WEEK-3

1.

Aim: Demostrate System calls with simple example Program.

Program:

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
int main( ){
  pid_t child_pid;
  child_pid = fork (); // Create a new child process;
  if (child_pid < 0) {
    printf("fork failed");
    return 1;
  } else if (child_pid == 0) {
    printf ("child process successfully created!\n");
    printf ("child_PID = %d,parent_PID = %d\n",
    getpid(), getppid( ) );
  } else {
   //   wait(NULL);
    printf ("parent process successfully created!\n");
    printf ("child_PID = %d, parent_PID = %d", getpid( ), getppid( ) );
  }
  return 0;
}
```

OUTPUT:

```
parent process successfully created!
child process successfully created!
child_PID = 3978, parent_PID = 8child_PID = 3979,parent_PID = 3978
```

2.

Aim: Demostrate following I/O system calls with examples

(Refere Text book : Begining Linux Progrogramming Programmer to Programmer)

i) open

ii) read

iii) Write

iv)lseek

v)ioctl

vi)stat

vi)syn

Program:

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

```
#include<sys/stat.h>
#define WR_VALUE _IOW('a','a',int*)
#define RD_VALUE _IOR('a','b',int*)
int main()
{
   // if file does not have in directory
   // then file foo.txt is created.
   int fd = open("sample.txt", O_RDONLY | O_CREAT);
   int fd2 = open("file2.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
   int fd3 = open("file3.txt", O_RDWR);
   char c[20];
   // reading from sample.txt
   read(fd, &c, 20);
   printf("Text in file is: %s\n",c);
   //writing into file2.txt which was stored in c
   write(fd2, c, read(fd, &c, 20));
   lseek(fd, 5, SEEK_CUR);
   char c1[20];
   read(fd, &c1, 20);
   printf("Text after lseek 10 characters: %s",c1);
   printf("Enter a value to write in the file: ");
   int number;
   scanf("%d",&number);
   ioctl(fd3, WR_VALUE,(int*) &number);
   struct stat sfile;
   stat("stat.c", &sfile);
   printf("st_mode = %o\n",sfile.st_mode);
   close(fd);
   close(fd2);
   close(fd3);
   return 0;
}
```

OUTPUT:

```
Text in file is: This is a sample texf
Text after lseek 10 characters: This is third line
Enter a value to write in the file: 12
st_mode = 400
```

sample - Notepad

File Edit Format View Help

```
This is a sample text.
This is second line
This is third line
This is fourth line
```

file2 - Notepad

File Edit Format View Help

```
t.
This is second l
```

file3 - Notepad

File Edit Format View Help

```
12
```

3.

Aim: Demonstrate following File system call with examples

i) link, ii) unlink

```
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week2$ cat hello.txt
hello world
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week2$ link hello.txt final.txt
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week2$ cat final.txt
hello world
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week2$ ls
final.txt  hello.txt  text.txt
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week2$ unlink final.txxt
unlink: cannot unlink 'final.txxt': No such file or directory
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week2$ unlink final.txt
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week2$ ls
hello.txt  text.txt
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week2$
```

iii) mount

```
sohail@LAPTOP-P1GHM93V:~$ mount
rootfs on / type wslfs (rw,noatime)
none on /dev type tmpfs (rw,noatime,mode=755)
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,noatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,noatime)
devpts on /dev/pts type devpts (rw,nosuid,noexec,noatime,gid=5,mode=620)
none on /run type tmpfs (rw,nosuid,noexec,noatime,mode=755)
none on /run/lock type tmpfs (rw,nosuid,nodev,noexec,noatime)
none on /run/shm type tmpfs (rw,nosuid,nodev,noatime)
none on /run/user type tmpfs (rw,nosuid,nodev,noexec,noatime,mode=755)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,relatime)
tmpfs on /sys/fs/cgroup type tmpfs (rw,nosuid,nodev,noexec,relatime,mode=755)
cgroup on /sys/fs/cgroup/devices type cgroup (rw,nosuid,nodev,noexec,relatime,devices)
C:\ on /mnt/c type drvfs (rw,noatime,uid=1000,gid=1000,case=off)
D:\ on /mnt/d type drvfs (rw,noatime,uid=1000,gid=1000,case=off)
E:\ on /mnt/e type drvfs (rw,noatime,uid=1000,gid=1000,case=off)
sohail@LAPTOP-P1GHM93V:~$
```

iv)unmount

```
sohail@LAPTOP-P1GHM93V:~$ sudo umount -a
[sudo] password for sohail:
umount: /dev: target is busy.
umount: /: target is busy.
sohail@LAPTOP-P1GHM93V:~$
```

v) chmod

```
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab$ ls
armstrong.sh       decimalToBinary.sh  hello.sh   palindrome.sh  week2
base5Todecimal.sh  decimalTobase5.sh   hello.txt  prime.sh
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab$ ls -lrt
total 0
-rwxrwxrwx 1 sohail sohail  513 Sep 10 09:15 decimalTobase5.sh
-rwxrwxrwx 1 sohail sohail  567 Sep 10 09:15 base5Todecimal.sh
-rwxrwxrwx 1 sohail sohail  262 Sep 10 09:15 armstrong.sh
-rwxrwxrwx 1 sohail sohail  378 Sep 10 09:15 decimalToBinary.sh
-rwxrwxrwx 1 sohail sohail  191 Sep 10 09:15 prime.sh
-rwxrwxrwx 1 sohail sohail  218 Sep 10 10:25 palindrome.sh
-rwxrwxrwx 1 sohail sohail   23 Sep 10 20:52 hello.txt
-rwxrwxrwx 1 sohail sohail  133 Sep 10 21:03 hello.sh
drwxrwxrwx 1 sohail sohail 4096 Sep 25 12:05 week2
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab$
```

vi) chown

```
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week2$ sudo chown -c root hello.
txt
[sudo] password for sohail:
changed ownership of 'hello.txt' from sohail to root
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week2$ ls -lrt
total 0
-rwxrwxrwx 1 sohail sohail 11 Sep 25 12:07 hello.txt
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week2$
```

4.

Aim: Demonstrate how process is created and allocate memory in /proc folder with example Program.

Program:

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
int main(int argc, char *argv[])
{
int a;
int fd1=open("kk.txt",O_RDONLY,742);  //open system call
int fd2=open("kk1.txt",O_RDONLY,S_IRWXU|S_IRGRP|S_IXOTH);
printf(" fd1=%d,fd2=%d\n", fd1,fd2);
printf("Process Id=%d\n",getpid());
scanf("%d",&a);
}
```

OUTPUT:

Process Program execution output:

```
 fd1=3,fd2=4
Process Id=135
```

Proc folder showing creation of new processes output:

ls:

```
bin   dev  home  lib    lib64  media  opt   root  sbin  srv  tmp  var
boot  etc  init  lib32  libx32  mnt    proc  run   snap  sys  usr
```

proc ls:

```
1   124  8  bus      cmdline  filesystems  loadavg  mounts  self  sys  uptime   version_signature
10  7    9  cgroups  cpuinfo  interrupts   meminfo  net     stat  tty  version
```

ls:

```
1   135  7  9    cgroups  cpuinfo      interrupts  meminfo  net   stat  tty     version
10  136  8  bus  cmdline  filesystems  loadavg     mounts   self  sys   uptime  version_signature
```

WEEK – 4

1.
Aim: Demonstrate System calls with simple example Program

Program:

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include <stdlib.h>
int main()
{
    printf(" Before fork");
    int fd1=open("kk.txt",O_RDONLY,742);
//kk.txt is a shared resource to parent and child
process
    int a;   // a is shared  variable to parent and
child process
    printf(" Process Id of parent %d \n",getpid());
     int rf=fork();
     if(rf==0)   // Upon successful execution fork()
-->  Child process will execute below code in if
block
     {
          printf("Inside child process \n");
           printf(" Process Id of child pid= %d
rf=%d \n ",getpid(),rf);
           scanf("%d",&a);
           printf(" Inside child a value is a=%d",a);

     }
     else if (rf> 0) // Upon successful execution
fork() system call --> Parent process will exceute
below code in elseif block
     {
       printf("Inside parent process");
       printf(" Process Id of parent pid=%d rf=%d
\n",getpid(),rf);
         scanf("%d",&a);
         printf(" Inside parent a value is a=%d",a);
     }
     else{   // Upon uncusccessful execution
fork() system call Parent process will exceute
below code in else block
```

```
            printf(" Fork unsucessful \n Indide
parent and Process Id of parent pid= %d  rf=%d
\n  ",getpid(),rf);
      }
      printf("End of process %d", getpid());
    return 0;
}
```

OUTPUT:

```
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$ ./1
 Before fork Process Id of parent 23
Inside parent process Process Id of parent pid=23 rf=24
Inside child process
 Process Id of child pid= 24  rf=0
 20
 Inside parent a value is a=20End of process 23sohail@LAPTOP-P1GHM93V:/mnt/
```

In proc folder:

```
sohail@LAPTOP-P1GHM93V:/proc$ ls
1   34  49  bus      cmdline  filesystems  loadavg  mounts  self  sys  uptime   version_signature
10  35  9   cgroups  cpuinfo  interrupts   meminfo  net     stat  tty  version
sohail@LAPTOP-P1GHM93V:/proc$
```

2.

Aim: Write a Programt to demonstrate the use of exe system calls

Parent process --> Execute factorial of given number

Child process --> Execute prime number

PROGRAM :

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

int main()
{
    printf(" Before fork");
    int fd1=open("kk.txt",O_RDONLY,742);
//kk.txt is a shared resource to parent and child
process
    int a;   // a is shared  variable to parent and
child process
    printf(" Process Id of parent %d \n",getpid());
     int rf=fork();
```

```
    if(rf==0)   // Upon successful execution fork()
-->  Child process will execute below code in if
block
    {
        printf("Inside child process \n");
         printf(" Process Id of child pid= %d
rf=%d \n ",getpid(),rf);
       printf("Child process: to execute the prime
number Program\nEnter a number: ");
         scanf("%d",&a);
       int flag = 0;
         for(int i=2;i<=a/2;i++){
         if (a%i==0){
           flag=1;
           break;
         }
       }
       if(flag==0){
        printf("Given number %d is a prime
number\n",a);
       }
       else{
        printf("Given number %d is not a prime
number\n",a);
       }
    }
    else if (rf> 0) // Upon successful execution
fork() system call --> Parent process will exceute
below code in elseif block
    {
      printf("Inside parent process");
      printf(" Process Id of parent pid=%d rf=%d
\n",getpid(),rf);
    printf("Parent process: to execute the
factorial of a number\nEnter a number: ");
       scanf("%d",&a);
       int total = 1, i = 1;
     while(i<=a){
        total=total*i;
        i=i+1;
     }
     printf("Factorial value is: %d\n",total);
    }
    else{   // Upon uncusccessful execution
fork() system call Parent process will exceute
below code in else block
```

```
            printf(" Fork unsucessful \n Indide
parent and Process Id of parent pid= %d  rf=%d
\n  ",getpid(),rf);
      }
      printf("End of process %d\n", getpid());
    return 0;
}
```

OUTPUT:

```
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$ gcc prime_factorial.c -o prime_fact
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$ ./prime_fact
 Before fork Process Id of parent 56
Inside parent process Process Id of parent pid=56 rf=57
Inside child process
Parent process: to execute the factorial of a number
 Process Id of child pid= 57  rf=0
Enter a number:  Child process: to execute the prime number program
Enter a number: 11
Given number 11 is a prime number
End of process 57
10
Factorial value is: 3628800
End of process 56
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$
```

3.

Aim:

Write a program to demonstrate use following exe functions

i. int execl(const char* path, const char* arg, …)
ii. int execlp(const char* file, const char* arg, …)
iii. int execle(const char* path, const char* arg, …, char* const envp[])
iv. int execv(const char* path, const char* argv[])
v. int execvp(const char* file, const char* argv[])
vi. int execvpe(const char* file, const char* argv[], char *const envp[])

Program:

Exec.c

```
#include<stdio.h>
#include<unistd.h>
int main()
{
   int i;
   printf("I am Demo file called by execvp() ");
   printf("\n");
   return 0;
}
```

Execv.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
      char *args[]={"./exec",NULL};
      execv(args[0],args);
      printf("Ending-----");
   return 0;
}
```

Output:

```
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$ ./3_4
I am Demo file called by family of exec() functions
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$
```

Execvp.c

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
      char *args[]={"./3_6_demo",NULL};
      execvp(args[0],args);
      printf("Ending-----");
   return 0;
}
```

Output:

```
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$ gcc execvp.c -o execcvp
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$ ./execcvp
I am Demo file called by family of exec() functions
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$
```

4.

Aim: Write a Program to demonstrate the use of vfork()

PROGRAM :

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

int main()
{
    printf(" Before fork");
```

```
    int fd1=open("kk.txt",O_RDONLY,742);
//kk.txt is a shared resource to parent and child
process
    int a,b;   // a is shared  variable to parent and
child process
    printf(" Process Id of parent %d \n",getpid());
     int rf=vfork();
     if(rf==0)   // Upon successful execution fork()
-->  Child process will execute below code in if
block
     {
          printf("Inside child process \n");
           printf(" Process Id of child pid= %d
rf=%d \n ",getpid(),rf);
           scanf("%d",&b);
           printf(" Inside child b value is b=%d",b);

     }

     else if (rf> 0) // Upon successful execution
fork() system call --> Parent process will exceute
below code in elseif block
     {

       printf("Inside parent process");
       printf(" Process Id of parent pid=%d rf=%d
\n",getpid(),rf);
         scanf("%d",&a);
         printf(" Inside parent a value is a=%d",a);
     }
     else{   // Upon uncusccessful execution
fork() system call Parent process will exceute
below code in else block
              printf(" Fork unsucessful \n Indide
parent and Process Id of parent pid= %d  rf=%d
\n  ",getpid(),rf);
       }
       printf("End of process %d\n", getpid());
     return 0;
}
```

OUTPUT:

```
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$ gcc vfork.c -o 4
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$ ./4
 Before fork Process Id of parent 71
Inside child process
 Process Id of child pid= 72  rf=0
 32
 Inside child b value is b=32End of process 72
Inside parent process Process Id of parent pid=71 rf=72
72
 Inside parent a value is a=72End of process 71
*** stack smashing detected ***: terminated
Aborted (core dumped)
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$
```

5.

Aim: Write a Program to demonstrate Orphan process

PROGRAM :

```
#include<stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
   // Create a child process
   int pid = fork();

   if (pid > 0)
      printf("in parent process\n");

   // Note that pid is 0 in child process
   // and negative if fork() fails
   else if (pid == 0)
   {
      sleep(9);
      printf("in child process\n");
   }
```

OUTPUT:

```
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$ gcc orphan.c -o orphan
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$ ./orphan
in parent process
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$ in child process
```

6.

Aim: Write a Program to demonstrate Zombie process

PROGRAM :

```
#include <stdlib.h>
#include <sys/types.h>
#include  <sys/wait.h>
```

```
#include  <unistd.h>
#include  <stdio.h>
int main()
{
pid_t pid;
char *message;
int n;
int exit_code;
printf("fork Program starting\n");
pid = fork();
switch(pid)
{
case -1:
   perror("fork failed");
   exit(1);
case 0:
    message = "This is the child";
    n = 3;
    exit_code = 37;
break;
default:
    message = "This is the parent";
    n = 5;
    exit_code = 0;
break;
}
for(; n > 0; n--) {
puts(message);
sleep(1);
}
}
```

OUTPUT:

```
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$ gcc forkex.c -o forkex
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$ ./forkex
fork program starting
This is the parent
This is the child
This is the child
This is the parent
This is the child
This is the parent
This is the parent
This is the parent
sohail@LAPTOP-P1GHM93V:/mnt/e/college/5th sem/os lab/week 3/week 3$
```