**//lexical analyzer code 2**

#include<stdio.h>

#include<ctype.h>

#include<string.h>

int main() {

FILE \*input, \*output;

int l=1, t=0, j=0, i, flag;

char ch, str[20];

char keyword[30][30] = {"int","main","if","else","do","while"};

input = fopen("in.txt","r");

output = fopen("out.txt","w");

fprintf(output,"Line no. \t Token no. \t\t Token \t\t\t Output\n\n");

while(!feof(input)) {

i=0;

flag=0;

ch=fgetc(input);

if( ch=='+' || ch== '-' || ch=='\*' || ch=='/' ) {

fprintf(output,"%7d\t\t %7d\t\t Operator\t %7c\n",l,t,ch);

t++;

} else if( ch==';' || ch=='{' || ch=='}' || ch=='(' || ch==')' || ch=='?' || ch=='@' || ch=='!' || ch=='%') {

fprintf(output,"%7d\t\t %7d\t\t Special symbol\t %7c\n",l,t,ch);

t++;

} else if(isdigit(ch)) {

fprintf(output,"%7d\t\t %7d\t\t Constant \t\t %7c\n",l,t,ch);

t++;

} else if(isalpha(ch)) {

str[i]=ch;

i++;

ch=fgetc(input);

while(isalnum(ch) && ch!=' ') {

str[i]=ch;

i++;

ch=fgetc(input);

}

str[i]='\0';

for(j=0;j<=30;j++) {

if(strcmp(str,keyword[j])==0) {

flag=1;

break;

}

}

if(flag==1) {

fprintf(output,"%7d\t\t %7d\t\t Keyword\t %7s\n",l,t,str);

t++;

} else {

fprintf(output,"%7d\t\t %7d\t\t Identifier\t %7s\n",l,t,str);

t++;

}

} else if(ch=='\n') {

l++;

}

}

fclose(input);

fclose(output);

return 0;

}

**// process scheduling algorithms rr ,fcfs,sjf code 4**

#include <stdio.h>

#include <stdbool.h>

struct Process {

int at, bt, ct, tat, wt;

};

void roundRobinScheduling(struct Process p[], int n, int q) {

int rt[n], t = 0, i;

for (i = 0; i < n; i++) rt[i] = p[i].bt;

while (1) {

bool done = true;

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

if (p[i].at <= t && rt[i] > 0) {

done = false;

if (rt[i] > q) {

t += q;

rt[i] -= q;

} else {

t += rt[i];

p[i].ct = t;

rt[i] = 0;

}

}

}

if (done) break;

}

}

void fcfs(struct Process p[], int n) {

int t = 0, i;

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

if (p[i].at > t) t = p[i].at;

p[i].ct = t + p[i].bt;

p[i].tat = p[i].ct - p[i].at;

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

t = p[i].ct;

}

}

void sjf(struct Process p[], int n) {

int i, j;

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

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

if (p[j].bt > p[j + 1].bt) {

struct Process temp = p[j];

p[j] = p[j + 1];

p[j + 1] = temp;

}

fcfs(p, n);

}

int main() {

int n, q, i;

printf("Enter the number of processes: ");

scanf("%d", &n);

struct Process p[n];

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

printf("Enter the Arrival Time for P%d: ", i + 1);

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

printf("Enter the Burst Time for P%d: ", i + 1);

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

}

printf("Choose a scheduling algorithm:\n1. FCFS\n2. SJF\n3. Round Robin\n");

int choice;

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("FCFS Scheduling\n");

fcfs(p, n);

break;

case 2:

printf("SJF Scheduling\n");

sjf(p, n);

break;

case 3:

printf("Enter the time quantum for Round Robin: ");

scanf("%d", &q);

printf("Round Robin Scheduling\n");

roundRobinScheduling(p, n, q);

break;

default:

printf("Invalid choice\n");

return 1;

}

printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");

int totalTAT = 0, totalWT = 0;

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

p[i].tat = p[i].ct - p[i].at;

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

totalTAT += p[i].tat;

totalWT += p[i].wt;

printf("P%d\t%d\t%d\t%d\t%d\t%d\n", i+1, p[i].at, p[i].bt, p[i].ct, p[i].tat, p[i].wt);

}

printf("\nAverage TAT: %.2f\n", (double)totalTAT/n);

printf("Average WT: %.2f\n", (double)totalWT/n);

return 0;

}

**// Process Synchronization code 5**

#include<stdio.h>

int empty=10, mutex=1, full=0, x=0;

void producer() {

mutex--;

empty--;

full++;

x++;

printf("Item is Produced %d",x);

mutex++;

}

void consumer() {

mutex--;

empty++;

full--;

x--;

printf("Item is consumed %d",x);

mutex++;

}

int main() {

int n, i;

do {

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

}

printf("\nDo you want to contrinue if yes then press 1:");

scanf("%d",&i);

} while(i==1);

return 0;

}

**// deadlock handling algorithms code 6**

#include<stdio.h>

int main() {

int p, c, count = 0, i, j, alc[5][3], max[5][3], need[5][3], safe[5], available[3], done[5], terminate = 0;

printf("Enter the number of process and resources");

scanf("%d %d", &p, &c);

printf("Enter allocation of resource of all process %dx%d matrix", p, c);

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

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

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

printf("Enter the max resource process required %dx%d matrix", p, c);

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

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

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

printf("Enter the available resource");

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

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

printf("\nNeed resources matrix are\n");

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

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

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

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

}

printf("\n");

}

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

done[i] = 0;

while (count < p) {

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

if (done[i] == 0) {

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

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

break;

if (j == c) {

safe[count] = i;

done[i] = 1;

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

available[j] += alc[i][j];

count++;

terminate = 0;

} else {

terminate++;

}

}

}

if (terminate == (p - 1)) {

printf("Safe sequence does not exist");

break;

}

}

if (terminate != (p - 1)) {

printf("\nAvailable resource after completion\n");

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

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

printf("\nSafe sequence are\n");

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

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

}

return 0;

}

**// Page Replacement Algorithm code 7**

**//FCFS**

#include<stdio.h>

int main() {

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

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

scanf("%d", &n);

printf("\n Enter the page number :\n");

for(i = 0; 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("\tref string\t page frames\n");

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

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

avail = 0;

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

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

avail = 1;

if (avail == 0) {

frame[j] = a[i];

j = (j + 1) % no;

count++;

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

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

}

printf("\n");

}

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

return 0;

}

**//LRU**

#include <stdio.h>

int findLRU(int time[], int n) {

int i, minimum = time[0], pos = 0;

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

if (time[i] < minimum) {

minimum = time[i];

pos = i;

}

}

return pos;

}

int main() {

int no\_of\_frames, no\_of\_pages, frames[10], pages[30], counter = 0, time[10], flag1, LRUflag2, i, j, pos, faults = 0;

printf("Enter number of frames: ");

scanf("%d", &no\_of\_frames);

printf("Enter number of pages: ");

scanf("%d", &no\_of\_pages);

printf("Enter reference string: ");

for (i = 0; i < no\_of\_pages; ++i) {

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

}

for (i = 0; i < no\_of\_frames; ++i) {

frames[i] = -1;

}

printf("\n F1 \t F2 \tF3");

for (i = 0; i < no\_of\_pages; ++i) {

flag1 = LRUflag2 = 0;

for (j = 0; j < no\_of\_frames; ++j) {

if (frames[j] == pages[i]) {

counter++;

time[j] = counter;

flag1 = LRUflag2 = 1;

break;

}

}

if (flag1 == 0) {

for (j = 0; j < no\_of\_frames; ++j) {

if (frames[j] == -1) {

counter++;

faults++;

frames[j] = pages[i];

time[j] = counter;

LRUflag2 = 1;

break;

}

}

}

if (LRUflag2 == 0) {

pos = findLRU(time, no\_of\_frames);

counter++;

faults++;

frames[pos] = pages[i];

time[pos] = counter;

}

printf("\n");

for (j = 0; j < no\_of\_frames; ++j) {

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

}

}

printf("\nTotal Page Faults = %d", faults);

return 0;

}

**//Disk Scheduling CODE 8**

**//FIFO**

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

int main() {

int i, n, req[50], mov=0, cp;

printf("Enter the current position\n");

scanf("%d", &cp);

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

scanf("%d", &n);

printf("Enter the request order\n");

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

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

}

mov = mov + abs(cp - req[0]);

printf("%d -> %d", cp, req[0]);

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

mov = mov + abs(req[i] - req[i-1]);

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

}

printf("\nTotal head movement = %d\n", mov);

return 0;

}

**//SSTF**

#include<math.h>

#include<stdio.h>

#include<stdlib.h>

int main() {

int i, n, k, req[50], mov=0, cp, index[50], min, a[50], j=0, mini, cp1;

printf("Enter the current position\n");

scanf("%d", &cp);

cp1 = cp;

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

scanf("%d", &n);

printf("Enter the request order\n");

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

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

}

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

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

index[i] = abs(cp - req[i]);

}

min = index[0];

mini = 0;

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

if(min > index[i]) {

min = index[i];

mini = i;

}

}

a[j] = req[mini];

j++;

cp = req[mini];

req[mini] = 999;

}

printf("Sequence is : ");

printf("%d", cp1);

mov = mov + abs(cp1 - a[0]);

printf(" -> %d", a[0]);

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

mov = mov + abs(a[i] - a[i-1]);

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

}

printf("\nTotal head movement = %d\n", mov);

return 0;

}