Lab # 02

Linux System Calls

System Calls

System calls provide programs running on the computer an interface to talk with the operating system. When a program needs to ask for a service (for which it does not have permission itself) from the kernel of the operating system it uses a system call. User level processes do not have the same permissions as the processes directly interacting with the operating system. For example, to communicate with and external I/O device or to interact with any other processes, a program has to use system calls.

1. PROGRAM USING SYSTEM CALL fork(), wait(), execve(),exit(),perror(),getpid() and getppid()

AIM: To write the program to create a Child Process using system call fork ().

ALGORITHM:

1. Step 1: Declare the variable pid.
2. Step 2: Get the pid value using system call fork ().
3. Step 3 : If pid value is less than zero then print as "Fork failed".
4. Step 4 : Else if pid value is equal to zero include the new process in the system's file using execlp system call.
5. Step 5 : Else if pid is greater than zero then it is the parent process and it waits till the child completes using the system call wait()
6. Step 6: Then print "Child complete".

SYSTEM CALLS USED:

1. Fork ( )

Used to create new processes. The new process consists of a copy of the address space of the original process. The value of process id for the child process is zero, whereas the value of process id for the parent is an integer value greater than zero.

Syntax: fork ( )

2. execve ( )

Used after the fork () system call by one of the two processes to replace the process‟ memory space with a new program. It loads a binary file into memory destroying the memory image of the program containing the execlp system call and starts its execution. The child process overlays its address space with the UNIX command /bin/ls using the execlp system call.

Syntax: execve ( )

3. **wait( )**

The parent waits for the child process to complete using the wait system call. The wait system call returns the process identifier of a terminated child, so that the parent can tell which of its possibly many children has terminated.

Syntax: wait (NULL)

4. **exit( )**
   A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit system call. At that point, the process may return data (output) to its parent process (via the wait system call).
   Syntax: exit (0)

5. **getpid( )**
   Each process is identified by its id value**.** This function is used to get the id value of a particular process.

6. **getppid ( )**
   Used to get particular process parent's id value.

7. **perror( )**
   Indicate the process error

PROGRAM CODING:

Example 1:

```
#include<stdio.h>
 #include<stdlib.h>
#include<unistd.h>
void main(int argc,char *arg[])
 {
   printf("%d",argc);
   argv[0]="/bin/ls";
   int pid;
   pid=fork();
   if(pid<0)
   {
    printf("fork failed");
    exit(1);
    }
    else if(pid==0)
    {
```

```
        execve( argv[0],argv,NULL);
    }
    else
    {
      printf("\n Process id is -%d\n",getpid());
      wait(NULL);
       exit(0);
  } }
```

Example 2:

```
#include<stdio.h>
 #include<unistd.h>
#include<stdlib.h>
int main( )
{
 int pid;
pid=fork( );
if(pid== -1)
{
perror("fork failed");
exit(0);
}
if(pid==0)
{ printf("\n Child process is under execution");
printf("\n Process id of the child process is %d", getpid());
printf("\n Process id of the parent process is %d", getppid());
}
else
{
printf("\n Parent process is under execution");
printf("\n Process id of the parent process is %d", getpid());
printf("\n Process id of the child process in parent is %d", pid());
printf("\n Process id of the parent of parent is %d", getppid());
}
 return(0);
}
```

2. PROGRAM USING SYSTEM CALLS opendir( ) readdir( ) closedir()

AIM: To write the program to implement the system calls opendir( ), readdir( ).

ALGORITHM:

Step 1 : Start.
Step 2 : In the main function pass the arguments.

Step 3 : Create structure as stat buff and the variables as integer.
Step 4 : Using the for loop, initialization

PROGRAM CODING:

Example 1:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/dir.h>
void main(int age,char *argv[])
{
DIR *dir;
struct dirent *rddir;
printf("\n Listing the directory content\n");
dir=opendir(argv[1]);
while((rddir=readdir(dir))!=NULL)
{
printf("%s\t\n",rddir->d_name);
}
closedir(dir);
}
```

3. PROGRAM USING SYSTEM CALL stat( ), creat(),open( ), stat(),fstat( ),gets() and lseek()

File Permissions and Number?

File permissions determine what you are allowed to do and who is allowed to do it.

	Owner	Group	World
Read	✓	✓	✓
Write	✓	✗	✗

Execute	✔	✔	✔

- Read is equal to 4.
- Write is equal to 2.
- Execute is equal to 1.
- No permissions for a user is equal to 0.

PROGRAM CODING:

Example 1:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<fcntl.h>
void main()
{
int fd1,fd2,n;
char source[30],ch[5];
struct stat s,t,w;
fd1=creat("text.txt",0644);
printf("Enter the file to be copied\n");
scanf("%s",source);
fd2=open(source,O_RDONLY);
if(fd2==-1)
{
perror("file doesnot exist");
exit(0);
}
while((n=read(fd2,ch,1))>0)
write(fd1,ch,n);
close(fd2);
stat(source,&s);
printf("Source file size=%d\n",s.st_size);
fstat(fd1,&t);
printf("Destination file size =%d\n",t.st_size);
close(fd1);
}
```

Example 2:

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

```
#include<string.h>
#include<fcntl.h>
int main( )
{
int fd[2];
char buf1[25]= ”just a test\n”;
char buf2[50];
fd[0]=open(“file1”, O_RDWR);
fd[1]=open(“file2”, O_RDWR);
write(fd[0], buf1, strlen(buf1));
printf(“\n Enter the text now….”);
gets(buf1);
write(fd[0], buf1, strlen(buf1));
lseek(fd[0], SEEK_SET, 0);
read(fd[0], buf2, sizeof(buf1));
write(fd[1], buf2, sizeof(buf2));
close(fd[0]);
close(fd[1]);
printf(“\n”);
return0;
}
```

Value	Meaning
SEEK_SET	Offset is to be measured in absolute terms.
SEEK_CUR	Offset is to be measured relative to the current location of the pointer.
SEEK_END	Offset is to be measured relative to the end of the file.