System Call

System call is a way to shift from user mode to kernel mode, and acces operating System.

☆ Kernel in the Sense Operating System. The main part or heart of O.S.

☆ System call is a way for a user program to interface with the O.S.

- File Related =>

Open(), Read(), write(), close(), Create file e.t.c.

- Device Related =>

Read, write, Reposition, ioctl, fcntl

- Information =>

get Pid, attributes, get System time and data.

january '21

Su	Mo	Tu	We	Th	Fr	Sa
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

NOTES

- Process control =>

Load, Execute, abort, Fork, Signal, wait, Allocate e.t.c.

- Communication =>

Pipe(), Create/delete Connections, &

Fork()

☆ Fork() is a System Call.

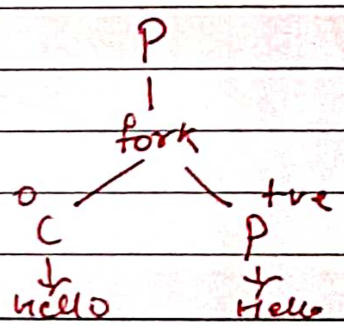
☆ It is used to create child Process.

☆ fork() generally returns value:

0 -> Child
+1 -> Parent
-1 -> No child created

Ex:-
```
Main(){
    fork();
    Printf("Hello");
}
```
Print 2 times Hello, by child & Parent Concurrently

NOTES



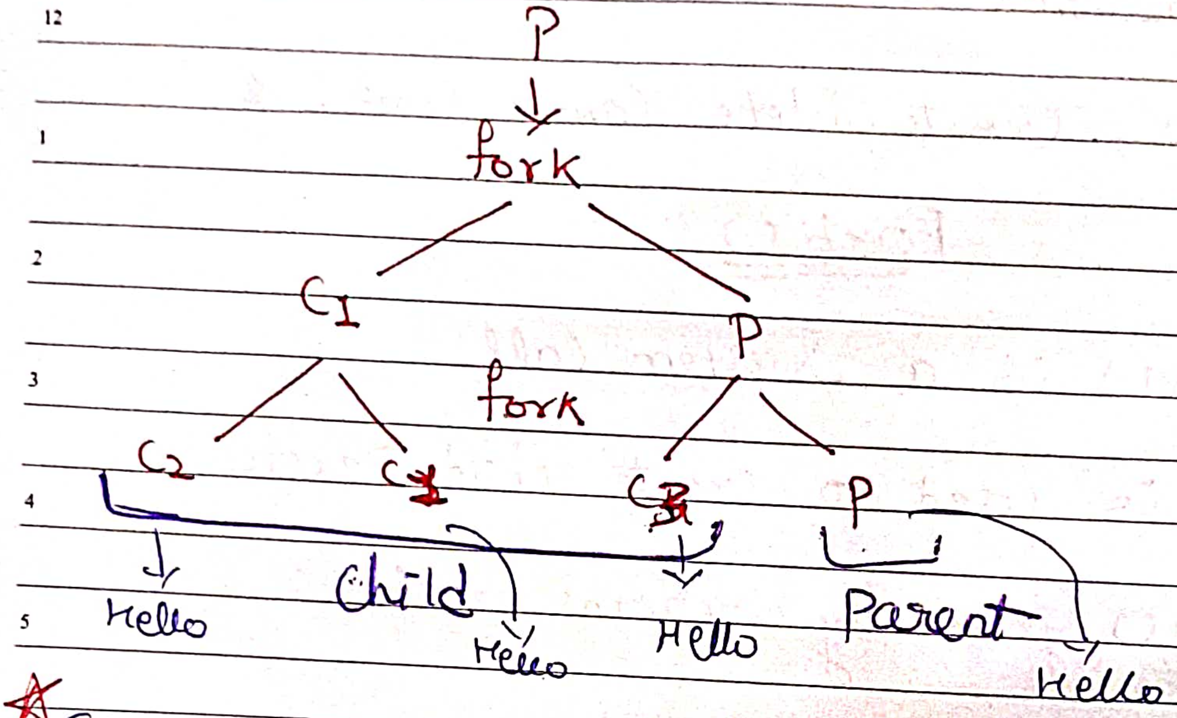
february '21

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28						

Dreams are illustration from the book your soul is writing about you. - *Marsha Norman*

Ex3:-

```
Main() {
    fork();
    fork();
    Printf ("Hello");
}
```



Formula :- 2^n

Child Procees :- $2^n - 1$

Fork() System Call is used for Parallel processing.

january '21

Su	Mo	Tu	We	Th	Fr	Sa
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

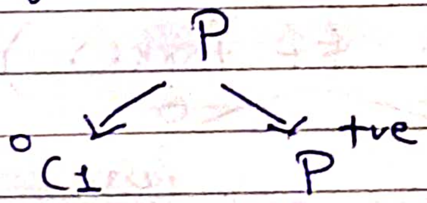
NOTES

You can't cross the sea merely by standing and staring at the water. - *Rabindaranath Tagore*

Questions on fork() System call

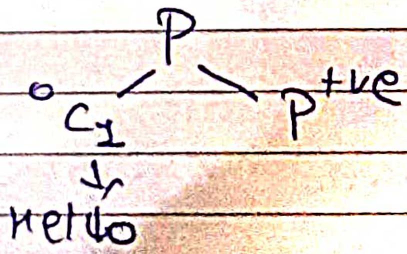
Q)
```
# include <Stdio.h>
# include<Unistd.h>
   int main()
 {
           0
     if (fork() && fork())
             +ve
        fork();
        Printf ("Hello");
        return 0;
   }
```

- Reads 1st fork() of if statement and Parent program's child is created.



- && is operator AND, Now the child process returns 0 So 0 && 0/1 returns 0. i.e it returns false. Therefore the inner fork() Statement will not invoke and print the Hello Statement.

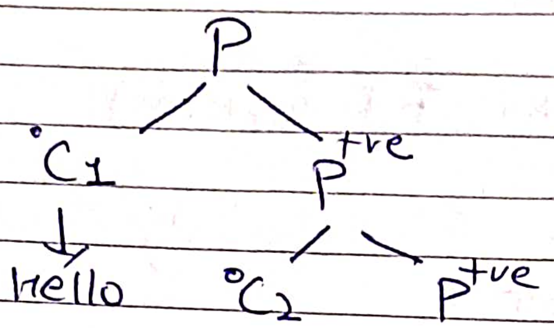
NOTES



February '21

Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28						

- But the parent 'p' returns +ve means 1 && 1 = 1 ∴ it invokes next fork() so parent will create its own child process.



But child process has return 0. Now ~~&&~~ parent's +ve and child's 0 returns false.

i.e. if (fork() && fork())

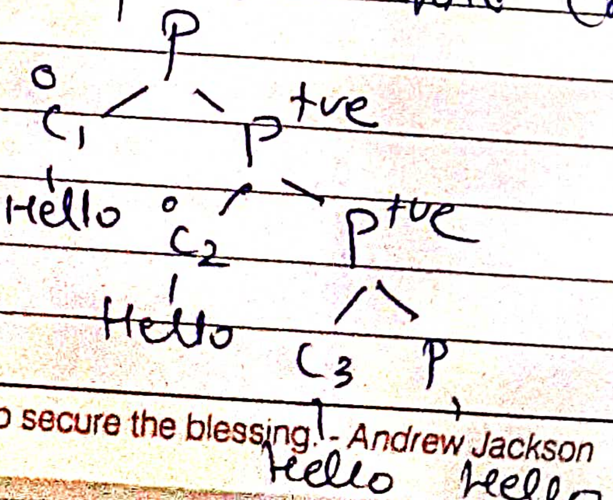
0 above the first fork(), with an arrow to the second fork()

+ve — 0

false.

Hence skipss fork() inside if. directly print Hello.

But parent's return +ve i.e +ve && +ve returns +ve true hence invoke fook() inside. So parent's child created.

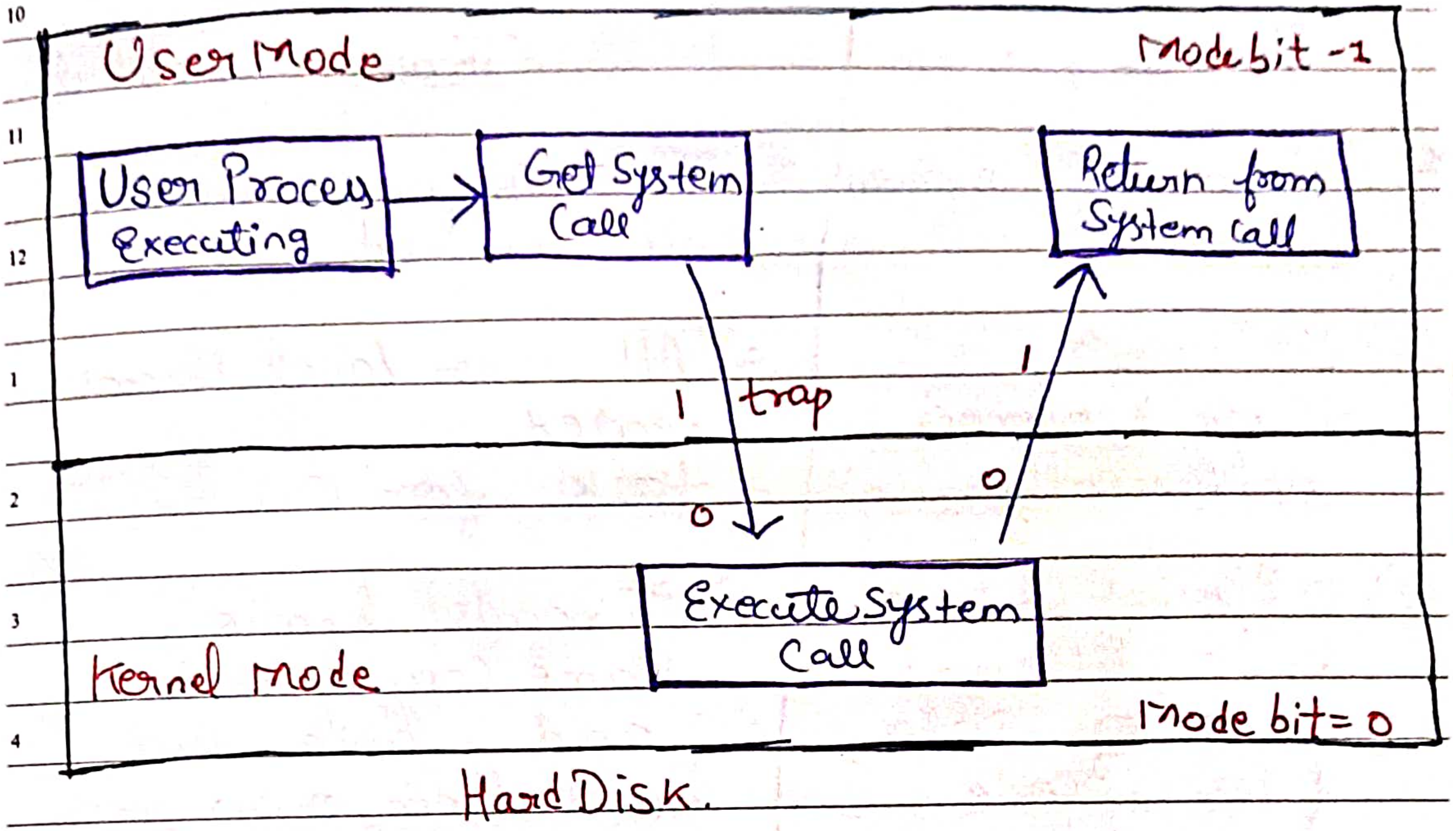


january '21

Su	Mo	Tu	We	Th	Fr	Sa
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

NOTES

★★ User mode Vs Kernel Mode



- ★ Switching of bits called trap.
- ★ Every application or process runs on User mode(
- ★ User can access hardware through Kernel Mode using System Call.

NOTES

- ★ CPU keeps switching between User mode and kernel mode.

FEBRUARY
MARCH
APRIL

Process and Threads in Operating System

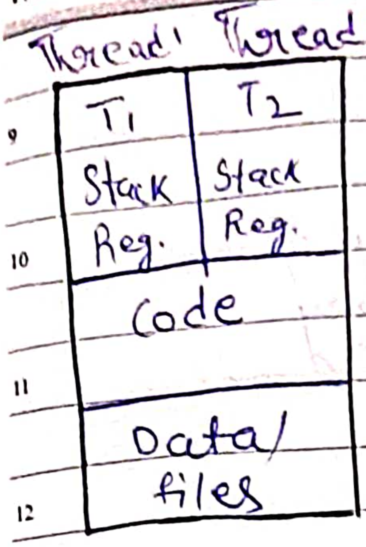
Process	Threads (User level)
1) System call involved in a process.	1) There is no system call involved.
(2) OS treats different process differently.	2) All user level threads treated as single task for O.S.
(3) Different process have different copies of Data, files, code.	3) Threads share same copy of code and data. But have their own stack and registers.
(4) Context switching is slower.	4) Context switching is faster.
(5) Blocking a process will not block the another process.	5) Blocking a thread will block entire process.
(6) Independent	(6) Interdependent

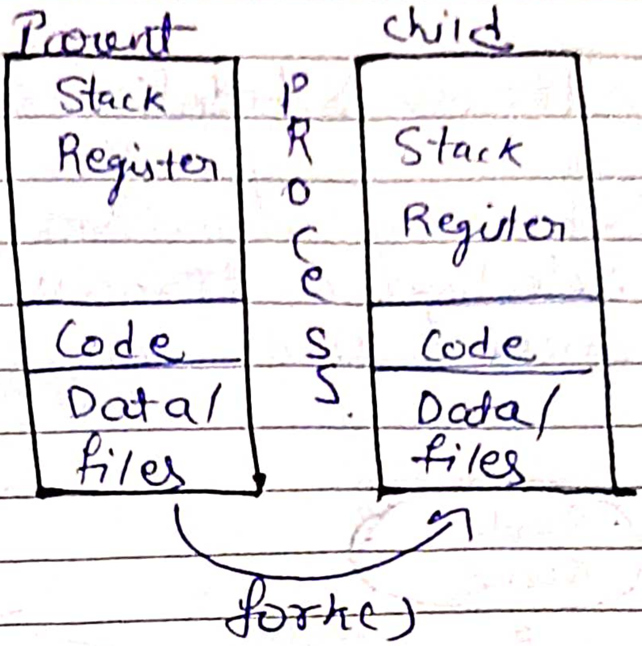
january '21

Su	Mo	Tu	We	Th	Fr	Sa
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

NOTES

Writing books is the closest men ever come to childbearing. - *Norman Mailer*





There are two levels of threads:

User level Threads	Kernel level Thread
1) User level threads are managed by the user level library.	(1) kernel level threads are managed by O.S.
2) User level threads are typically fast.	(2) kernel level threads are slower than user level.
3) Context switching is faster.	(3) Context switching is Slower.
4) If one user level threads perform blocking operation than entire Process get blocked	4) If one kernel level thread blocked, No affect on others.