NATIONAL INSTITUTE OF TECHNOLOGY ROURKELA

LABORATORY REPORT

OPERATING SYSTEM

Submitted to:

Prof. Arun Kumar

(Operating System)

------------------------------…….------------------------------

Submitted by:

Komal

(Roll No.:120CS0123)

(3rd year, Computer Science

and Engineering)

LAB-1

DATE- 03/08/2022

1.Write all Linux commands and its description.

COMMANDS

1. ***--help*** : Find help on any command
2. ***man*** : Show help (the manual)
3. **pwd** : Print Working Directory. Shows current location in the directory tree.
4. ***cd*** : Change directory, when typed alone, it returns to your home directory.
5. ***cd ..*** : Move up one directory
6. ***ls*** : List all files in the current directory
7. ***ls -a*** : List all files including hidden files
8. ***ls -l*** : List files in the long format, one file per line
9. ***ls -ld directory :*** A long list of directory, but instead showing the directory contents and detailed information.
10. ***cat*** : Display the contents of a text file on the screen
11. ***chmod*** : Change the permissions of a file or directory.
12. ***diff*** : Show the difference between two files
13. ***head*** : Display the first few lines of a text file. Example: head /etc/services
14. ***tail*** : Display the last few lines of a text file. Example: tail /etc/services
15. ***cp*** : Copies a file from one location to another
16. ***mv*** : Moves a file to a new location, or renames it
17. ***rm*** : Deletes a file
18. ***mkdir*** : Make Directory
19. ***rmdir*** : Remove Directory. Example: rmdir /tmp/myfiles/
20. ***locate*** : A quick way to search for files anywhere on the filesystem. For example, you can find all files and directories that contain the name “mozilla” by typing: locate mozilla
21. ***find*** : . It can be used to search for files matching certain patterns, as well as many other types of searches. A simple example is: find . -name *mp3
22. ***ps*** : Lists currently running processes (programs).
23. ***id*** : Print your user-id and group id’s
24. ***du*** : Disk Usage in a particular directory.
25. ***top*** : Displays CPU processes in a full-screen.
26. ***free*** : Displays amount of free and used memory in the system
27. ***clear*** : Clear the screen
28. ***echo*** : Display text on the screen. For example: echo “Hello World”

29. ***grep*** : Search for a pattern in a file or program output. For e.g - grep "nfs" /etc/services
    This looks for any line that contains the string "nfs" in the file "/etc/services" and displays only those lines

SPECIAL CHARACTERS

30. \ : If you want to reference a special character, you must "escape" it with a backslash first. Example: touch /tmp/filename\*
31. ***/*** : Directory separator
32. ***..*** : parent directory
33. ***~*** : Home directory
34. ******* : Represents 0 or more characters in a filename, or by itself, all files in a directory.
    Example: pic*2002 can represent the files pic2002, picJanuary2002, picFeb292002, etc.
35. ***?*** : Represents a single character in a filename.
    Example: hello?.txt can represent hello1.txt, helloz.txt, but not hello22.txt
36. ***[ ]*** : Can be used to represent a range of values, e.g. [0-9], [A-Z], etc.
    Example: hello[0-2].txt represents the names hello0.txt, hello1.txt, and hello2.txt
37. ***|*** : "Pipe". Redirect the output of one command into another command.
    Example: ls | more
38. ***>*** : Redirect output of a command into a new file. If the file already exists, over-write it.
    Example: ls > myfiles.txt
39. ***>>*** : Redirect the output of a command onto the end of an existing file.
    Example: echo "Mary 555-1234" >> phonenumbers.txt
40. ***<*** : Redirect a file as input to a program.
    Example: more < phonenumbers.txt
41. ***&&*** : Command separator as above, but only runs the second command if the first one
42. ***Up/Down Arrow Keys*** : Scroll through your most recent commands
43. ***TAB Completion*** : Auto Complete
44. ***Complete recent commands with "!"*** : Type "!" followed by the first couple of letters of a recent command and press ENTER
45. ***Search command history with CTRL-R*** : Press CTRL-R and then type any portion of a recent command. It will search the commands
46. ***Scrolling the screen with Shift-PageUp and Page Down*** : Scroll back and forward through your terminal.

LAB-2

DATE- 10/08/2022

1.)fork(), getpid(), getppid(), wait(0), wait(p,0,0) and related questions

_fork() with wait() command (Program-1)

```
#include <stdio.h>
#include <sys/types.h>
#include<sys/wait.h>
#include <unistd.h>
void main() {
    fork();
    wait(NULL);
    fork();
    wait(NULL);
    printf("My PID: %d, My Parent's PID: %d\n", getpid(), getppid());
}
```

Output:

```
My PID: 146, My Parent's PID: 145

My PID: 145, My Parent's PID: 144

My PID: 147, My Parent's PID: 144

My PID: 144, My Parent's PID: 9
```

fork() command (Program-2)

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

int main() {
    pid_t p = fork();
        if(p==-1){
        perror("fork");
        exit(1);
    }

    printf("%d\n",p);
    if(p==0) printf("I am a child\n");
    if(p>0) printf("I am Parent\n");
    return 0;
}
```

Output:

```
163
I am Parent
0
I am a child
```

<u>fork(), wait(), wait(p,0,0), getpid(), getppid() command (Program-3)</u>

```c
#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <stdlib.h>


int main() {

    int i;

    pid_t p=fork();

    if(p==-1) {

        perror("fork");

        exit(1);

    }


    if(p>0) {

        waitpid(p,0,0); //put this o=ut -- orphan child

        printf("\nI am parent of PID %d", p);

        sleep(3);

        exit(0);

    }

    for(i=0;i<5;i++) {

        printf("\nMy PID is %d", getpid());

        printf("\nMy parent PID is %d", getppid());

        sleep(1);

    }

}
```

Output:

```
My PID is 8598

My parent PID is 8597
My PID is 8598
My parent PID is 8597
My PID is 8598
My parent PID is 8597
My PID is 8598
My parent PID is 8597
My PID is 8598
My parent PID is 8597
I am parent of PID 8598
```

LAB-3

DATE- 17/08/2022

1.) write a program to implement the IPC using PIPE in which one process sends an array to another process and another process performs the sorting of the array and send back the sorted array to first process.

source code:

```
#include<stdio.h>

#include<errno.h>

#include<string.h>

#include<stdlib.h>

#include<unistd.h>


int main(){

    int fd[2],n;

    int arr[6]={5,2,3,1,8,4};

    int comp(const void*x,const void*y){
```

```
return *(int*)x > *(int*)y;
}

if(pipe(fd)==-1) exit(1);
else printf("PIPE created by successfully\n");

pid_t pid=fork();

if(pid<0) exit(1);

else if(pid>0){
        close(fd[0]);
        write(fd[1],arr,6);
        printf("UnSorted array:");
        int start=0;
        while(start<6) printf("%d ",arr[start++]);
        printf("\n");
}

else{
        close(fd[1]);
        n=read(fd[0],arr,6);
        qsort(arr,6,sizeof(int),comp);
        write(fd[1],arr,6);
        printf("Sorted array:");
        int start=0;
        while(start<6) printf("%d ",arr[start++]);
        printf("\n");
```

```
    }
}
```

output:

```
PIPE created by successfully
UnSorted array:5 2 3 1 8 4
Sorted array:1 2 3 4 5 8
```

LAB-4

DATE- 24/08/2022

1.) write a c program to illustrate use of fork()system call and check if variables in the parent process are shared with child process or not.

source code:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<stdlib.h>
int main(){
        int pid;
        pid=fork();
        if(pid<0) exit(1);
        else if(pid>0){
                int a=2;
                printf("hello from parent:%d\n",a);
        }
        else{
                int a=3;
                printf("hello from child:%d\n",a);
        }
}
```

output:

```
hello from parent:2
hello from child:3
```

2.) write a program to create a FULL process tree of depth 'D' and degree 'N'.For each process print the children PIDs and all its ancestors PIDs.The values of 'D' and 'N' are to read dynammically from the keyboard.

source code:

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

int main(){
        int depth=0;
        int D;
        int degree;
        int N;
        int pid;
        printf("Enter depth and degree:");
        scanf("%d%d",&D,&N);
        for(degree=0;depth<D-1 && degree<N;degree++){
            pid=fork();
            if(pid==0){
                depth++;
                printf("level:%d\n",depth);
                printf("child process id:%d\n",getpid());
                printf("parent process id:%d\n",getppid());
                degree=-1;
            }
        }
        waitpid(pid,0,0);
}
```

output:

```
Enter depth and degree:2 3
level:1
child process id:36551
parent process id:36528
level:1
child process id:36552
parent process id:36528
level:1
child process id:36553
parent process id:36528
```

3.) write a C program to create a process chain like p1->p2->....pn.
Here -> defines the parent->child.

source code:

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

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

#include<stdlib.h>

#include<sys/wait.h>


int main(){

        int node,ct=1;

        int pid;

        printf("Enter degree:");

        scanf("%d",&node);

        for(int i=1;i<node && ct==1;i++){

            ct=0;

            pid=fork();

            if(pid==0){

                ct=1;

                printf("%d\n",i);

                printf("child process id:%d\n",getpid());

                printf("parent process id:%d\n",getppid());

            }

        }

        wait(0);

}
```

output:

```
Enter degree:4

1

child process id:38242

parent process id:38225

2

child process id:38243

parent process id:38242
```

```
3
child process id:38244
parent process id:38243
```

4.)

source code:

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

int main(){
        int f1[2];
        char *z = "INITIAL";
        int p;
        if(pipe(f1) == -1) exit(1);
        p = fork();
            if(p == -1) exit(1);
            else if(p == 0){
                        printf("\nChild process : %d\n", getpid());
                        z = "CHANGED";
                        write(f1[1], &z, sizeof(z));
                                close(f1[1]);
        }
        else{
        //      wait(NULL);
                        printf("\nParent process: %d\n", getpid());
                        printf("Data before Modification :%s\n", z);
```

```
                read(f1[0], &z, sizeof(z));
                close(f1[0]);
                printf("Data after Modification :%s\n", z);
        }
}
```

output:

```
Parent process: 40995
Data before Modification :INITIAL
Child process : 40996
Data after Modification :CHANGED
```

LAB-5

DATE- 07/09/2022

1.)Write the description of all the exec() commands.

ANS:

The following are the syntaxes for each function of exec:

```
int execl(const char* path, const char* arg, …)
int execlp(const char* file, const char* arg, …)
int execle(const char* path, const char* arg, …, char* const envp[])
int execv(const char* path, const char* argv[])
int execvp(const char* file, const char* argv[])
int execvpe(const char* file, const char* argv[], char *const envp[])
```

2.) Write a program to execute a system call (ls,ls-l,ls-a) from main program using exec().

source code:

```
#include<stdio.h>
#include<unistd.h>
int main(){
    execl("/bln/ls","ls","-l","-a",NULL);
```

```
        printf("Exec failed\n");
}
```

output:

```
Exec failed
```

3.) write two programs ,make two ELF(executable and linkable format).Execute one ELF from another ELF.

source code(first ELF):

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

int main(){
        char*ptr[]={"./EXEC",NULL};
        printf("Before execv call,we are in main program\n");
        execv(ptr[0],ptr);
        printf("after execv nothing will execute,we are in another program\n");
}
```

source code(Second ELF):

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

int main(){
        int arr[5]={1,2,3,4,5};
        int sum=0;
        for(int i=0;i<5;i++) sum+=arr[i];
        printf("sum of array elements is %d\n",sum);
}
```

output:

```
Before execv call,we are in main program
sum of array elements is 15
```

LAB-6

DATE- 14/09/2022

1.) Implement one way continuous chat between two separately process.

source code:(second person)

```c
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
   int fd1;
   mkfifo("myfile2.txt", 0666);

   char str1[80];
   while (1)
   {
      fd1 = open("myfile2.txt",O_RDONLY);
      read(fd1, str1, 80);
      printf("Me: %s\n", str1);
```

```
        close(fd1);
    }
    return 0;
}
```

source code:(First person)

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
int main()
{
    int fd;
    mkfifo("myfile2.txt", 0666);
    char arr1[80];
    while (1){
        fd = open("myfile2.txt", O_WRONLY);
        gets(arr1);
        write(fd, arr1, strlen(arr1)+1);
        close(fd);
    }
    return 0;
}
```

output:

```
First person chat:
Hi second person
what are you doing?
```

```
How are you?
^C
Second person chat:
Me: Hi second person
Hi first person
Me: what are you doing?
Me: How are you?
Me: How are you?
^C
```

2.) Implement two way continuous chat between two separately process.

source code:(second person)

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
   int fd1;
   mkfifo("myfile.txt", 0666);

   char str1[80], str2[80];
   while (1)
   {
        fd1 = open("myfile.txt",O_RDONLY);
        read(fd1, str1, 80);
        printf("First  person : %s\n", str1);
        close(fd1);
        fd1 = open("myfile.txt",O_WRONLY);
        fgets(str2, 80, stdin);
        write(fd1, str2, strlen(str2)+1);
        close(fd1);
    }
    return 0;
}
```

source code:(First person)

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>

int main()
{
    int fd;
    mkfifo("myfile.txt", 0666);
    char arr1[80], arr2[80];
    while (1){
        fd = open("myfile.txt", O_WRONLY);
        fgets(arr2, 80, stdin);
        write(fd, arr2, strlen(arr2)+1);
        close(fd);

        fd = open("myfile.txt", O_RDONLY);

        read(fd, arr1, sizeof(arr1));

        printf("Second person: %s\n", arr1);

        close(fd);

    }

    return 0;

}
```

output:

```
First person chat:

Hi second person

Second  person : Hi first person


How are you?

Second  person : I am fine,How are you?

I am fine too.

^C

Second person chat:

First  person : Hi second person

Hi first person

First  person : How are you?

I am fine,How are you?

First  person : I am fine too.

^C
```