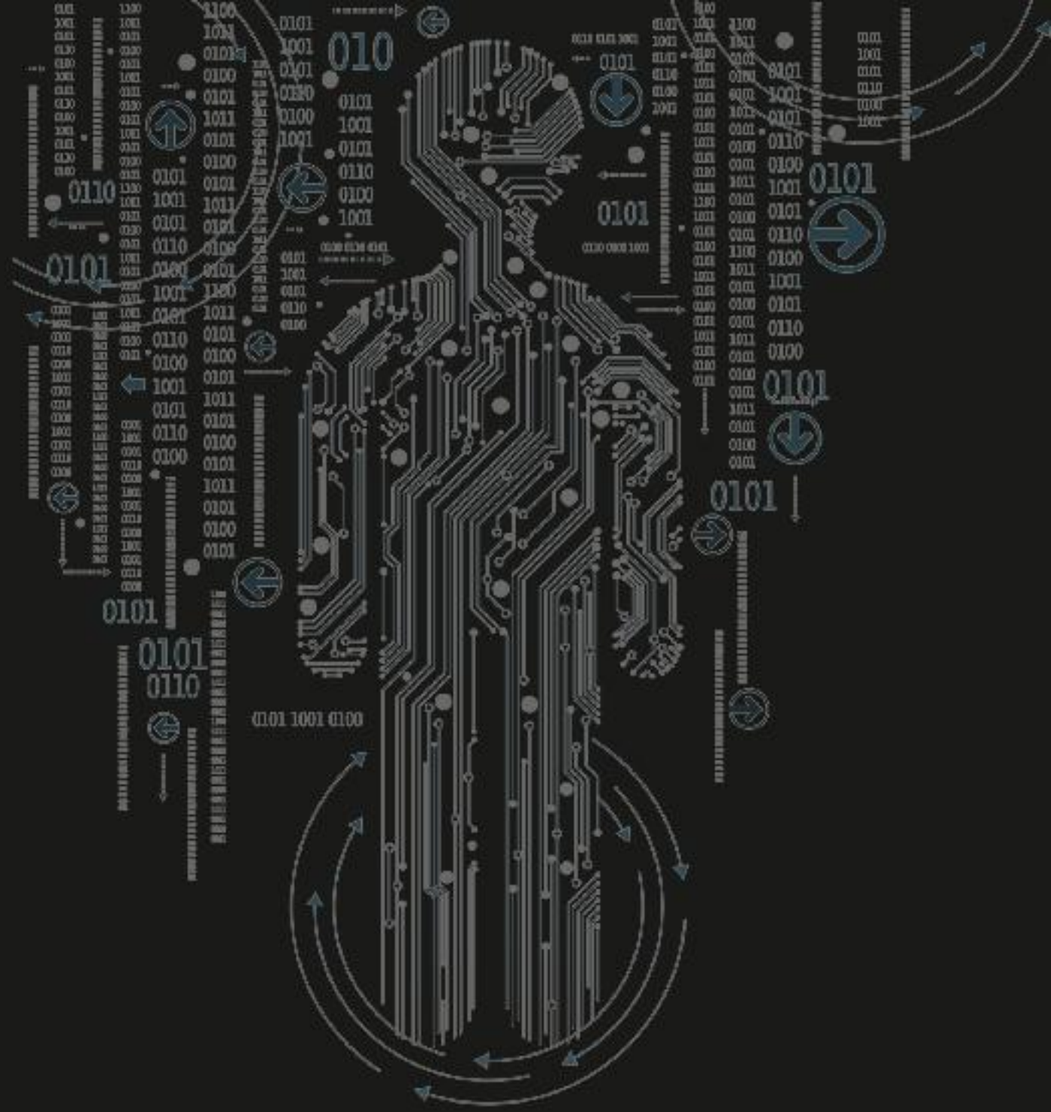
BYJU'S EXAM PREP

OPERATING SYSTEMS

System Calls



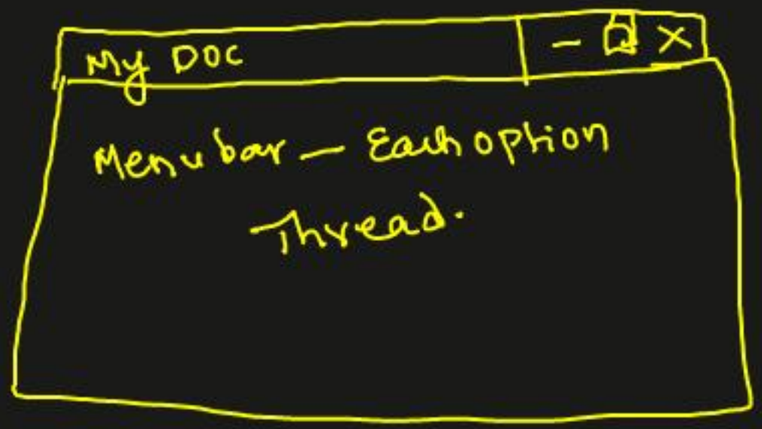
Guidelines to Attend Live Class

BYJU'S EXAM PREP

- ✓ Pen, Notebook and Calculator are must while attending the online class.
- ✓ Regularity and Punctuality is necessary.
- ✓ Hold chat while attending the class. We will allow you to ask and put your questions in the comment box.
- ✓ Attempt Quizzes Daily as per the schedule.
- ✓ Strictly follow the day-wise study plan.

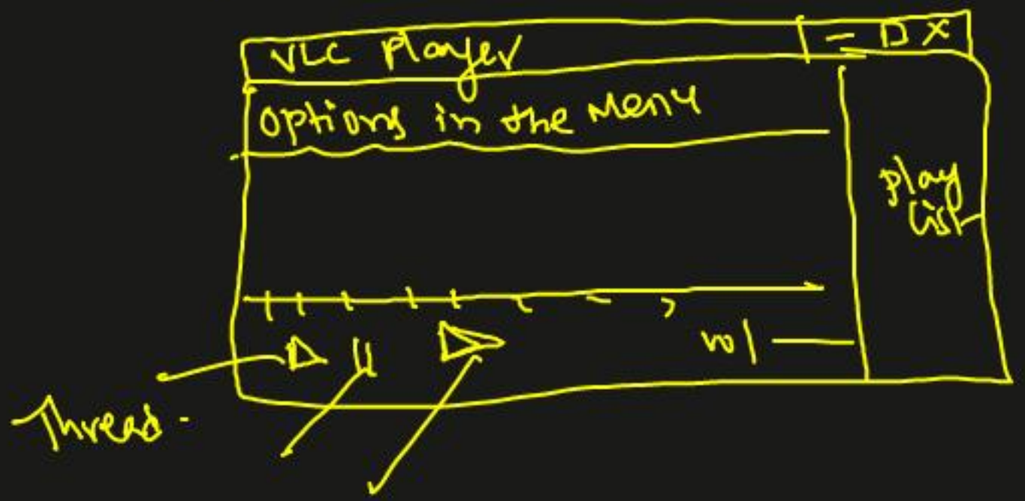
Thread

A Thread is a Lightweight Process.

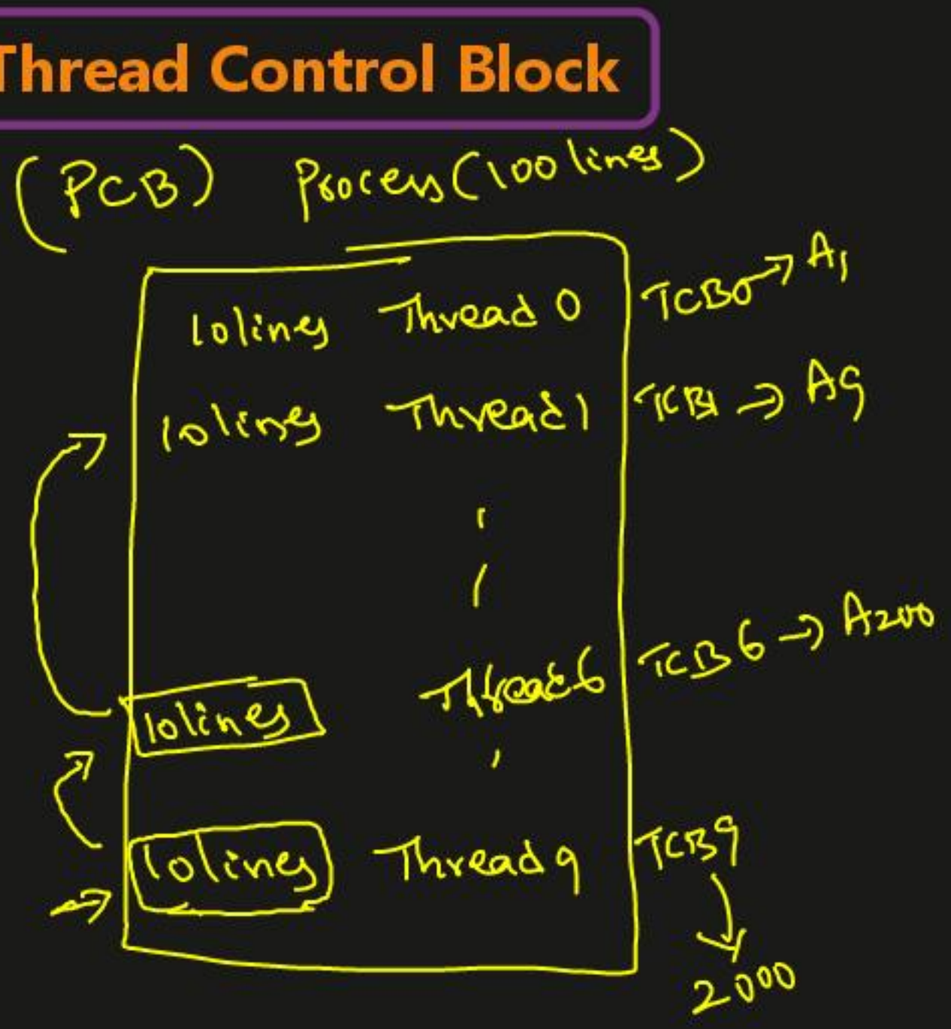


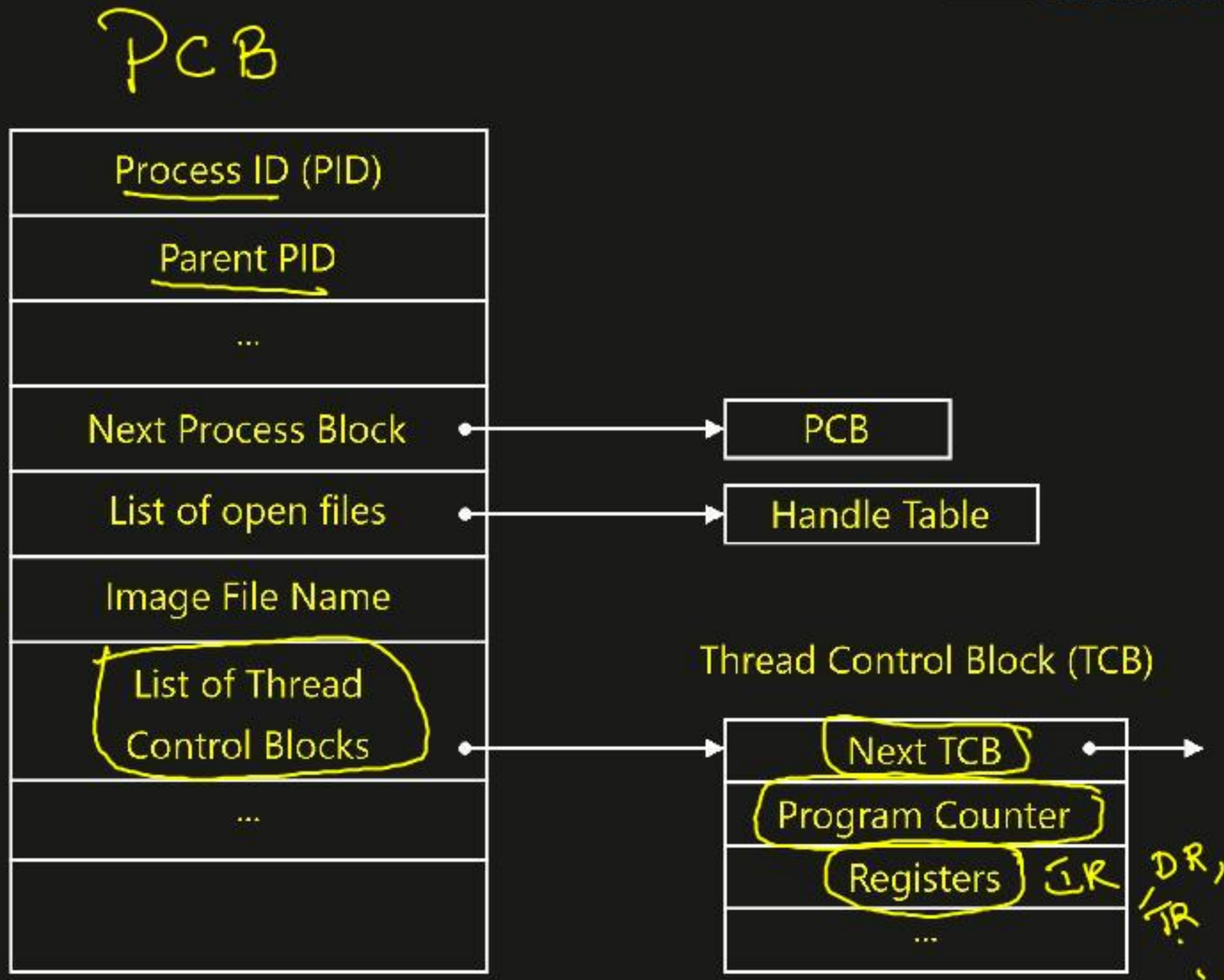
- Less Memory Space
- Less Execution time
- Less resources

Compared with Process.



Thread Control Block





TCB

① Stack ② Heap ③ Text area [Program]

Memory Layout Of Program / Process

⑦ Preprocessor directives

① int a; ← Not initialized

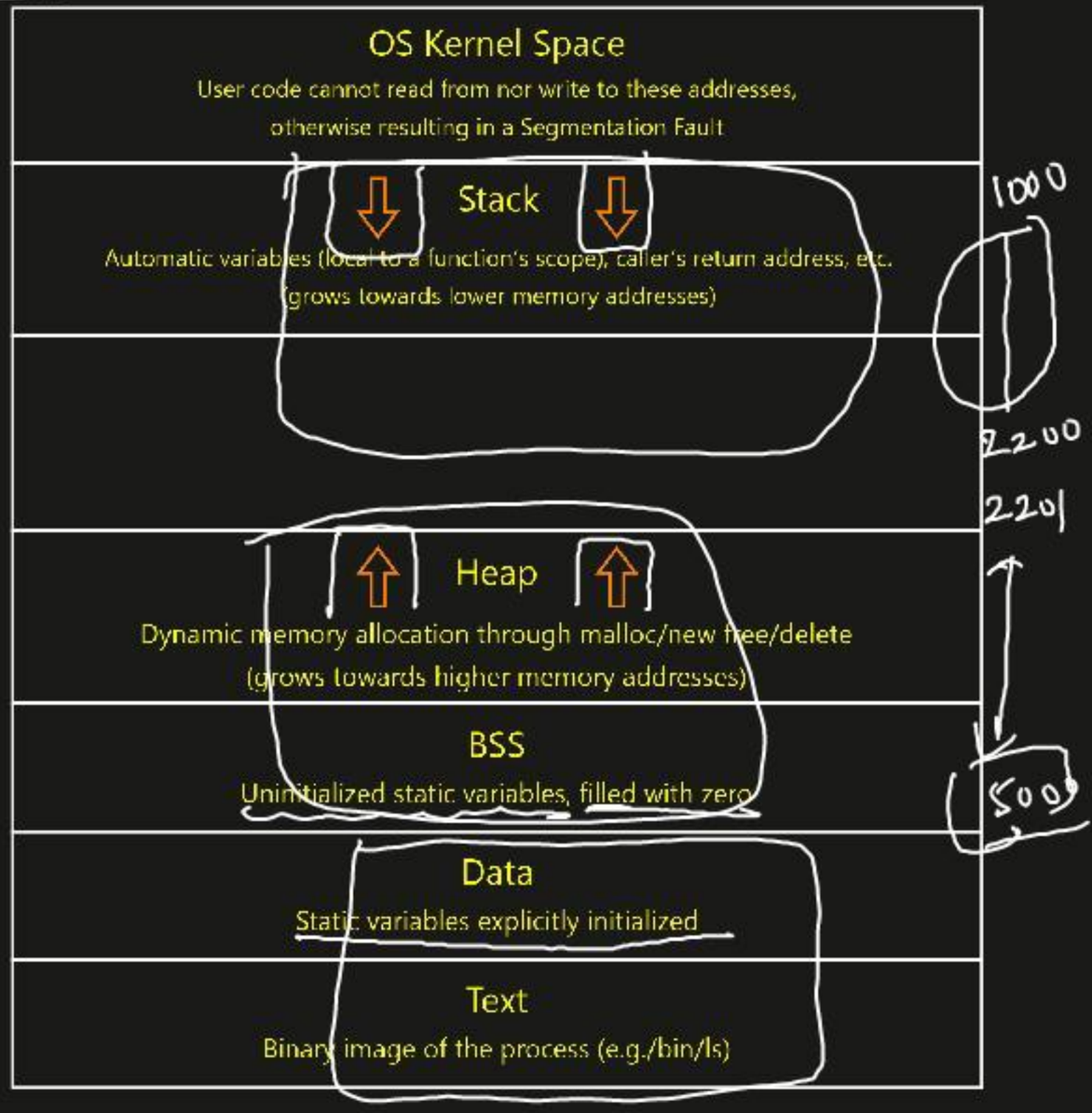
② int a = 7; ← Initialized

③ Static int a; ← static without initialization

④ Static int a = 9; ← static with initial

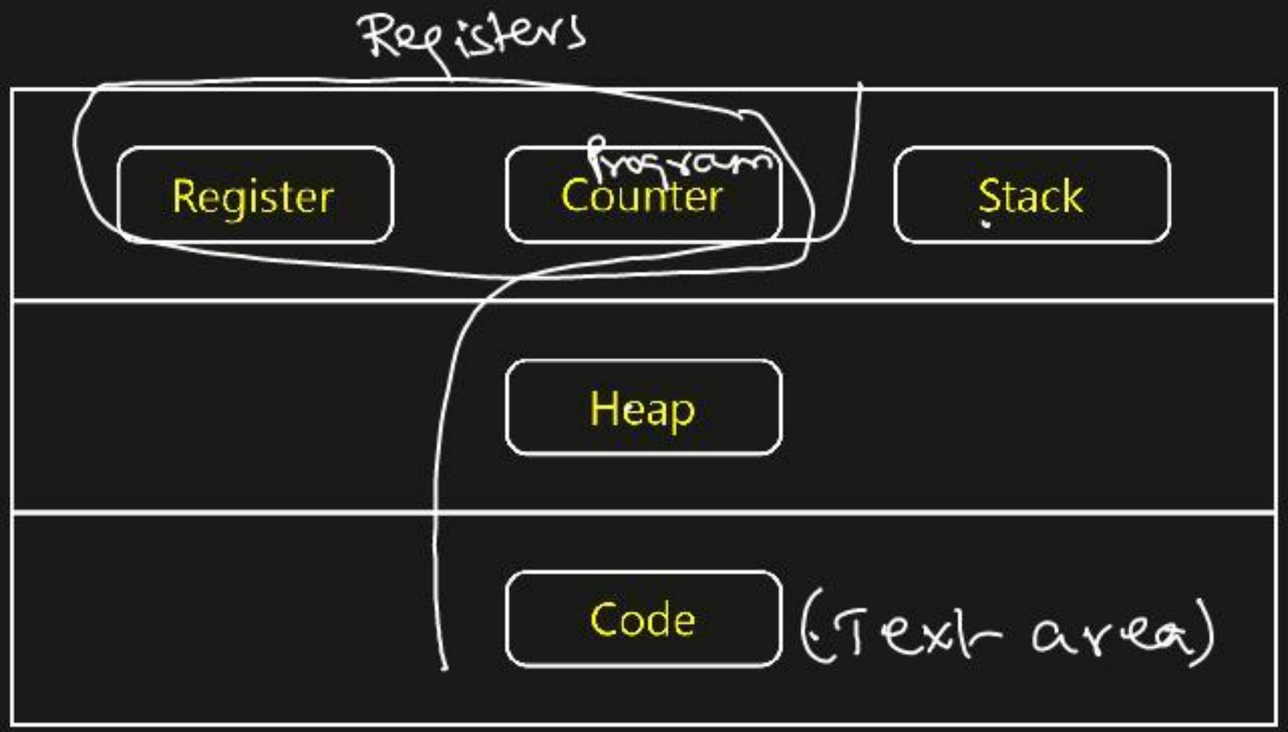
⑤
```
int *p;
p = (int *) malloc ( /* size of(int) */ )
```
— Dynamic memory allocated Variable

⑥
```
int a;
{ --- Program
}
```
Global Variable



Memory Layout

Threads

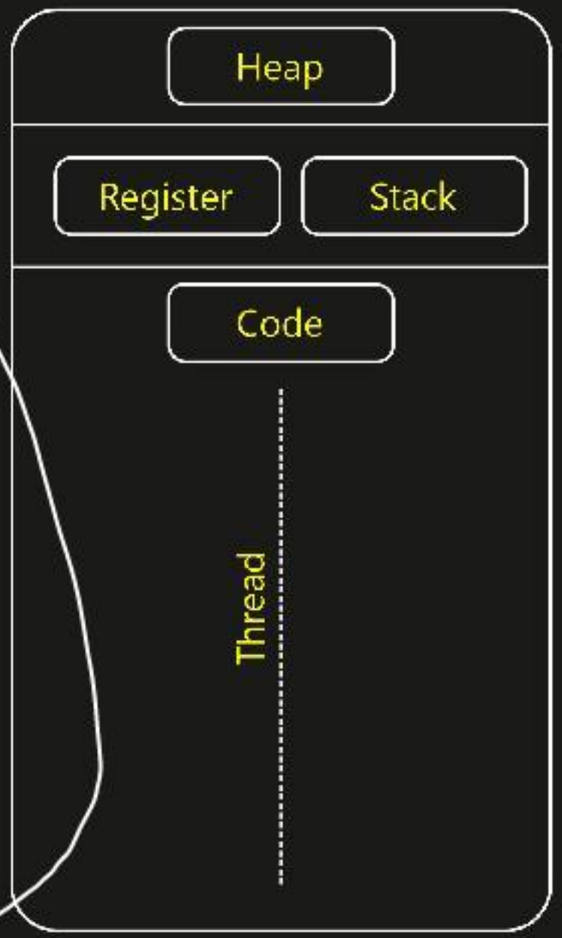


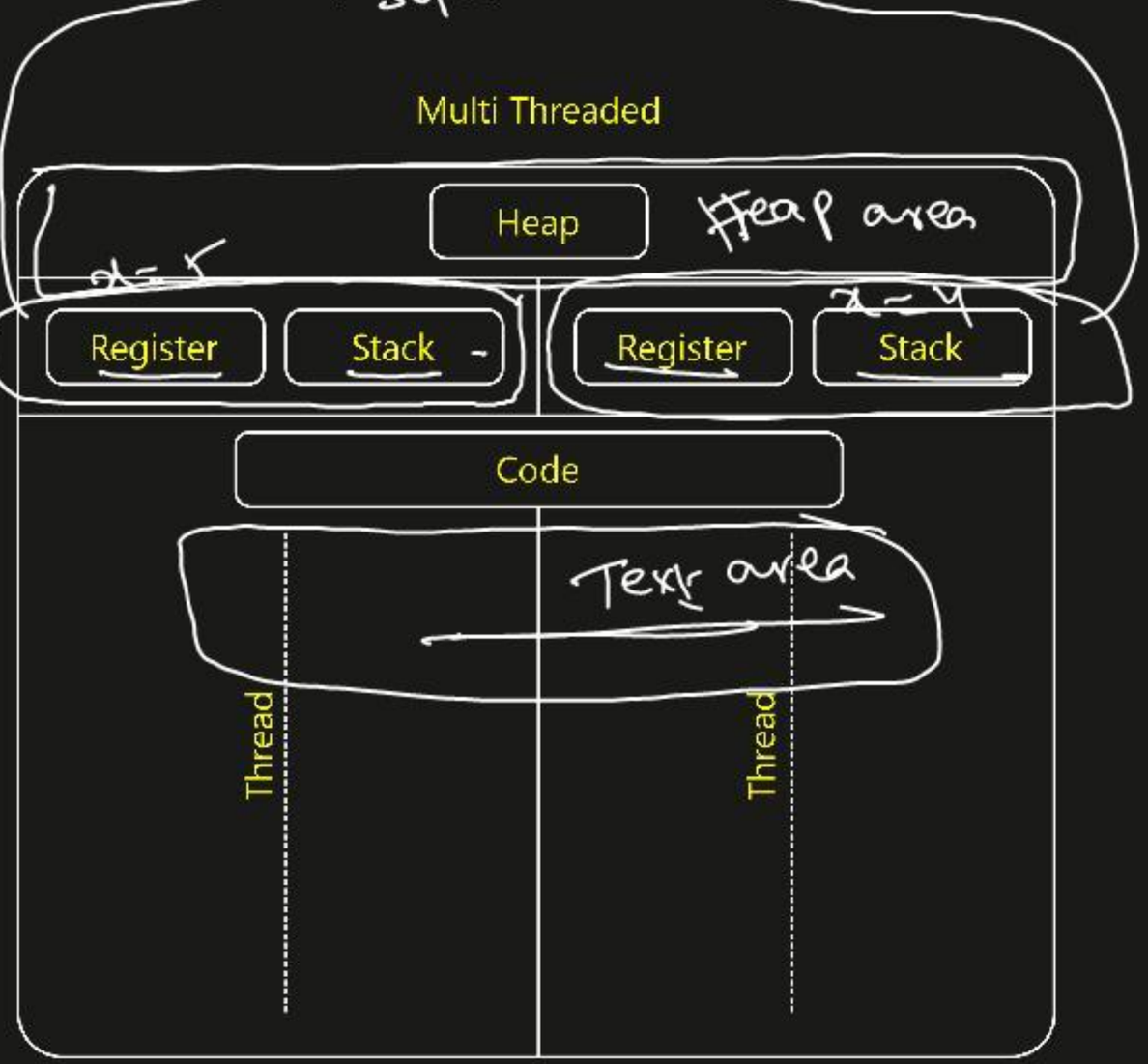
Single Thread

A(5) → A(4) → A(3) → A(2) → A(1)
x=5 x=4 x=3 x=2 x=1

Threads

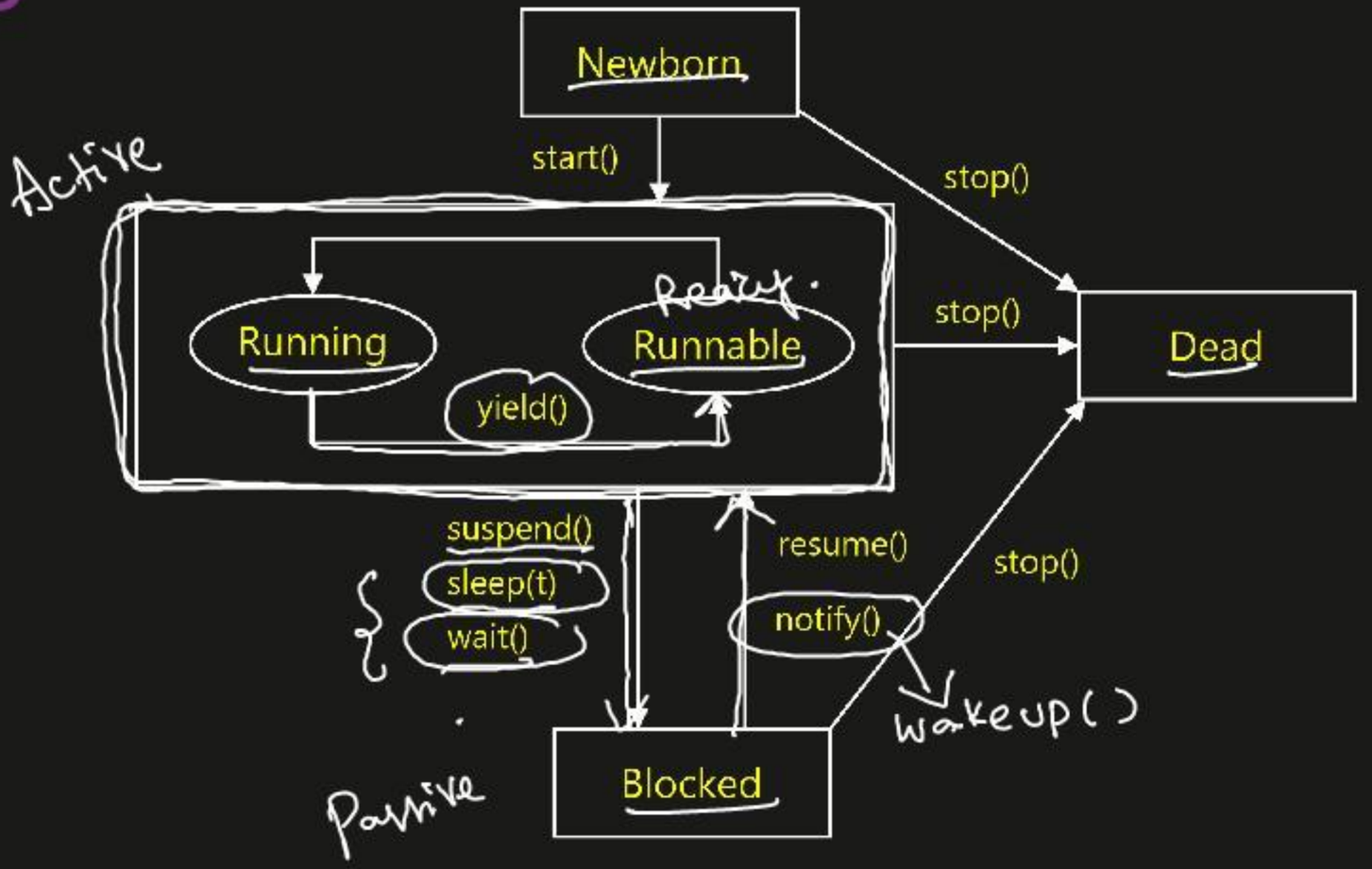
```
1 Void A(int x)
2 {
3   if(x<=1)
      return x;
4   else
5     Print x;
      A(x-1);
6 }
```





Multi Threaded Architecture

Thread States



Thread State Diagram

Process Vs Threads

SN	Process	Thread
1.	Process is heavy weight or resource intensive.	Thread is light weight taking lesser resources than a process.
2.	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3.	In multiple processing environments each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
4.	If one process is blocked then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, second thread in the same task can run.

Process Vs Threads

SN	Process	Thread
5.	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6.	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.
7.	Unlike threads, processes don't share the same address space.	Multithreading requires careful programming since threads share data strucures that should only be modified by one thread at a time.

Process Vs Threads

Less
ULT < KLT < Process (More time)

Context Switching.

ULT vs KLT

Maintained by External Source (Thread library)

Extend Thread class

Implementing Runnable interface.

S.N.	User-Level Threads	Kernel-Level Thread
1.	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manager.
2.	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3.	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4.	Multi-threaded application cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Blocking of one ULT blocks entire Process.

ULTs Vs KLTs

Blocking of one KLT does not block entire Process.

Q2. Consider the following statements about user level threads and kernel level threads. Which one of the following statement is FALSE?

A. Context switch time is longer for kernel level threads than for user level threads. True

B. User level threads do not need any hardware support. True

C. Related kernel level threads can be scheduled on different processors in a multi-processor system. True

D. Blocking one kernel level thread blocks all related threads. False.

Q3. Consider the following statements with respect to user-level threads and kernel supported threads

i. Context switch is faster with kernel-supported threads. False

ii. For user-level threads, a system call can block the entire process. True

iii. Kernel supported threads can be scheduled independently. True

iv. User level threads are transparent to the kernel. True

Which of the above statements are true?

A (ii), (iii) and (iv) only

B (ii) and (iii) only

C (i) and (iii) only

D (i) and (ii) only

Q4. A thread is usually defined as a 'light weight process' because an Operating system (OS) maintains smaller data structures for a thread than for a process. In relation to this, which of the following is TRUE ?

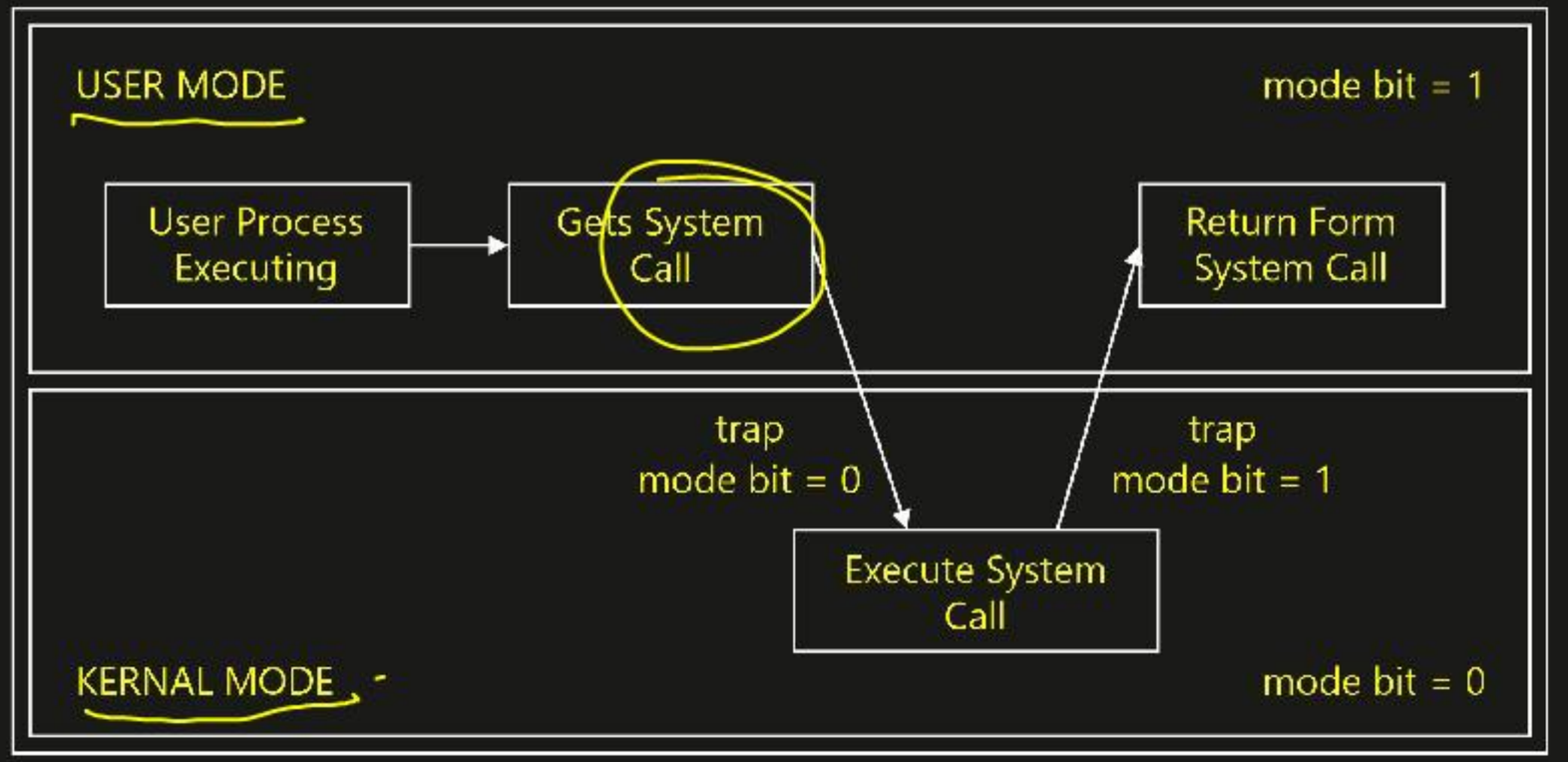
A. On per-thread basis, the OS maintains only CPU register state. False

B. The OS does not maintain a separate stack for each thread. False

C. On per-thread basis, the OS does not maintain virtual memory state. True

D. On per thread basis, the OS maintains only scheduling and accounting information. False

User Mode and Kernel Mode



User and Kernel Modes

System Call

The means through which, Operating System Kernel Services are acquired.

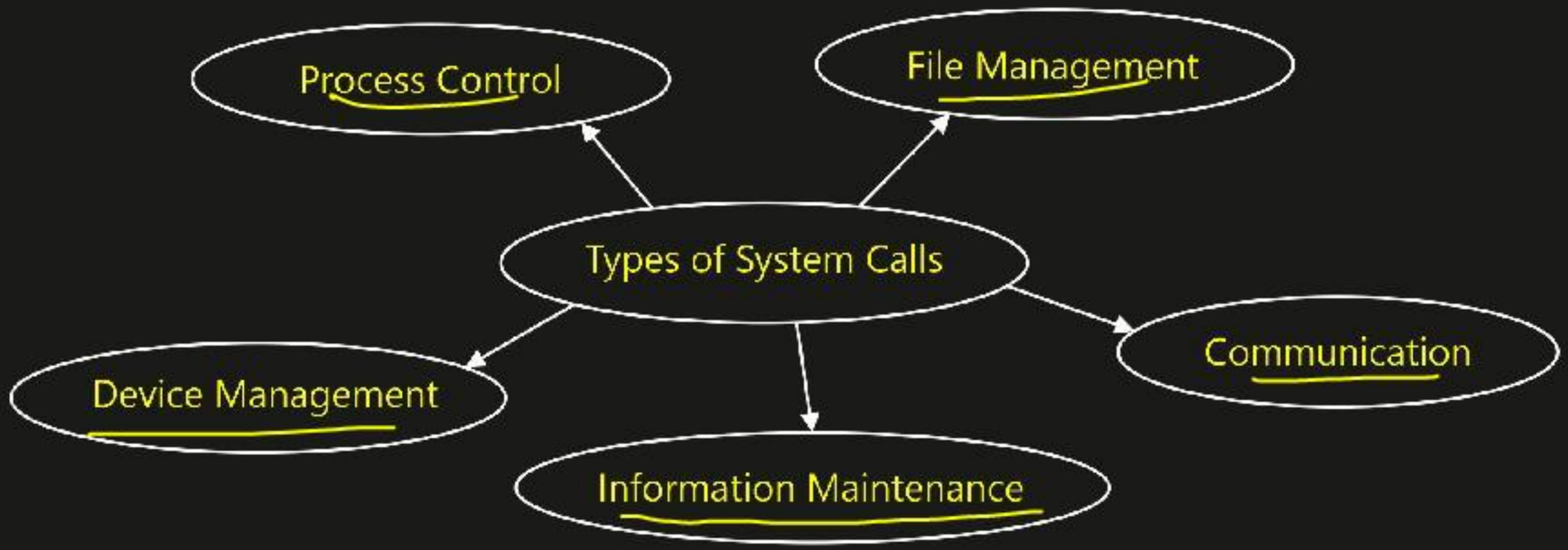
Ex: admit (p); Process P is loaded into Memory

Suspend (P_2); Process P_2 will be suspended from RAM.

dispatch (P_x); Process P_x will be assigned to cpu.

Preempt (P_y); Process P_y will be interrupted.

Types Of System Calls



Types of System Calls

Process Control

This system call performs the task of process creation, process termination, etc.

Functions :

- End and Abort (abnormal Termination)
- Load and Execute
- Create Process and Terminate Process
- Wait and Signal Event (resume)
- Allocate and free memory

File Management

File management system calls handle file manipulation jobs like creating a file, reading, writing, etc.

Functions :

- Create a file
- Delete file
- Open and close the file
- Read, write and reposition
- Get and set file attributes

Device Management

Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.

Functions :

- Request and release device
- Logically attach/ detach devices
- Get and Set device attributes

Information Maintenance

It handles information and its transfer between the OS and the user program.

Functions :

- Get or set time and date
- Get process and device attributes

Communication

These types of system calls are specially used for interprocess communications.

Functions :

- Create, and delete communications connections
- Send, receive the message
- Help OS to transfer status information
- Attach or detach remote devices

System Calls

fork() : without arg, with return value

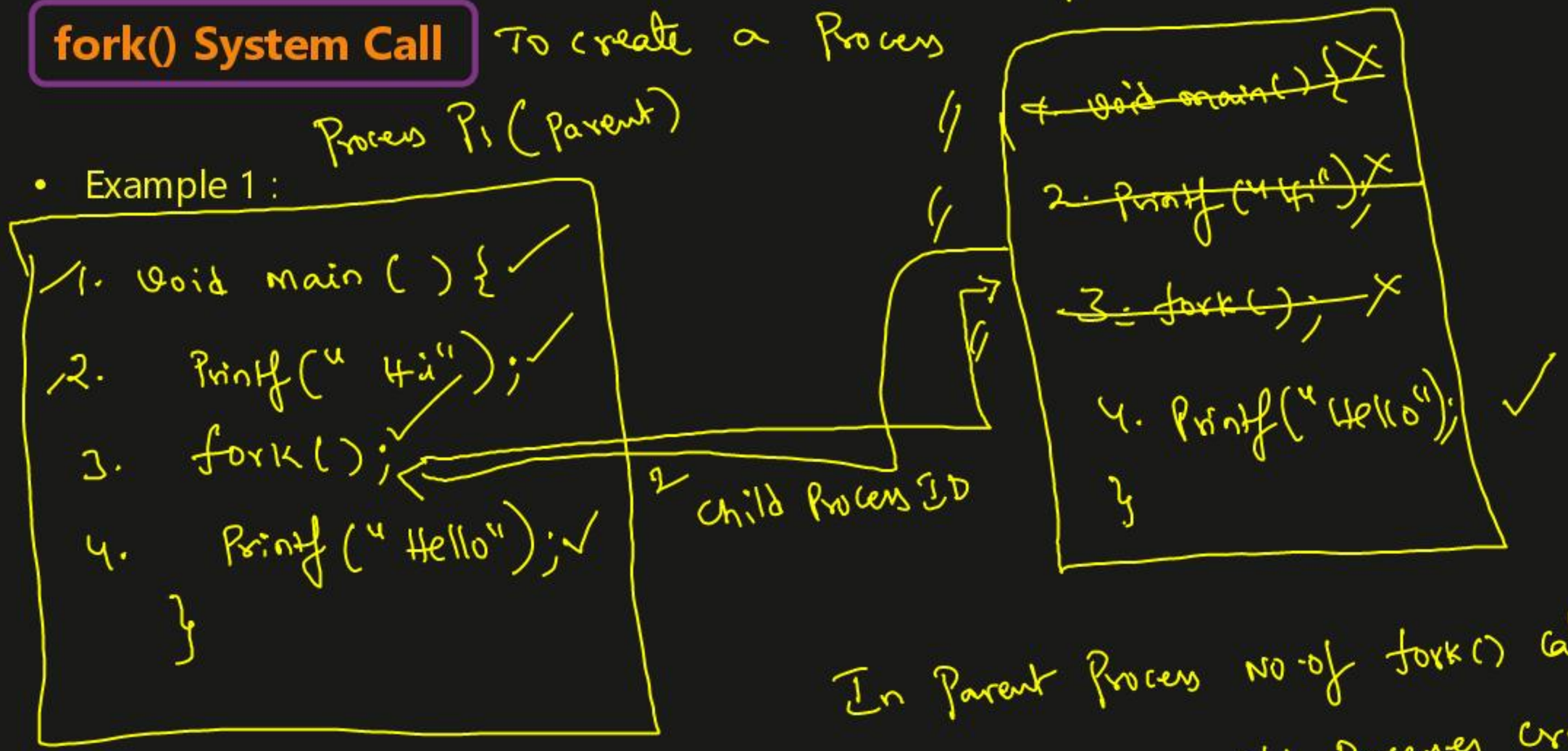
fork() System Call

To create a Process

Process P_1 (Parent)

Process P_2 (child)

- Example 1 :



O/p: Hi Hello Hello

In Parent Process no. of fork() calls = 1

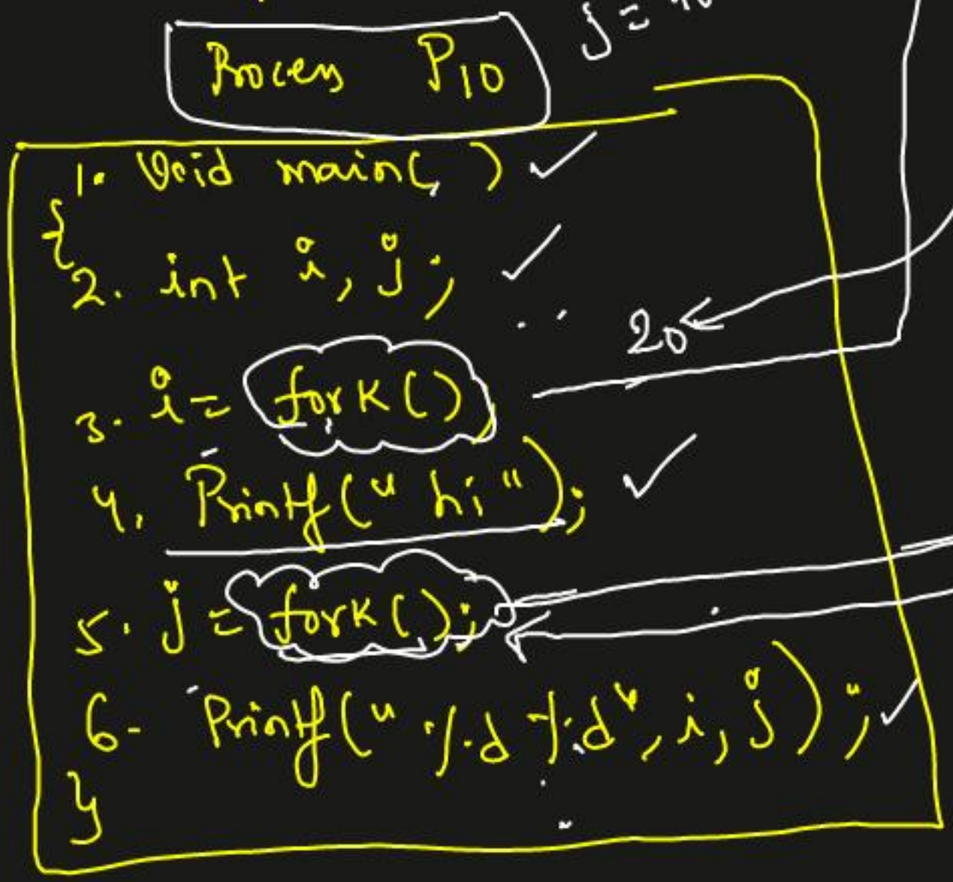
how many child Processes created ?

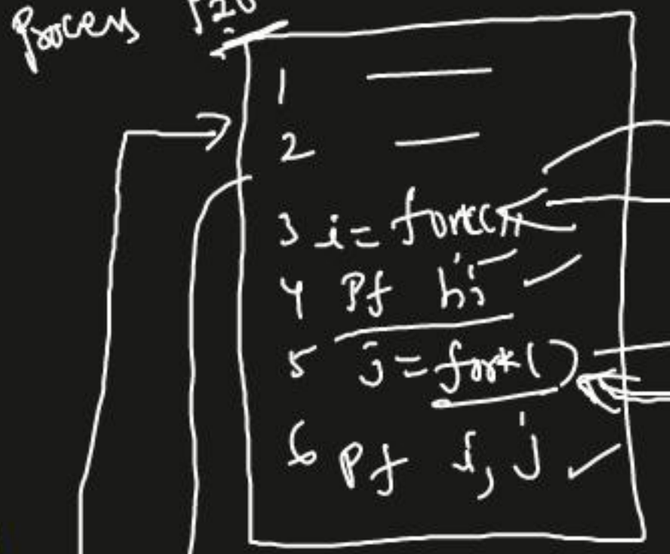
ANS: 1

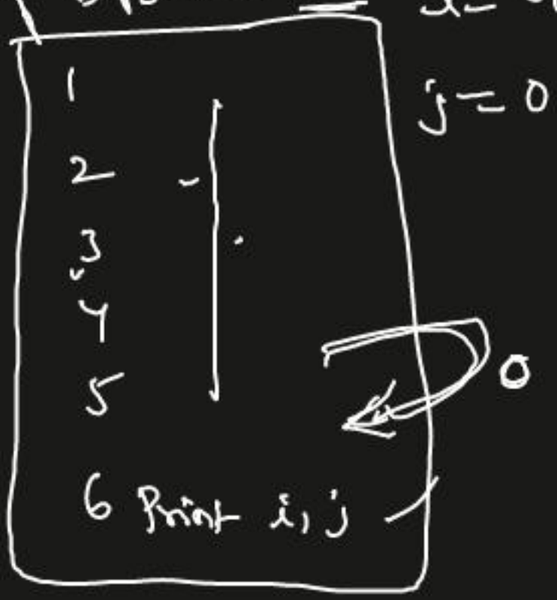
System Calls

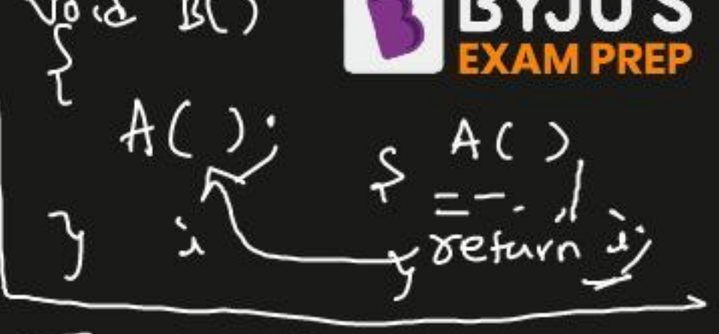
fork() System Call

- Example 2 : i = 20 j = 40





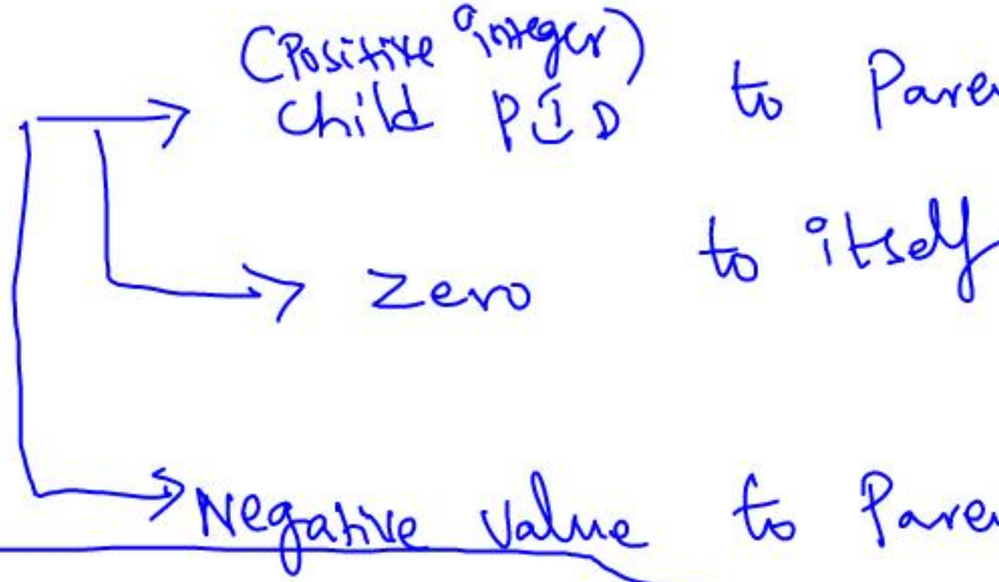




o/p: hi hi 20 40 0 30 Garbage 0 Garbage 0

P10 P20

⇒ When fork() is executed, a child Process is created and it returns



1 fork() call	1 child Process
2 fork() calls	3 child Processes
3 fork() calls	7 child Processes
⋮	
n fork() calls	$(2^n - 1)$ child Processes

fork() System Call

- Example 3 :

Most Important System Calls

To be contd ...

wait()

In some systems, a process needs to wait for another process to complete its execution. This type of situation occurs when a parent process creates a child process, and the execution of the parent process remains suspended until its child process executes.

The suspension of the parent process automatically occurs with a wait() system call. When the child process ends execution, the control moves back to the parent process.

fork()

Processes use this system call to create processes that are a copy of themselves. With the help of this system call, the parent process creates a child process, and the execution of the parent process will be suspended till the child process executes.

Most Important System Calls

exec()

This system call runs when an executable file in the context of an already running process replaces the older executable file. However, the original process identifier remains as a new process is not built, but stack, data, head, data, etc. are replaced by the new process.

kill()

The kill() system call is used by OS to send a termination signal to a process that urges the process to exit. However, a kill system call does not necessarily mean killing the process and can have various meanings.

Most Important System Calls

exit()

The exit() system call is used to terminate program execution. Especially in the multi-threaded environment, this call defines that the thread execution is complete. The OS reclaims resources that were used by the process after the use of the exit() system call.

System Calls

BYJU'S EXAM PREP

Q1. Which of the following standard C library functions will always invoke a system call when executed from a single-threaded process in a UNIX/Linux operating system ?

[MSQ]

A exit

B malloc

C sleep

D strlen

H/W

Q2. The following C program is executed on a Unix/Linux system :

```
#include<unistd.h>
int main()
{
    int i;
    for(i=0; i<10; i++)
        if(i%2 == 0)
            fork();
    return 0;
}
```

The total number of child processes created is _____ .

Q3. The following C program:

```
{
    fork(); fork(); printf("yes");


}
```

If we execute this core segment, how many times the string yes will be printed?

A. Only once

B. 2 times

C. 4 times

D. 8 times

Q4. What is the output of the following program?

```
main()
{
    int a = 10;
    if(fork()) == 0))
        a++;
    printf("%d\n",a);
}
```

y/w

A. 10 and 11

B. 10

C. 11

D. 11 and 11

Q5. Consider the following code fragment:

```
if (fork() == 0)
{
  a = a + 5;
  printf("%d, %p n", a, &a);
}
else
{
  a = a - 5;
  printf ("%d, %p n", a,& a);
}
```

Q6. Let u, v be the values printed by the parent process and x,y be the values printed by the child process. Which one of the following is TRUE ?

A. u=x+10 and v=y

B. u=x+10 and v!=y

C. u+10=x and v=y

D. u+10=x and v!=y


BYJU'S
EXAM PREP
4.5
Google Play Review
GET IT ON
Google Play