```
int d ;
Srand (getpid()) ;
d=rand() %10+1 ;
Printf (" In C2 %d \n", d) ;
Sleep (d)
exit(2)
}
else
{
    if (fork() ==0)
    {
        // child 3
        int d ;
        Srand (getpid()) ;
        d=rand() %10 +1 ;
        Printf (" In C3  %d \n", d) ;
        Sleep (d) ;
        exit(3) ;
    }
    else
    {
        int s ;
        Printf (" In Parent ----\n") ;
        while (wait(&s) != -1)
        {
            s = s >>8 ;
            if (s==1)
                Printf(" C1---\n") ;
            else if (s==2)
                Printf (" C2 ----\n") ;
            else if (s==3)
                Printf (" C3 ----\n") ;
        }
    Printf (" Done----\n") ;
    }
}
```

Output :-

```
In C3  1
In C2  9
In C1  2
C3 ----
C1 ----
C2 ----
Done ----
```

Example-3 Parent has conform the which child is complete first using return Value of the wait().

⇒
```
#include <stdio.h>
#include <stdlib.h>
main()
{   int a[3];
    if ((a[0] = fork()) == 0)
    {
        // child 1
        int d;
        srand (getpid());
        d = rand () % 10 + 1;
        Printf ("In C1 %d\n", d);
        sleep(d);
        exit (1);
    }
    else
    {
        if ((a[1] = fork()) == 0)
        {
            // child 2
            int d;
            srand (getpid());
            d = rand() % 10 + 1;
            Printf ("In C2 - %d\n", d);
            sleep (d);
            exit(a);
        }
        else
        {
            if ((a[2] = fork()) == 0)
            {
                // child 3
                int d;
```

```
    srand (getpid ());
    d=rand () %10 +1 ;
    Printf (" Im C3   %d \n",d);
    Sleep(d) ;
    exit(3) ;
}
else
{
    int s ;
    Printf (" Im Parent ---\n");
    While ((s=wait(0)) != -1)
    {
        if (s==a[0])
        Pf (" C1---\n");
        else if (s==a[1])
            Printf (" C2----\n");
        else if (s==a[2])
            Printf (" C3----\n");
    }
    Printf (" Done----\n");
  }
 }
}
```

Ⓐ Vi copy.c :-

```
#include <stdio.h>
main()
{
    int i=10 ;
        if (fork()==0)
        {
            // child
            i = getpid () ;
            Printf (" Im child. i = %d \n", i) ;
        }
```

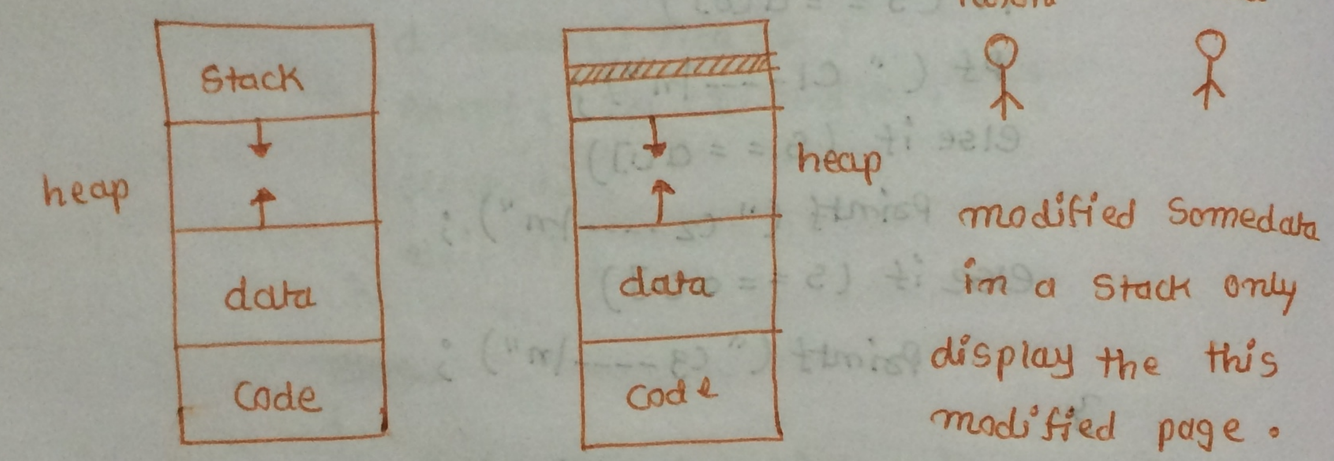
```
else
{
    sleep(1) ;
    Printf (" In Parent i = %d\n ", i) ;
}
}
```

Output :-

In child i = 1604

In parent i = 10

Copy on Write :-



modified Somedata in a Stack only display the this modified page.

⇒ In newer Version of the fork() whenever child process is created no memory block is duplicated.

⇒ If either the parent or the child modifing the data then separate page is created for that process this mechanism is called "Copy on write" mechanism.

① WIFEXITED (Status) → Child process terminate normally

② WEXTSTATUS (Status) →

③ WIFSIGNALED (Status) → child process terminate using some signal.

④ WTREMSIG (Status) →

① WIFEXITED (status) :-

return true if the child terminated normally, that is by calling exit() or _exit() or by returning from main().

② WEXTSTATUS :-

return the exit status of the child. This consits of the least significant 8 bits of the status argument that the child specified in a call exit() or _exit().

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
main()
{
    if (fork()==0)
    {
        Printf ("In child ----%d\n", getpid());
        sleep(10);
        Printf ("In child after sleep ----\n");
        exit(1);
    }
    else
    {   int s,r;
        Printf(" In Parent ---\n");
        r = wait(&s);
        if ( WIFEXITED (S))
        {
            printf (" Normal ---- %d\n", WEXITSTATUS(S));
        }
        else if (WIFSIGNALED (S))
            Printf (" Signal ---- %d\n", WTREMSIG(S));
    }
}
```

Wait() :- The wait() system call suspended execution of the calling process until one of its children terminates. The call wait(&status) is equivalent to

```
Waitpid (-1, &status, 0) ;
```

① **WNOHANG** :- return immediately if no child has exited.

```
main()
{
    if (fork() == 0)
    {
        Printf (" In child ---- %d\n", getpid());
        sleep(10);
        Printf (" In child after sleep ---\n");
        exit(1) ;
    }
    else
    {
        int s,r ;
        Printf (" In Parent Before wait ----\n");
        waitpid (-1, &s, 0) ; // waitpid (-1, &s, WNOHANG);
        Printf (" In parent After wait ---\n");
    }
}
```

Output

```
In parent Before wait
In child ---- 1800
In child after sleep ----
In parent after wait ---
```

Output

```
In parent Before wait ---
In parent After wait ---
In child ---- 1800
In child after sleep --
```

② WUNTRACED :- also return if a child has stopped not traced via ptrace(2). Status for traced children which have stopped is provided even if this option is not specified.

Waitpid (-1,0,0) ⟶ child is terminated then comeout.

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    if (fork()==0)
    {
        Printf(" In child ---- %d\n", getpid());
        while(1);
    }
    else
    {
        int s,x;
        Printf(" In parent Before wait ----\n");
        wait pid(-1,0,0); || waitpid(-1,0,WUNTRACED);
        Printf(" In parent after wait ----\n");
    }
}
```

Output

In parent Before wait---
In child ----1820
Process terminate then only
In parent after wait---

Output

In parent before wait--
In child...1820
When process is suspended OR terminate then display the
In parent After wait----

(*) WIFSTOPPED (status) :-

→ return true if the child process was stopped by delivery of a signal, this is only possible if the call was done using WUNTRACED or when the child is being traced.

(3) WCONTINUED (9) :-

also return if a stopped child has been resumed by delivery of SIGCONT.

→ already child process is stopped and you will resume it then come out.

```
waitpid(-1, &s, WUNTRACED);
```

```
waitpid(-1, &s, WCONTINUED);
```

$ man exec

Synopsis :- #include <unistd.h>

```
extern char **environ;
```

(1) int execl (const char *path, const char * arg, ---);

(2) int execlp (const char *file, const char * arg, ---);

(3) int execv (const char *path, char * const argv[]);

(4) int execvp (const char *file, char * const argv[]);

Description :-

The exec() family of functions replaces the current process image with a new process image. The function described in this manual page are front-ends for execv(2).