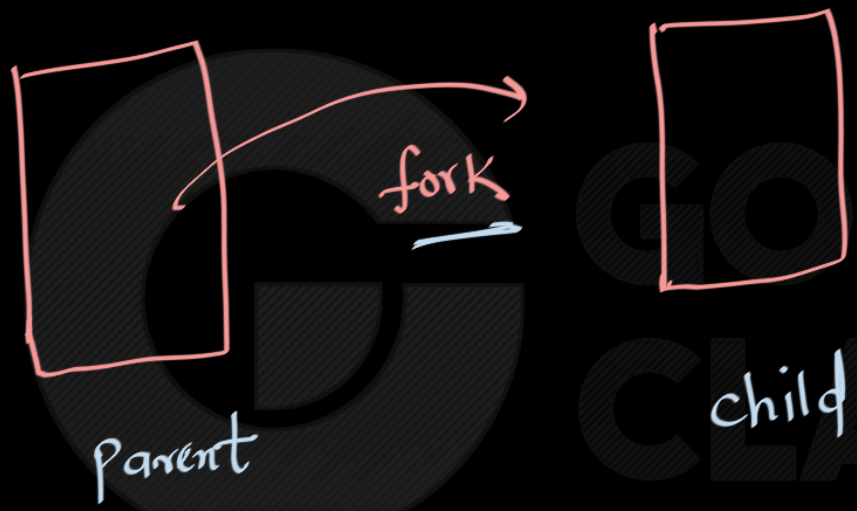
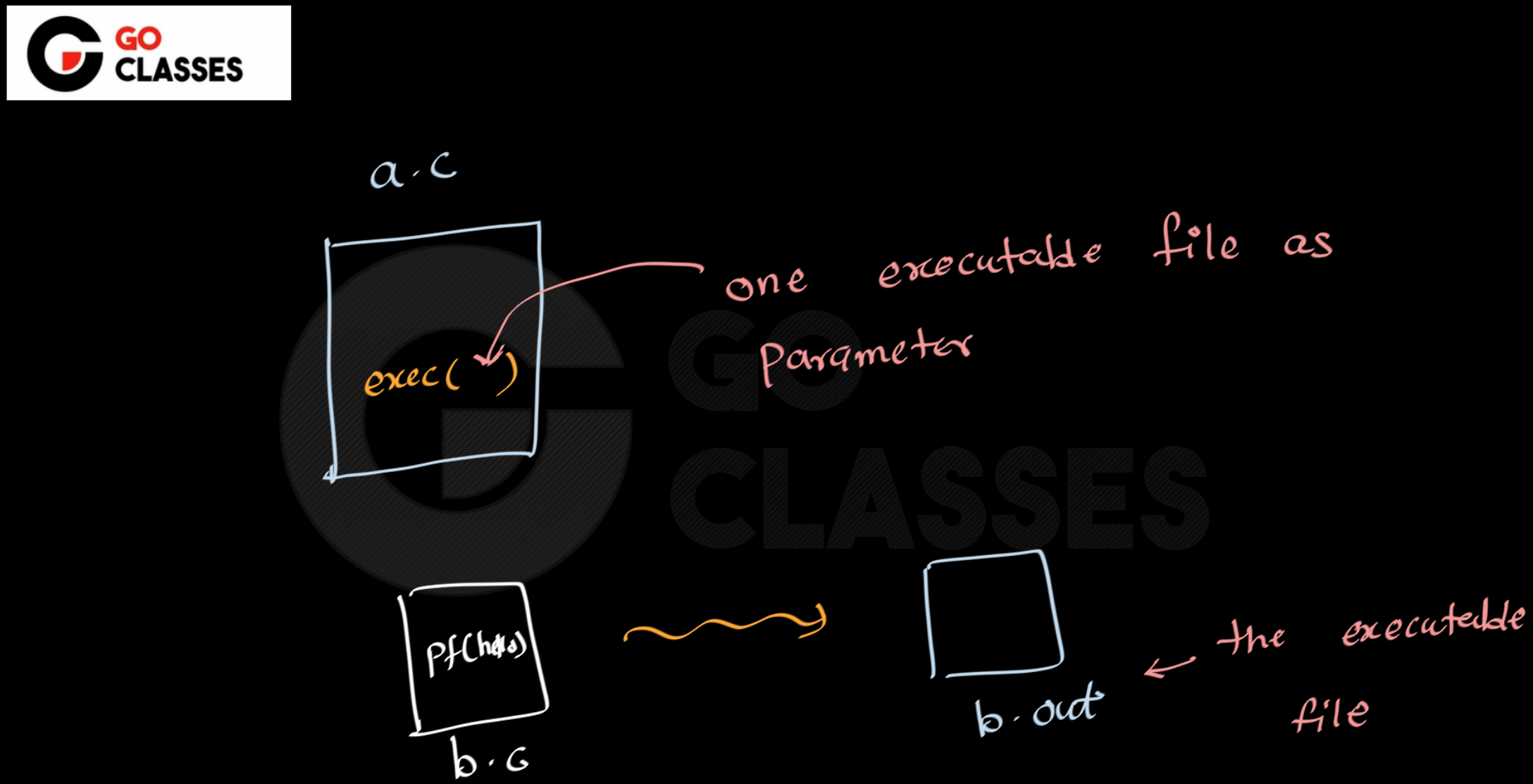
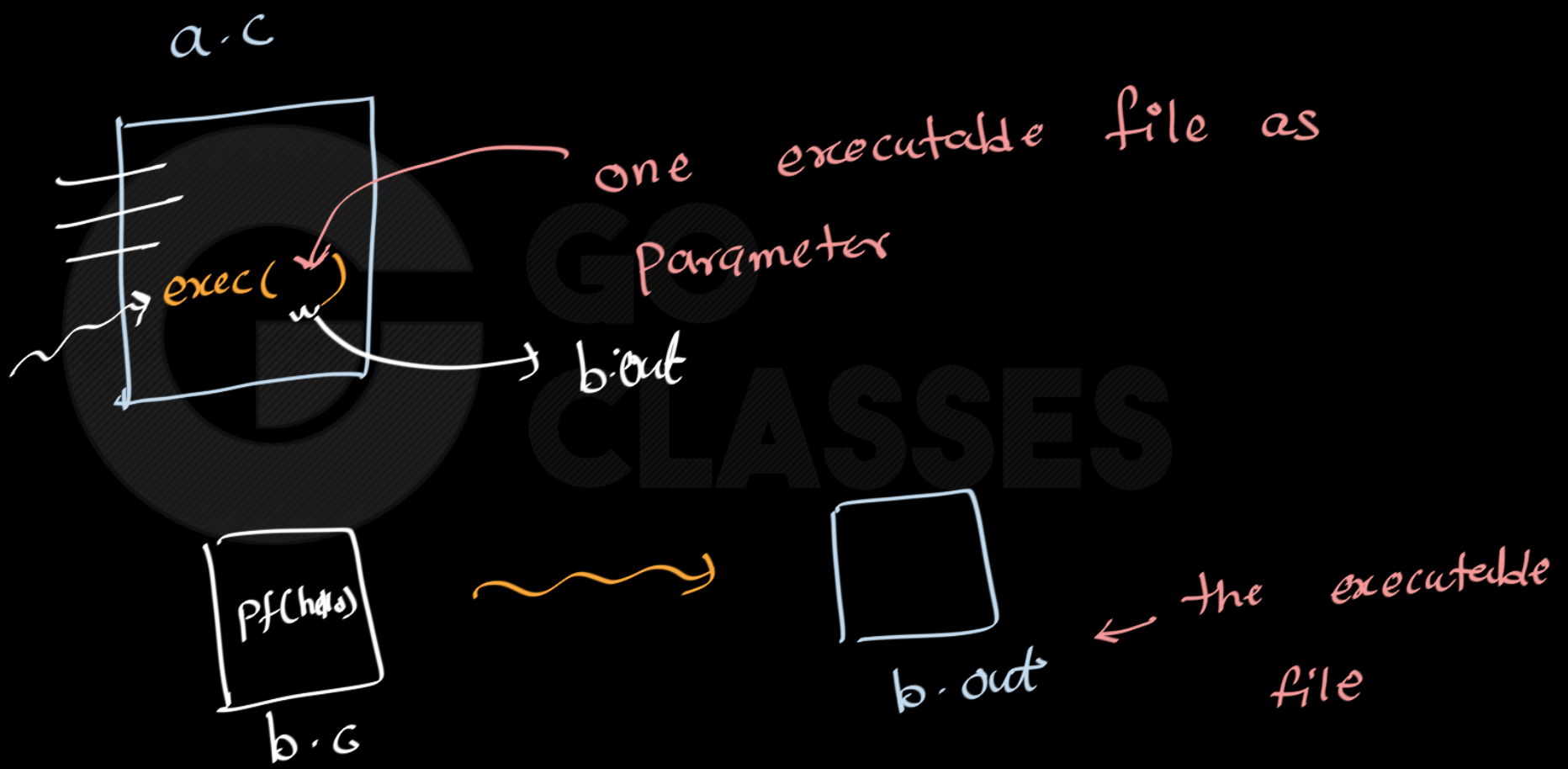
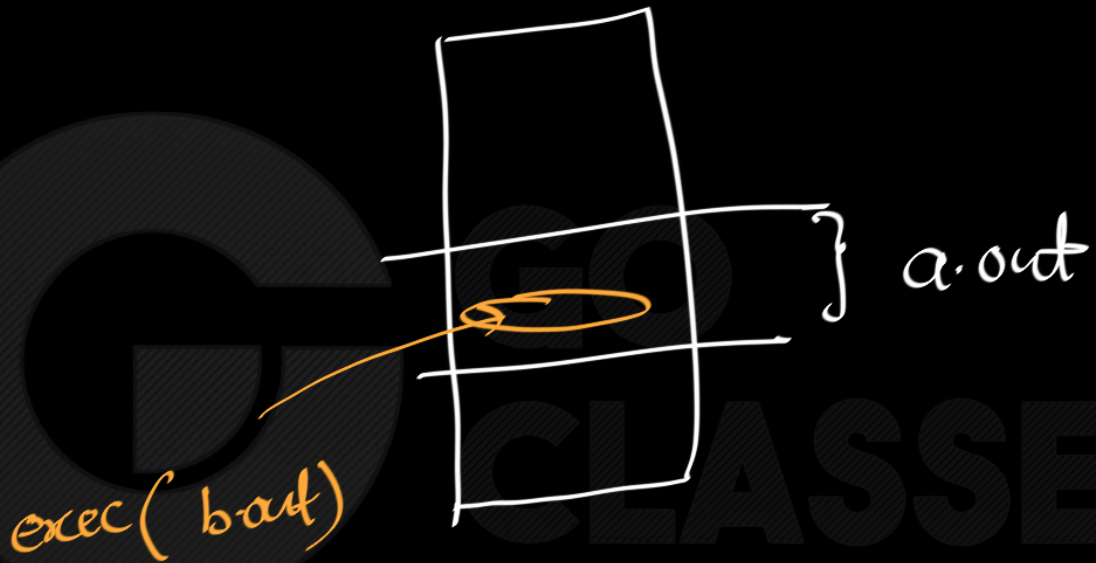
Lecture 4


fork
parent
child

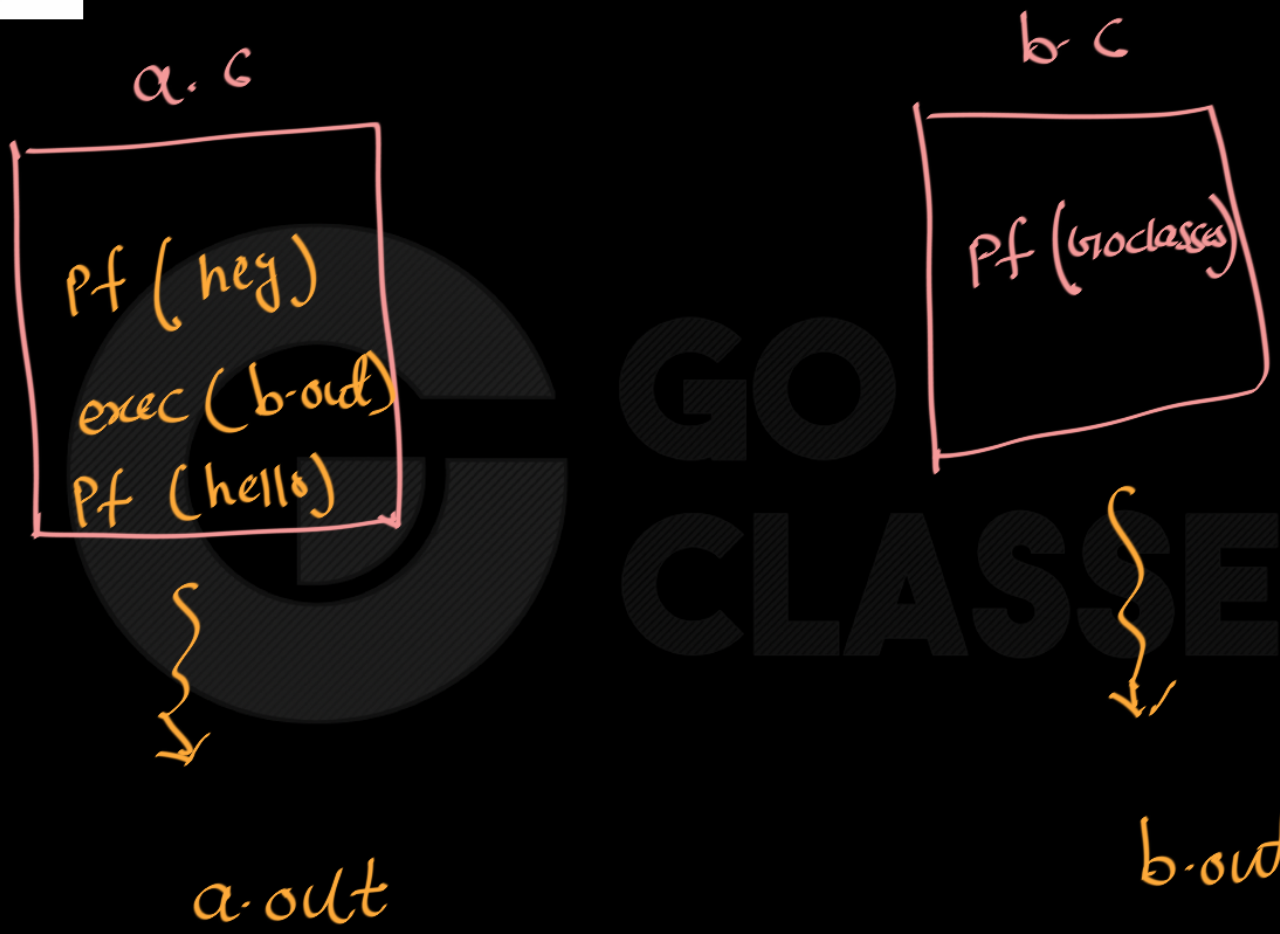
exec System call


a.c
exec( )
one executable file as parameter
Pf(hello)
b.c
b.out
the executable file


a.c
exec( )
one executable file as parameter
b.out
Pf(hello)
b.c
b.out
the executable file



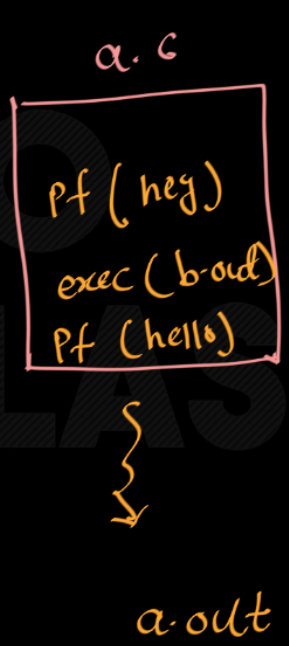
a.out will get replaced with b.out

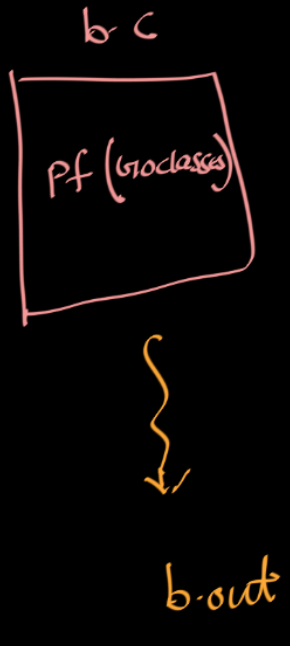

a.c
Pf (hey)
exec (b.out)
Pf (hello)
a.out
b.c
Pf (Goclasses)
b.out

GO CLASSES

question :

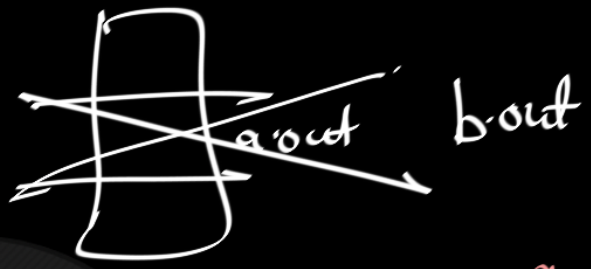
What is the output we get if we run a.out file?





GO CLASSES

question :



What is the output we get if we run a.out file?

a.c

```
Pf (hey)
exec (b.out)
Pf (hello)
```

b.c

```
Pf (GOclasses)
```

a.out

b.out

Answer hey GOclasses

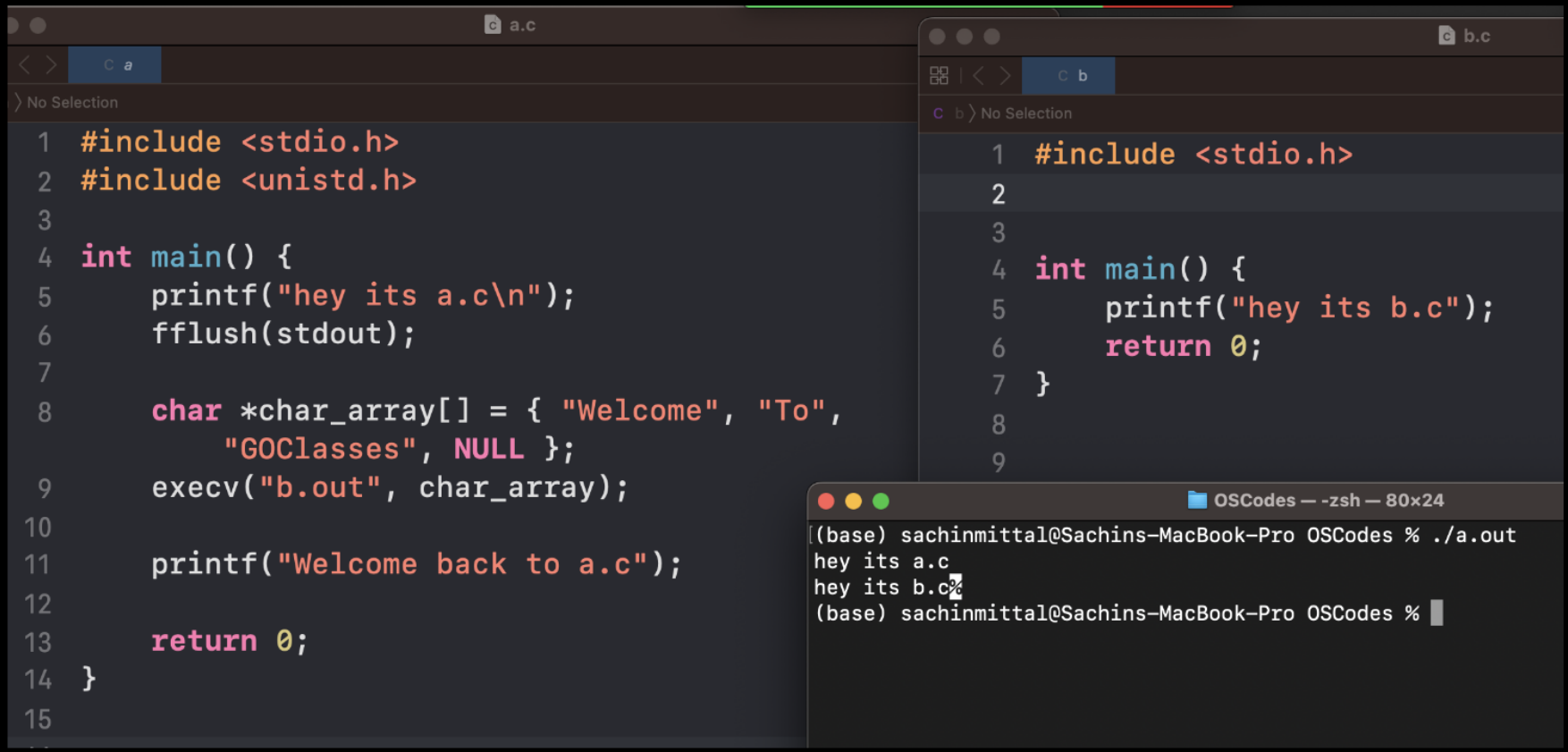
a.c

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    printf("hey its a.c\n");
    fflush(stdout);
    char *char_array[] = { "Welcome", "To", "GOClasses", NULL };
    execv("b.out", char_array);

    printf("Welcome back to a.c");

    return 0;
}
```

fflush(stdout); ← ignore

b.c

```c
int main() {
    printf("hey its b.c");
    return 0;
}
```


a.c
a
No Selection
1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main() {
5 printf("hey its a.c\n");
6 fflush(stdout);
7
8 char *char_array[] = { "Welcome", "To",
"GOClasses", NULL };
9 execv("b.out", char_array);
10
11 printf("Welcome back to a.c");
12
13 return 0;
14 }
15
b.c
b
No Selection
1 #include <stdio.h>
2
3
4 int main() {
5 printf("hey its b.c");
6 return 0;
7 }
8
9
OSCodes — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro OSCodes % ./a.out
hey its a.c
hey its b.c%
(base) sachinmittal@Sachins-MacBook-Pro OSCodes %

b.c

b

No Selection

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
    printf("hey its b.c");
    return 0;
}

```

After Running a.c we get ⇒

```
hey its a.c
hey its b.c
```

Note that we will not get `Welcome back to a.c`

Next Topic:
Process Creation

GO CLASSES

Objective :

understand how does the 2 steps get perform.

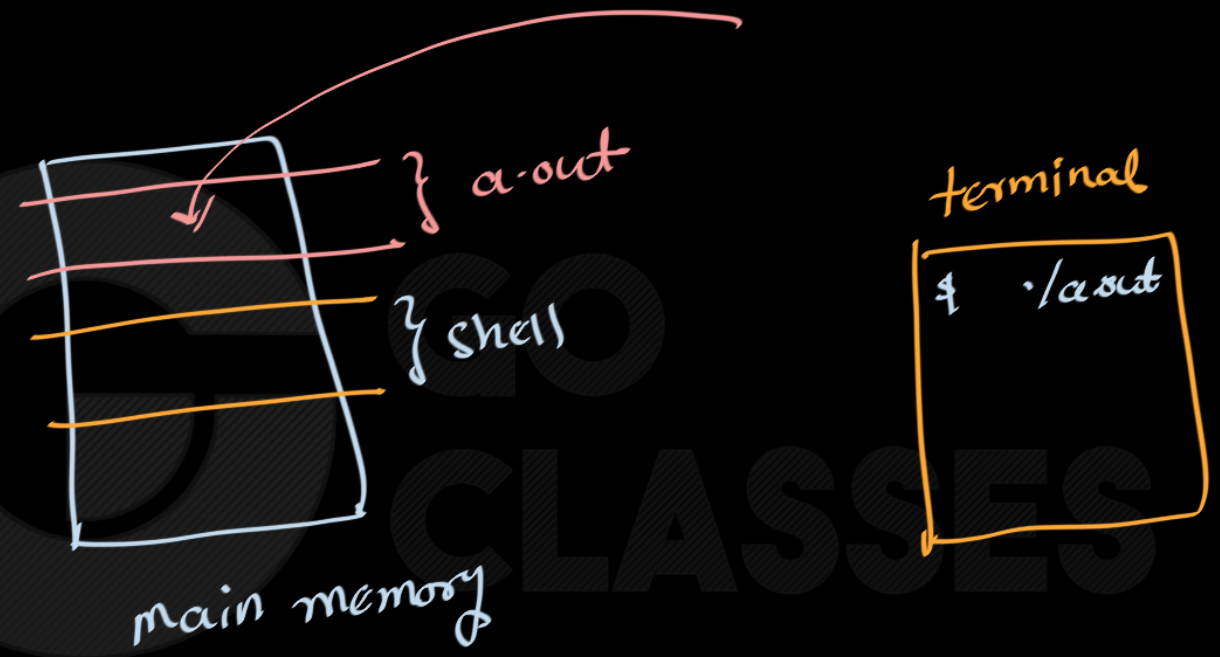
fork-exec

1. Create space for hello.out
2. load hello.out in that space

→ understand fork

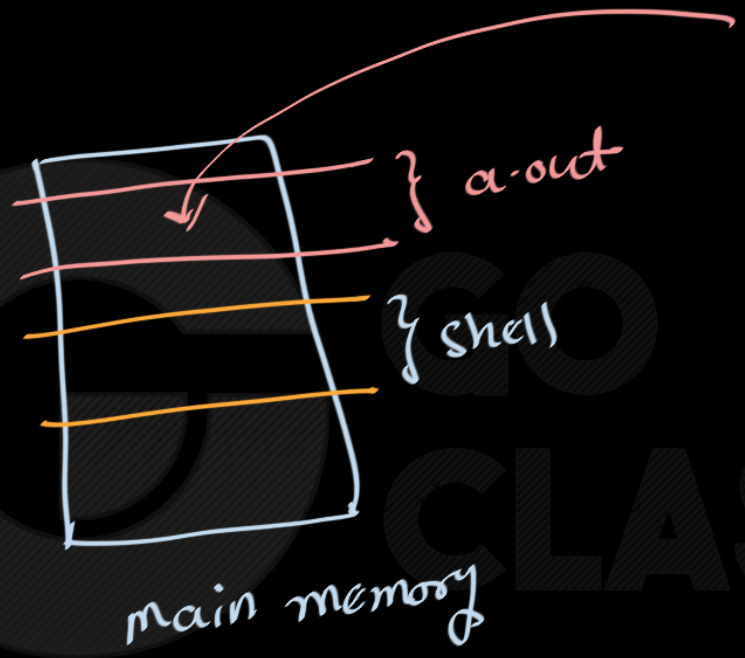
→ understand exec

→ we will come back to process creation.



⇒ create space for a.out

⇒ load the a.out in that space.



terminal

```
$ ./a.out
```

Sangeet Bhowmick to Everyone 8:13 PM

SB

first fork will create a replica of parent process and exec will replace that process

Satyam Naik to Everyone 8:14 PM

SN

fork exec

⇒ create space for a.out

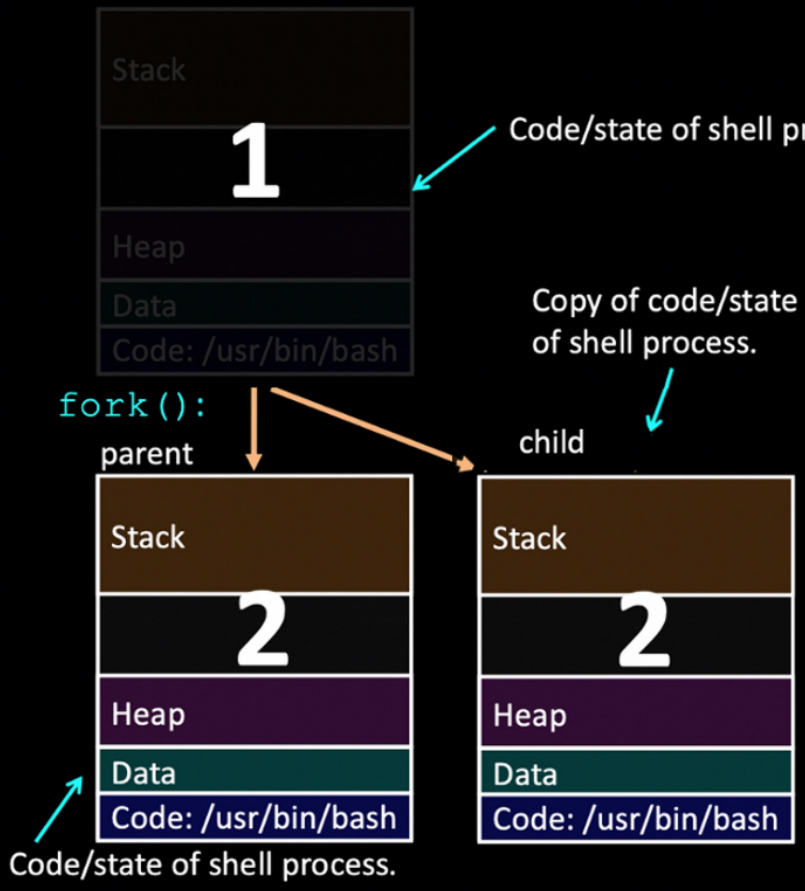
⇒ load the a.out in that space.

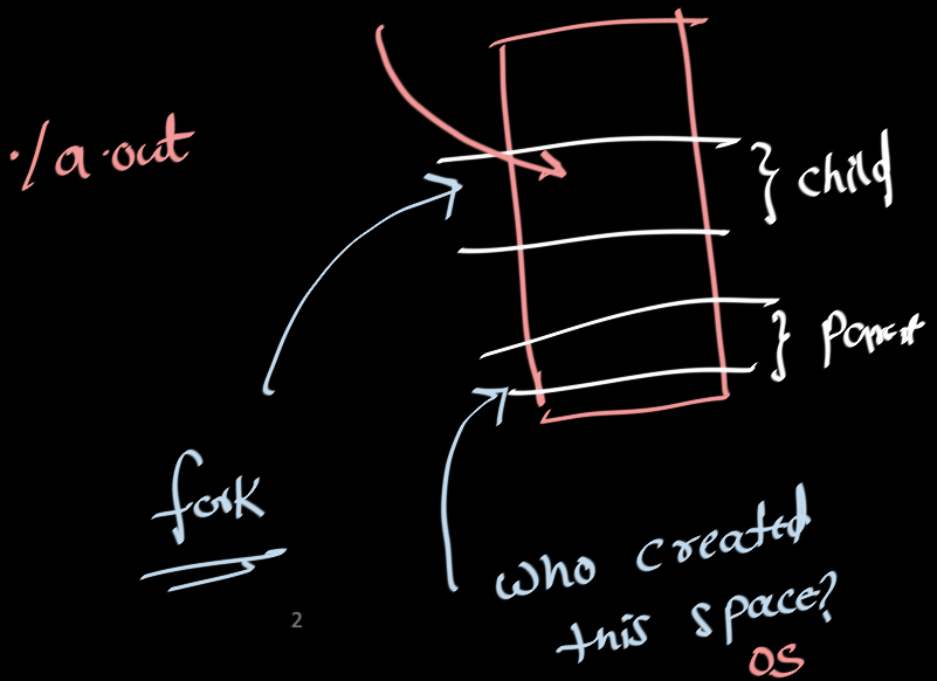
Running the command `ls` in a shell:

Executing a new program

Running the command `ls` in a shell:

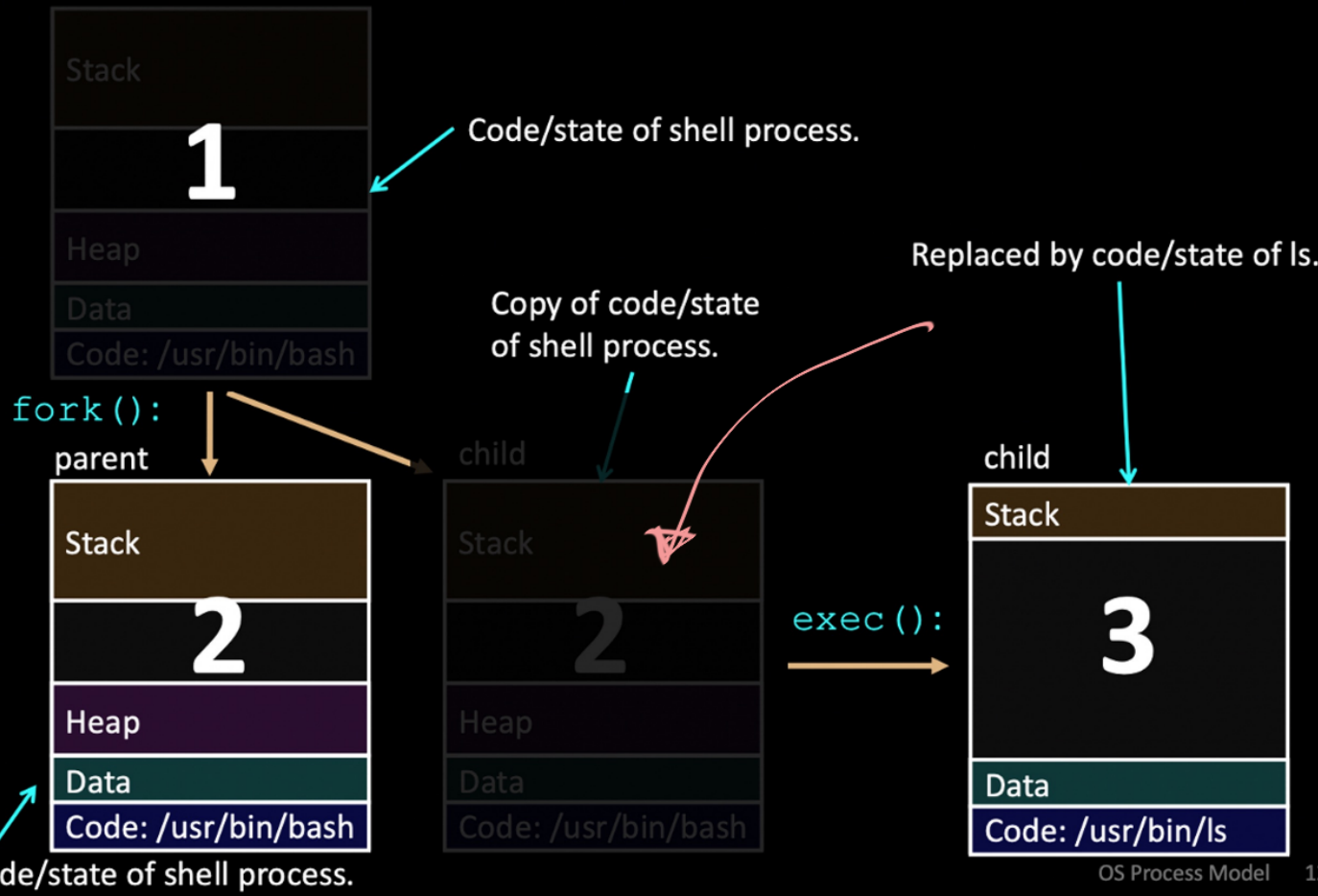
Executing a new program

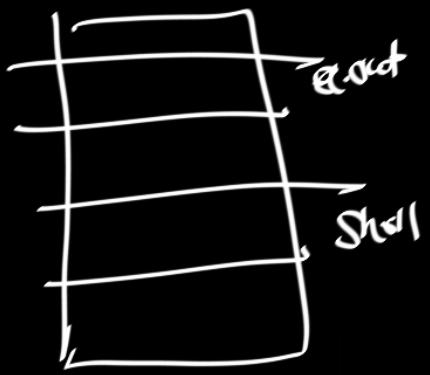




Running the command `ls` in a shell:

Executing a new program





GO CLASSES

Shell code -

(optional)

```
while (true)
{
    scanf ("%s", &t)
    pid = fork()
    if (pid == 0)
        exec (t)
    else    wait() → block state
}
```

some command

a.out

Question

You write a UNIX shell, but instead of calling fork() then exec() to launch a new job, you instead insert a subtle difference: the code first calls exec() and then calls fork() like the following:

```
shell (..) {
   .. ..
   exec (cmd, args);
   fork();
   .. ..
}
```

b.out

Does it work? What is the impact of this change to the shell, if any? (Explain)

means just like earlier

https://www.eecg.toronto.edu/~yuan/teaching/ece344/pastExams/2013midterm_solution.pdf

https://www.cs.princeton.edu/courses/archive/fall13/cos318/exams/midterm-solutions-2013.pdf

Question

You write a UNIX shell, but instead of calling fork() then exec() to launch a new job, you instead insert a subtle difference: the code first calls exec() and then calls fork() like the following:

```
shell (..) {
   .. ..
   exec (cmd, args);
   fork();
   .. ..
}
```

Does it work? What is the impact of this change to the shell, if any? (Explain)

Answer: NO

→ we are destroying the shell

→ we are just replacing shell with bloat

→ fork() will never get executed

https://www.eecg.toronto.edu/~yuan/teaching/ece344/pastExams/2013midterm_solution.pdf

https://www.cs.princeton.edu/courses/archive/fall13/cos318/exams/midterm-solutions-2013.pdf

Question 11

Part (a) [3 MARKS] whatA

```
int main() {
        int i = 0;
        printf("%d ", i);
        i = i + 1;
        if (i >= 2)
                exit(0);
        execv( "./whatA");
}
```

A: 0 1 2
B: 0 0 0
C: 0 0 0 0 0 0 ... (forever)
D: 0 1 2 0 1 2 ... (forever)
E: none of the above

Part (a) [3 MARKS] whatA

```
int main() {
        int i = 0;
        char *argv[2] = { "./whatA", NULL };

        printf("%d ", i);
        i = i + 1;
        if (i >= 2)
                exit(0);
        execv(argv[0], argv);
}
```

A: 0 1 2
B: 0 0 0
C: 0 0 0 0 0 0 ... (forever)
D: 0 1 2 0 1 2 ... (forever)
E: none of the above

GO CLASSES

Meeting Chat

Akash Maji to Everyone 8:33 PM

AM

does that mean a new process is running with same PID ? and same privileges?

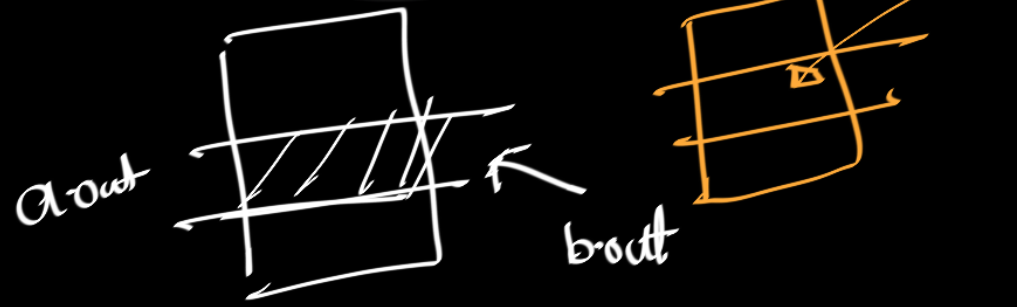
Meeting Chat

Jahnavi P to Everyone 8:38 PM

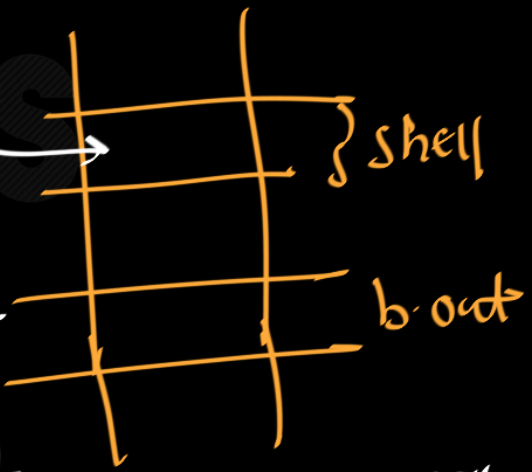
JP

So Shell doen't discoutinue when the file is replaced?

Yes PID is Same



block state → then it will not even get scheduled unless bout finish



main memory

System Calls

- What is System call ?
- Why they are always in limelight?

User Trust Issue

- Most of the users are not experts in computers and do not understand how hardware works. Thus they may not know how to properly manage the resources.
- An user (expert or not) can make incorrect operations while managing the resources (like handling h/w)
- An user can be malicious i.e. he/she can deliberately make incorrect operations. This can lead to system failure.

Solution from OS side: Never trust a user while dealing with critical tasks

```
int a, b, c = 0
a = b+c
```

Normal task: $a = b + c;$ ← Can we done without permission of OS

Privileged task: asking for the hardware access

GO CLASSES

User wants to do both.

Normal task: `a = b+c;` ← Can we done without permission of OS

Privileged task: asking for the hardware access

GO CLASSES

User wants to do both.

usermode

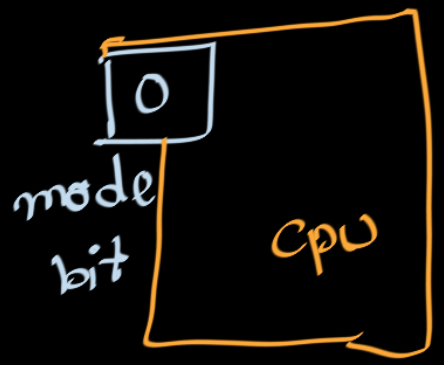
mode bit = 1

Normal task: `a = b+c;` ← Can we done without permission of OS

kernel mode or Privileged mode

Privillaged task: asking for the hardware access

mode bit = 0

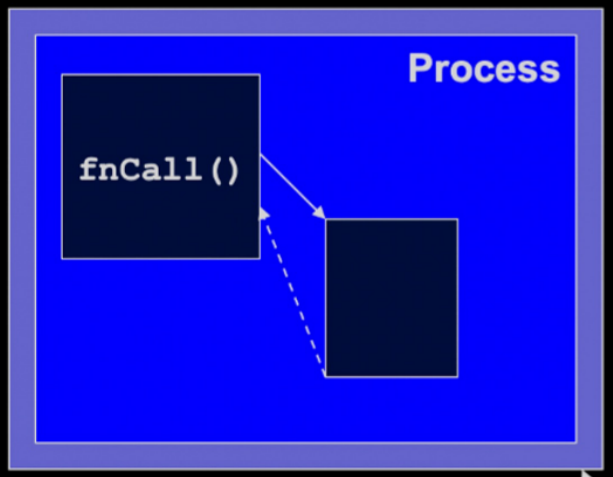


How user ask for OS Services ?

- Using System call ← entry points to the kernel

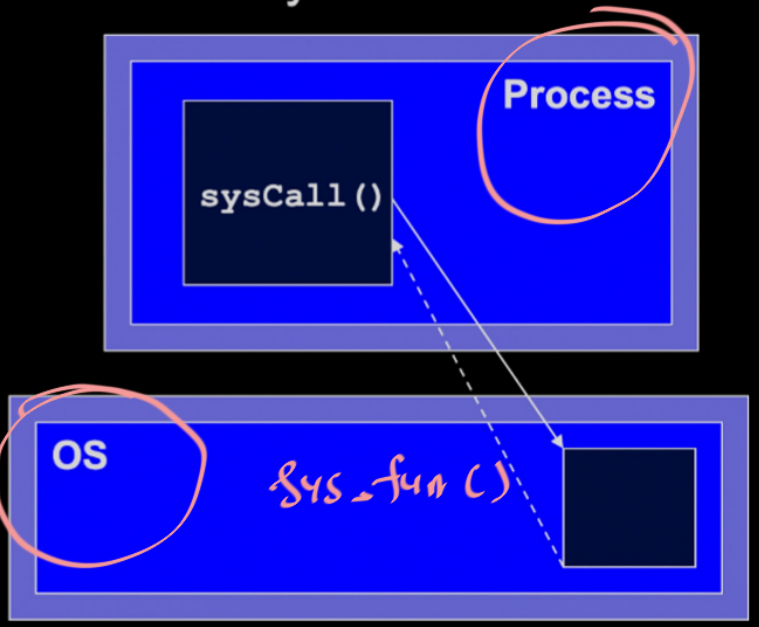
System Calls versus Function Calls

Function Call

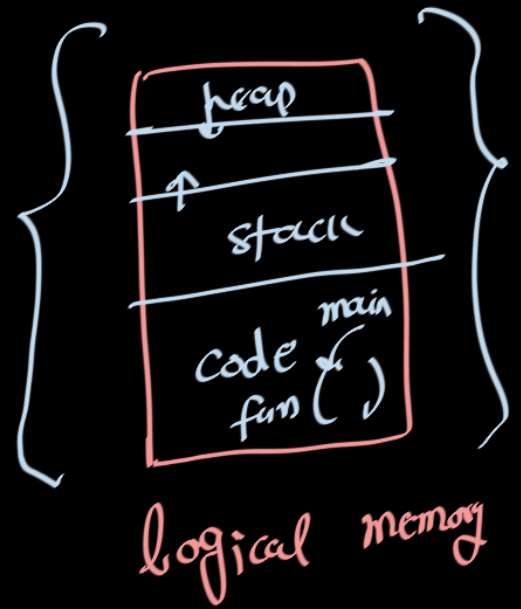


Caller and callee are in the same Process
- Same user
- Same "domain of trust"

System Call

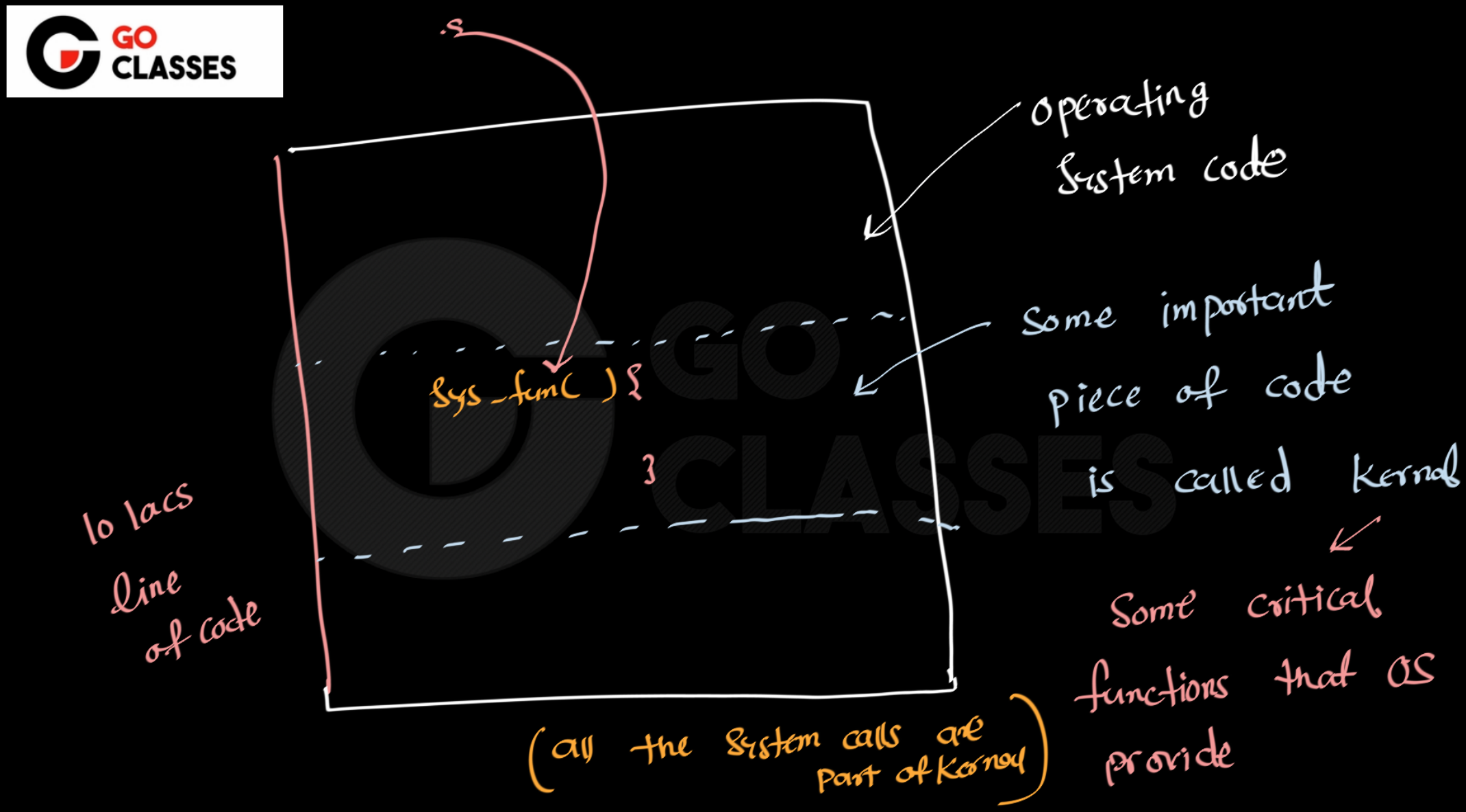


- OS is trusted; user is not.
- OS has super-privileges; user does not
- Must take measures to prevent abuse

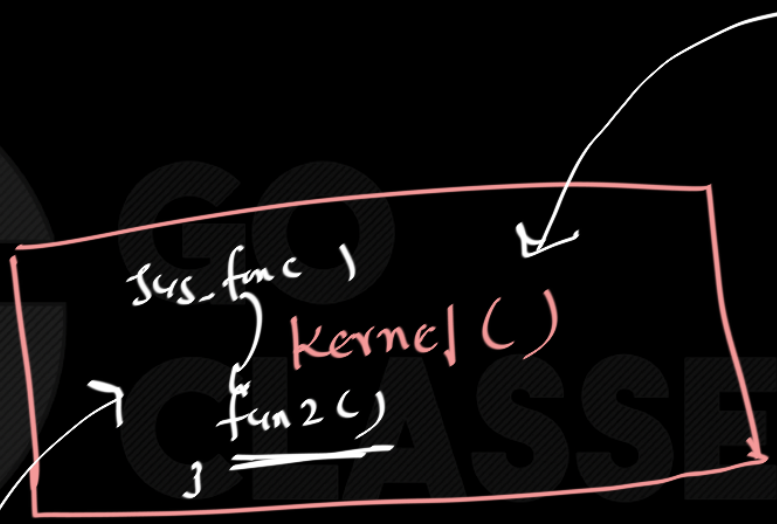


logical memory

Address space


GO CLASSES
Operating System code
Some important piece of code is called kernal
Some critical functions that OS provide
sys_fun( ){
}
10 lacs line of code
(all the System calls are Part of Kernal)

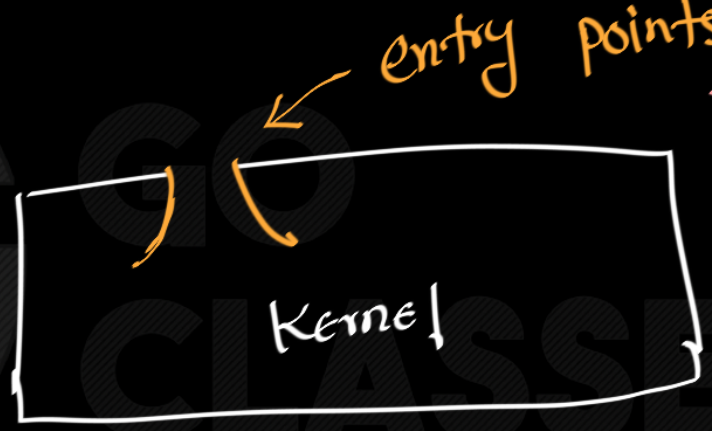
user code



1000 functions

400 are System Calls

600 are Not System Call



System calls

- interrupts
- exceptions

Dual Mode of Operations

- User Mode
  - Compute operations like addition, subtraction, etc.
  - Reading and writing into program's memory
- Kernel Mode
  - CPU and memory management
  - I/O handling

a = b + c

a = 10;

user is not supposed to do directly

How to ensure that user does not perform some thing illegal? Before running each instruction check whether user is allowed to run it or not. This check can be done efficiently in hardware.

1 ← usermode

Processor Modes

- The OS must restrict what a user process can do
  - What instructions can execute
  - What portions of the address space are accessible
- Supervisor mode (or kernel mode)
  - Can execute any instructions in the instruction set
    - Including halting the processor, changing mode bit, initiating I/O
  - Can access any memory location in the system
    - Including code and data in the OS address space
- User mode
  - Restricted capabilities
    - Cannot execute privileged instructions
    - Cannot directly reference code or data in OS address space
  - Any such attempt results in a fatal “protection fault”
    - Instead, access OS code and data indirectly via system calls

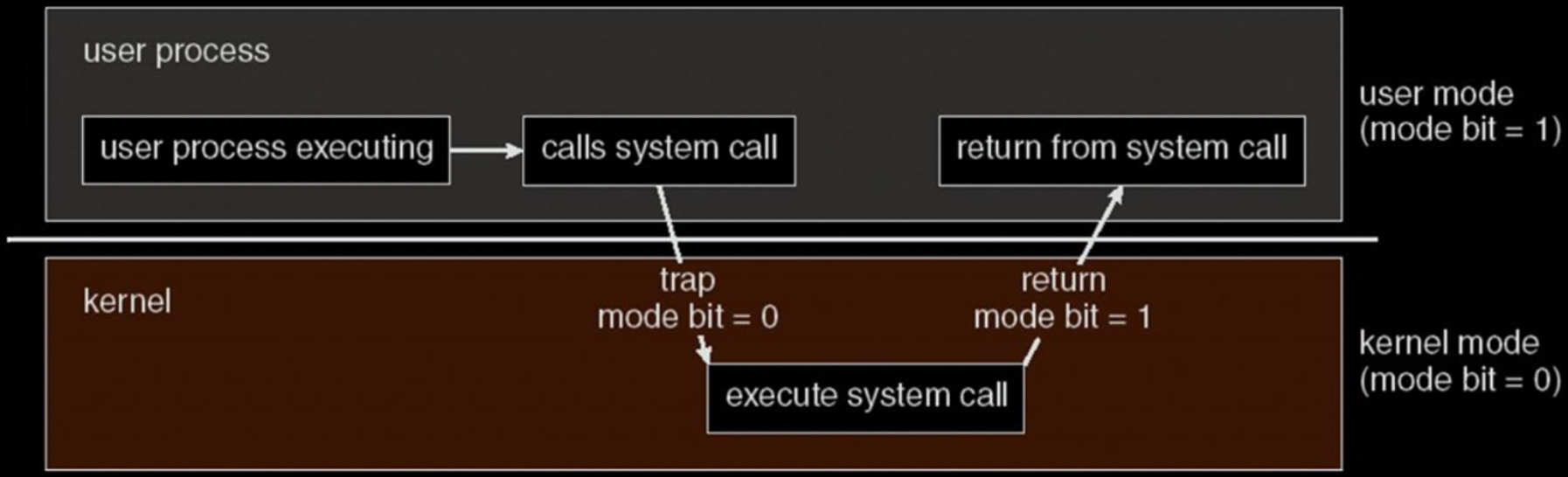
Mode Bit and Privileged Instructions

- CPU maintains a special mode bit. If mode = 0 then system is in kernel/privileged mode; otherwise in user mode
- Instructions that are considered critical will run only when mode = 0. For example, changing the value of program counter (PC).

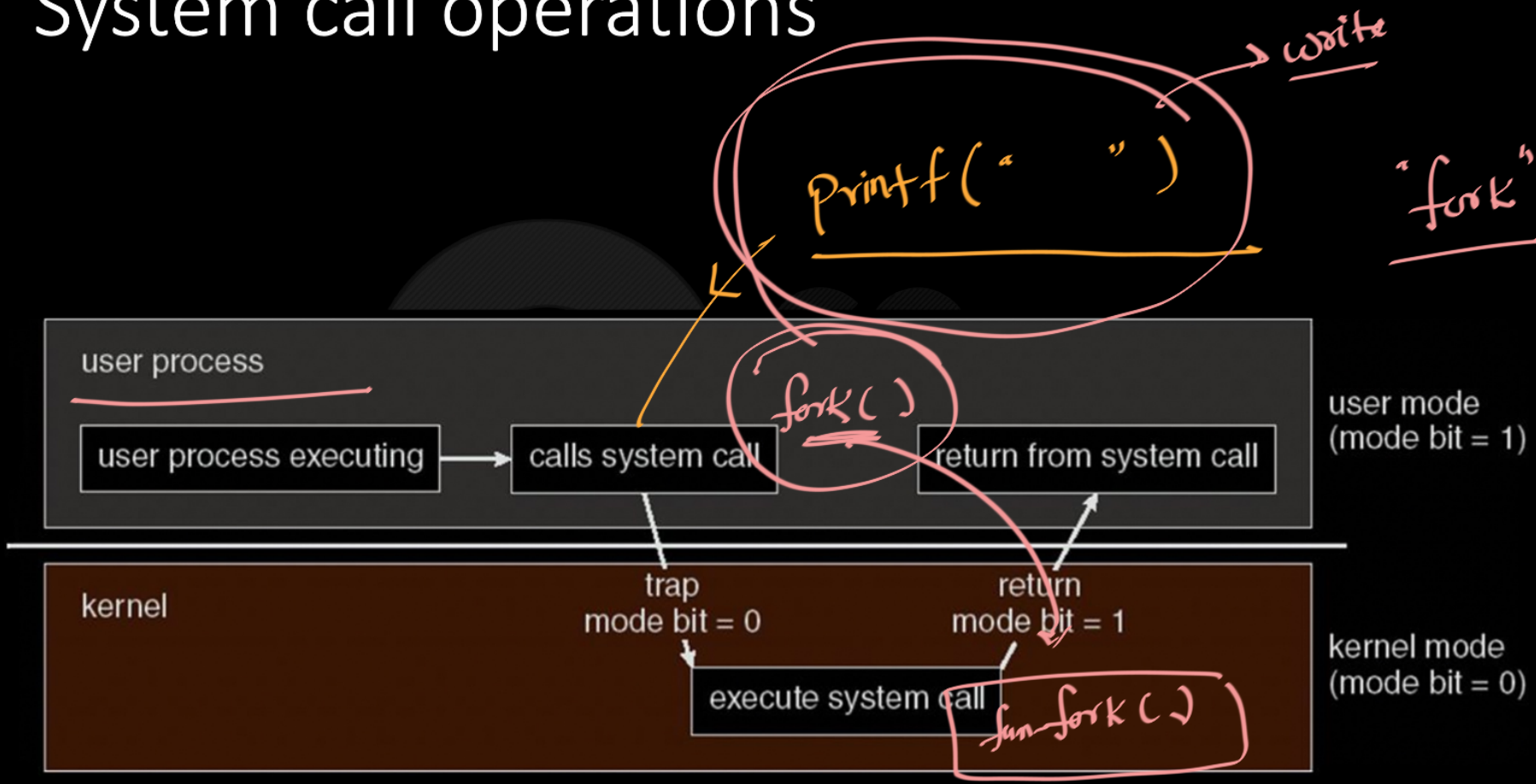
system calls

- Note that we have blocked user (user program) from doing certain tasks to protect the illegal use of the system.
- Instead, user can will now request OS to perform these tasks. These tasks will run privileged instructions → in OS code
- User program can initiate such request using a mechanism called system calls.
- Note that we need to change the mode here

System call operations



System call operations

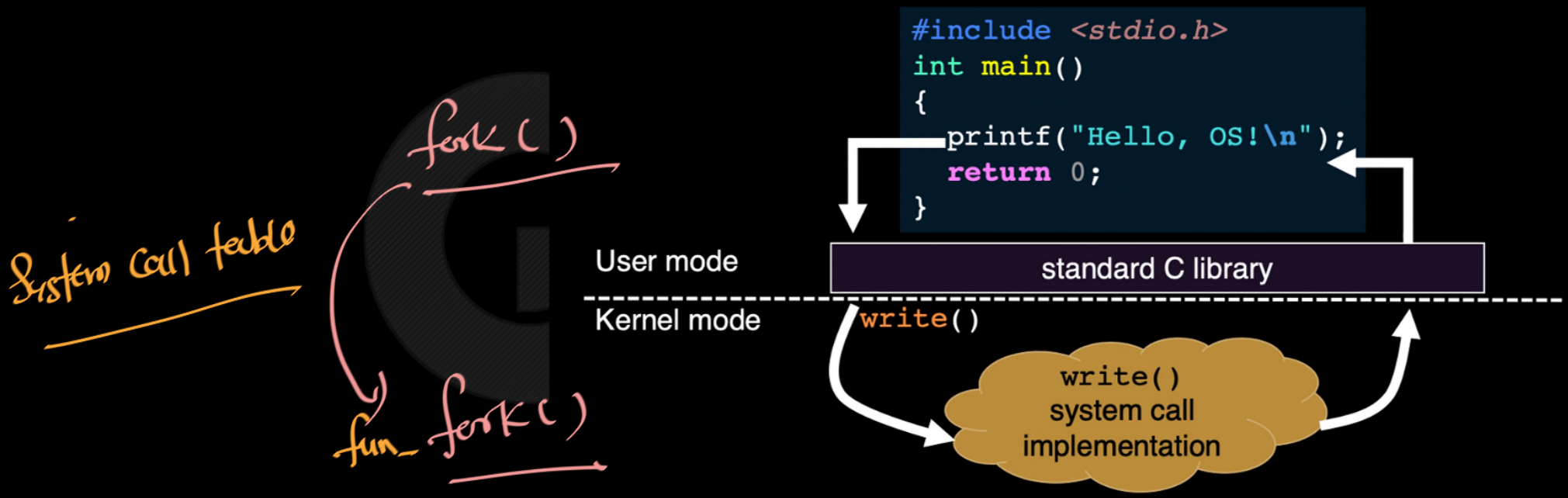


Invoking System Calls

A user process cannot invoke system calls like regular functions because they are part of the kernel-space. Modern CPU provide a special instruction generally called a *software interrupt* , which can be performed by a user-space software and which causes the CPU to switch to the system mode and give the control to the kernel, just like when serving a hardware interrupt or an exception.

https://courses.cs.washington.edu/courses/cse333/20su/lectures/lec-07/07-posix-20su.pdf

System call Interface



https://www.cs.jhu.edu/~huang/cs318/fall20/lectures/lec1_intro.pdf

GO CLASSES

As a user can i ask directly from monitor ?

No

if i directly use monitor without asking then it is a problem

Application Programming Interface

(API)

User processes usually do not directly invoke system calls. Instead they use something which is called an *Application Programming Interface* (an API, for short). It is a set of functions, variables and other software constructs that allow the processes to perform the most needed operations and also to cooperate with the operating system.

→ library

Application Programming Interface

User processes usually do not directly invoke system calls. Instead they use something which is called an *Application Programming Interface* (an API, for short). It is a set of functions, variables and other software constructs that allow the processes to perform the most needed operations and also to cooperate with the operating system.

Optinal read

Linux has an API defined by the POSIX and SUS standards, which is inherited from the Unix system. Some of the functions from this library do not use any system calls (like the `strcpy()` function),

some of them are simple wrappings for a system call (like the `write()` function) and some of them perform other operations beside invoking a system call (like the `printf()` function).

printf()
↓
System call + Do some other thing

4. System calls must be used to:

   (a) modify global variables

   (b) call a user-written function

   (c) write to a file

   (d) All of the above

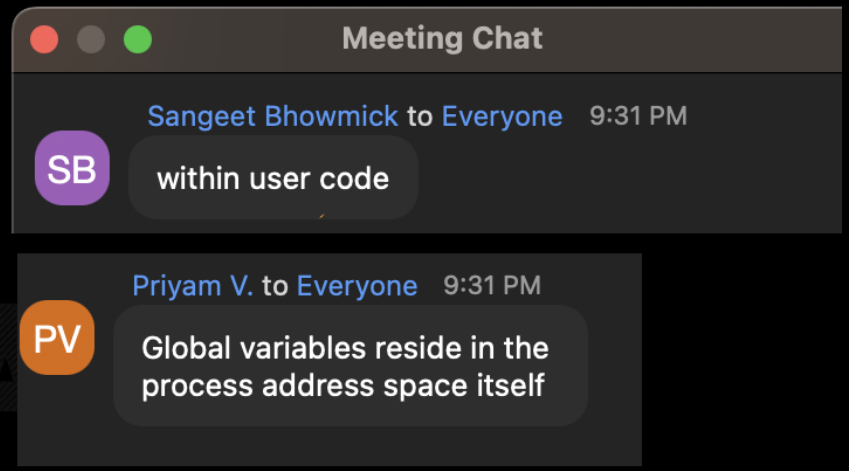
4. System calls must be used to:

(a) modify global variables

(b) call a user-written function

(c) write to a file

(d) All of the above



Answer C

GO CLASSES

Next topic :

System Call Execution

System Call Execution

(or System call implementation)

(interrupt execution in CoA)