PROCESS MANAGEMENT

Multitasking/Multiprocessing

- In a multiprogramming environment, several processes are kept in memory at one time.
- When one process is currently using the CPU and after some time it encounters an I/O request, the OS takes the CPU from that process and gives it to another process which requires it. And this pattern continues.
- Generally done so that CPU never sits idle.
- This maximizes CPU utilization thereby, improving system's performance.

Process Identification

getpid( ) - Unix Function (System Call)

It is used to get the ID no. of a process.

Example:

```
main()
{
    int  pid;
    pid = getpid();
    printf("Process Id = %d ",pid);
}
```

Parent and Child Process

getppid( ) - Unix Function (System Call)

It is used to get the ID no. of a parent process.

Example :

```
main()
{
    Int ppid;
    ppid=getppid();
```

```
        Printf("Parent process Id=%d ",ppid);
}
```

fork( ) - It is a system call used to create child process.

Example :

```
main()
{
        printf("WelCome\n");
        fork();
        printf("\nHello World");
}
```

Output:

```
WelCome
Hello World
Hello World
```

Discussion :

1. Fork creates a child that is duplicate of the parent process.
2. Two identical processes are created in memory.
3. Child process begins with fork(). All statements after call to fork() will be executed twice- once by parent and once by child. All statements before fork() will be only executed by the parent process, So WelCome is executed once by the parent process.

Example :

```
main()
{
        printf("WelCome\n");
        fork();
        fork();
        printf("\nHello World");
}
```

Output :

WelCome

Hello World

Hello World

Hello World

Hello World

Discussion (on execution point of view)

This gives a total of 4 processes in memory, the parent and its two children and one grandchild.

The first call to fork() creates one child process. Now there are two processes. Both processes begin executing from the second call to the fork(), thus producing four processes.

Note :

1. The progression with each fork() that is added is 2^n where *n* is number of calls to fork().
2. pid of child is always 0.
3. pid of parent is greater than 0.
4. If pid is less than 0 then, Error occurred, process is not created.
5. pid returns two values – 0 to the child, and 1 to its parent.

Example:

```
#include <stdio.h>
#include<sys/types.h>
        int  main( )
        {
          int pid;
          pid = fork( ) ;                  /* fork a process  */
          if ( pid < 0 )                   /* child process not created, error occurred  */
           {
              printf( stderr, “Error Occurred” );
              exit ( -1 );
           }
           elseif ( pid == 0 )             /* child process created  */
           {
              printf ( “I am in child process” );
              printf ( “I am exiting child process” );
```

```
    }
    else                            /* parent process  */
    {
       printf ( “I am in parent process” );
       printf ( “I am exiting parent process” );
    }
}
```

Output:

First run …

Output (1) :
I am in child process
I am in parent process
I am exiting child process
I am exiting parent process

Next run …

Output (2) :
I am in child process
I am in parent process
I am exiting parent process
I am exiting child process

Conclusion …

Different run produces different outputs.

We use **wait( )** in the parent process, so that the parent will wait until the child has completed execution. Hence, the modified code in parent process becomes :

```
else                                /*  parent process  */
    {
       wait ( );                /* this system call suspend execution of the current process
                                   until one of its children terminates */
       printf ( “I am in parent process” );
       printf ( “I am exiting parent process” );
    }
```

Output :

I am in child process
I am exiting child process
I am in parent process
I am exiting parent process

sleep(int seconds) - This system call halts the process for the given number of seconds.

Example :

```
#include <stdio.h>
#include<sys/types.h>
int main( )
{
        int pid;
        pid = fork( ) ;                    /* fork another process  */
        if ( pid == 0 )                    /* child process created  */
        {
                printf ( "I am in child process" );
                sleep(10);
                printf ( "I am exiting child process" );
        }
        else                               /* parent process  */
        {
                printf ( "I am in parent process" );
        }

}
```