NAME: **Meet Raut**

DIV: **S21**

ROLL.NO: **2201084**

Experiment 4:

- **AIM: To create a child process in Linux using system calls; fork().**

- **THEORY:**

1] System call:

When a program is in user mode and requires access to Ram or hardware resources, it must ask the kernel to provide access to that resource. This is done via something called a system call. When a program makes a system call, the mode is switched from user mode to Kernel mode .This is called context switch. The kernel provides the resources which the program requested. System calls are made by user level programmes in following cases :

- Creating, opening, closing and delete files in the system
- Creating and managing new processes
- Creating a connection in the network, sending and receiving packets
- Requesting access to a hardware device like a mouse or a printer

2] Fork ( ) :

The fork system call is used to create processes. When a process makes a fork().call, an exact copy of the process is created.

There are now two processes, one being the parent process and the other being the child process. The process which is called fork()call is the parent process and the process which is created is called child process. The child process will be exactly the same as the parent. The process state of the parent i.e the address space, variables, open files etc is copied into the child process. The change of values in the parent process doesn’t affect the child and vice versa.

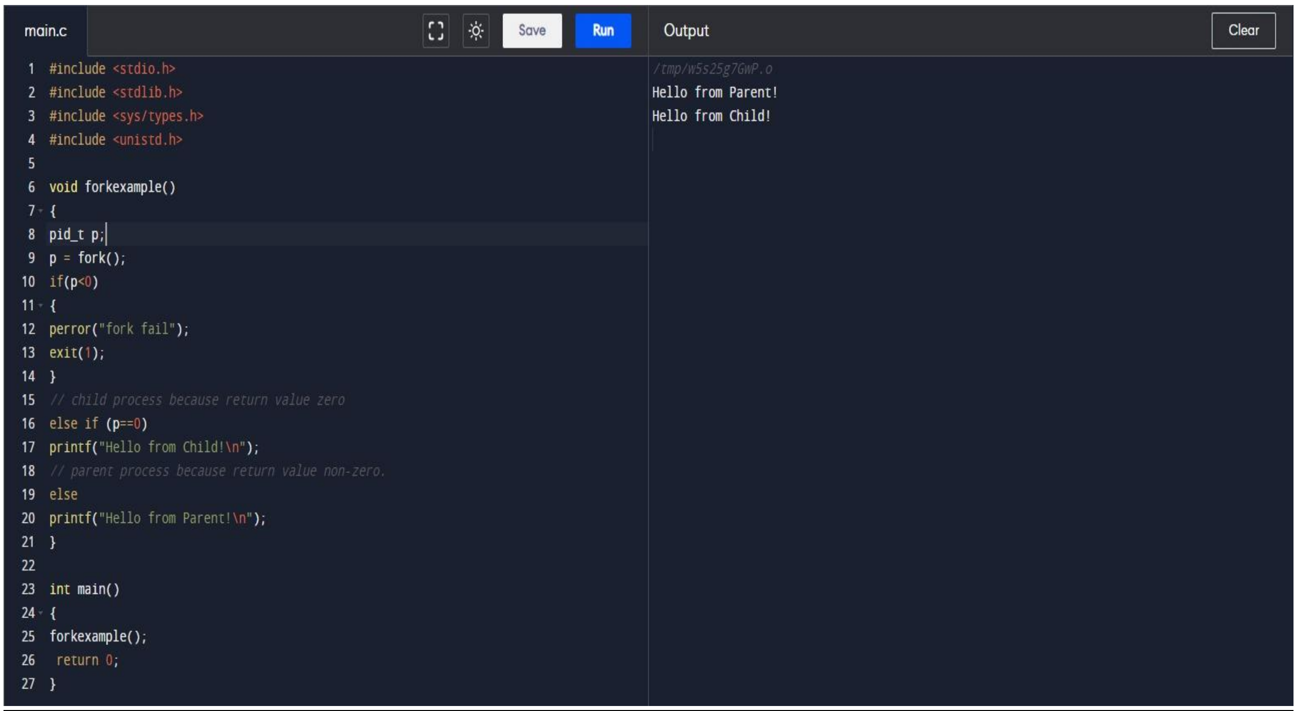
- **1) C PROGRAM:**

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

void forkexample()
{
pid_t p;
p = fork();
if(p<0)
{
perror("fork fail");
exit(1);
}
// child process because return value zero
else if (p==0)
printf("Hello from Child!\n");
// parent process because return value non-zero.
else
printf("Hello from Parent!\n");
}

int main()
{
forkexample();
 return 0;
}
```

- **OUTPUT:**



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

void forkexample()
{
pid_t p;
p = fork();
if(p<0)
{
perror("fork fail");
exit(1);
}
// child process because return value zero
else if (p==0)
printf("Hello from Child!\n");
// parent process because return value non-zero.
else
printf("Hello from Parent!\n");
}

int main()
{
forkexample();
 return 0;
}
```

```
/tmp/w5s25g7GwP.o
Hello from Parent!
Hello from Child!
```

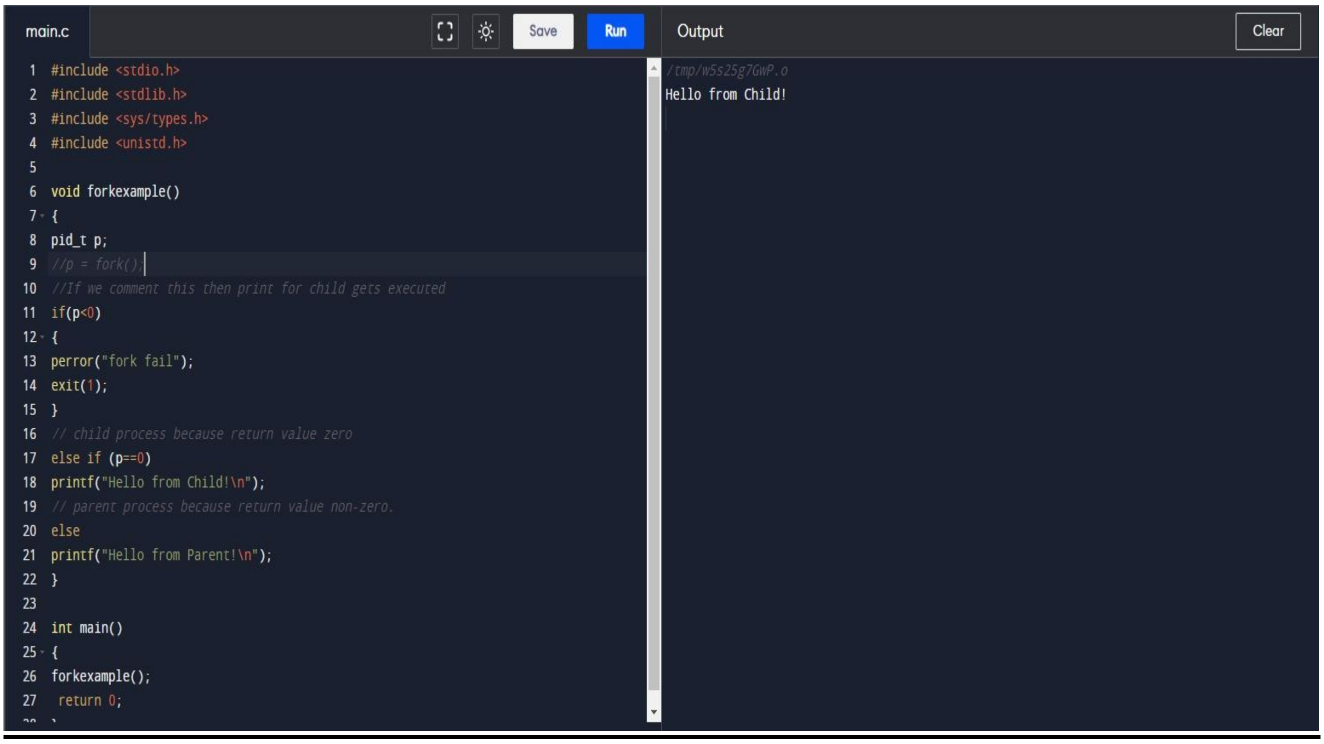
➢ **2) C PROGRAM:**

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

void forkexample()
{
pid_t p;
//p = fork();
//If we comment this then print for child gets executed
if(p<0)
{
perror("fork fail");
exit(1);
}
// child process because return value zero
else if (p==0)
printf("Hello from Child!\n");
// parent process because return value non-zero.
else
printf("Hello from Parent!\n");
}

int main()
{
forkexample();
 return 0;
}
```

- **OUTPUT:**



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

void forkexample()
{
pid_t p;
//p = fork();
//If we comment this then print for child gets executed
if(p<0)
{
perror("fork fail");
exit(1);
}
// child process because return value zero
else if (p==0)
printf("Hello from Child!\n");
// parent process because return value non-zero.
else
printf("Hello from Parent!\n");
}

int main()
{
forkexample();
 return 0;
```

```
/tmp/w5s25g7GwP.o
Hello from Child!
```

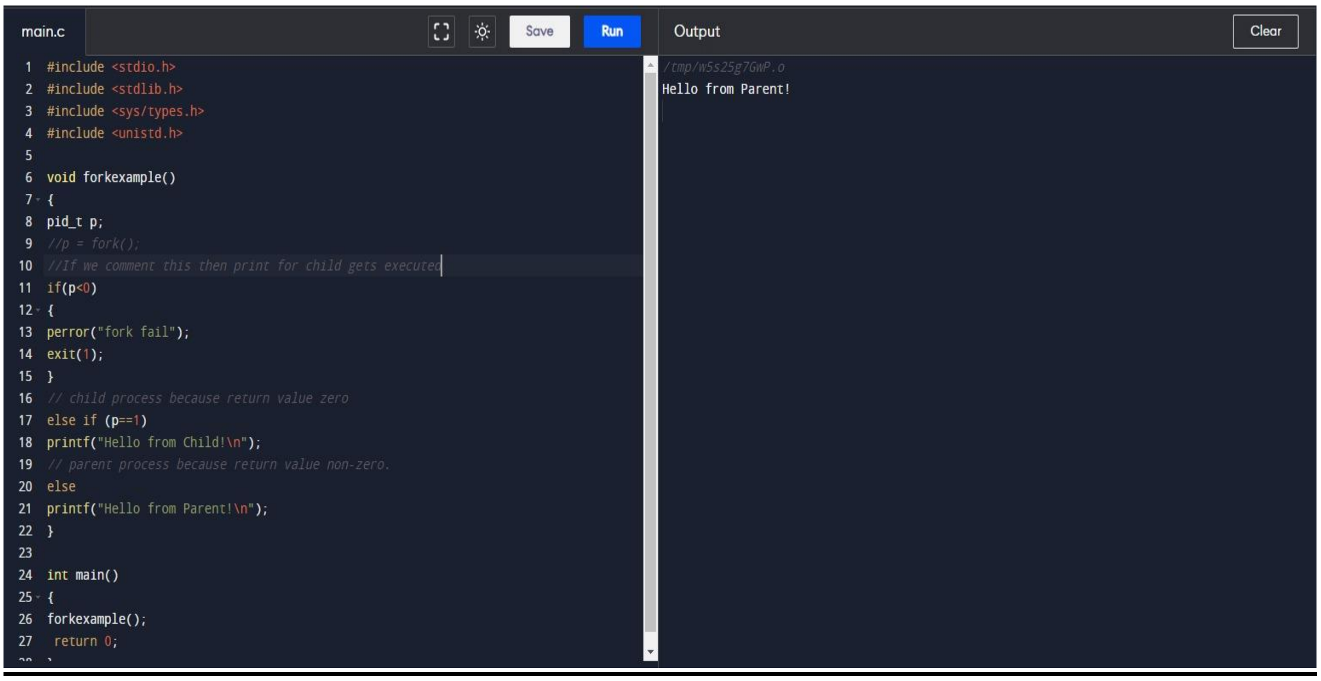
- **3) C PROGRAM:**

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

void forkexample()
{
pid_t p;
//p = fork();
//If we comment this then print for child gets executed
if(p<0)
{
perror("fork fail");
exit(1);
}
// child process because return value zero
else if (p==1)
printf("Hello from Child!\n");
// parent process because return value non-zero.
else
printf("Hello from Parent!\n");
}

int main()
{
forkexample();
 return 0;
}
```

- **OUTPUT:**



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

void forkexample()
{
pid_t p;
//p = fork();
//If we comment this then print for child gets executed
if(p<0)
{
perror("fork fail");
exit(1);
}
// child process because return value zero
else if (p==1)
printf("Hello from Child!\n");
// parent process because return value non-zero.
else
printf("Hello from Parent!\n");
}

int main()
{
forkexample();
 return 0;
```

```
/tmp/w5s25g7GwP.o
Hello from Parent!
```

➢ **4) C PROGRAM:**

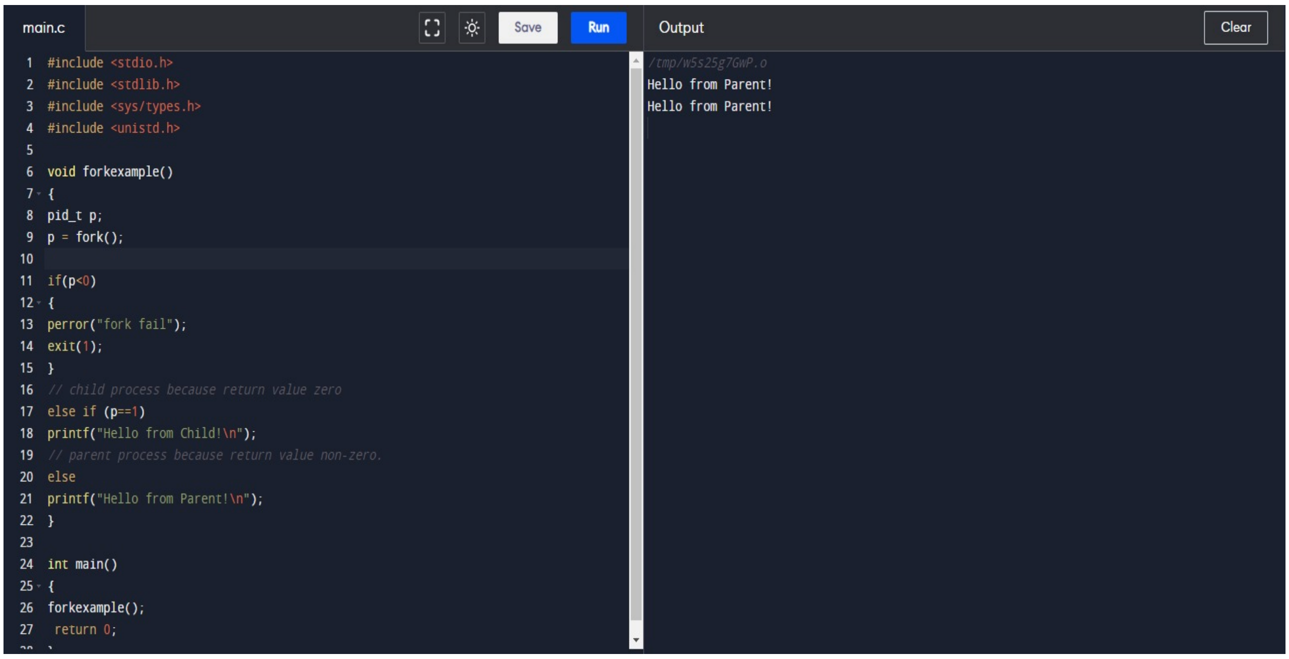
```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

void forkexample()
{
pid_t p;
p = fork();

if(p<0)
{
perror("fork fail");
exit(1);
}
// child process because return value zero
else if (p==1)
printf("Hello from Child!\n");
// parent process because return value non-zero.
else
printf("Hello from Parent!\n");
}

int main()
{
forkexample();
 return 0;
}
```

- **OUTPUT:**



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

void forkexample()
{
pid_t p;
p = fork();

if(p<0)
{
perror("fork fail");
exit(1);
}
// child process because return value zero
else if (p==1)
printf("Hello from Child!\n");
// parent process because return value non-zero.
else
printf("Hello from Parent!\n");
}

int main()
{
forkexample();
 return 0;
```

```
/tmp/w5s25g7GwP.o
Hello from Parent!
Hello from Parent!
```

<u>CONCLUSION:</u> Hence, we have successfully implemented and studied how to create child process in Linux using fork() system calls.