

***Learning Objectives:** Students will be able to learn UNIX process management system calls and library functions.*

Exercise 1

Write a C program to print the process ID of the process and it's parent process ID.

getpid() – to get process ID

getppid() – to get parent process ID

```
#include<stdio.h>
#include<unistd.h>
int main(){
    int pid= getpid();
    int ppid=getppid();

    printf("process ID :%d",pid);
    printf("\nparent process ID :%d",ppid);
}
```

```
~$ vi ex01.c
~$ gcc -o ex01 ex01.c
~$ ./ex01
process ID :588
parent process ID :448~$
```

Handwritten notes: "compile" with an arrow pointing to the gcc command, and "Run" with an arrow pointing to the ./ex01 command.

fork () System call – used to create a duplicate process(child) from original process(parent)

Exercise 2

```
#include <stdio.h>
main()
{
    printf("I am Parent\n");
    fork();
    printf("Hello World...\n");
}
```

Parent process

```
#include <stdio.h>
main()
{
    printf("I am Parent\n");
    fork();
    printf("Hello World...\n");
}
```

Child process

```
#include <stdio.h>
main()
{
    printf("I am Parent\n");
    fork();
    printf("Hello World...\n");
}
```

```
I am Parent
Hello World...!
Hello World...!
```

Exercise 3

```
#include <stdio.h>
main()
{
    int ret;
    printf("I am Parent\n");
    ret = fork();
    printf("Return Value: %d\n", ret);
}
```

Child process – 0

Parent process – any positive value

Error – negative value

```
I am Parent
Return Value: 794
Return Value: 0
```

getpid () and getppid () system calls

Exercise 5

```
#include <stdio.h>
main()
{
    int ret;
    printf("Hello World\n");
    ret = fork();

    if(ret == 0){
        printf("I am Child and Return Value=%d\n", ret);
        printf("Child PID: %d\n", getpid());
        printf("Child's Parent PID: %d\n", getppid());
    }
    else{
        printf("I am Parent and Return Value=%d\n", ret);
        printf("Parent PID: %d\n", getpid());
    }

    sleep(20);
}
```

Handwritten annotations:

- child** (written vertically next to the if block)
- parent** (written vertically next to the else block)
- 4** (written above the first printf in the child block)
- 5** (written above the second printf in the child block)
- 6** (written above the third printf in the child block)
- 2** (written above the first printf in the parent block)
- 3** (written above the second printf in the parent block)
- ←** (arrow pointing to the sleep(20); line)

```
Hello World
I am Parent and Return Value=838
Parent PID: 837
I am Child and Return Value=0
Child PID: 838
Child's Parent PID: 837
```

execl () system call-used to run unix in-buld commands**Exercise 6**

```
#include <stdio.h>
main()
{
    printf("Here comes the date. \n");
    execl("/bin/date", "date", 0); /*0 means end-of-arguments */
    printf("That was the date. \n");
}
```

```
Here comes the date.
Tue Oct 11 06:18:42 UTC 2022
```

Exercise 7

```
#include <stdio.h>
main()
{
    printf("Here comes the date. \n");
    fork();
    execl("/bin/date", "date", 0);
    printf("That was the date. \n");
}
```


Why did you get date two times? and Why didn't you get first print statement two times?

```
Here comes the date.
Tue Oct 11 06:20:39 UTC 2022
Tue Oct 11 06:20:39 UTC 2022
```

Exercise 8

system () library function-used to run pre-defined commands

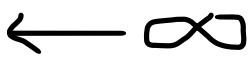
```
#include <stdio.h>
main()
{
    printf("Here comes the date. \n"); ✓
    system("date");
    printf("That was the date"); ✓
}
~$ ./ex01
Here comes the date.
Tue Oct 11 06:27:43 UTC 2022
that was the day~$
```



Zombi Process-if parent process is running again and again so that child process become a zombie process.

Exercise 10

```
#include <stdio.h>
main()
{
    int id;
    if ((id = fork())== 0)
    {
        printf("I am child process \n");
    }
    else
    {
        while(1) ← ∞
        {
            sleep(100);
        }
    }
}
I am child process
```



Orphan Process -processes that are still running if parent process are terminated**Exercise 11**

```
#include <stdio.h>
main()
{
    int id;
    if ((id = fork())== 0)
    {
        printf("I am child process \n");
        sleep(10);
    }
    else
    {
        printf("I am parent process \n");
    }
}
```

```
I am parent process
I am child process
```