. of times * is 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]

H.W.

#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

2 mins Summary

Topic Multithreading

Topic System Call: Fork()

Happy Learning

THANK - YOU