1. Execution of basic linux commands

man, mkdir, pwd, cd, cat, vi editor, rm, rmdir, mv

Solution:

man: The man (manual) command in Linux is used to display the user manual of any command that we can run on the terminal.

>> man ls:

This command is used to display the documentation on the 'ls'

command in Linux.

>>mkdir

The mkdir command in Linux is used to create new directories inside an existing working directory from the terminal.

>> mkdir [Option].. <Directory Name>

The best part about this command is that it can be simultaneously used to set user permissions pertaining to any directory you're creating directly from the terminal.

>> pwd

The pwd command in Linux translates to "Print Working Directory" and is used to display the path of the current working directory inside the terminal.

pwd: Displays the full path of the current working directory you're in.

>> cd

The cd command in Linux expands to 'change directory'. it is used to change the current working directory to a specified folder inside the terminal.

cd - This command is used to navigate to the parent directory of the current directory

>>cat

The cat command in Linux is used to create file and read the contents of one or more files and display their contents inside the terminal.

vi editor: The default editor that comes with the UNIX operating system is called vi (visual editor). Using vi editor, we can edit an existing file or create a new file from scratch. we can also use this editor to just read a text file.

$vi <file name> -------- to edit txt file

The vi editor has two modes:

Command Mode: In command mode, actions are taken on the file. The vi editor starts in command mode. Here, the typed words will act as commands in vi editor. To pass a command, you need to be in command mode.

Insert Mode: In insert mode, entered text will be inserted into the file. The Esc key will take you to the command mode from insert mode.

rm

The rm command in Linux helps you delete files and directories. To be more specific, rm deletes all references to objects from the file system, where those objects may have several references.

rm <<Filename>>: Used to delete the corresponding existing file from inside the working directory.

rm -i <<File name>>: The '-i' tag enables the rm command to request confirmation from the user before deleting the corresponding file.

$ rm -i "<Filename>"

rmdir <<Directory__Name>>:: Used to **delete any directory** provided it is empty.

rmdir

The rmdir command in Linux only allows you to delete empty directories. So if a directory has some files/folders inside it, rmdir will display an error.

Syntax:

$rmdir [Option] [Directory_Name]

rmdir <<Directory__Name>>:: Used to delete any directory provided it is empty.

mv

The mv command in Linux translates to 'move'. It performs two major functions in Linux:

• You can rename a file/directory using this command.

• You can easily move a file/directory from one location to another.

mv <<Initial Filename>> <<New__Filename>>:

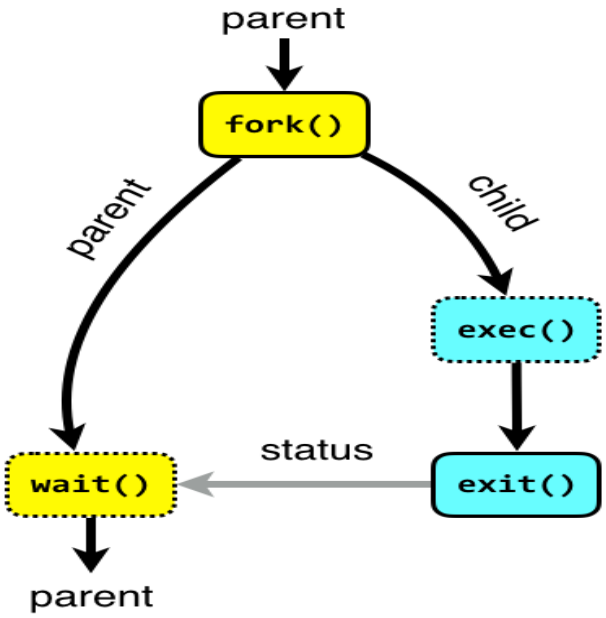
Used to rename a file to a different name as specified.

2. Familiarity and usage of Linux System calls:

fork(), exec(), exit(), wait(), sleep()

Solution:

The basic process management is done with a number of system calls, each with a simple purpose. These system calls can be combined to implement more complex behaviours.



fork(): A parent process uses fork() system call to create a new child process. After a successful fork, the child process is a copy of the parent. After fork(), both parent and child executes the same program but in separate processes.

Exec(): Replaces the program executed by a process. The child may use exec after a fork to replace the process' memory space with a new program executable making the child execute a different program than the parent.

Exit(): Terminates the process with an exit status.

Wait(): The parent may use wait to suspend execution until a child terminates. Using wait the parent can obtain the exit status of a terminated child.

Sleep(): A sleep system call takes a time value as a parameter, specifying the minimum amount of time that the process is to sleep before resuming execution.

The parameter typically specifies seconds, although some operating systems provide finer resolution, such as milliseconds or microseconds.

Pfork.c

```
#include <stdio.h>
#include <unistd.h>
int main()
{
// make two process which run same
// program after this instruction
fork();
printf("Hello world!\n");
return 0;
}
```

Pfork1.c

```
#include<stdio.h>
#include<unistd.h>
main(){
int pid,pid1,pid2;
pid=fork();
if(pid==-1)
{
        printf(“ERROR IN PROCESS CREATION \n”);
        exit(1);
}
if(pid!=0)
{
        pid1=getpid();
```

```
        printf(“\n the parent process ID is %d\n”, pid1);
}
else{
        pid2=getpid();
        printf(“\n the child process ID is %d\n”, pid2);
}
}
```

3. Write a C program that reads file.txt line by line and prints the first 10 digit numbers in the given file(digits should be continuous), If not found then print the first 10 characters excluding numbers.

Solution:

$cat>myfile.txt

1234567891 abcdefghijklmn

CTRL+D ------→ to SAVE the file

```
$cat>filename.c
#include <stdio.h>
int main()
{
 //Reading the file
 FILE *fp = fopen("myfile.txt" , "r");
int count,i;
int c=0;

while(1)
 {
     count = 0;
     for(i=0;i<10;i++)
     {
         c = getc(fp);
         if( c == -1)
         break;
         if(c>='0' && c<='9')
         count++;
     }
     if(c == -1)
       break;
     if(count==10)
       {
         char s[10];
         fseek(fp,-10,SEEK_CUR);// sets the file position to the given offset
         fgets(s,11,fp);
         printf("%s",s);
         break;
```

```
        }
    }

if (count!=10)
 {
        rewind(fp);//sets the position to the beginning of the file
        for(i=0;i<10;)
        {
                c = getc(fp);
                if(c>='0' && c<='9')
                continue;
                printf("%c",c);
                i++;
        }
}
   return 0;
}
```

To Compile and Execute:

```
$  gcc filename.c

$  ./a.out
```

4. Write a C / python code to implement First come first serve CPU scheduling algorithm.

Solution:

FCFS without Arrival Time

```
#include<stdio.h>
int main()
{
	// Create variable for n process and BT values in integer
	int n,b[10];
	// Create variable for initially CT, TAT and WT are Zero
	int ct=0,wt=0,tat=0;
	// i using "for loop"
	int i;
	// Create variable for ATAT and AWT values in float
	float awt,atat;
	// Create tempariry variables
		// "temp" for sum of WT values but intially "temp" is zero
	  // "temp1" for sum of TAT values but intially "temp1" is zero
	int temp=0,temp1=0;

	printf("Enter no of processes:\n");
	scanf("%d",&n);

	printf("\nEnter burst time:\n");
	for(i=1;i<=n;i++)
		scanf("%d",&b[i]);

	printf("\n Note: Assume all process AT is 0.\n");

	printf("\nGantt Chart::\n\n");
	printf("-------------------------------------------\n");
	for(i=1;i<=n;i++)
		printf("P%d\t| ",i);
	printf("\n-------------------------------------------\n");

	printf("\nTable form::");
	printf("\n-----------------------------------------------------");
	printf("\nProcess|  BT | CT  |  WT  |  TAT |\n");
	printf("-----------------------------------------------------\n");
	for(i=1;i<=n;i++)
	{
		ct = ct + b[i];
		tat = ct;   // or tat = tat + b[i];
		wt = tat - b[i];
		printf("P%d   |  %d  |  %d   |  %d  |  %d  |\n",i,b[i],ct,wt,tat);
		temp = temp + wt;
		temp1 = temp1 + tat;
	}
```

```
        printf("-------------------------------------------------\n");

        awt = (float)temp/n;
        atat = (float)temp1/n;
        printf("\n Average Waiting Time(AWT)= %f",awt);
        printf("\n Average Turn Arround Time(ATAT)= %f",atat);
        return 0;
}
```

5. Write a C / python code to implement Shortest job first CPU scheduling algorithm.

Solution:

SJF using Non-preemitive mode without Arrival Time

```
#include<stdio.h>
int main()
{
        int n,p[20],bt[20],wt[20],tat[20];
        int i,j;
        int temp1=0,temp2=0;
        int pos;
        float avg_wt,avg_tat;

        printf("Enter number of process:");
        scanf("%d",&n);

        printf("\nEnter Burst Time:\n");
        for(i=0;i<n;i++)
        {
                printf("p%d:",i+1);
                scanf("%d",&bt[i]);
                p[i]=i+1;          //it contains process number for table form
        }
   //sorting burst time in ascending order using selection sort
        int temp;
        for(i=0;i<n;i++)
        {
                pos=i;
                for(j=i+1;j<n;j++)
                {
                        if(bt[j]<bt[pos])
                        pos=j;
                }
                // BurstTime moving one position to another position
      temp=bt[i];
                bt[i]=bt[pos];
                bt[pos]=temp;
                // at the same time process also
      temp=p[i];
                p[i]=p[pos];
                p[pos]=temp;
        }

        //calculate waiting time

   wt[0]=0;   //waiting time for first process will be zero
```

```
        for(i=1;i<n;i++)
        {
                wt[i]=0;

                for(j=0;j<i;j++)
                        wt[i]+=bt[j];

      temp1+=wt[i];
   }

   avg_wt=(float)temp1/n;   //average waiting time

   printf("\nProcess\t    Burst Time    \tWaiting Time\t Turnaround Time");
        for(i=0;i<n;i++)
        {
                tat[i]=bt[i]+wt[i];  //calculate turnaround time
                temp2+=tat[i];
                printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
   }
        avg_tat=(float)temp2/n;     //average turnaround time
        printf("\n\nAverage Waiting Time=%f",avg_wt);
        printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```