1. Study of Basic Linux Commands: echo, ls, read, cat, touch, test, loops, arithmetic comparison, conditional loops, grep,sed etc.
2. Write a program to implement an address book with options given below: a) Create address book. b) View address book. c) Insert a record. d) Delete a record. e) Modify a record. f) Exit

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 "Delete record\nEnter Name/Phone Number"

read pattern

temp="temp"

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

rm $fileName

cat $temp | cat >> $fileName

rm $temp

;;

5)

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

1. Implement the C program in which main program accepts the integers to be sorted. Main program uses the FORK system call to create a new process called a child process. Parent process sorts the integers using sorting algorithm and waits for child process using WAIT system call to sort the integers using any sorting algorithm. Also demonstrate zombie and orphan states.

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

#include<stdlib.h>

void sortAssending(int arr[30],int n)//bubble sorting

{

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");

}

int main()

{

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 fork returns value 0, then it is the child process If it returns some positive number then

it is parent process. The positive number is the pid of a child.\*/

if(pid==0)//this will be executed by child

{

sleep(10);

printf("\nchild process\n");

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

sortAssending(arr,n);

printf("\nElements are sorted by child process");

printf("\n");

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

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

printf("\n\n");

}

else//executed by parent

{

printf("\nparent process\n");

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

sortAssending(arr,n);

printf("Elements Sorted by parent");

printf("\n");

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

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

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

}

return 0;

}

1. Implement the C program in which main program accepts an array. Main program uses the FORK system call to create a new process called a child process. Parent process sorts an array and passes the sorted array to child process through the command line arguments of EXECVE system call. The child process uses EXECVE system call to load new program which display array in reverse order.

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h> // For fork() system call and pid\_t data type

#define MAX 20

void sortAssending(int arr[30],int n)//bubble sorting

{

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;

}

}

}

}

void sortDecending(int arr[30],int n)//bubble sorting

{

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;

}

}

}

}

int main()

{

pid\_t pid; // Decleration of pid which will store process ID

int a[MAX],n;

int i;

// Accepting Elements of an array

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]);

}

/\* =====Performing fork() system call==== \*/

pid=fork();

if(pid<0)// If Process not created successfully

{

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

}

else if(pid==0) // For Child process

{

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

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

sortDecending(a,n); //Performing decending sort in child process

printf("\n\n\tSorted array in decending order:\n\t");

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

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

printf("\n");

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

}

else // For Parent process

{

// For Zombie process

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

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

sortAssending(a,n); //Performing sort in parent process

printf("\n\n\tSorted array in accending order:\n\t");

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

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

printf("\n");

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

}

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

return 0;

}

1. **Implement the C program for CPU Scheduling Algorithms: Shortest Job First (Preemptive)**

**#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);**

**}**

1. Implement the C program for CPU Scheduling Algorithms: Round Robin with different arrival time.

#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 ID \t\t\t Burst Time \t\t Turnaround Time \t\t Waiting Time");

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\t\t%d\t\t\t%d\t\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\t\t\t%f", average\_wait\_time);

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

return 0;

}

1. **Implement the C program for Deadlock Avoidance Algorithm: Bankers Algorithm.**

// Banker's Algorithm it is resource allocation and deadlock allocation algorithm.

#include <stdio.h>

int main()

{

// P0, P1, P2, P3, P4 are the Process names here

int n, m, i, j, k,a;

n = 5; // Number processes to be run

m = 3; // Number of resources i.e.cpu etc

int allocatedResources[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix

{ 2, 0, 0 }, // P1

{ 3, 0, 2 }, // P2

{ 2, 1, 1 }, // P3

{ 0, 0, 2 } };//P4

int maximumNeed[5][3] = { { 7, 5, 3 }, // P0 // MAX need Matrix

{ 3, 2, 2 }, // P1

{ 9, 0, 2 }, // P2

{ 2, 2, 2 }, // P3

{ 4, 3, 3 } };//P4

int availableResources[3] = { 3, 3, 2 }; // Available Resources

int f[n],ind = 0;//finished process storage

int ans[n];//in ans[n]array we will store the sequence by which the processes runs

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

f[k] = 0;//we will set flag for all process as 0 as they are not completed yet

}

int need[n][m];//to store number of required resources

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

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

need[i][j] = maximumNeed[i][j] - allocatedResources[i][j];//here the remaining need of resources to the processes are stored

}

int y = 0;

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

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

if (f[i] == 0) {//if f[i]==0 then process not completed && if f[i]==1 the it is completed

int flag = 0;

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

if (need[i][j] > availableResources[j]){// if need is greater than available resources then we will not allocate resources to that process

flag = 1;

break;

}

}

if (flag == 0) { // here need is less than available resources then we will allocate resources to that process and the process will start

ans[ind] = i; //here we are storing the running process number in the array to maintain running sequence

printf("\n now P%d will run and complete \n", ans[ind]);

ind++;

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

//here we will free the resources of that process which completed just and add those resources to the available resources

availableResources[y] += allocatedResources[i][y];

printf("Now available resources are ");

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

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

}

f[i] = 1;//we will set flag of finished process as 1

}

}

}

}

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

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

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

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

return (0);

}

1. **Implement the C program for Page Replacement Algorithms: FCFS**

#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;

} }

}

1. **Inter process communication in Linux using FIFO.**

**#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;

}

1. **Inter-process Communication using Shared Memory using System V.**

#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;

}

1. **Implement the C program for Disk Scheduling Algorithms: SSTF**

#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;

}