**EXPERIMENT NO 3**

**1 opening a file**

//open system call

#include<stdio.h>

#include<fcntl.h>

#include<errno.h> extren int

errno; int main()

{

int fd = open("foo.txt", O\_RDONLY | O\_CREAT);

printf("fd = %d/n",fd);

if(fd==-1)

{

printf("error number %d\n",errno);

perror("Program");

}

return 0;

}

**2. reading from file**

#include<stdio.h>

#include<fcntl.h>

#include<stdlib.h>

#include<unistd.h>

int main() {

int fd, sz;

char \*c = (char \*) calloc(100, sizeof(char));

if (c == NULL) {

perror("Memory allocation failed");

exit(1);

}

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

if (fd < 0) {

perror("Error opening file");

exit(1);

}

sz = read(fd, c, 10);

if (sz < 0) {

perror("Error reading file");

exit(1);

}

printf("called read(%d, c, 10). returned that %d bytes were read.\n", fd, sz);

c[sz] = '\0';

printf("Those bytes are as follows: %s\n", c);

free(c); // Free dynamically allocated memory

close(fd); // Close the file descriptor

return 0;

}

**3 writing a file**

#include<stdio.h>

#include <fcntl.h>

#include<stdlib.h>

#include <unistd.h>

#include <string.h>

int main(void) {

int fd[2];

char buf1[12] = "hello world";

char buf2[12];

// Open the file for reading and writing

fd[0] = open("foobar.txt", O\_RDWR | O\_CREAT, 0666);

if (fd[0] == -1) {

perror("Error opening file");

exit(1);

}

// Open the file again for reading and writing

fd[1] = open("foobar.txt", O\_RDWR);

if (fd[1] == -1) {

perror("Error opening file");

exit(1);

}

// Write data to the file

if (write(fd[0], buf1, strlen(buf1)) == -1) {

perror("Error writing to file");

exit(1);

}

// Read data from the file and write to stdout

ssize\_t bytesRead = read(fd[1], buf2, sizeof(buf2));

if (bytesRead == -1) {

perror("Error reading from file");

exit(1);

}

write(1, buf2, bytesRead);

// Close the file descriptors

if (close(fd[0]) == -1) {

perror("Error closing file");

exit(1);

}

if (close(fd[1]) == -1) {

perror("Error closing file");

exit(1);

}

return 0;

}

**4 closing a file**

// C program to illustrate close system Call

#include<stdio.h>

#include<stdlib.h>

#include <fcntl.h>

#include <unistd.h>

int main(){

// assume that foo.txt is already created

int fd1= open("foo.txt", O\_RDONLY, 0);

close(fd1);

// assume that baz.tzt is already created

int fd2= open("baz.txt", O\_RDONLY, 0);

printf("fd2 = %d\n", fd2);

exit(0);

}

**5 process management-getpid()&getppid()**

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

int main() {

printf("Process ID = %d\n", getpid());

printf("Parent Process ID = %d\n", getpid()); return 0;}

**6 accessing user & groupids**

#include <stdio.h>

#include <unistd.h>

int main(int argc,char \*\* argv) {

pid\_t pid\_tmp;

pid\_tmp = getuid();

printf("the ID of the actural user is:%d\n",pid\_tmp);

pid\_tmp = geteuid();

printf("the ID of the effective user is :%d\n",pid\_tmp);

pid\_tmp = getgid();

printf("the ID of the actural group is:%d\n",pid\_tmp);

pid\_tmp = getegid();

printf("the ID of the effective group is:%d\n",pid\_tmp);

return 0;

}

**EXPERIMENT NO 5**

**1 CPU Scheduling FCFS**

/\* Program to Simulate First Come First Serve CPU Scheduling Algorithm \*/

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

int i,j,n,bt[10],compt[10],at[10],

wt[10],tat[10]; float

sumwt=0.0,sumtat=0.0,avgwt,avgtat;

printf("Enter number of processes: ");

scanf("%d",&n);

printf("Enter the burst time of %d process\n",

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

{

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

}

printf("Enter the arrival time of %d process\n",

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

{

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

}

compt[0]=bt[0]-at[0];

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

compt[i]=bt[i]+compt[i-

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

{

tat[i]=compt[i]-at[i];

wt[i]=tat[i]-bt[i];

sumtat+=tat[i];

sumwt+=wt[i];

}

avgwt=sumwt/n;

avgtat=sumtat/n;

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

; printf("PN\tBt\tCt\tTat\tWt\n");

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

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

{

printf("%d\t%2d\t%2d\t%2d\t%2d\n",i,bt[i],compt[i],tat[i],wt[i]);

}

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

printf(" Avgwt = %.2f\tAvgtat = %.2f\n",avgwt,avgtat);

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

}

**2.shortest job first cpu scheduling**

/\* Program to Simulate Shortest Job First CPU Scheduling Algorithm \*/

#include<stdio.h>

#include<conio.h>

#include<string.h>

void main()

{

int i,j,n,bt[10],compt[10],

wt[10],tat[10],temp;

float sumwt=0.0,sumtat=0.0,avgwt,avgtat;

printf("Enter number of processes: ");

scanf("%d",&n);

printf("Enter the burst time of %d process\n", n);

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

{

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

}

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

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

if(bt[i]>bt[j])

{

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

}

compt[0]=bt[0];

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

compt[i]=bt[i]+compt[i-1];

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

{

tat[i]=compt[i];

wt[i]=tat[i]-

bt[i];

sumtat+=tat[i];

sumwt+=wt[i];

}

avgwt=sumwt/n;

avgtat=sumtat/n;

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

printf("Bt\tCt\tTat\tWt\n");

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

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

{

printf("%2d\t %2d\t %2d\t %2d\n",i,bt[i],compt[i],tat[i],wt[i]);

}

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

printf(" Avgwt = %.2f\tAvgtat = %.2f\n",avgwt,avgtat);

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

}

**EXPERIMENT NO 6**

**1.producre and consumer using semaphores**

#include<stdio.h>

#include<stdlib.h>

int

mutex=1,full=0,empty=3,x=0;

int main()

{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

printf("\n1.Producer\n2.Consumer\n3.Exit");

while(1)

{

printf("\nEnter your choice:");

scanf("%d",&n);

switch(n)

{

case 1: if((mutex==1)&&(empty!=0))

producer();

else

printf("Buffer is full!!");

break;

case 2: if((mutex==1)&&(full!=0))

consumer();

else

printf("Buffer is empty!!");

break;

case 3:

exit(0);

break;

}

}

return 0;

}

int wait(int s)

{

return (--s);

}

int signal(int s)

{

return(++s);

}

void producer()

{

mutex=wait(mutex);

full=signal(full);

empty=wait(empty);

x++;

printf("\nProducer produces the item %d",x);

mutex=signal(mutex);

}

void consumer()

{

mutex=wait(mutex);

full=wait(full);

empty=signal(empty);

printf("\nConsumer consumes item %d",x);

x--;

mutex=signal(mutex);

}

**EXPERIMENT NO 7**

**Banker algo**

// Banker's Algorithm

#include <stdio.h>

int main()

{

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

int n, m, i, j, k;

n = 5; // Number of processes

m = 3; // Number of resources

int alloc[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 max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix

{ 3, 2, 2 }, // P1

{ 9, 0, 2 }, // P2

{ 2, 2, 2 }, // P3

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

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

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;

}

}

}

}

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);

}

**EXPERIMENT NO 8**

**1.best fit**

#include <stdio.h>

//#include <conio.h> // Removed conio.h

#define max 25

int main() // Changed void main() to int main()

{

int frag[max], b[max], f[max], i, j, nb, nf, temp, lowest = 10000;

int bf[max] = {0}, ff[max] = {0}; // Initialize arrays

//clrscr(); // Removed clrscr()

printf("\nEnter the number of blocks and files: "); // Combined prompts

scanf("%d %d", &nb, &nf); // Combined inputs

printf("\nEnter the size of the blocks:\n");

for (i = 0; i < nb; i++) // Changed loop index start from 1 to 0

{

printf("Block %d: ", i + 1); // Adjusted index display

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

}

printf("Enter the size of the files :-\n");

for (i = 0; i < nf; i++) // Changed loop index start from 1 to 0

{

printf("File %d: ", i + 1); // Adjusted index display

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

}

for (i = 0; i < nf; i++) // Changed loop index start from 1 to 0

{

for (j = 0; j < nb; j++) // Changed loop index start from 1 to 0

{

if (bf[j] != 1)

{

temp = b[j] - f[i];

if (temp >= 0)

{

if (lowest > temp)

{

ff[i] = j;

lowest = temp;

}

}

}

}

frag[i] = lowest;

bf[ff[i]] = 1;

lowest = 10000;

}

printf("\nFile No\tFile Size\tBlock No\tBlock Size\tFragment\n"); // Adjusted formatting

for (i = 0; i < nf && ff[i] != 0; i++) // Changed loop index start from 1 to 0

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, f[i], ff[i] + 1, b[ff[i]], frag[i]); // Adjusted index display

//getch(); // Removed getch()

return 0; // Added return statement

}

//43 23 32 32 32 24

**2. first fit**

#include <stdio.h>

//#include <conio.h> // Removed conio.h

#define max 25

int main() // Changed void main() to int main()

{

int frag[max], b[max], f[max], i, j, nb, nf, temp, highest = 0;

int bf[max] = {0}, ff[max] = {0}; // Initialize arrays

//clrscr(); // Removed clrscr()

printf("\n\tMemory Management Scheme - Worst Fit\n");

printf("Enter the number of blocks and files: ");

scanf("%d %d", &nb, &nf); // Combined inputs

printf("\nEnter the size of the blocks:\n");

for (i = 0; i < nb; i++) // Changed loop index start from 1 to 0

{

printf("Block %d: ", i + 1); // Adjusted index display

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

}

printf("Enter the size of the files:\n");

for (i = 0; i < nf; i++) // Changed loop index start from 1 to 0

{

printf("File %d: ", i + 1); // Adjusted index display

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

}

for (i = 0; i < nf; i++) // Changed loop index start from 1 to 0

{

for (j = 0; j < nb; j++) // Changed loop index start from 1 to 0

{

if (bf[j] != 1) // if bf[j] is not allocated

{

temp = b[j] - f[i];

if (temp >= 0)

{

if (highest < temp)

{

ff[i] = j;

highest = temp;

}

}

}

}

frag[i] = highest;

bf[ff[i]] = 1;

highest = 0;

}

printf("\nFile\_no:\tFile\_size:\tBlock\_no:\tBlock\_size:\tFragment\n"); // Corrected formatting

for (i = 0; i < nf && ff[i] != 0; i++) // Changed loop index start from 1 to 0 and loop condition

printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", i + 1, f[i], ff[i] + 1, b[ff[i]], frag[i]); // Adjusted index display

//getch(); // Removed getch()

return 0; // Added return statement

} //43 48 23 84

**EXPERMENT NO 9**

**1 FIFO page replacement**

//Program to implement FIFO page replacement algorithm/

#include<stdio.h>

#include<conio.h>

void main()

{

int i, j, k, f, pf=0, count=0, rs[25], m[10], n;

//clrscr();

printf("\n Enter the length of reference string -- ");

scanf("%d",&n);

printf("\n Enter the reference string -- ");

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

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

printf("\n Enter no. of frames -- ");

scanf("%d",&f);

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

m[i]=-1;

printf("\n The Page Replacement Processis -- \n");

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

{

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

{

if(m[k]==rs[i])

break;

}

if(k==f)

{

m[count++]=rs[i];

pf++;

}

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

printf("\t%d",m[j]);

if(k==f)

printf("\tPF No. %d",pf);

printf("\n");

if(count==f)

count=0;

}

printf("\n The number of Page Faults using FIFO are %d",pf);

//getch();

} //7 1,3,0,3,5,6,3

**2. Least recently used(LRU)**

#include<stdio.h>

#include<conio.h>

void main()

{

int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0,

next=1;

//clrscr();

printf("Enter the length ofreference string -- ");

scanf("%d",&n);

printf("Enter the reference string -- ");

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

{

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

flag[i]=0;

}

printf("Enterthe number of frames -- ");

scanf("%d",&f);

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

{

count[i]=0;

m[i]=-1;

}

printf("\nThe Page Replacement process is-- \n");

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

{

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

{

if(m[j]==rs[i])

{

flag[i]=1;

count[j]=next;

next++;

}

}

if(flag[i]==0)

{

if(i<f)

{

m[i]=rs[i];

count[i]=next;

next++;

}

else

{

min=0;

for(j=1;j<f;j++)

if(count[min] > count[j])

min=j;

m[min]=rs[i];

count[min]=next;

next++;

}

pf++;

}

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

printf("%d\t", m[j]);

if(flag[i]==0)

printf("PF No. -- %d" , pf);

printf("\n");

}

printf("\nThe number of page faults using LRU are %d",pf);

//getch();

} //7 1,3,0,3,5,6,3

**EXPERIMENT NO 10**

**1 FCFS Disk Scheduling**

#include<stdio.h>

int main()

{ int queue[20],n,head,i,j,k,seek=0,max,diff;

float avg;

printf("Enter the max range of disk\n");

scanf("%d",&max);

printf("Enter the size of queue request\n");

scanf("%d",&n);

printf("Enter the queue of disk positions to be read\n");

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

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

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

scanf("%d",&head);

queue[0]=head;

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

{ diff=abs(queue[j+1]-queue[j]); seek+=diff;

printf("Disk head moves from %d to %d with seek %d\n",queue[j],queue[j+1],diff);

}

printf("Total seek time is %d\n",seek);

avg=seek/(float)n;

printf("Average seek time is %f\n",avg);

return 0;

} //255 5 176 78 60 34 111 0

**2. scan disk scheduling**

#include <stdio.h>

int main() {

int queue[20], n, head, i, j, seek = 0, max, diff, temp, queue1[20], queue2[20], temp1 = 0, temp2 = 0;

float avg;

printf("Enter the max range of disk\n");

scanf("%d", &max);

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

scanf("%d", &head);

printf("Enter the size of queue request\n");

scanf("%d", &n);

printf("Enter the queue of disk positions to be read\n");

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

scanf("%d", &temp);

if (temp >= head) {

queue1[temp1] = temp;

temp1++;

} else {

queue2[temp2] = temp;

temp2++;

}

}

// Sorting queue1 in ascending order

for (i = 0; i < temp1 - 1; i++) {

for (j = i + 1; j < temp1; j++) {

if (queue1[i] > queue1[j]) {

temp = queue1[i];

queue1[i] = queue1[j];

queue1[j] = temp;

}

}

}

// Sorting queue2 in descending order

for (i = 0; i < temp2 - 1; i++) {

for (j = i + 1; j < temp2; j++) {

if (queue2[i] < queue2[j]) {

temp = queue2[i];

queue2[i] = queue2[j];

queue2[j] = temp;

}

}

}

// Merging sorted queues into a single queue

for (i = 1, j = 0; j < temp1; i++, j++)

queue[i] = queue1[j];

queue[i++] = max;

for (j = 0; j < temp2; i++, j++)

queue[i] = queue2[j];

queue[i++] = 0;

queue[0] = head;

// Calculating seek time

for (j = 0; j < n + 2; j++) {

diff = abs(queue[j + 1] - queue[j]);

seek += diff;

printf("Disk head moves from %d to %d with seek %d\n", queue[j], queue[j + 1], diff);

}

printf("Total seek time is %d\n", seek);

avg = (float)seek / n;

printf("Average seek time is %f\n", avg);

return 0;

}//255 0 5 12 215 48 110 83

**3. c-scan disk scheduling**

#include<stdio.h>

int main() {

int queue[20], n, head, i, j, seek = 0, max, diff, temp, queue1[20], queue2[20], temp1 = 0, temp2 = 0;

float avg;

printf("Enter the max range of disk\n");

scanf("%d", &max);

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

scanf("%d", &head);

printf("Enter the size of queue request\n");

scanf("%d", &n);

printf("Enter the queue of disk positions to be read\n");

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

scanf("%d", &temp);

if (temp >= head) {

queue1[temp1] = temp;

temp1++;

}

else {

queue2[temp2] = temp;

temp2++;

}

}

// Sorting queue1 in ascending order

for (i = 0; i < temp1 - 1; i++) {

for (j = i + 1; j < temp1; j++) {

if (queue1[i] > queue1[j]) {

temp = queue1[i];

queue1[i] = queue1[j];

queue1[j] = temp;

}

}

}

// Sorting queue2 in ascending order

for (i = 0; i < temp2 - 1; i++) {

for (j = i + 1; j < temp2; j++) {

if (queue2[i] > queue2[j]) {

temp = queue2[i];

queue2[i] = queue2[j];

queue2[j] = temp;

}

}

}

// Merging sorted queues into a single queue

for (i = 1, j = 0; j < temp1; i++, j++)

queue[i] = queue1[j];

queue[i++] = max;

for (j = 0; j < temp2; i++, j++)

queue[i] = queue2[j];

queue[i++] = 0;

queue[0] = head;

// Calculating seek time

for (j = 0; j < n + 2; j++) {

diff = abs(queue[j + 1] - queue[j]);

seek += diff;

printf("Disk head moves from %d to %d with seek %d\n", queue[j], queue[j + 1], diff);

}

printf("Total seek time is %d\n", seek);

avg = (float)seek / n;

printf("Average seek time is %f\n", avg);

return 0;

}// 255 0 4 254 9 84 110