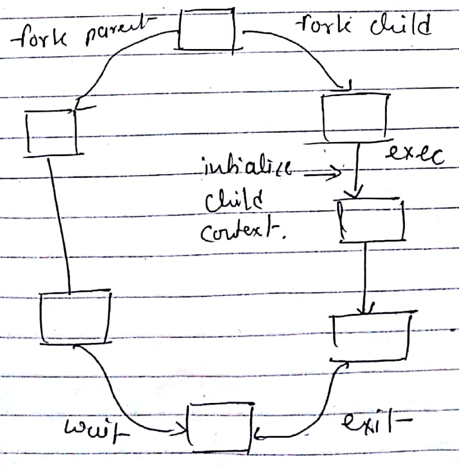
I.

```
Int pid = fork();
```
Create a new process that is a clone of its Parent.

```
exec* ("program" [, argvp, envp]);
```
overlay the calling process virtual memory with a new program and transfer control to it.

```
exit (status);
```
Exit with status, destroying the process.

```
Int pid = wait (&status);
```
wait for exit (or other status change) of a child



II

```
int pid;
int status = 0;

if (pid = fork()) {
 /* Parent */
 ...
pid = wait (&status);
} 
else {
 /* child */

 exit (status);
}
```

The fork syscall returns twice: it return a zero to the child and the child process ID (pid) to the parent.

Parent uses <u>wait</u> to sleep until the child exits; wait returns child pid and status.

wait variants allow wait on a specific child, or notification of stops and other signals.

```
int main()
{
  int pid ;
 /* fork another process */
  pid = fock ();
  if (pid <0) { /* error occured */
    fprint (stderr, " fork failed");
    exit (-1);
  }
else if (pid ==0)
  { /* child process */
   execlp("/bin/ls", "ls",
           NULL);
  }

else { /* parent process */
       /* parent will wait for
       the child to complete */
   wait (NULL);
   printf ("child Complete");
   exit (0);
  }
}
```

Process Termination

{

}

write a prog. to crete the process

:-

child should calculte Area of the circle

parant Simple interest.

```
#
# include <uni std.h>
int main ()
{
 int pid ;
  pid = fock ();
 if (pid <0)
 fprint ( stderr, " fork failed");
  exit (-1);
}
else if (pid ==0)
 {
  scanf (" %d ", x )
 fprint (" The area of circle is
   ", 3.14 * x * x );

 }

else {
  scanf (" %d %d %d ", &P
          &R, &T);
printf (" The simple interest is %d
     (P*R*T)/100);
   exit(0);
 }.
```

Zombie

A process that has terminated but whose parent

interprocess Communication

-> independent if it cannot affect or be affected by the othe executing processes

-> Cooperating

Reasons for process cooperation

↳ information sharing
↳ speedup
↳ Modularity

UNIX
FORK FORK

```
#include <stdio.h>
#include <sys/types.h>
int main()

fork();
printf("Hello world!\n");
return 0;
}.
```

```
#
#
# include <unistd.h>
void forkexample ()
{
  if(fork()==0)
    printf("Hello from Child\n");
  else
    printf ("Hello from Parent\n");
}
int main()
{ forkexample ();
  return 0;
}
```

Output-

1. Hello from Child
Hello from Parent.
or
~~Hello from child~~

Hello from ~~Parent~~

2. Hello from Parent
Hello from Child