FORK SYSTEM CALL

The fork system call is used to create a new process. The newly created process is the child process. The process which calls fork and creates a new process is the parent process. The child and parent processes are executed concurrently.
But the child and parent processes reside on different memory spaces. These memory spaces have same content and whatever operation is performed by one process will not affect the other process.

***Negative Value*:** creation of a child process was unsuccessful.
Zero: Returned to the newly created child process.
Positive value: Returned to parent or caller. The value contains process ID of newly created child process.

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

    // make two process which run same
    // program after this instruction
    fork();

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

Output:
Hello world!
Hello world!

When working with fork(), <sys/types.h> can be used for type *pid_t* for processes ID's as pid_t is defined in <sys/types.h>.
The header file <unistd.h> is where fork() is defined so you have to include it to your program to use fork(). The return type is defined in <sys/types.h> and fork() call is defined in <unistd.h>. Therefore, you need to include both in your program to use fork() system call.

```
#include <stdio.h>
#include <sys/types.h>
int main()
{
    fork();
    fork();
    printf("hello\n");
    return 0;
}
```

Output-

hello
hello
hello
hello

CREATE CHILD PROCESS

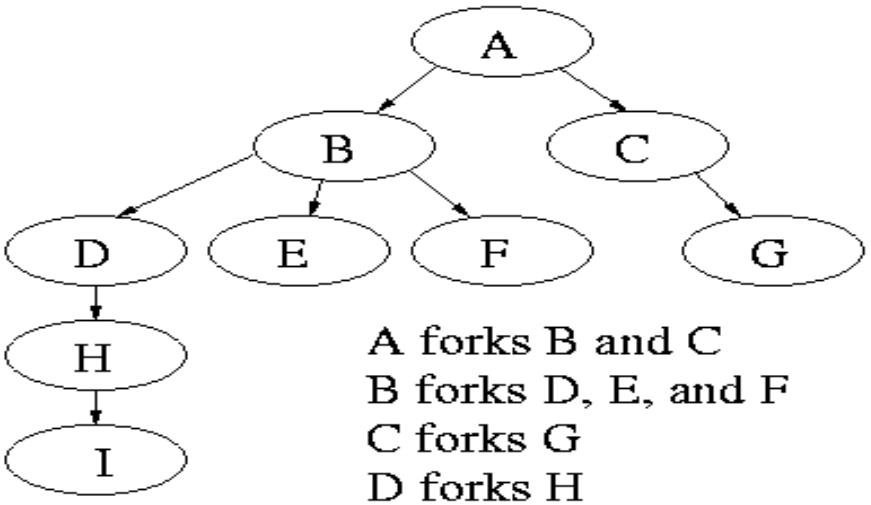
```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    pid_t p;
    p = fork();
    if(p==-1)
    {
        printf("There is an error while calling fork()");
    }
    if(p==0)
    {
        printf("We are in the child process");
    }
    else
    {
        printf("We are in the parent process");
    }
    return 0;
}
```

Both getppid() and getpid() are inbuilt functions defined in **unistd.h** library.

```
#include<unistd.h>
#include<stdio.h>
#include <sys/types.h>
int main()
{
Int r;
      r=fork();
if(r==0)
{
printf(“The process is child process\n”);
printf(“child id=%d\n”, getpid());
printf(“parent id=%d\n”,getppid());
}
else
{
printf(“The process is the parent process\n”);
printf(“The process id=%d\n”,getpid());
printf(“parent id=%d\n”,getppid());
}
return 0;
}
```

distinguishes the parent from the child.

```
#include <stdio.h>

 #include <sys/types.h>

#include<unistd.h>
#define MAX_COUNT 10
void  ChildProcess(void);               /* child process prototype */
void ParentProcess(void); /* parent process prototype*/

int main(void)
{
      pid_t pid;

      pid = fork();if
      (pid == 0)
            ChildProcess();else
            ParentProcess();return
0;
}

void ChildProcess(void)
{
      int     i;

      for (i = 1; i <= MAX_COUNT; i++)
            printf("This line is from child, value = %d\n", i);
            printf(" *** Child process is done ***\n");
}


void ParentProcess(void)
{
      int     i;

      for (i = 1; i <= MAX_COUNT; i++)
      printf("This line is from parent, value = %d\n", i);
      printf("*** Parent is done ***\n");
}
```




```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
int a ;
int b ;
int c ;
int d ;
int e ;
int f ;
int g ;
int h ;
int i ;
```

```
b=fork();

if (b == 0) //it's child
{
d= fork();
if(d==0)
{
h=fork();
if(h==0)
{
i=fork();
if(i==0)
printf("%d: I\n", getpid());
else
printf("%d: H\n", getpid());
}
else
printf("%d: D\n", getpid());
}
else
{
e=fork();
if(e==0)
printf("%d: E\n", getpid());

else
{
f=fork();
if(f==0)
printf("%d: F\n", getpid());
else
printf("%d: B\n", getpid());
}
}
}
else
```

```
{
c=fork();
if(c==0){
g=fork();
if(g==0)
printf("%d: G\n", getpid());
else
printf("%d: C\n", getpid());
}
else
printf("%d: A\n", getpid());
}
return 0;
}
```

Output (UNIX):

```
10201: A
10203: C
10202: B
10204: G
10207: F
10206: E
10205: D
10208: H
10209: I
```