(*) RETURN VALUE OF FORK() :-

⇒ on sucess, the PID of the child process is returned in the parent and 0 is returend in the child.

⇒ on failure -1 is returned in the parent, no child process is created and errno is set appropriately.

(1) Program :-

```
#include <stdio.h>
main()
{
    int ret;
    Printf (" Hello ---\n");
    ret = fork();
    Printf (" ret = %d\n", ret);
    while (1);
}
```

output :-

```
Hello ---
ret = 1686   (Parent)
ret = 0      (child)
```

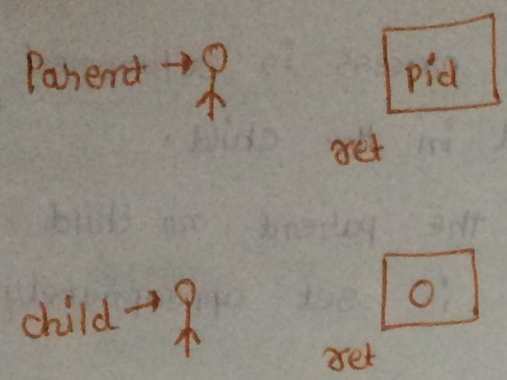
(2) Program :-

```
#include <stdio.h>
main()
{
    int ret;
    Printf (" Hello ---\n");
    ret = 400;
    ret = fork();
    if (ret==0)  // child process
    {
        Printf (" In child ret = %d\n", ret);
    }
    else // Parent process
    {
        Printf (" In parent ret = %d\n", ret);
    }
    common code
    while(1);
}
```

exclusive child code

exclusive Parent Code

child and parent boths are executes.



Note If we write command code between if and else then it gives syntical error

③ **Program :-**

```
#include <stdio.h>
main()
{
    int ret;
    Printf ("Hello---\n");
    ret = 400;
    ret = fork();
    if (ret==0)
    {
        Printf ("In child---\n");
    }
    else
    {
        Printf ("In Parent---\n");
    }
}
```

Output

Hello---
In parent-
In child

④ **Program :-**

```
#include <stdio.h>
main()
{
    int ret;
    Printf ("Hello---\n");
    ret = 400;
    ret = fork();
```

Output

Hello---
In child
In parent

```
if (ret==0)
{
    Printf(" In Child ----\n");
}
else
{
    Printf(" In parent\n"); ②
    Sleep(1); ①
}
}
```

⑤ Program :-

```
#include <stdio.h>
main()
{
    int ret;
    Printf(" Hello ---\n");
    ret=400;
    ret = fork();
    if (ret==0)
    {
        Printf(" In child \n");
        sleep(10);
        Printf(" In child after sleep\n");
    }
    else
    {
        printf(" In Parent \n");
        sleep(a);
    }
}
```

output :-

```
hello ---
In parent
In child
Vector@ubuntu :~ /vi5be3 /linux/
pm  In child after
    sleep
```

⊛ If want to execute the command in concurrently to whole the system function in fork like

ex:- 3.a

ex:- 3

```
{
 if (fork() ==0)
 {
    system ("pwd");
 }
 else
 {
   system ("ls");
 }
 while(1);
}
```

ex:- 8.a

```
if (fork() ==0)
{
   system ("Pwd");
}
else
{
   system ("ls");
}
```

ex:-

```
if (fork() ==0)
{
    Printf (" In child \n");
    system (" .|P1");
}
else
{
    Printf (" In Parent \n");
    system (" ls");
}
Printf (" Hai---- \n");
}
```

output :-

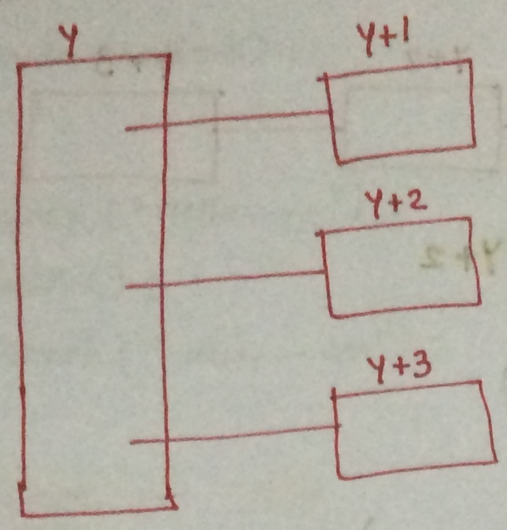
In parent
In child
ls command execute
Hai ---
after command process execute.

→ Here one "Hai" statement is printed, because child is not printing the "hai" due to the system(" .|P1") is in while loop, so system call block the child process.

Program :-

```
#include <stdio.h>
main()
{
    if (fork() ==0)
    {
        system (" pwd");
    }
    else
    {
        system (" ls");
    }
}
```

Example -1



Print ppid then get Y

```
if (fork()==0)
{ // Y+1
}
else
{ // Y
    if (fork()==0)
    { // Y+2
    }
    else
    { // Y
        if (fork()==0)
        { // Y+3
        }
        else
        { // Y
        }
    }
}
```

① execlusive child 1 code.

② execlusive Parent code.

③ execlusive child 2 code.

④ execlusive Parent code.

⑤ execlusive child 3 code.

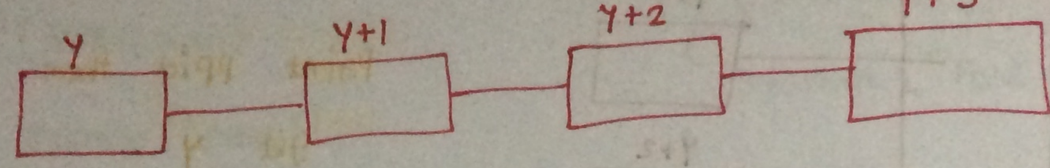
⑥ execlusive Parent code.

Y, Y+3

Y, Y+2, Y+3

Y, Y+1, Y+2, Y+3

Example-2



PPID of Y+3 is = Y+2

```
if (fork()==0)
{
    // Y+1
    if (fork()==0)
    {
        // Y+2
        if (fork()==0)
        {
            // Y+3
        }
        else
        {
            // Y+2
        }
        ≡ → Y+2, Y+3
    }
    else
    {
        // Y+1
    }
    ≡ → Y+1, Y+2, Y+3
}
else
{
    // Y
}
≡ → Y, Y+1, Y+2, Y+3
```

① Program :-

```
#include <stdio.h>
main()
{
    Printf ("Hello----");
    fork();
    Printf (" Hai----\n");
}
```

Output :-

Hello --- Hai ---

Hello --- Hai ---

② Program :-

```
#include <stdio.h>
main()
{
    Printf ("Hello---");
    while(1);
}
```

Output :-

Hello --- is not display.

③ Program :-

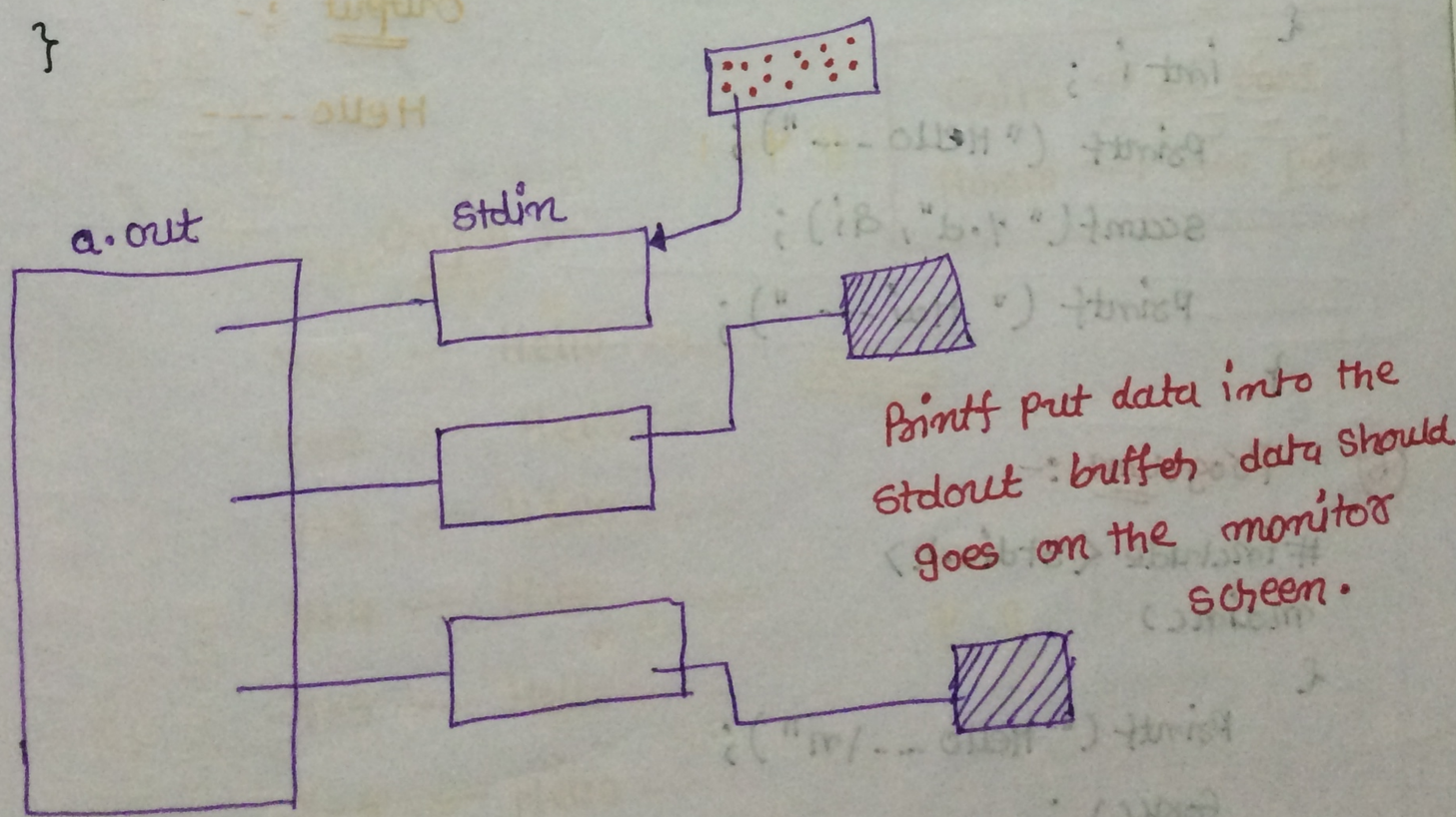
```
#include <stdio.h>
main()
{
    Printf ("Hello--- ");
    Sleep(5);
    printf (" Hai---\n");
}
```

Output :-

after 5 second

both are execute

Hello --- Hai ---



⊛ Possibility of a.out & stdout :-

1) \n
2) Buffer is full.
3) scanf
4) internally if you flused the data.
5) process terminated normally.

④ Program :-

```
#include <stdio.h>
main()
{
    Printf(" Hello---");
    sleep(5);
    Printf (" Hai---\n");
}
```

after 5 Second

Hello--- Hai---

Vector@ubuntu :~ linux/Pm

Note \n is useful for the immediately flushing the data on the monitor screen.

⑤ Program :-

```
#include <stdio.h>
main()
{
    int i;
    Printf (" Hello---");
    scanf(" %d", &i);
    Printf (" Hai--- ");
}
```

Output :-

Hello----

⑥ Program :-

```
#include <stdio.h>
main()
{
    Printf (" Hello--\n");
    fork();
    Printf (" Hai--\n");
}
```