**Producer Consumer Problem:**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

int buf[5], f, r;

sem\_t mutex, full, empty;

void \*produce(void \*arg)

{

int i;

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

{

sem\_wait(&empty);

sem\_wait(&mutex);

printf("Produced item is %d\n", i);

buf[(++r) % 5] == i;

sleep(1);

sem\_post(&mutex);

sem\_post(&full);

printf("full %u\n", full);

}

}

void \*consume(void \*arg)

{

int item, i;

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

{

sem\_wait(&full);

printf("Full %u\n", full);

sem\_wait(&mutex);

item = buf[(++f) % 5];

printf("Consumed item is %d\n", i);

sleep(i);

sem\_post(&mutex);

sem\_post(&empty);

}

}

int main()

{

pthread\_t tid1, tid2;

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

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

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

pthread\_create(&tid1, NULL, produce, NULL);

pthread\_create(&tid2, NULL, consume, NULL);

pthread\_join(tid1, NULL);

pthread\_join(tid2, NULL);

return 0;

}

Output:

Full 0

Consumed item is 0

Produced item is 0

full 129

Produced item is 1

Full 0

full 0

Consumed item is 1

Full 0

Produced item is 2

full 0

Consumed item is 2

Full 0

Consumed item is 3

Produced item is 3

full 129

Produced item is 4

Full 0

full 0

Consumed item is 4

Full 0

Produced item is 5

full 0

Consumed item is 5

Full 0

Produced item is 6

full 0

Consumed item is 6

Full 0

Consumed item is 7

Produced item is 7

full 129

Produced item is 8

Full 0

full 0

Consumed item is 8

Full 0

Produced item is 9

full 0

Consumed item is 9

**Reader writer:**

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

sem\_t mutex, writeblock;

int data = 0, rcount = 0;

void \*reader(void \*arg)

{

int f;

f = ((int)arg);

sem\_wait(&mutex);

rcount++;

if (rcount == 1)

{

sem\_wait(&writeblock);

}

sem\_post(&mutex);

printf("Data read by the reader%d is %d\n", f, data);

sleep(1);

sem\_wait(&mutex);

rcount--;

if (rcount == 0)

{

sem\_post(&writeblock);

}

sem\_post(&mutex);

}

void \*writer(void \*arg)

{

int f;

f = ((int)arg);

sem\_wait(&writeblock);

data++;

printf("Data written by the writer%d is %d\n", f, data);

sleep(1);

sem\_post(&writeblock);

}

void main()

{

int i, b;

pthread\_t rtid[5], wtid[5];

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

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

for (i = 0; i <= 2; i++)

{

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

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

}

for (i = 0; i <= 2; i++)

{

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

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

}

}

Ouptut:

Data read by the reader0 is 0

Data read by the reader1 is 0

Data read by the reader2 is 0

Data written by the writer0 is 1

Data written by the writer1 is 2

Data written by the writer2 is 3

**Dining Philosopher:**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#define N 5

#define THINKING 2

#define HUNGRY 1

#define EATING 0

#define LEFT (phnum + 4) % N

#define RIGHT (phnum + 1) % N

int state[N];

int phil[N] = { 0, 1, 2, 3, 4 };

sem\_t mutex;

sem\_t S[N];

void test(int phnum){

if (state[phnum] == HUNGRY

&& state[LEFT] != EATING

&& state[RIGHT] != EATING) {

// state that eating

state[phnum] = EATING;

sleep(2);

printf("Philosopher %d takes fork %d and %d\n",phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is Eating\n", phnum + 1);

// sem\_post(&S[phnum]) has no effect during takefork used to wake up hungry philosopher during putfork

sem\_post(&S[phnum]);

}

}

// take up chopsticks

void take\_fork(int phnum){

sem\_wait(&mutex);

// state that hungry

state[phnum] = HUNGRY;

printf("Philosopher %d is Hungry\n", phnum + 1);

// eat if neighbours are not eating

test(phnum);

sem\_post(&mutex);

// if unable to eat wait to be signalled

sem\_wait(&S[phnum]);

sleep(1);

}

// put down chopsticks

void put\_fork(int phnum){

sem\_wait(&mutex);

// state that thinking

state[phnum] = THINKING;

printf("Philosopher %d putting fork %d and %d down\n", phnum + 1, LEFT + 1, phnum + 1);

printf("Philosopher %d is thinking\n", phnum + 1);

test(LEFT);

test(RIGHT);

sem\_post(&mutex);

}

void\* philosopher(void\* num){

while (1) {

int\* i = num;

sleep(1);

take\_fork(\*i);

sleep(0);

put\_fork(\*i);

}

}

int main(){

int i;

pthread\_t thread\_id[N];

// initialize the semaphores

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

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

sem\_init(&S[i], 0, 0);

}

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

// create philosopher processes

pthread\_create(&thread\_id[i], NULL, philosopher, &phil[i]);

printf("Philosopher %d is thinking\n", i + 1);

}

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

pthread\_join(thread\_id[i], NULL);

}

}

Output:

Philosopher 1 is thinking

Philosopher 2 is thinking

Philosopher 3 is thinking

Philosopher 4 is thinking

Philosopher 5 is thinking

Philosopher 1 is Hungry

Philosopher 5 is Hungry

Philosopher 4 is Hungry

Philosopher 2 is Hungry

Philosopher 3 is Hungry

Philosopher 3 takes fork 2 and 3

Philosopher 3 is Eating

Philosopher 3 putting fork 2 and 3 down

Philosopher 3 is thinking

Philosopher 2 takes fork 1 and 2

Philosopher 2 is Eating

Philosopher 4 takes fork 3 and 4

Philosopher 4 is Eating

Philosopher 2 putting fork 1 and 2 down

Philosopher 2 is thinking

Philosopher 1 takes fork 5 and 1

Philosopher 1 is Eating

Philosopher 3 is Hungry

Philosopher 4 putting fork 3 and 4 down

Philosopher 4 is thinking

Philosopher 3 takes fork 2 and 3

Philosopher 3 is Eating

Philosopher 2 is Hungry

Philosopher 1 putting fork 5 and 1 down

Philosopher 1 is thinking

Philosopher 5 takes fork 4 and 5

Philosopher 5 is Eating

Philosopher 4 is Hungry

Philosopher 3 putting fork 2 and 3 down

Philosopher 3 is thinking

so on...

**FIFO:**

#include <iostream>

using namespace std;

int main(){

int x;

cout<<"Enter the length of reference string: ";

cin>>x;

int incomingStream[x];

cout<<"Enter the reference string: ";

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

cin>>incomingStream[i];

}

int pageFaults = 0;

int frames;

cout<<"Enter frame size: ";

cin>>frames;

int m, n, s, pages;

pages = sizeof(incomingStream)/sizeof(incomingStream[0]);

cout<<"Incoming \t Frame 1 \t Frame 2 \t Frame 3";

int temp[frames];

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

{

temp[m] = -1;

}

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

{

s = 0;

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

{

if(incomingStream[m] == temp[n])

{

s++;

pageFaults--;

}

}

pageFaults++;

if((pageFaults <= frames) && (s == 0))

{

temp[m] = incomingStream[m];

}

else if(s == 0)

{

temp[(pageFaults - 1) % frames] = incomingStream[m];

}

cout<<"\n";

cout<<incomingStream[m]<<"\t\t\t";

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

{

if(temp[n] != -1)

cout<<temp[n]<<"\t\t\t";

else

cout<<" - \t\t\t";

}

}

cout<<"\nTotal Page Faults:\t"<<pageFaults<<"\n";

return 0;

}

Output:

Enter the length of reference string: 20

Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter frame size: 3

Incoming Frame 1 Frame 2 Frame 3

7 7 - -

0 7 0 -

1 7 0 1

2 2 0 1

0 2 0 1

3 2 3 1

0 2 3 0

4 4 3 0

2 4 2 0

3 4 2 3

0 0 2 3

3 0 2 3

2 0 2 3

1 0 1 3

2 0 1 2

0 0 1 2

1 0 1 2

7 7 1 2

0 7 0 2

1 7 0 1

Total Page Faults: 15

**LRU:**

#include<iostream>

#include<limits.h>

using namespace std;

int checkHit(int incomingPage, int queue[], int occupied){

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

if(incomingPage == queue[i])

return 1;

}

return 0;

}

void printFrame(int queue[], int occupied)

{

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

cout<<queue[i]<<"\t\t\t";

}

int main(){

int n;

cout<<"Enter the size of reference string: ";

cin>>n;

int incomingStream[n];

cout<<"Enter the reference string: ";

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

cin>>incomingStream[i];

}

int frames;

cout<<"Enter frame size: ";

cin>>frames;

int queue[n];

int distance[n];

int occupied = 0;

int pagefault = 0;

printf("Page\t\tFrame1\t\t\tFrame2\t\t\t Frame3\n");

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

{

cout<<incomingStream[i]<<": \t\t";

if(checkHit(incomingStream[i], queue, occupied)){

printFrame(queue, occupied);

}

// filling when frame(s) is/are empty

else if(occupied < frames){

queue[occupied] = incomingStream[i];

pagefault++;

occupied++;

printFrame(queue, occupied);

}

else{

int max = INT\_MIN;

int index;

// get LRU distance for each item in frame

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

distance[j] = 0;

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

++distance[j];

if(queue[j] == incomingStream[k])

break;

}

if(distance[j] > max){

max = distance[j];

index = j;

}

}

queue[index] = incomingStream[i];

printFrame(queue, occupied);

pagefault++;

}

cout<<"\n";

}

cout<<"Page Fault: "<<pagefault;

return 0;

}

Output:

Enter the size of reference string: 20

Enter the reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Enter frame size: 3

Page Frame1 Frame2 Frame3

7: 7

0: 7 0

1: 7 0 1

2: 2 0 1

0: 2 0 1

3: 2 0 3

0: 2 0 3

4: 4 0 3

2: 4 0 2

3: 4 3 2

0: 0 3 2

3: 0 3 2

2: 0 3 2

1: 1 3 2

2: 1 3 2

0: 1 0 2

1: 1 0 2

7: 1 0 7

0: 1 0 7

1: 1 0 7

Page Fault: 12

**Optimal:**

#include<iostream>

using namespace std;

int search(int key, int frame\_items[], int frame\_occupied){

for (int i = 0; i < frame\_occupied; i++)

if (frame\_items[i] == key)

return 1;

return 0;

}

void printOuterStructure(int max\_frames){

cout<<"Stream ";

for(int i = 0; i < max\_frames; i++)

cout<<"Frame "<< i+1;

}

void printCurrFrames(int item, int frame\_items[], int frame\_occupied, int max\_frames){

cout<<"\n"<<item<< "\t\t";

for(int i = 0; i < max\_frames; i++){

if(i < frame\_occupied)

cout<<frame\_items[i]<<" \t\t";

else

cout<<"- \t\t";

}

}

int predict(int ref\_str[], int frame\_items[], int refStrLen, int index, int frame\_occupied){

int result = -1, farthest = index;

for (int i = 0; i < frame\_occupied; i++) {

int j;

for (j = index; j < refStrLen; j++)

{

if (frame\_items[i] == ref\_str[j])

{

if (j > farthest) {

farthest = j;

result = i;

}

break;

}

}

if (j == refStrLen)

return i;

}

return (result == -1) ? 0 : result;

}

void optimalPage(int ref\_str[], int refStrLen, int frame\_items[], int max\_frames){

int frame\_occupied = 0;

printOuterStructure(max\_frames);

int hits = 0;

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

// If found already in the frame items : HIT

if (search(ref\_str[i], frame\_items, frame\_occupied)) {

hits++;

printCurrFrames(ref\_str[i], frame\_items, frame\_occupied, max\_frames);

continue;

}

if (frame\_occupied < max\_frames){

frame\_items[frame\_occupied] = ref\_str[i];

frame\_occupied++;

printCurrFrames(ref\_str[i], frame\_items, frame\_occupied, max\_frames);

}

else {

int pos = predict(ref\_str, frame\_items, refStrLen, i + 1, frame\_occupied);

frame\_items[pos] = ref\_str[i];

printCurrFrames(ref\_str[i], frame\_items, frame\_occupied, max\_frames);

}

}

cout<<"\n\nHits: "<<hits<<"\n";

cout<<"Misses: "<< refStrLen - hits;

}

int main(){

int refStrLen;

cout<<"Enter size of reference string: ";

cin>>refStrLen;

int ref\_str[refStrLen];

cout<<"Enter the reference string: ";

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

cin>>ref\_str[i];

}

int max\_frames;

cout<<"Enter the size of frame: ";

cin>>max\_frames;

int frame\_items[max\_frames];

optimalPage(ref\_str, refStrLen, frame\_items, max\_frames);

return 0;

}

Output:

Enter size of reference string: 20

Enter the reference string: 1 2 3 4 2 5 3 4 2 6 7 8 7 9 7 8 2 5 4 9

Enter the size of frame: 3

Stream Frame 1Frame 2Frame 3

1 1 - -

2 1 2 -

3 1 2 3

4 4 2 3

2 4 2 3

5 4 5 3

3 4 5 3

4 4 5 3

2 4 5 2

6 6 5 2

7 7 5 2

8 7 8 2

7 7 8 2

9 7 8 9

7 7 8 9

8 7 8 9

2 2 8 9

5 5 8 9

4 4 8 9

9 4 8 9

Hits: 7

Misses: 13