ork is done (success)
ork is not done
(faiture)
went thats called
is work.
er
is logout then
exit Status of
l server.
e Sometime call
are zombie.
to send a
ess to wait
ber exit
minate
ct the

$ man 3 exit :-

	Exit()	_Exit()
1)	It is a library function.	1) It is a System call.
2)	Prototype Void exit (int Status)	2) prototype :- Void _exit (int status)
3)	The exit() function cause normal process termination	3) The function _exit() terminates the process "immediately".
4)	all function registered with atexit(3) and on exit(3) are called, in the reverse order of their registration.	

$ Man atexit

atexit :- register a function to be called a normal process termination.

Synopsis :- #include <stdlib.h>

int atexit (void (*function) (void));

Program exit program :-

```
# include <stdio.h>
# include <stdlib.h>
Void abc (Void)
{
    Printf (" In abc ___ \n");
}
main()
{
```

```
Printf ("In main----\n");
atexit (abc);
printf ("hai---\n");
sleep(10);
exit(0);
}
```

Output :-

In main----

hai-----

(after 10 second)

In abc----

⇒ Before terminate by exit() function, it is check their is registered function is there or not after it is terminate.

⇒ If there it is called. it is in a reverse order.

✱ Program :- _exit() :-

```
#include <stdio.h>
#include <stdlib.h>
Void abc (Void)
{
    Printf ("In abc---\n");
}
main()
{
    Printf("In main----\n");
    atexit(abc);
    Printf ("hai--- ");
    sleep(10);
    _exit(0);
}
```

Output:-

In main---

(after 10 second)

Vector @ubuntu :~ V15be3/

Linux.

⇒ hai is not print bcoz of Hai--- buffer is not flush and _exit(0) is terminate the process that why Hai---- is not print after 10 second.

⇒ If exit(0); is used then After 10 second Hai is Printed and also registered function also printed.

$ man wait :-

Process to change the state.

Synopsis :- #include <sys/types.h>
#include <sys/wait.h>

pid_t wait (int *status);

pid_t waitpid (pid_t pid, int *status, int options);

int waitpid (idtype_t idtype, id_t id, siginfo_t * infoop, int options);

⊛ program :-

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
main()
{
    if (fork() == 0)
    {
        printf ("In child----\n");
        sleep(10);
        printf ("In child after sleep---\n");
        exit(1);
    }
    else
    {
        int s;
        printf ("In parent ---\n");
        wait(&s);
        printf ("In parent after sleep--- %d\n", s);
        while(1);
    }
}
```

Output :- In parent ----
In child ----
In child after sleep ----
In parent after sleep ---- 256

✱ Few point for wait() :-

① we used the wait() system Call in a parent.
② wait() will block the parent process till the completion of the child.
③ upon successful waiting wait return the Pid of the child.
④ if there is not Child process is there wait() is failed it return -1.
⑤ wait() function Can be used in a two different way :-
1) wait(&s);
2) wait(0);

wait(&s);	wait(0);
wait is waiting for the Child process termination and also interested to collect exit status.	Parent is waiting for Child process termination but not interested to Collect the exit Status.

✱ Program :-

```
#include <stdio.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);
        Printf (" In parent after sleep ---%d\n", s);
        r = wait (0);
        while(1);
    }
}
```

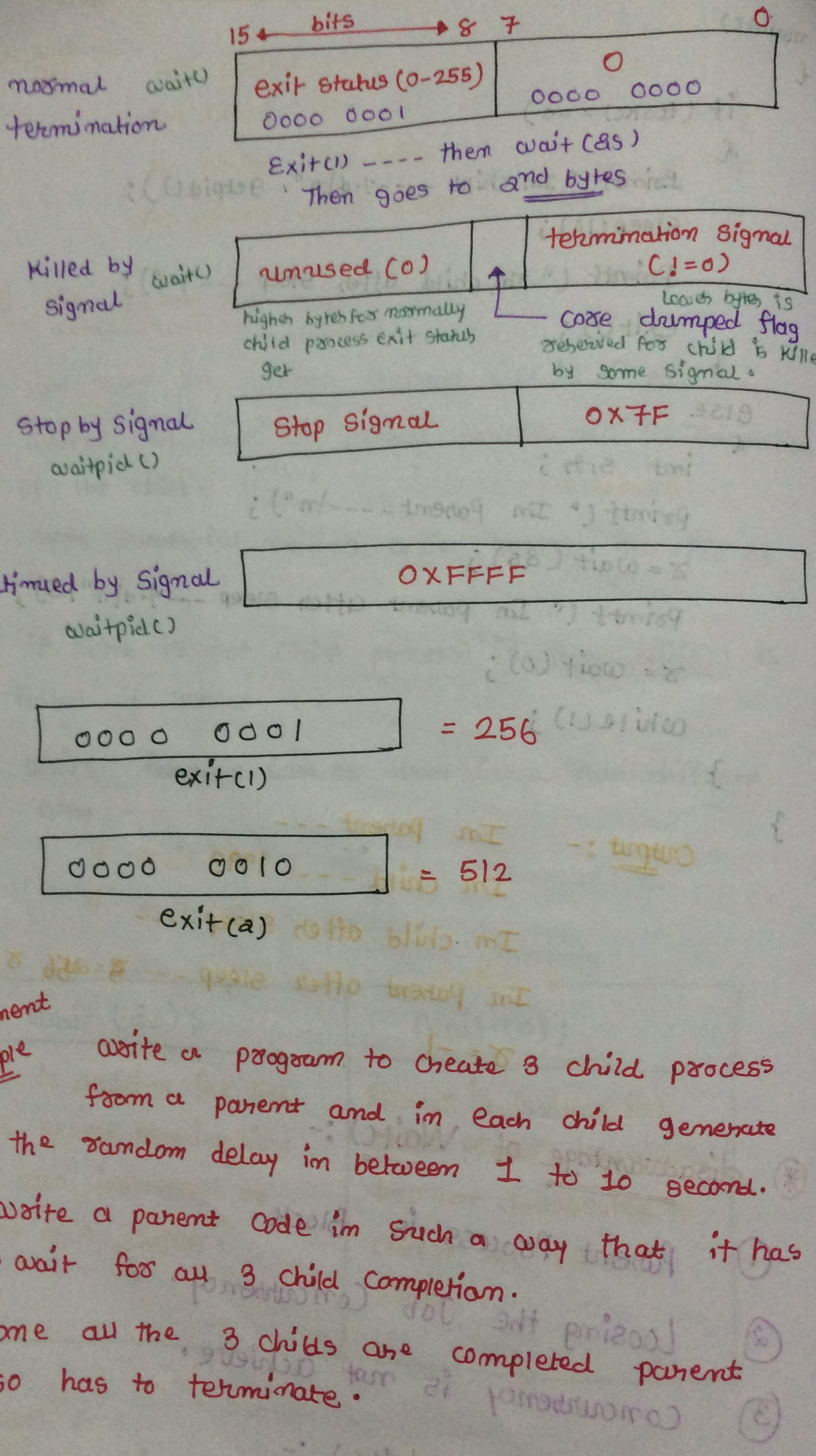
Output :- In parent ---
In child ---- 1993
In child after sleep ---
In parent after sleep ---- s = 256 r = 1993
r = -1

(*) disadvantage of Wait() :-

(1) Parent Process is Block.
(2) Loosing the Job Concurrency.
(3) Concurrency is not achieve.

(*) advantage of Wait() :-

(1) Child process is remove from Zombie.

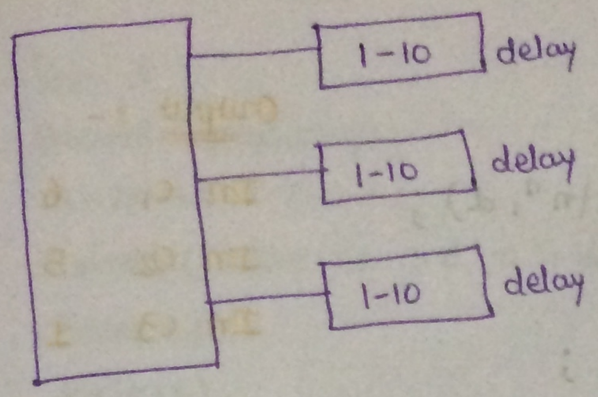


Assignment Example

Write a program to create 3 child process from a parent and in each child generate the random delay in between 1 to 10 second.

① write a parent code in such a way that it has to wait for all 3 child completion.

② one all the 3 childs are completed parent also has to terminate.




```
# include <stdio.h>
main()
{
    if ( fork() ==0)
    {
        // child 1
    int  d ;
         srand (getpid());
         d=rand() %. 10 +1 ;
         printf (" In. c1   %.d\n ", d);
         sleep (d);
         Printf (" c1 ----\n") ;
}
else
{
        if(fork() ==0)
        {
            // child 2
        int d ;
            srand (getpid());
            d = rand () %.10 +1
            printf (" In  c2 .  %.d\n",d) ;
            sleep(d) ;
            printf (" c2 ----\n") ;
        }
        else
        {
            if (fork() ==0)
            {
                // child 3
            int d ;
```

```
        srand (getpid());
        d=rand () %10 +1 ;
        printf (" In C3  %d \n", d);
        sleep (d);
        Printf (" C3 ----\n");
    }
    else
    {
        Printf (" In Parent ----\n");
        while (wait (0) != -1);
        Printf (" Done ---\n");
    }
  }
}
```

Output :-

```
In C1   6
In C2   5
In C3   1

C3 ----
C2 ----
C1 ----
Done ----
```

Example-2 Parent should wait which child complete first? using exit status.
(above program modification).

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