* **Assignment 1**

#!/bin/bash

# Assignment no 1

createAddressBook()

{

echo

read -p "Enter no. of records: " n

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

do

echo

echo "Enter ID"

read ID[i]

echo "Enter name"

read name[i]

echo "Enter address"

read address[i]

echo "Enter phone\_no"

read phone\_no[i]

done

}

displayRecords()

{

echo

echo "ID NAME ADDRESS PHONE\_NO"

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

do

echo "${ID[i]} ${name[i]} ${address[i]} ${phone\_no[i]}"

done

}

insertRecord()

{

echo

echo "Enter ID to insert"

read ins\_id

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

do

if test "$ins\_id" == "${ID[i]}"

then

echo

echo "ID already exists"

dup\_id=1

fi

done

if test "$dup\_id" != 1

then

ID[n]=$ins\_id

echo "Enter name"

read name[n]

echo "Enter address"

read address[n]

echo "Enter phone\_no"

read phone\_no[n]

n=$((n+1))

fi

}

deleteRecord()

{

echo

echo "Enter index of record to delete"

read del\_pos

for((i=del\_pos; i<n-1; i++))

do

ID[i]=${ID[i+1]}

name[i]=${name[i+1]}

address[i]=${address[i+1]}

phone\_no[i]=${phone\_no[i+1]}

done

n=$((n-1))

}

modifyRecord()

{

echo

echo "Enter ID to modify"

read mod\_id

is\_mod=0

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

do

if test "$mod\_id" == "${ID[i]}"

then

echo "Enter name"

read name[i]

echo "Enter address"

read address[i]

echo "Enter phone\_no"

read phone\_no[i]

is\_mod=1

fi

done

if test "$is\_mod" == 0

then

echo

echo "ID does not exist"

fi

}

exitFunction()

{

echo

echo "EXITING"

echo

break

}

createAddressBook

while :

do

echo

echo "ADDRESS BOOK"

echo "1. View"

echo "2. Insert"

echo "3. Delete"

echo "4. Modify"

echo "5. Exit"

echo

echo "Enter choice"

read choice

if test "$choice" == 1

then

displayRecords

fi

if test "$choice" == 2

then

insertRecord

fi

if test "$choice" == 3

then

deleteRecord

fi

if test "$choice" == 4

then

modifyRecord

fi

if test "$choice" == 5

then

exitFunction

fi

done

* **Assignment 2a: Merge Sort | Quick sort**

#include <stdio.h>

#include<unistd.h>

void mergesort();

void merge();

void quicksort();

int main()

{

pid\_t pid ;

int a[40] , n, i ;

printf("Enter the no. of integers to be sorted :");

scanf("%d", &n);

printf("Enter integers :- ");

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

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

pid = fork();

if(pid>0)

{

printf("\nInside parent process");

printf("\nProcess ID:%d" , getpid());

printf("\n\n");

quicksort(a,0,n-1);

printf("Integers sorted using quicksort : ");

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

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

printf("\n");

}

else if(pid==0)

{

printf("\nInside Child process");

printf("\nProcess ID:%d" , getpid());

printf("\n\n");

mergesort(a,0,n-1);

printf("Integers sorted using mergesort : ");

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

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

printf("\n");

}

else

printf("\nChild Process could not be created!\n");

return 0 ;

}

void mergesort (int a[] , int first , int last)

{

int mid ;

if (first < last)

{

mid = (first + last) / 2 ;

mergesort(a, first , mid);

mergesort(a,mid+1 , last);

merge(a,first,mid,last);

}

}

void merge(int a[] , int first , int mid , int last)

{

int b[50];

int i , j , k ;

i = first ;

j = mid + 1 ;

k = first ;

while(i <= mid && j<= last)

{

if(a[i]<= a[j])

b[k++]=a[i++];

else

b[k++]=a[j++];

}

if(i>mid)

{

while(j<=last)

b[k++] = a[j++];

}

else

{

while(i<=mid)

b[k++] = a[i++];

}

for(i=first;i<=last;i++)

a[i] = b[i];

}

void quicksort(int a[40] , int first , int last)

{

int i , j , pivot , temp ;

if(first < last)

{

pivot = first ;

i = first ;

j = last ;

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[pivot] ;

a[pivot] = a[j];

a[j] = temp;

quicksort(a,0,j-1);

quicksort(a,j+1,last);

}

}

* **Assignment 3: SJF**

#include <stdio.h>

int main()

{

int arrival\_time[10], burst\_time[10], temp[10];

int i, smallest, count = 0, time, limit;

double wait\_time = 0, turnaround\_time = 0, end;

float average\_waiting\_time, average\_turnaround\_time;

printf("Enter the Total Number of Processes:");

scanf("%d", &limit);

printf("Enter Details of %d Processes", limit);

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

{

printf("Enter Arrival Time:");

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

printf("Enter Burst Time:");

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

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

}

burst\_time[9] = 9999;

for(time = 0; count != limit; time++)

{

smallest = 9;

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

{

if(arrival\_time[i] <= time && burst\_time[i] < burst\_time[smallest] &&

burst\_time[i] > 0)

{

smallest = i;

}

}

burst\_time[smallest]--;

if(burst\_time[smallest] == 0)

{

count++;

end = time + 1;

wait\_time = wait\_time + end - arrival\_time[smallest] - temp[smallest];

turnaround\_time = turnaround\_time + end - arrival\_time[smallest];

}

}

average\_waiting\_time = wait\_time / limit;

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

printf("Average Waiting Time:%lf\n", average\_waiting\_time);

printf("Average Turnaround Time:%lf\n", average\_turnaround\_time);

return 0;

}

* **Assignment 3 : Round Robin**

#include <stdio.h>

#include <stdlib.h>

void main()

{

// initlialize the variable name

int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10],

temp[10];

float avg\_wt, avg\_tat;

printf(" Total number of process in the system: ");

scanf("%d", &NOP);

y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the

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

{

printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);

printf(" Arrival time is: \t"); // Accept arrival time

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

printf(" \nBurst time is: \t"); // Accept the Burst time

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

temp[i] = bt[i]; // store the burst time in temp array

}

// Accept the Time qunat

printf("Enter the Time Quantum for the process: \t");

scanf("%d", &quant);

// Display the process No, burst time, Turn Around Time and the waiting

printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");

for(sum=0, i = 0; y!=0; )

{

if(temp[i] <= quant && temp[i] > 0) // define the conditions

{

sum = sum + temp[i];

temp[i] = 0;

count=1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - quant;

sum = sum + quant;

}

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

{

y--; //decrement the process no.

printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-bt[i]);

wt = wt+sum-at[i]-bt[i];

tat = tat+sum-at[i];

count =0;

}

if(i==NOP-1)

{

i=0;

}

else if(at[i+1]<=sum)

{

i++;

}

else

{

i=0;

}

}

}

* **Assignment 4a : Thread Sync using semaphore**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<pthread.h>

#include<semaphore.h>

sem\_t empty, full;

pthread\_mutex\_t pmt;

int arr[5],in=0,out=0;

void\* prodFunc(void\* arg)

{

int item=1;

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

{

sem\_wait(&empty);

pthread\_mutex\_lock(&pmt);

printf("Item number : %d\n",(i+1));

printf(" Producer produced item %d\n", item);

arr[in]=item;

in=(in+1)%5;

item++;

pthread\_mutex\_unlock(&pmt);

sem\_post(&full);

sleep(1);

}

}

void\* consFunc(void\* arg)

{

int item;

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

{

sem\_wait(&full);

pthread\_mutex\_lock(&pmt);

item=arr[out];

out=(out+1)%5;

printf(" Consumer consumed item %d\n\n", item);

pthread\_mutex\_unlock(&pmt);

sem\_post(&empty);

sleep(1);

}

}

int main()

{

pthread\_t pt1, pt2;

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

sem\_init(&empty, 0, 5);

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

pthread\_create(&pt1, NULL, consFunc, NULL);

pthread\_create(&pt2, NULL, prodFunc, NULL);

pthread\_join(pt1, NULL);

pthread\_join(pt2, NULL);

printf("\nCompleted\n");

return 0;

}

* **Assignment 4b: Thread sync and mutual exclusion using mutex**

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

#include<sys/unistd.h>

int data=0;

pthread\_mutex\_t mutex;

sem\_t wrt;

int readcount=0;

void \*reader(void \*rno)

{

int rn=\*(int \*)rno;

//printf("\nReader %d is arrived",rn);

int limit=1;

do

{

pthread\_mutex\_lock(&mutex);

readcount++;

if(readcount==1)

{

sem\_wait(&wrt);

}

pthread\_mutex\_unlock(&mutex);

printf("\nReader %d is reading %d",rn,data);

pthread\_mutex\_lock(&mutex);

readcount--;

if(readcount==0)

{

sem\_post(&wrt);

}

pthread\_mutex\_unlock(&mutex);

sleep(2);

limit++;

} while(limit<3);

}

void \*writer(void \*wno)

{

int wn=\*(int \*)wno;

int limit1;

do

{

// printf("\nWriter %d is arrived",wn);

sem\_wait(&wrt);

printf("\nWriter %d is Entering in critical section",wn);

data++;

printf("\nWriter %d is writing %d",wn,data);

printf("\nWriter %d is Leaving from critical section",wn);

sem\_post(&wrt);

sleep(2);

limit1++;

} while(limit1<3);

}

void main()

{

pthread\_t rd[3];

pthread\_t wr[3];

char buff[20];

int i;

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

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

for(i=0;i<3;i++)

{

pthread\_create(&wr[i],NULL,writer,&i);

pthread\_create(&rd[i],NULL,reader,&i);

}

for(i=0;i<3;i++)

{

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

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

}

}

* **Assignment 5:**

#include <stdio.h>

int main()

{

int n, m, i, j, k, y,alloc[20][20],max[20][20],avail[50],ind=0;

printf("Enter the no of Proceses:");

scanf("%d",&n);

printf("Enter the no of Resources:");

scanf("%d",&m);

printf("Enter the Allocation Matrix:");

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

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

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

}

printf("Enter the Max Matrix:");

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

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

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

}

printf("Enter the Available Matrix");

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

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

int finish[n], safesequence[n],work[m],need[n][m];

//calculating NEED matrix

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

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

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

}

printf("NEED matrix is");

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

{

printf("\n");

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

printf(" %d ",need[i][j]);

}

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

{

work[i]=avail[i];

}

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

finish[i] = 0;

}

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

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

{

if (finish[i] == 0)

{

int flag = 0;

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

{

if (need[i][j] > work[j])

{

flag = 1;

break;

}

}

if (flag == 0) {

safesequence[ind++] = i;

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

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

finish[i] = 1;

}

}

}

}

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

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

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

}

* **Assignment 6:**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int i,j,n,a[50],frame[10],no,k,avail,count=0;

printf("\n ENTER THE NUMBER OF PAGES:\n");

scanf("%d",&n);

printf("\n ENTER THE PAGE NUMBER :\n");

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

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

printf("\n ENTER THE NUMBER OF FRAMES :");

scanf("%d",&no);

for(i=0;i<no;i++)

frame[i]= -1;

j = 0 ;

printf("\n ref string \t page frames \tHit/Fault\n");

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

{

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

avail=0;

for(k=0;k<no;k++)

if(frame[k]==a[i])

{

avail=1;

for(k=0;k<no;k++)

printf("%d\t",frame[k]);

printf("H");

}

if (avail==0)

{

frame[j]=a[i];

j=(j+1)%no;

count++;

for(k=0;k<no;k++)

printf("%d\t",frame[k]);

printf("F");

}

printf("\n");

}

printf("Page Fault Is %d",count);

return 0;

}

* **Assignment 8a | 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;

}

* **Assignment 8b | SCAN**

#include<conio.h>

#include<stdio.h>

int main() {

int i, j, sum = 0, n; int d[20];

int disk; //loc of head int temp, max;

int dloc; //loc of disk in array

printf("enter number of location\t"); scanf("%d", & n);

printf("enter position of head\t"); scanf("%d", & disk);

printf("enter elements of disk queue\n");

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

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

}

d[n] = disk; n = n + 1;

for (i = 0; i < n; i++) // sorting disk locations

{

for (j = i; j < n; j++) { if (d[i] > d[j]) {int temp = d[i];

d[i] = d[j]; d[j] = temp;

}

}

}

int max = d[n];

for (i = 0; i < n; i++) // to find loc of disc in array

{

if (disk == d[i]) { dloc = i;

break;

}

}

for (i = dloc; i >= 0; i--) { printf("%d -->", d[i]);

}

printf("0 -->");

for (i = dloc + 1; i < n; i++) { printf("%d-->", d[i]);

}

sum = disk + max;

printf("\nmovement of total cylinders %d", sum); getch();

return 0;

}