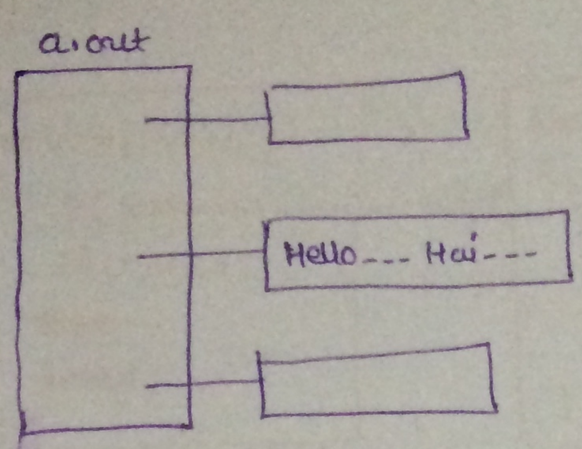
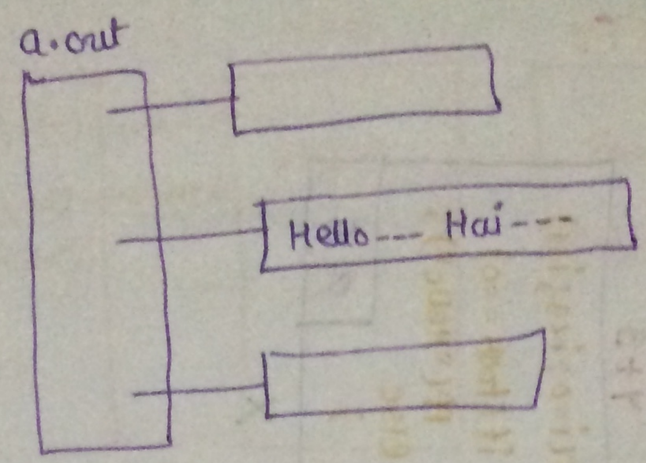




→ In a system programming every printf used with "\n".

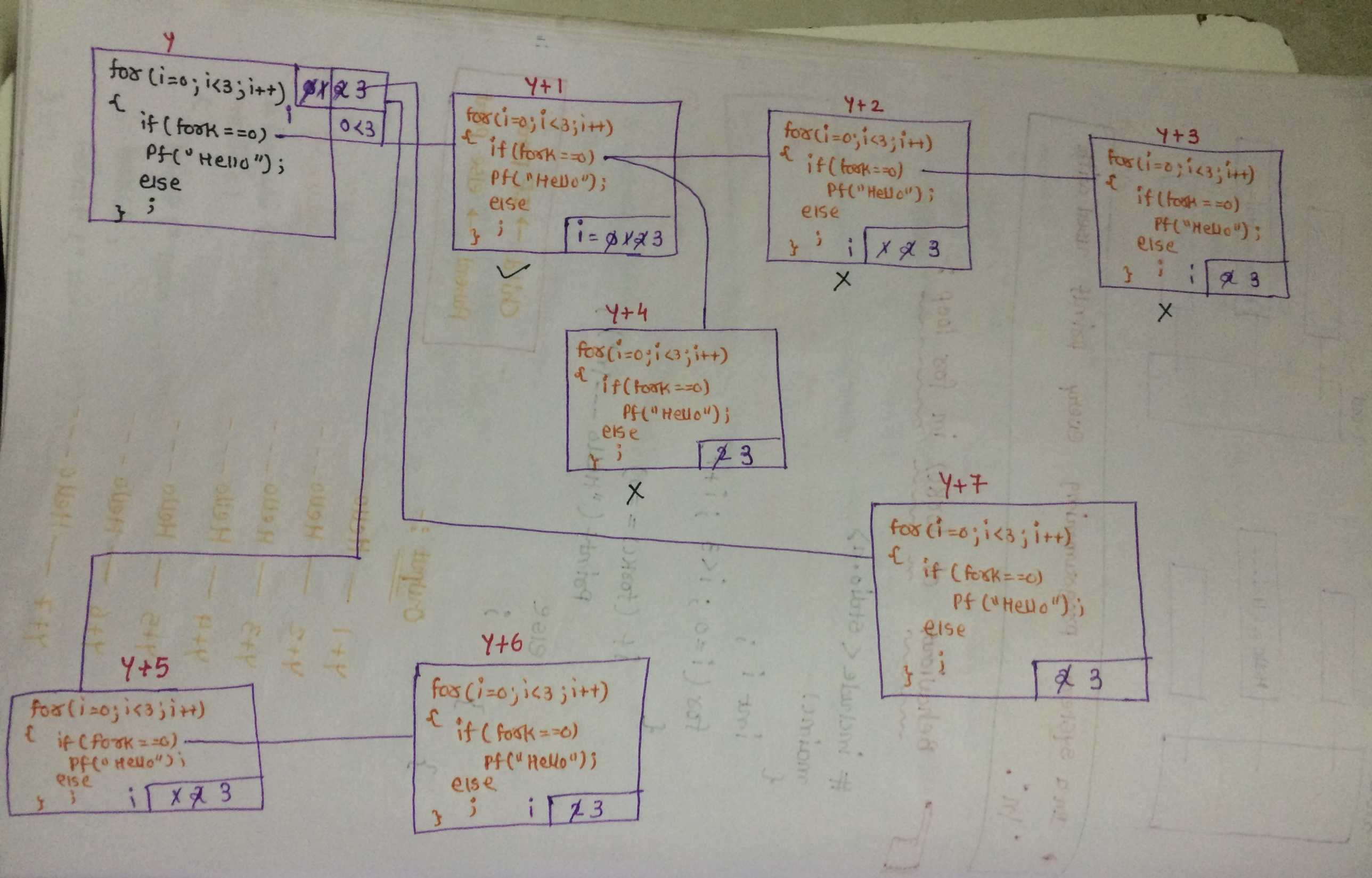
Behaviour of fork() in for loop :-

```
# include < stdio.h>
main()
{
    int i ;
    for (i=0 ; i<3 ; i++)
    {
        if (fork() ==0)
            Printf ("Hello ---\n") ;
        else
            ;
    }
}
```

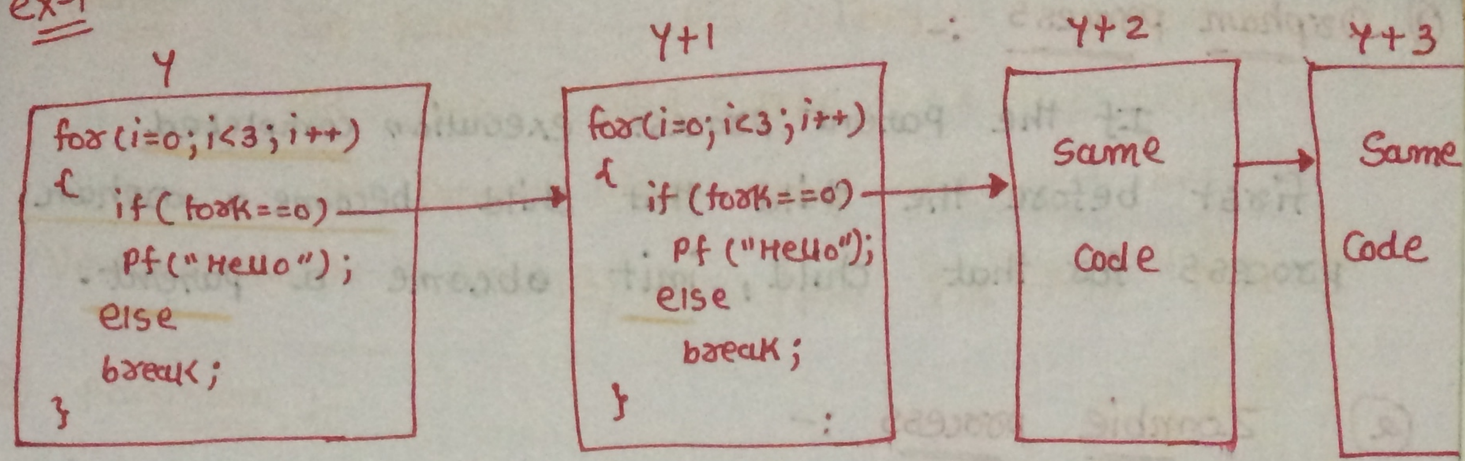
Child → If part
Parent → else part

Output :-

Y+1 — Hello---
Y+2 — Hello---
Y+3 — Hello---
Y+4 — Hello---
Y+5 — Hello---
Y+6 — Hello---
Y+7 — Hello---


Y
for (i=0; i<3; i++)
{
if (fork==0)
Pf("Hello");
else
;
}
i
0<3
Y+1
for(i=0;i<3;i++)
{ if (fork==0)
Pf("Hello");
else
;
}
i=0
Y+2
for(i=0;i<3;i++)
{ if(fork==0)
Pf("Hello");
else
;
}
i
Y+3
for(i=0;i<3;i++)
{ if(fork==0)
Pf("Hello");
else
;
}
i
Y+4
for(i=0;i<3;i++)
{ if(fork==0)
Pf("Hello");
else
;
}
Y+5
for(i=0;i<3;i++)
{ if(fork==0)
pf("Hello");
else
;
}
i
Y+6
for(i=0;i<3;i++)
{ if(fork==0)
Pf("Hello");
else
;
}
i
Y+7
for(i=0;i<3;i++)
{ if (fork==0)
Pf("Hello");
else
;
}

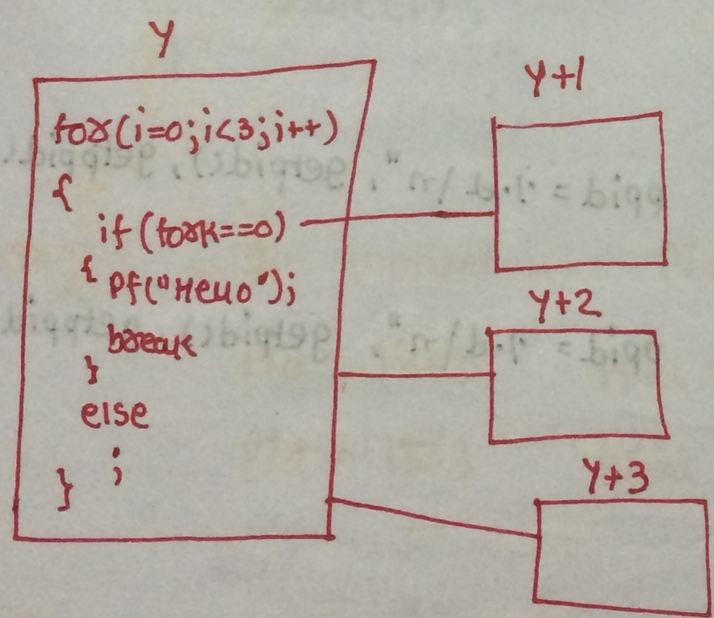
ex-1



Code

```
# include <stdio.h>
main()
{
        int i ;
        for(i=0 ; i<3 ; i++)
        {
                if (fork() = =0)
                        printf (" Hello---\n");
                else
                        break ;
        }
}
```

ex-2



Code

```
#include <stdio.h>
main()
{
        int i ;
        for (i=0 ; i<3 ; i++)
        {
                if (fork() = =0)
                {
                        Printf("Hello--\n");
                        break ;
                }
                else
                        ;
        }
}
```

① Orphan process :-

If the parent process execution completed first before the child that child become a orphan process for that child, init abcome a parent.

② Zombie process :-

If child process execution is completed first before a parent that child becomes a Zombie process. (dead process).

⇒ The process which doesn't have any pending instruction to execute but it is present in a Ram as a dead because of some reason that process is called as a dead process or Zombie process.

⊛ program :-

```
#include <stdio.h>
main()
{
    if (fork()==0)
    {
        // child
        Printf (" Pid= %d  ppid= %d \n", getpid(), getppid());
        sleep(10);
        Printf (" pid= %d  ppid= %d\n", getpid(), getppid());
    }
    else
    {
        // Parent
        Printf (" In parent ---- Pid= %d\n ", getpid());
        sleep(5);
        Printf (" In parent after sleep ---- \n");
    }
}
```

Output In parent ---- Pid = 1709

Pid = 1710 ppid = 1709

In parent after sleep

Vector@ubuntu :~ V15be3/Linux/Pm Pid = 1710 ppid = 1

⊛ program :-

```
#include <stdio.h>
main()
{
    if (fork() ==0)
    {
        //child
        Printf (" Pid = %d  ppid = %d \n", getpid(), getppid());
        sleep(10);
        Printf (" Pid = %d  ppid = %d\n", getpid(), getppid());
    }
    else
    {
        // Parent
        Printf (" In parent ---- Pid = %d \n", getpid());
        while(1);
    }
}
```

Output In parent ---- Pid = 1802

Pid = 1803 ppid = 1802

Pid = 1803 ppid = 1802

Man perror :-

Perror → Print a system error message

synopsis :-

```
#include <stdio.h>
Void perror (const char *s)
```

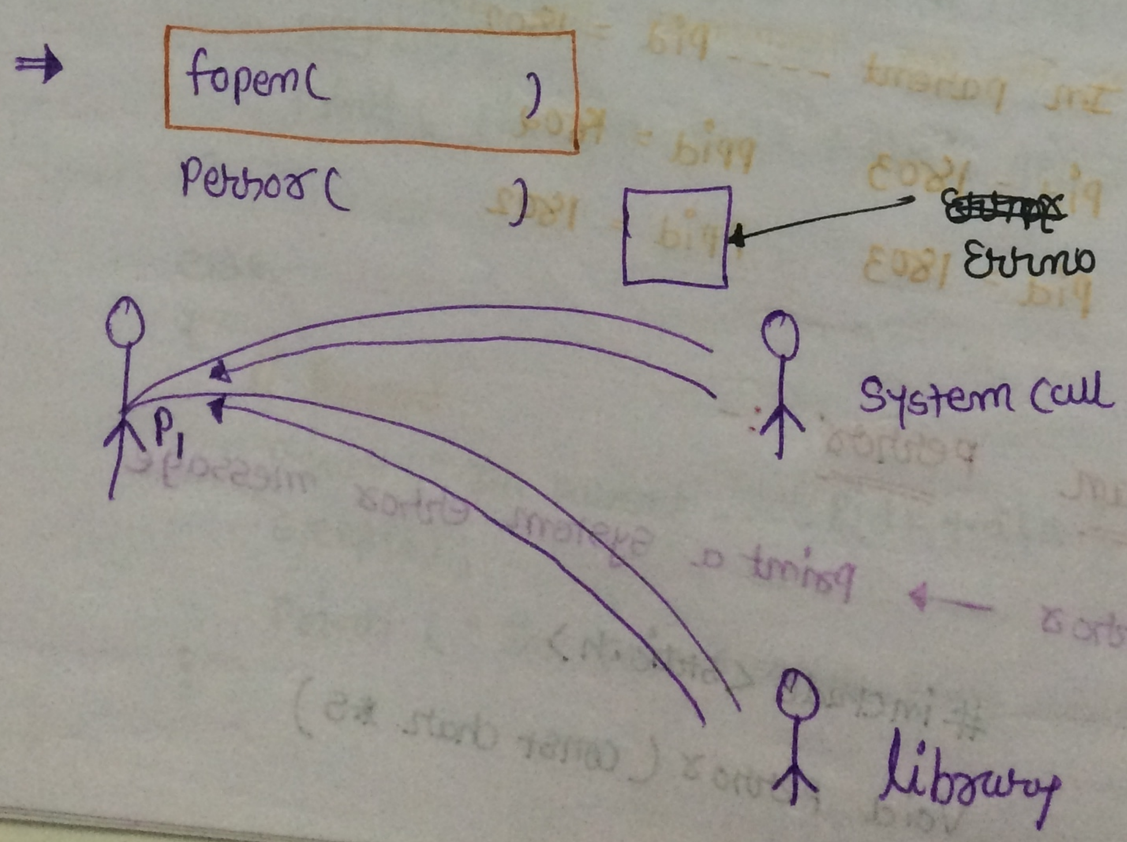
```
# include <errno.h>
Const char *sys_errlist [] ;
int sys_nerr ;
int errno ;
```

Chmod u-r data → Permission for not read the file.

file is present but display as a "file is not present"

Program

```
#include <stdio.h>
main()
{
    FILE *fp ;
    fp = fopen ("data", "r");
    if (fp==0)
    {
        Perror ("fopen") ;
    }
    Printf (" File present----\n");
}
```



Man errno :-

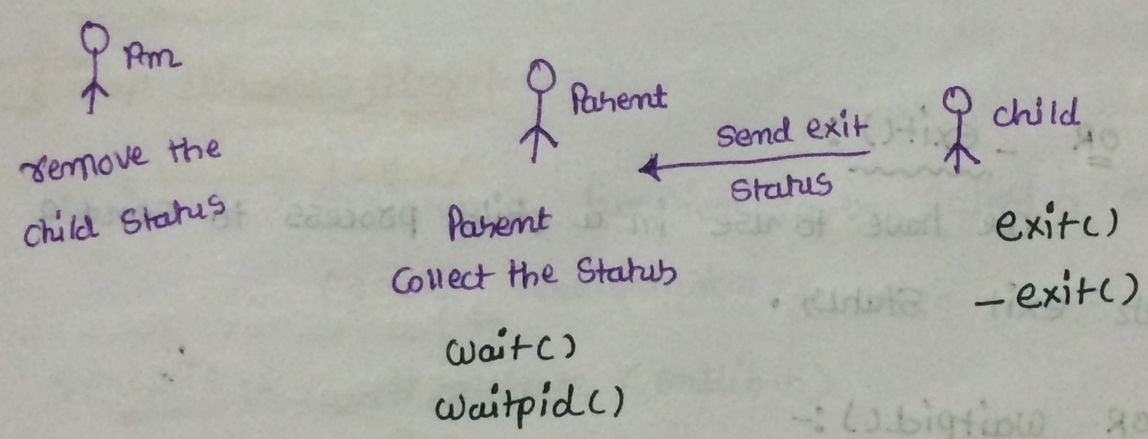
The <errno.h> headerfile define the integer Variable errno, which is System Calls and some library function in the event of an error to indicate what went wrong.

⇒ errno is defined by the ISO C Standard to be a modifiable lvalue of type int.

Name :- error → nuber of last error

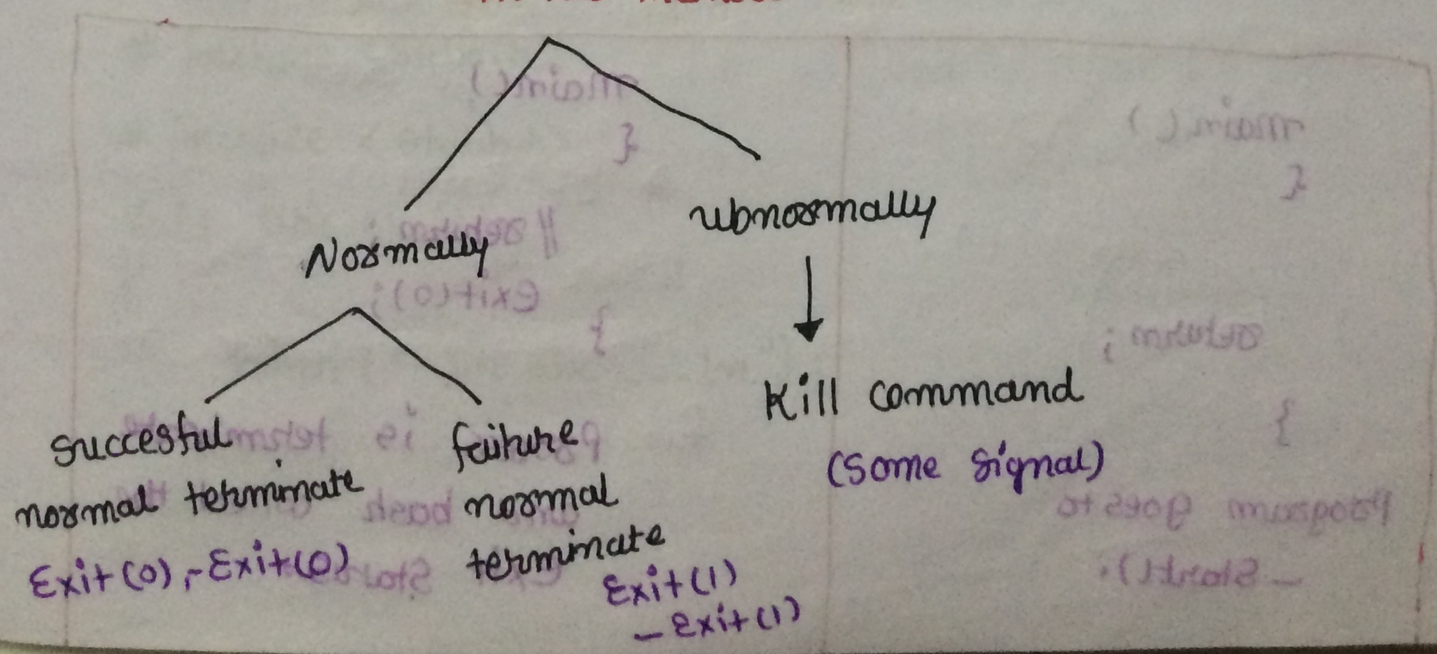
Synopsis :- # include <errno.h>

⇒ when child process is completed child has to send exit status to the parent and an parent collect the exit status. process manager removes the child from the Zombie.



⇒ exit() is use for terminate the process.

Process terminated in two method



```
if ( work is done )
    Exit(0);
else
    exit(1)
```

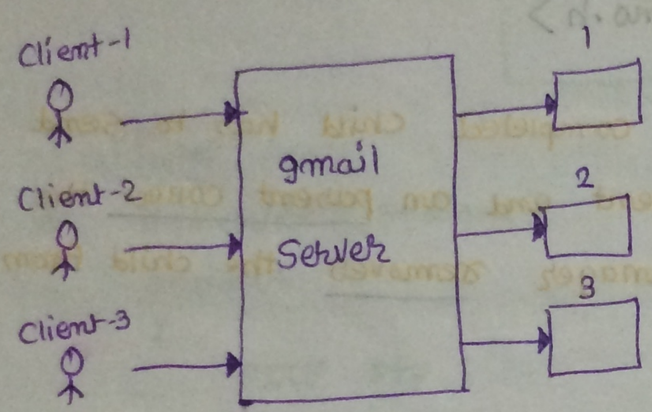
exit(0) = work is done (success)

exit(1) = work is not done (failure)

⇒ If in a system no space for the new event thats called a Dead process.

⇒ If is use in a where multiple process is work.

ex :- gmail server, Yahoo server



⇒ when 1 is logout then send a exit status of the gmail server.

⇒ Otherwise sometime all process are zombie.

exit() OR _exit() :-

we have to use in a child process to send a proper exit status.

wait() OR waitpid() :-

we have to use in a parent process to wait for child process completion and collect the proper exit status.

main()
{
 	
return;
}
Program goes to
_Start().
	main()
	{
	// return;
	exit(0);
	}
	Process is terminate and bash collect the exit status.