ASSINGMENT 1

1. Linux Commands –
2. echo: - **echo** command in linux is used to display line of text/string that are passed as an argument. This is a built-in command that is mostly used in shell scripts and batch files to output status text to the screen or a file.



1. pwd: - **pwd** stands for **P**rint **W**orking **D**irectory. It prints the path of the working directory, starting from the root.



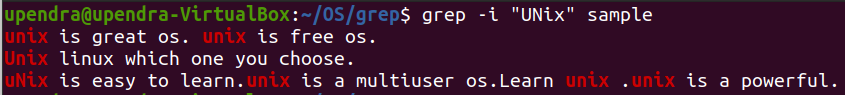
1. cd: - **cd** command in linux known as change directory command. It is used to change current working directory.



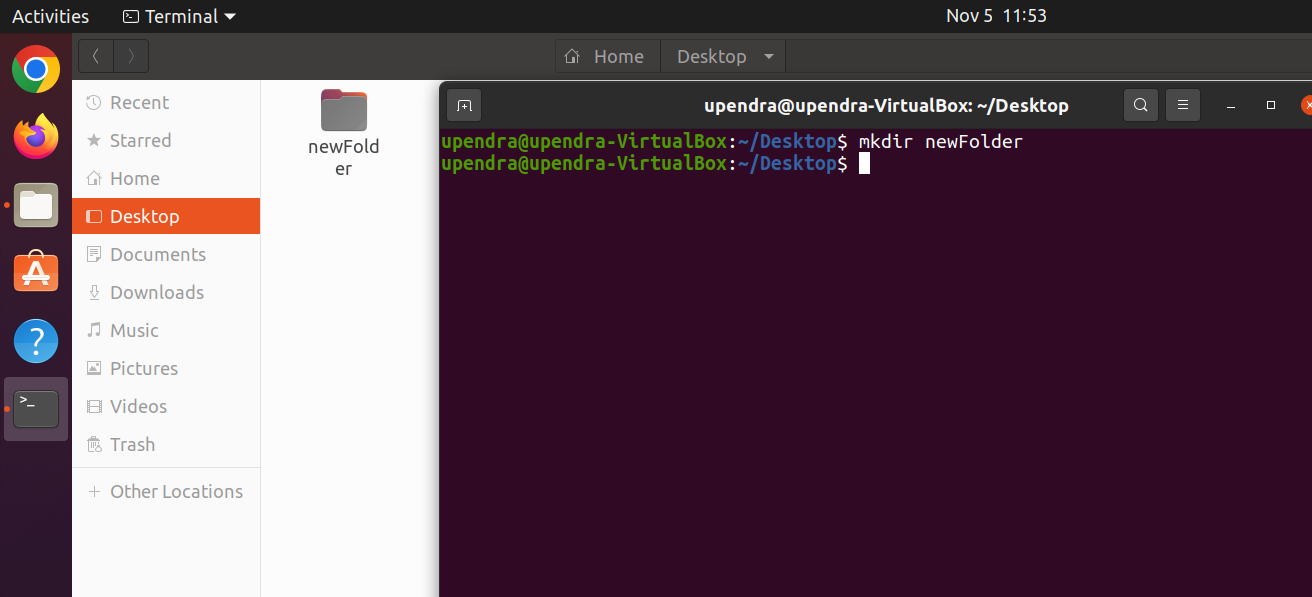
1. find: - The **find** command in UNIX is a command line utility for walking a file hierarchy. It can be used to find files and directories and perform subsequent operations on them. It supports searching by file, folder, name, creation date, modification date, owner and permissions. By using the ‘-exec’ other UNIX commands can be executed on files or folders found.



1. grep: - The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for global search for regular expression and print out).



1. mkdir: - **mkdir** command in Linux allows the user to create directories (also referred to as folders in some operating systems ). This command can create multiple directories at once as well as set the permissions for the directories.



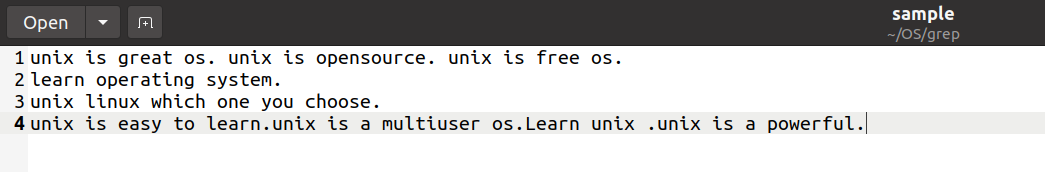
1. rmdir: - **rmdir** command is used remove empty directories from the filesystem in Linux. The rmdir command removes each and every directory specified in the command line only if these directories are empty. So if the specified directory has some directories or files in it then this cannot be removed by *rmdir*command.



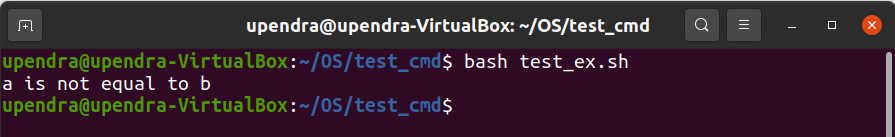
1. mv: - **mv** stands for **move**. mv is used to move one or more files or directories from one place to another in a file system like UNIX. It has two distinct functions:   
   **(i)** It renames a file or folder.   
   **(ii)** It moves a group of files to a different directory.



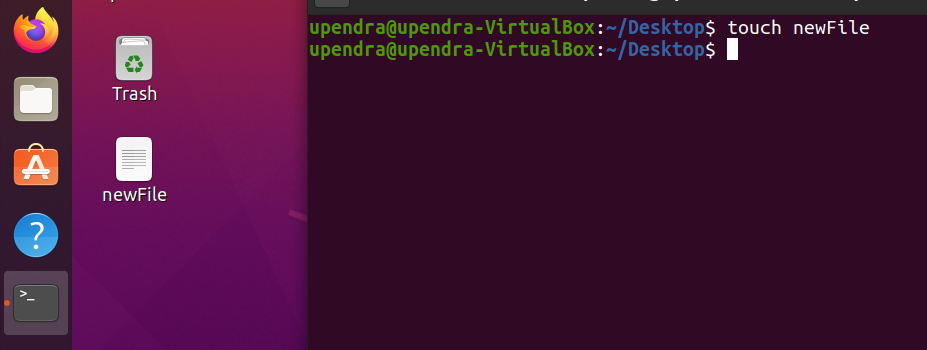
1. sed: - SED command in UNIX stands for stream editor and it can perform lots of functions on file like searching, find and replace, insertion or deletion. Though most common use of SED command in UNIX is for substitution or for find and replace. By using SED you can edit files even without opening them, which is much quicker way to find and replace something in file, than first opening that file in VI Editor and then changing it.
   * SED is a powerful text stream editor. Can do insertion, deletion, search and replace(substitution).
   * SED command in unix supports regular expression which allows it perform complex pattern matching.



1. test: - A test command is a command that is used to test the validity of a command. It checks whether the command/expression is true or false. It is used to check the type of file and the permissions related to a file. Test command returns 0 as a successful exit status if the command/expression is true, and returns 1 if the command/expression is false.



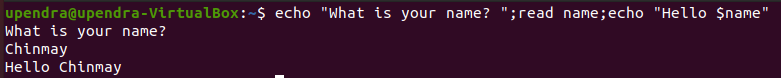
1. touch: - The touch command is a standard command used in UNIX/Linux operating system which is used to create, change and modify timestamps of a file. Basically, there are two different commands to create a file in the Linux system which is as follows:
   * cat command: It is used to create the file with content.
   * touch command: It is used to create a file without any content. The file created using touch command is empty. This command can be used when the user doesn’t have data to store at the time of file creation.



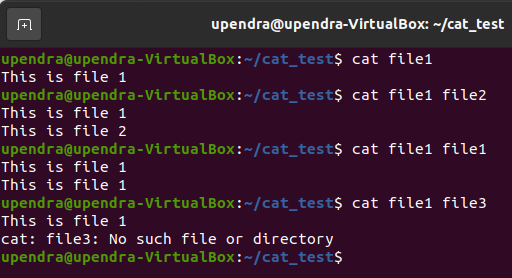
1. ls: - ls is a Linux shell command that lists directory contents of files and directories. Some practical examples of ls command are shown below.



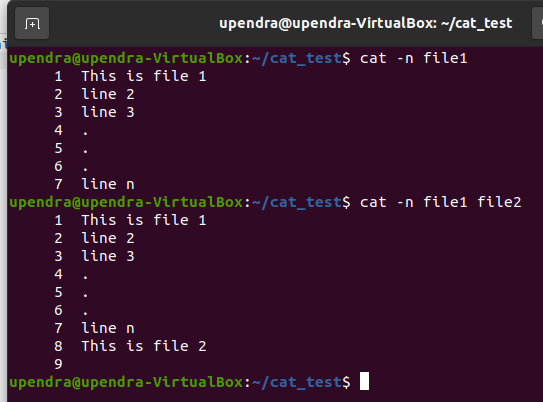
1. read: - read command in Linux system is used to read from a file descriptor. Basically, this command read up the total number of bytes from the specified file descriptor into the buffer. If the number or count is zero then this command may detect the errors. But on success, it returns the number of bytes read. Zero indicates the end of the file. If some errors found then it returns -1.



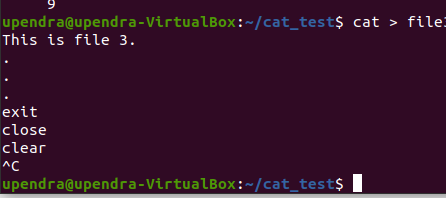
1. cat: - Cat(concatenate) command is very frequently used in Linux. It reads data from the file and gives their content as output. It helps us to create, view, concatenate files. So let us see some frequently used cat commands.



//concatenate files

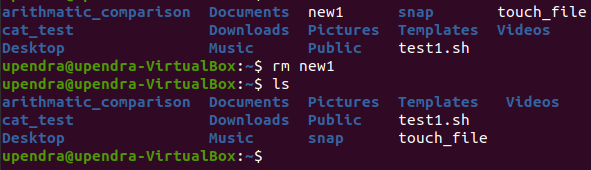


//add numbers on file content.



//create and write in the file.

1. rm: - rm stands for remove here. rm command is used to remove objects such as files, directories, symbolic links and so on from the file system like UNIX.



1. Arithmetic Comparison: -

Code:

#!/bin/sh

a=50

b=10

val=`expr $a + $b`

echo "$a + $b : $val"

val=`expr $a - $b`

echo "$a - $b : $val"

val=`expr $a \\* $b`

echo "$a \* $b : $val"

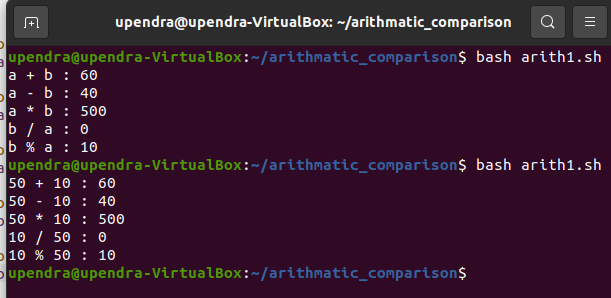
val=`expr $a / $b`

echo "$b / $a : $val"

val=`expr $b % $a`

echo "$b % $a : $val"

Output:



1. Shell Program –

Program Code: -

opt=1

while [ "$opt" -lt 7 ]

do

echo -e "Choose one of the Following\n1. Create a New Address Book\n2. View Records\n3. Insert new Record\n4. Delete a Record\n5. Modify a Record\n6. Exit"

# echo -e, enables special features of echo to use \n \t \b etc.

read opt

case $opt in

1)

echo "Enter filename"

read fileName

if [ -e $fileName ] ; then # -e to check if file exists, if exits remove the file

rm $fileName

fi

cont=1

echo "NAME\t NUMBER\t\tADDRESS\n===============================\n" | cat >> $fileName

while [ "$cont" -gt 0 ]

do

echo "Enter Name:"

read name

echo "Enter Phone Number of $name"

read number

echo "Enter Address of $name"

read address

echo "$name\\t$number\\t$address" | cat >> $fileName

echo "Enter 0 to Stop, 1 to Enter next"

read cont

done

;;

2)

cat $fileName

;;

3)

echo "\nEnter Name"

read name

echo "Enter Phone Number of $name"

read number

echo "Enter Address of $name"

read address

echo "$name\t$number\t\t$address" | cat >> $fileName

;;

4)

echo "Enter address name"

read name

grep -v $name

;;

5)

echo "Delete record\nEnter Name/Phone Number"

read pattern

temp="temp"

grep -v $pattern $fileName | cat >> $temp

rm $fileName

cat $temp | cat >> $fileName

rm $temp

;;

6)

echo "Modify record\nEnter Name/Phone Number"

read pattern

temp="temp"

grep -v $pattern $fileName | cat >> $temp

rm $fileName

cat $temp | cat >> $fileName

rm $temp

echo "Enter Name"

read name

echo "Enter Phone Number of $name"

read number

echo "Enter Address of $name"

read address

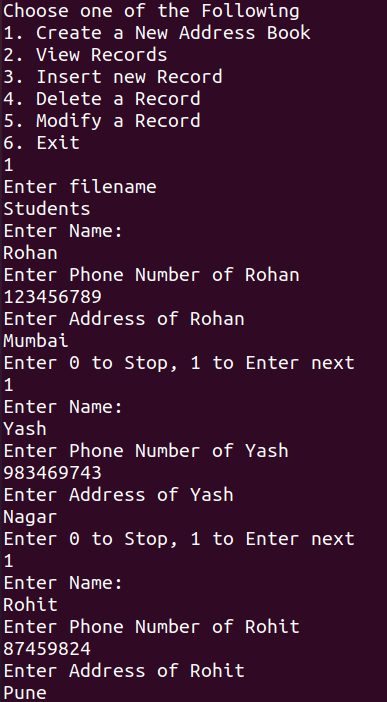
echo -e "$name\t$number\t$address" | cat >> $fileName

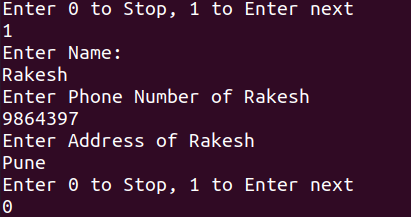
;;

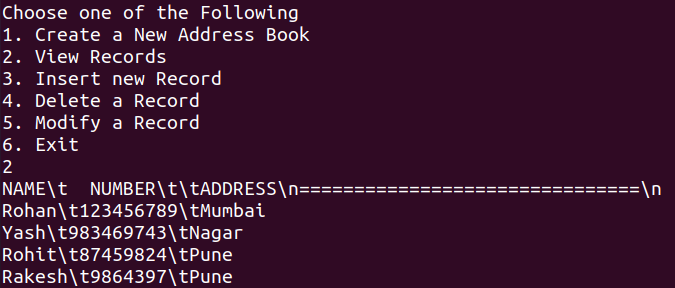
esac

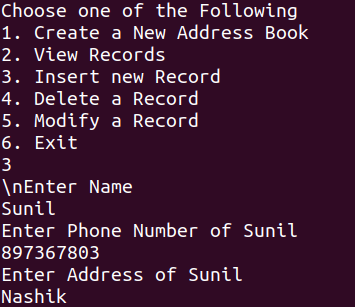
done

Output: -

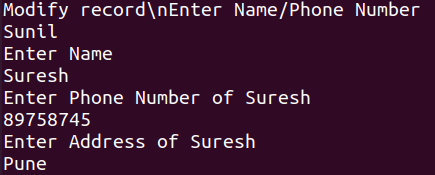


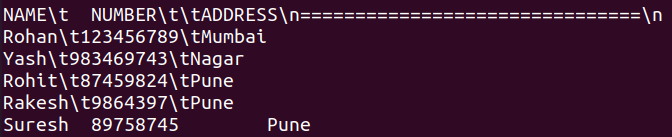






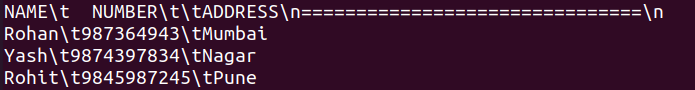
Modify



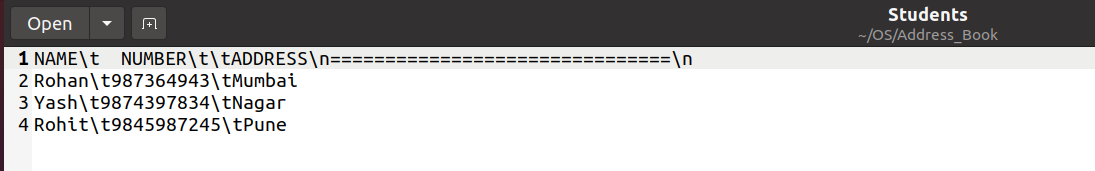


Delete





File



ASSINGMENT 2

1. Zombie Process: -

Program Code: -

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

#define MAX 20

void quicksort(int a[], int, int);

void merge(int a[], int low, int mid, int high);

void divide(int a[], int low, int high);

void zombie(pid\_t pid, int a[], int n, int i) {

printf("\n\tEnter the no. of elements: ");

scanf("%d", & n);

printf("\n\tEnter the elements: \n");

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

printf("\t");

scanf("%d", & a[i]);

}

pid = fork();

if (pid < 0) {

printf("Error While creating a new process.....!!!!!!");

} else if (pid == 0) {

printf("\n\t==============Child process started=============");

printf("\n\tI am a child process with pid=%d and ppid=%d", getpid(), getppid());

quicksort(a, 0, n - 1);

printf("\n\n\tSorted array by quick sort:\n\t");

for (i = 0; i < n; i++)

printf("%d\t", a[i]);

printf("\n");

printf("\n\t==============Child process terminated=============\n");

} else {

wait();

printf("\n\t==============Parent process started=============");

printf("\n\n\tI am a parent process with pid=%d ", getpid());

divide(a, 0, n - 1);

printf("\n\n\tSorted array by merge sort:\n\t");

for (i = 0; i < n; i++)

printf("%d\t", a[i]);

printf("\n");

printf("\n\t==============Parent process terminated=============\n");

}

}

void orphan(pid\_t pid, int a[], int n, int i) {

printf("\n\tEnter the no. of elements: ");

scanf("%d", & n);

printf("\n\tEnter the elements: \n");

for (i = 0; i < n; i++)

{

printf("\t");

scanf("%d", & a[i]);

}

pid = fork();

if (pid < 0)

{

printf("Error While creating a new process.....!!!!!!");

} else if (pid == 0)

{

wait();

printf("\n\t==============Child process started=============");

printf("\n\tI am a child process with pid=%d and ppid=%d", getpid(), getppid());

quicksort(a, 0, n - 1);

printf("\n\n\tSorted array by quick sort:\n\t");

for (i = 0; i < n; i++)

printf("%d\t", a[i]);

printf("\n");

printf("\n\t==============Child process terminated=============\n");

} else {

printf("\n\t==============Parent process started=============");

printf("\n\n\tI am a parent process with pid=%d ", getpid());

divide(a, 0, n - 1);

printf("\n\n\tSorted array by merge sort:\n\t");

for (i = 0; i < n; i++)

printf("%d\t", a[i]);

printf("\n");

printf("\n\t==============Parent process terminated=============\n");

}

}

int main()

{

pid\_t pid;

int a[MAX], n;

int i, op;

printf("\n\tEnter 1) for zombie and 2) for orphan ");

scanf("%d", & op);

if (op == 1) {

zombie(pid, a, n, i);

} else if (op == 2) {

orphan(pid, a, n, i);

}

execl("/bin/ps", "ps", NULL);

return 0;

}

void quicksort(int a[MAX], int first, int last)

{

int pivot, j, i, temp;

if (first < last)

{

i = first;

j = last;

pivot = first;

while (i < j)

{

while (a[i] <= a[pivot] && i < last)

i++;

while (a[j] > a[pivot])

j--;

if (i < j)

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

}

}

temp = a[j];

a[j] = a[pivot];

a[pivot] = temp;

quicksort(a, first, j - 1);

quicksort(a, j + 1, last);

}

}

void divide(int a[MAX], int low, int high)

{

if (low < high)

{

int mid = (low + high) / 2;

divide(a, low, mid);

divide(a, mid + 1, high);

merge(a, low, mid, high);

}

}

void merge(int a[MAX], int low, int mid, int high)

{

int i, j, k, m = mid - low + 1, n = high - mid;

int first\_half[m], second\_half[n];

for (i = 0; i < m; i++)

first\_half[i] = a[low + i];

for (i = 0; i < n; i++)

second\_half[i] = a[mid + i + 1];

i = j = 0;

k = low;

while (i < m || j < n)

{

if (i >= m)

{

a[k++] = second\_half[j++];

continue;

}

if (j >= n)

{

a[k++] = first\_half[i++];

continue;

}

if (first\_half[i] < second\_half[j])

a[k++] = first\_half[i++];

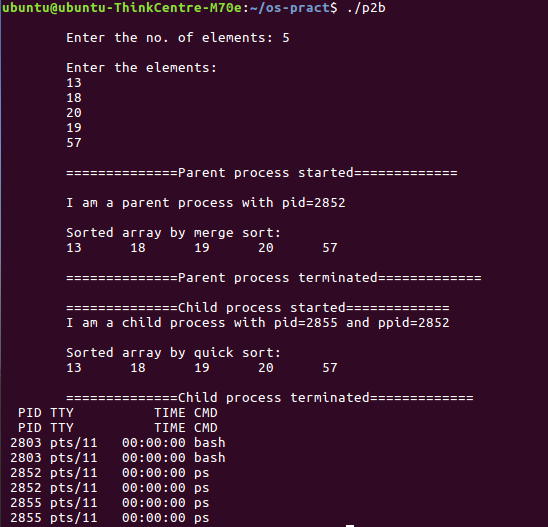
else

a[k++] = second\_half[j++];

}

}

Output: -



1. SYS CALL

Program Code:

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

void bass(int arr[30], int n)

{

int i, j, temp;

for (i = 0; i < n; i++)

{

for (j = 0; j < n - 1; j++)

{

if (arr[j] > arr[j + 1])

{

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

printf("\n Ascending Order \n");

for (i = 0; i < n; i++)

printf("\t%d", arr[i]);

printf("\n\n\n");

}

void bdsc(int arr[30], int n)

{

int i, j, temp;

for (i = 0; i < n; i++)

{

for (j = 0; j < n - 1; j++)

{

if (arr[j] < arr[j + 1])

{

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

printf("\n Descending Sorting \n\n");

for (i = 0; i < n; i++)

printf("\t%d", arr[i]);

printf("\n\n\n");

}

void forkeg()

{

int arr[25], n, i, status;

printf("\nEnter the no of values in array: ");

scanf("%d", &n);

printf("\nEnter the array elements: ");

for (i = 0; i < n; i++)

scanf("%d", &arr[i]);

int pid = fork();

if (pid == 0)

{

sleep(10);

printf("\nchild process\n");

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

bdsc(arr, n);

for (i = 0; i < n; i++)

printf("%d,", arr[i]);

printf("\b");

printf("\nparent process id=%d\n", getppid());

}

else

{

printf("\nparent process\n");

printf("\nparent process id=%d\n", getppid());

bass(arr, n);

for (i = 0; i < n; i++)

printf("%d,", arr[i]);

printf("\n\n\n");

}

}

int main()

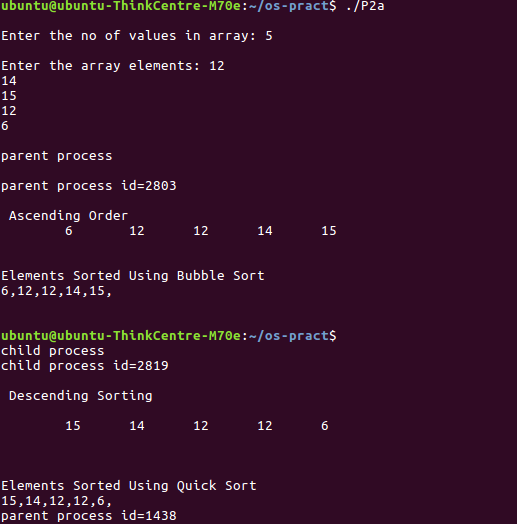
{

forkeg();

return 0;

}

Output:



ASSINGMENT 3

1. SJF

Program Code: -

#include <stdio.h>

int main()

{

int A[100][4]; // Matrix for storing Process Id, Burst

// Time, Average Waiting Time & Average

// Turn Around Time.

int i, j, n, total = 0, index, temp;

float avg\_wt, avg\_tat;

printf("Enter number of process: ");

scanf("%d", &n);

printf("Enter Burst Time:\n");

// User Input Burst Time and alloting Process Id.

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

printf("P%d: ", i + 1);

scanf("%d", &A[i][1]);

A[i][0] = i + 1;

}

// Sorting process according to their Burst Time.

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

index = i;

for (j = i + 1; j < n; j++)

if (A[j][1] < A[index][1])

index = j;

temp = A[i][1];

A[i][1] = A[index][1];

A[index][1] = temp;

temp = A[i][0];

A[i][0] = A[index][0];

A[index][0] = temp;

}

A[0][2] = 0;

// Calculation of Waiting Times

for (i = 1; i < n; i++) {

A[i][2] = 0;

for (j = 0; j < i; j++)

A[i][2] += A[j][1];

total += A[i][2];

}

avg\_wt = (float)total / n;

total = 0;

printf("P BT WT TAT\n");

// Calculation of Turn Around Time and printing the

// data.

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

A[i][3] = A[i][1] + A[i][2];

total += A[i][3];

printf("P%d %d %d %d\n", A[i][0],

A[i][1], A[i][2], A[i][3]);

}

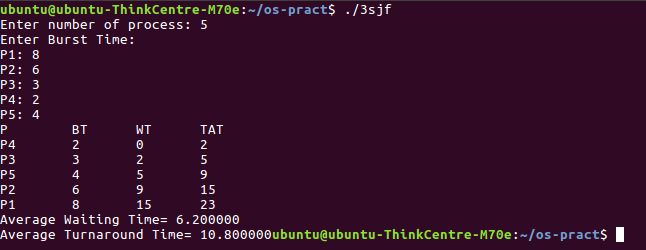
avg\_tat = (float)total / n;

printf("Average Waiting Time= %f", avg\_wt);

printf("\nAverage Turnaround Time= %f", avg\_tat);

}

Output: -



1. RR

Program Code: -

#include<stdio.h>

int main()

{

int i, limit, total = 0, x, counter = 0, time\_quantum;

int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10], temp[10];

float average\_wait\_time, average\_turnaround\_time;

printf("Enter Total Number of Processes:\n\t");

scanf("%d", &limit);

x = limit;

for(i = 0; i < limit; i++)

{

printf("Enter Details of Process[%d]\n", i + 1);

printf("Arrival Time:\t");

scanf("%d", &arrival\_time[i]);

printf("Burst Time:t");

scanf("%d", &burst\_time[i]);

temp[i] = burst\_time[i];

}

printf("Enter Time Quantum:\n\t");

scanf("%d", &time\_quantum);

printf("\nProcess IDttBurst Timet Turnaround Timet Waiting Timen");

for(total = 0, i = 0; x != 0;)

{

if(temp[i] <= time\_quantum && temp[i] > 0)

{

total = total + temp[i];

temp[i] = 0;

counter = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - time\_quantum;

total = total + time\_quantum;

}

if(temp[i] == 0 && counter == 1)

{

x--;

printf("\nProcess[%d]\t%d\t %d\t %d", i + 1, burst\_time[i], total - arrival\_time[i], total - arrival\_time[i] - burst\_time[i]);

wait\_time = wait\_time + total - arrival\_time[i] - burst\_time[i];

turnaround\_time = turnaround\_time + total - arrival\_time[i];

counter = 0;

}

if(i == limit - 1)

{

i = 0;

}

else if(arrival\_time[i + 1] <= total)

{

i++;

}

else

{

i = 0;

}

}

average\_wait\_time = wait\_time \* 1.0 / limit;

average\_turnaround\_time = turnaround\_time \* 1.0 / limit;

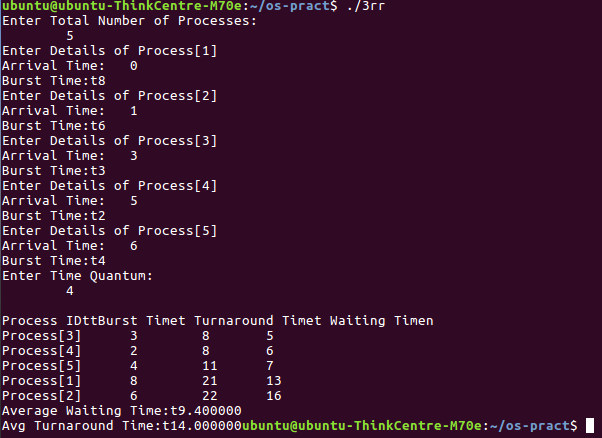
printf("\nAverage Waiting Time:t%f", average\_wait\_time);

printf("\nAvg Turnaround Time:t%f", average\_turnaround\_time);

return 0;

}

Output: -



ASSINGMENT 4

1. Producer - Consumer

Program Code: -

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#include <stdio.h>

#define MaxItems 5 // Maximum items a producer can produce or a consumer can consume

#define BufferSize 5 // Size of the buffer

sem\_t empty;

sem\_t full;

int in = 0;

int out = 0;

int buffer[BufferSize];

pthread\_mutex\_t mutex;

void \*producer(void \*pno)

{

int item;

for(int i = 0; i < MaxItems; i++) {

item = rand(); // Produce an random item

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

buffer[in] = item;

printf("Producer %d: Insert Item %d at %d\n", \*((int \*)pno),buffer[in],in);

in = (in+1)%BufferSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

}

void \*consumer(void \*cno)

{

for(int i = 0; i < MaxItems; i++) {

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

int item = buffer[out];

printf("Consumer %d: Remove Item %d from %d\n",\*((int \*)cno),item, out);

out = (out+1)%BufferSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

}

int main()

{

pthread\_t pro[5],con[5];

pthread\_mutex\_init(&mutex, NULL);

sem\_init(&empty,0,BufferSize);

sem\_init(&full,0,0);

int a[5] = {1,2,3,4,5}; //Just used for numbering the producer and consumer

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

pthread\_create(&pro[i], NULL, (void \*)producer, (void \*)&a[i]);

}

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

pthread\_create(&con[i], NULL, (void \*)consumer, (void \*)&a[i]);

}

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

pthread\_join(pro[i], NULL);

}

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

pthread\_join(con[i], NULL);

}

pthread\_mutex\_destroy(&mutex);

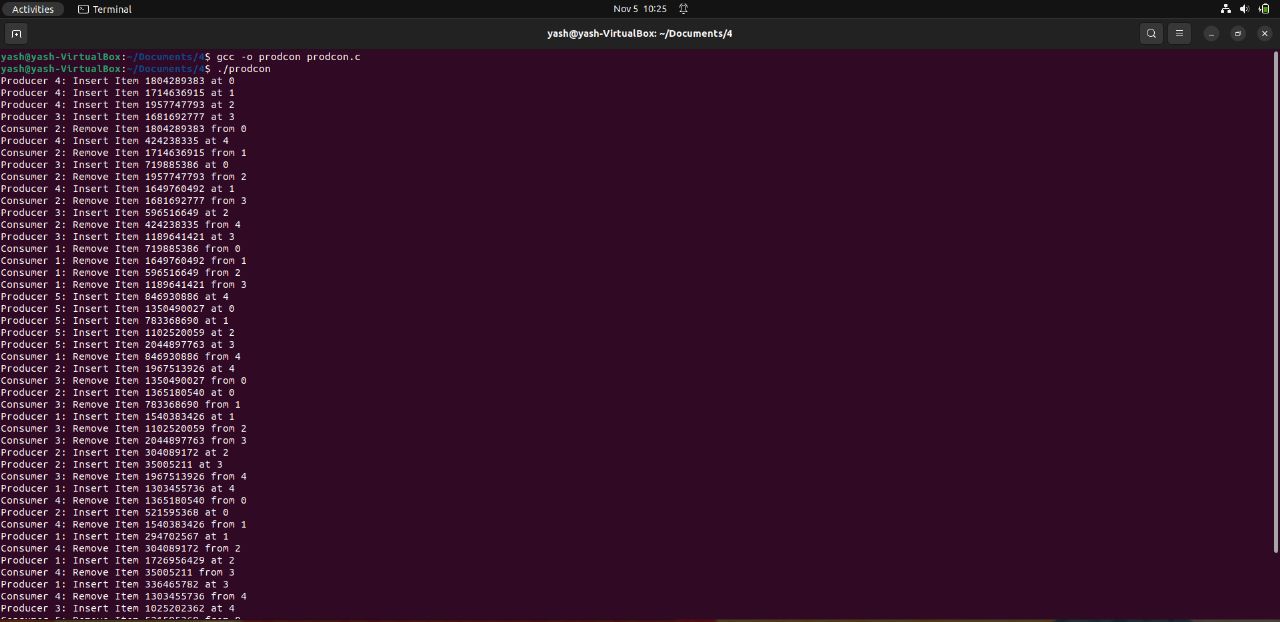
sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

Output: -



1. Reader – Writer

Program Code: -

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

sem\_t wrt;

pthread\_mutex\_t mutex;

int cnt = 1;

int numreader = 0;

void \*writer(void \*wno)

{

sem\_wait(&wrt);

cnt = cnt\*2;

printf("Writer %d modified cnt to %d\n",(\*((int \*)wno)),cnt);

sem\_post(&wrt);

}

void \*reader(void \*rno)

{

// Reader acquire the lock before modifying numreader

pthread\_mutex\_lock(&mutex);

numreader++;

if(numreader == 1) {

sem\_wait(&wrt); // If this id the first reader, then it will block the writer

}

pthread\_mutex\_unlock(&mutex);

// Reading Section

printf("Reader %d: read cnt as %d\n",\*((int \*)rno),cnt);

// Reader acquire the lock before modifying numreader

pthread\_mutex\_lock(&mutex);

numreader--;

if(numreader == 0) {

sem\_post(&wrt); // If this is the last reader, it will wake up the writer.

}

pthread\_mutex\_unlock(&mutex);

}

int main()

{

pthread\_t read[10],write[5];

pthread\_mutex\_init(&mutex, NULL);

sem\_init(&wrt,0,1);

int a[10] = {1,2,3,4,5,6,7,8,9,10}; //Just used for numbering the producer and consumer

for(int i = 0; i < 10; i++) {

pthread\_create(&read[i], NULL, (void \*)reader, (void \*)&a[i]);

}

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

pthread\_create(&write[i], NULL, (void \*)writer, (void \*)&a[i]);

}

for(int i = 0; i < 10; i++) {

pthread\_join(read[i], NULL);

}

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

pthread\_join(write[i], NULL);

}

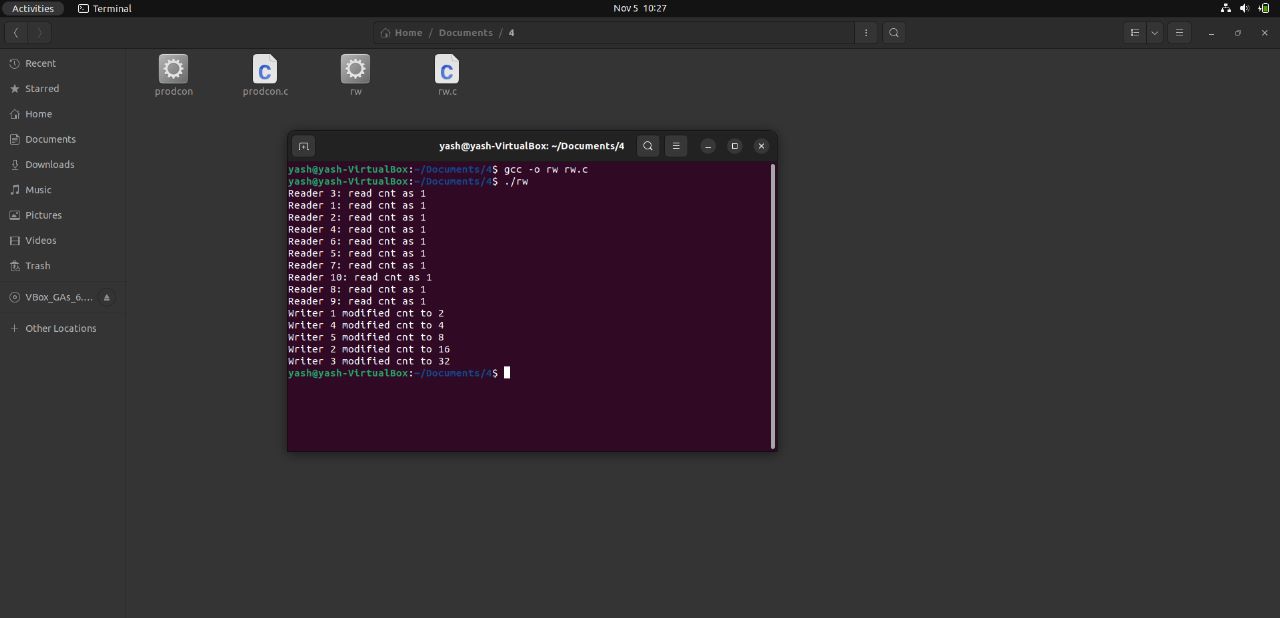
pthread\_mutex\_destroy(&mutex);

sem\_destroy(&wrt);

return 0;

}

Output: -



ASSINGMENT 5

Program Code: -

// Banker's Algorithm

#include <stdio.h>

int main(){

int n, m, i, j, k;

printf("Enter number of Processes: ");

scanf("%d", &n);

printf("Enter number of Resources: ");

scanf("%d", &m);

int alloc[n][m];

int max[n][m];

int avail[m];

printf("Enter Allocation Matrix: \n");

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

scanf("%d", &alloc[i][j]);

}

}

printf("Enter Max Matrix: \n");

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

scanf("%d", &max[i][j]);

}

}

printf("Enter Available Resources: \n");

for (int i = 0; i < m; i++) {

scanf("%d", &avail[i]);

}

int f[n], ans[n], ind = 0;

for (k = 0; k < n; k++) {

f[k] = 0;

}

int need[n][m];

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

for (j = 0; j < m; j++)

need[i][j] = max[i][j] - alloc[i][j];

}

int y = 0;

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

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

if (f[i] == 0) {

int flag = 0;

for (j = 0; j < m; j++) {

if (need[i][j] > avail[j]){

flag = 1;

break;

}

}

if (flag == 0) {

ans[ind++] = i;

for (y = 0; y < m; y++)

avail[y] += alloc[i][y];

f[i] = 1;

}

}

}

}

int flag = 1;

for(int i=0;i<n;i++){

if(f[i]==0){

flag=0;

printf("The following system is not safe");

break;

}

}

if(flag==1)

{

printf("Following is the SAFE Sequence\n");

for (i = 0; i < n - 1; i++)

printf(" P%d ->", ans[i]);

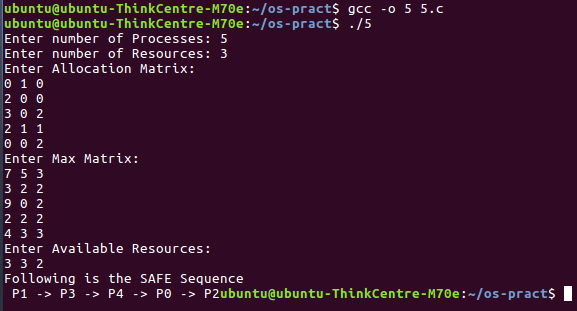
printf(" P%d", ans[n - 1]);

}

return (0);

}

Output: -



ASSINGMENT 6

Program Code: -

#include<stdio.h>

int n,nf;

int in[100];

int p[50];

int hit=0;

int i,j,k;

int pgfaultcnt=0;

void getData()

{

printf("\nEnter length of page reference sequence:");

scanf("%d",&n);

printf("\nEnter the page reference sequence:");

for(i=0; i<n; i++)

scanf("%d",&in[i]);

printf("\nEnter no of frames:");

scanf("%d",&nf);

}

void initialize()

{

pgfaultcnt=0;

for(i=0; i<nf; i++)

p[i]=9999;

}

int isHit(int data)

{

hit=0;

for(j=0; j<nf; j++)

{

if(p[j]==data)

{

hit=1;

break;

}

}

return hit;

}

int getHitIndex(int data)

{

int hitind;

for(k=0; k<nf; k++)

{

if(p[k]==data)

{

hitind=k;

break;

}

}

return hitind;

}

void dispPages()

{

for (k=0; k<nf; k++)

{

if(p[k]!=9999)

printf(" %d",p[k]);

}

}

void dispPgFaultCnt()

{

printf("\nTotal no of page faults:%d",pgfaultcnt);

}

void fifo()

{

initialize();

for(i=0; i<n; i++)

{

printf("\nFor %d :",in[i]);

if(isHit(in[i])==0)

{

for(k=0; k<nf-1; k++)

p[k]=p[k+1];

p[k]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault");

}

dispPgFaultCnt();

}

void optimal()

{

initialize();

int near[50];

for(i=0; i<n; i++)

{

printf("\nFor %d :",in[i]);

if(isHit(in[i])==0)

{

for(j=0; j<nf; j++)

{

int pg=p[j];

int found=0;

for(k=i; k<n; k++)

{

if(pg==in[k])

{

near[j]=k;

found=1;

break;

}

else

found=0;

}

if(!found)

near[j]=9999;

}

int max=-9999;

int repindex;

for(j=0; j<nf; j++)

{

if(near[j]>max)

{

max=near[j];

repindex=j;

}

}

p[repindex]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault");

}

dispPgFaultCnt();

}

void lru()

{

initialize();

int least[50];

for(i=0; i<n; i++)

{

printf("\nFor %d :",in[i]);

if(isHit(in[i])==0)

{

for(j=0; j<nf; j++)

{

int pg=p[j];

int found=0;

for(k=i-1; k>=0; k--)

{

if(pg==in[k])

{

least[j]=k;

found=1;

break;

}

else

found=0;

}

if(!found)

least[j]=-9999;

}

int min=9999;

int repindex;

for(j=0; j<nf; j++)

{

if(least[j]<min)

{

min=least[j];

repindex=j;

}

}

p[repindex]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault!");

}

dispPgFaultCnt();

}

int main()

{

int choice;

while(1)

{

printf("\nPage Replacement Algorithms\n1.Enter data\n2.FIFO\n3.Optimal\n4.LRU\n5.Exit\nEnter your choice:");

scanf("%d",&choice);

switch(choice)

{

case 1:

getData();

break;

case 2:

fifo();

break;

case 3:

optimal();

break;

case 4:

lru();

break;

default:

return 0;

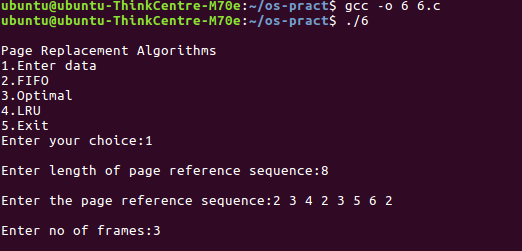
break;

}

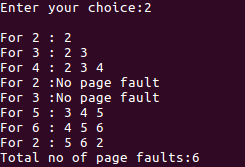
}

}

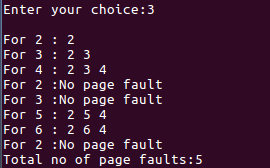
Output: -



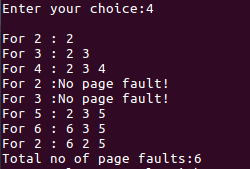
1. FCFS



1. Optimal



1. LRU



ASSINGMENT 7

1. FIFO

Client Program Code: -

#include<stdio.h>

#include<stdlib.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<unistd.h>

#include<fcntl.h>

#include<string.h>

int main()

{

puts("\n\tClient - Listening\n");

int code6 = mkfifo("fifo6.txt",0666);

int code7 = mkfifo("fifo7.txt",0666);

char strMessage[5000];

if(code6 == -1)

perror("\n\tmkfifo6 returned an error-file any already exist\n");

if(code7 == -1)

perror("\n\tmkfifo7 returned an error-file any already exist\n");

int fd = open("fifo6.txt", O\_RDONLY);

int fd2 = open("fifo7.txt", O\_WRONLY);

if(fd == -1)

{

perror("Cannot open FIFO6 for read");

return EXIT\_FAILURE;

}

if(fd2 == -1)

{

perror("Cannot open FIFO7 for write");

return EXIT\_FAILURE;

}

puts("FIFO OPEN");

//read string up to(5000 characters)

char stringBuffer[5000];

memset(stringBuffer, 0, 5000);

int res;

char Len;

//while(1)

{

res = read(fd, &Len, 1);

//if(Len == 1)//since null counts 1

//break;

read(fd, stringBuffer, Len); //Read String Characters

stringBuffer[(int)Len] = 0;

printf("\nClient Received: %s\n", stringBuffer);

int j = 0,w=0, line = 0;

while(stringBuffer[j]!='\0'){

char ch = stringBuffer[j];

if((ch==' ')||(ch=='\n')){

w++;

if(ch=='\n')

line++;

}

j++;

}

char LC = (char) strlen(strMessage);

char str1[256];

char str2[256];

char str3[256];

sprintf(str1," No.of Words : %d:::", w); strcat(strMessage,str1);

sprintf(str2," No.of Charecters: %d:::",(j-1)); strcat(strMessage,str2);

sprintf(str3," No.of Lines: %d",line); strcat(strMessage,str3);

strcat(strMessage,"\0");

printf("\n\tString: %s",strMessage);

write(fd2, &LC, 1);

write(fd2, strMessage, strlen(strMessage));

fflush(stdin);

strMessage[0] = 0;//reseting the character array

//if(LC==1)

//break;

}

printf("\n");

puts("CLIENT CLOSED");

puts("SERVER CLOSED");

close(fd);

close(fd2);

return 0;

}

Server Program Code: -

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/types.h>

#include<sys/types.h>

#include<fcntl.h>

#include<string.h>

int main()

{

int n;

puts("Server");

char strMessage[5000];//[] = {"welcome", "to", "the", "module.", "This", "will", "now", "stop"};

int fd = open("fifo6.txt", O\_WRONLY);

int fd2 = open ("fifo7.txt", O\_RDONLY);

if(fd == -1)

{

perror("cannot open fifo6");

return EXIT\_FAILURE;

}

if(fd2 == -1)

{

perror("cannot open fifo7");

return EXIT\_FAILURE;

}

puts("FIFO OPEN");

//read string up to(5000 characters)

char stringBuffer[5000];

memset(stringBuffer, 0, 5000);

int res;

char Len;

//while(1)

{

printf("\n\n\t\tEnter the Message to be passed (hitting ENTER without any string will terminate program): ");

fgets(strMessage, 100, stdin);

char L = (char) strlen(strMessage);

//printf("\n\tLength of the given string: %d\n", (L-1));

write(fd, &L, 1);

write(fd, strMessage, strlen(strMessage));

fflush(stdin);

strMessage[0] = 0;//reseting the character array

//if(L==1)//since null counts 1

//break;

int len2;

res = read(fd2, &len2, 1);

//if(len2 == 1)//since null counts 1

//break;

read(fd2, stringBuffer, 5000); //Read String Characters

printf("\nServer Received: %s\n", stringBuffer);

stringBuffer[(int)len2] = 0;

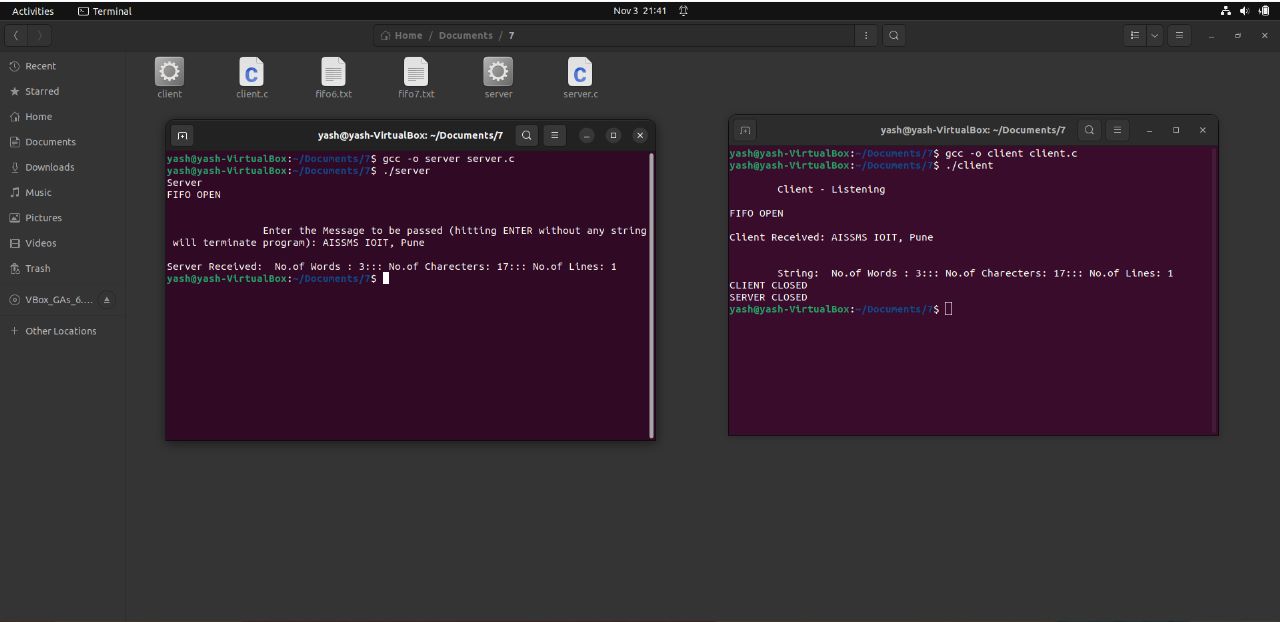
};

//printf("\n\nCLIENT CLOSED\n")

//return 0;

}

Output: -



1. Shared Memory: -

Client Program Code: -

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#define SHMSZ 27

main()

{

int shmid;

key\_t key;

char \*shm, \*s;

/\*

\* We need to get the segment named

\* "5678", created by the server.

\*/

key = 5678;

/\*

\* Locate the segment.

\*/

if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {

perror("shmget");

exit(1);

}

/\*

\* Now we attach the segment to our data space.

\*/

if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1) {

perror("shmat");

exit(1);

}

/\*

\* Now read what the server put in the memory.

\*/

for (s = shm; \*s != NULL; s++)

putchar(\*s);

putchar('\n');

/\*

\* Finally, change the first character of the

\* segment to '\*', indicating we have read

\* the segment.

\*/

\*shm = '\*';

exit(0);

}

Server Program Code: -

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#define SHMSZ 27

main()

{

char c;

int shmid;

key\_t key;

char \*shm, \*s;

/\*

\* We'll name our shared memory segment

\* "5678".

\*/

key = 5678;

/\*

\* Create the segment.

\*/

if ((shmid = shmget(key, SHMSZ, IPC\_CREAT | 0666)) < 0) {

perror("shmget");

exit(1);

}

/\*

\* Now we attach the segment to our data space.

\*/

if ((shm = shmat(shmid, NULL, 0)) == (char \*) -1) {

perror("shmat");

exit(1);

}

/\*

\* Now put some things into the memory for the

\* other process to read.

\*/

s = shm;

for (c = 'a'; c <= 'z'; c++)

\*s++ = c;

\*s = NULL;

/\*

\* Finally, we wait until the other process

\* changes the first character of our memory

\* to '\*', indicating that it has read what

\* we put there.

\*/

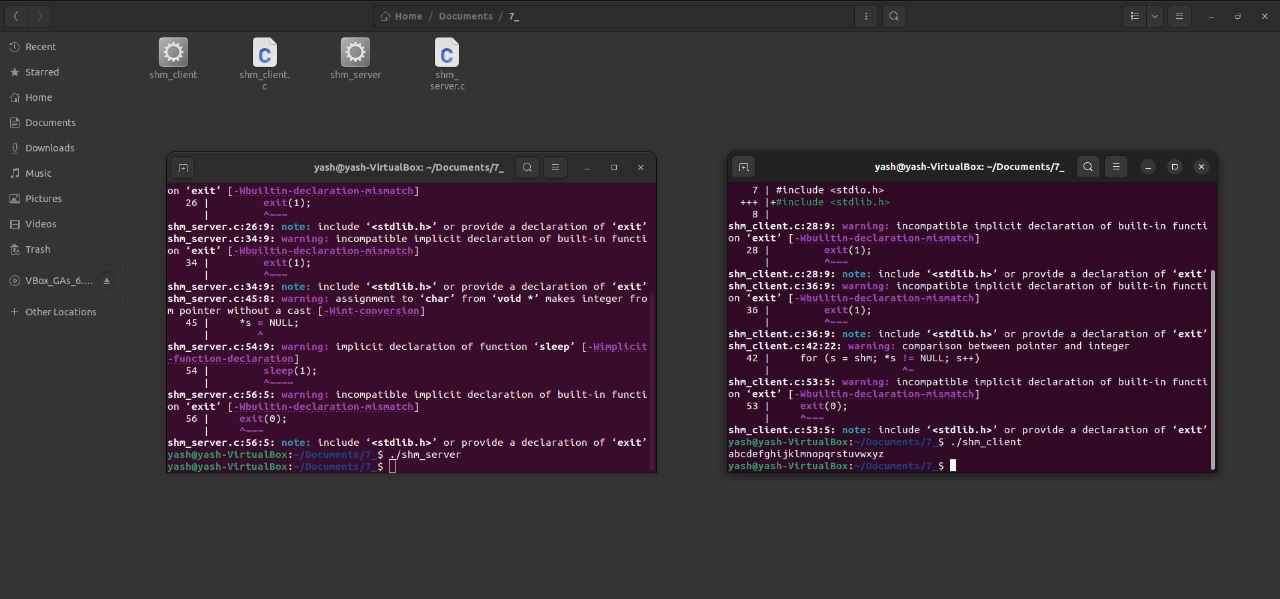
while (\*shm != '\*')

sleep(1);

exit(0);

}

Output: -



ASSINGMENT 8

1. SSTF

Program Code: -

#include<stdio.h>

#include<stdlib.h>

int main()

{

int RQ[100],i,n,TotalHeadMoment=0,initial,count=0;

printf("Enter the number of Requests\n");

scanf("%d",&n);

printf("Enter the Requests sequence\n");

for(i=0;i<n;i++)

scanf("%d",&RQ[i]);

printf("Enter initial head position\n");

scanf("%d",&initial);

// logic for sstf disk scheduling

/\* loop will execute until all process is completed\*/

while(count!=n)

{

int min=1000,d,index;

for(i=0;i<n;i++)

{

d=abs(RQ[i]-initial);

if(min>d)

{

min=d;

index=i;

}

}

TotalHeadMoment=TotalHeadMoment+min;

initial=RQ[index];

// 1000 is for max

// you can use any number

RQ[index]=1000;

count++;

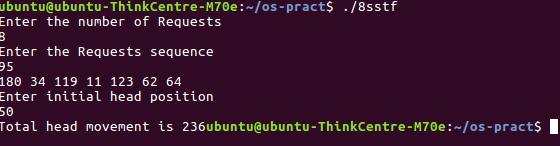
}

printf("Total head movement is %d",TotalHeadMoment);

return 0;

}

Output: -



1. SCAN

Program Code: -

#include<stdio.h>

#include<stdlib.h>

int main()

{

int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;

printf("Enter the number of Requests\n");

scanf("%d",&n);

printf("Enter the Requests sequence\n");

for(i=0;i<n;i++)

scanf("%d",&RQ[i]);

printf("Enter initial head position\n");

scanf("%d",&initial);

printf("Enter total disk size\n");

scanf("%d",&size);

printf("Enter the head movement direction for high 1 and for low 0\n");

scanf("%d",&move);

// logic for Scan disk scheduling

/\*logic for sort the request array \*/

for(i=0;i<n;i++)

{

for(j=0;j<n-i-1;j++)

{

if(RQ[j]>RQ[j+1])

{

int temp;

temp=RQ[j];

RQ[j]=RQ[j+1];

RQ[j+1]=temp;

}

}

}

int index;

for(i=0;i<n;i++)

{

if(initial<RQ[i])

{

index=i;

break;

}

}

// if movement is towards high value

if(move==1)

{

for(i=index;i<n;i++)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

// last movement for max size

TotalHeadMoment=TotalHeadMoment+abs(size-RQ[i-1]-1);

initial = size-1;

for(i=index-1;i>=0;i--)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

}

// if movement is towards low value

else

{

for(i=index-1;i>=0;i--)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

// last movement for min size

TotalHeadMoment=TotalHeadMoment+abs(RQ[i+1]-0);

initial =0;

for(i=index;i<n;i++)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

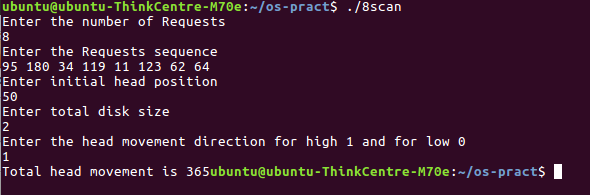
}

printf("Total head movement is %d",TotalHeadMoment);

return 0;

}

Output: -



1. C-Look

Program Code: -

#include<stdio.h>

#include<stdlib.h>

int main()

{

int RQ[100],i,j,n,TotalHeadMoment=0,initial,size,move;

printf("Enter the number of Requests\n");

scanf("%d",&n);

printf("Enter the Requests sequence\n");

for(i=0;i<n;i++)

scanf("%d",&RQ[i]);

printf("Enter initial head position\n");

scanf("%d",&initial);

printf("Enter total disk size\n");

scanf("%d",&size);

printf("Enter the head movement direction for high 1 and for low 0\n");

scanf("%d",&move);

// logic for C-look disk scheduling

/\*logic for sort the request array \*/

for(i=0;i<n;i++)

{

for( j=0;j<n-i-1;j++)

{

if(RQ[j]>RQ[j+1])

{

int temp;

temp=RQ[j];

RQ[j]=RQ[j+1];

RQ[j+1]=temp;

}

}

}

int index;

for(i=0;i<n;i++)

{

if(initial<RQ[i])

{

index=i;

break;

}

}

// if movement is towards high value

if(move==1)

{

for(i=index;i<n;i++)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

for( i=0;i<index;i++)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

}

// if movement is towards low value

else

{

for(i=index-1;i>=0;i--)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

for(i=n-1;i>=index;i--)

{

TotalHeadMoment=TotalHeadMoment+abs(RQ[i]-initial);

initial=RQ[i];

}

}

printf("Total head movement is %d",TotalHeadMoment);

return 0;

}

Output: -

