PRODUCER-CONSUMER USING MONITORS

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

#include<stdlib.h>

#include<unistd.h>

sem\_t empty, full;

pthread\_mutex-t mutex;

int buffer[s];

int count = 0;

int b=s,c=0,d=5,p=0;

void \*producer(void\*arg){

int a;

long int num = (long int ) arg;

a = sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

buffer[count]=rand()%10;

b=b-1;

c=c+1;

printf(“\n producer:%cdproducer:%dbuffer empty:%dbufferfull%d”,num+1,buffer[count],b,c);

count++;

sleep(1);

pthread\_mutex\_unlock(&mutex);

r=sem\_post(&full); }

Void\*consumer(void\*arg){

Long int num = (long int)arg;

int a = sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

count--;

b=b+1;

c=c-1;

printf(“\n consumer:%d consumed:%d buffer empty:%d bufferfull:%d”,num+1,buffer[count],b,c);

sleep(1);

pthread\_mutex\_unlock(&mutex);

a=sem\_post(&empty);

}  
int main(){

Int np,nc;

Pthread\_t p[10],c[10];

Unsigned long int I,j,k,l;

Printf(“enter no.of.producer and consumer:”);

Scanf(“%d%d”, &np,$nc);

Printf(“buffer\_size:5 buffer\_empty:5 buffer\_full:0”);

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

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

Pthread\_mutex\_init(&mutex,null);

For(i=0;i<np:i++)

Pthread\_create(&p[i],null,producer,(void\*)i);

For(j=0;j<nc;j++)

Pthread\_create(&c[j],null,consumer,(void\*)j);

For(k=0;k<np;k++)

Pthread\_join(p[k],null);

For(i=0;i<nc;i++)

Pthread\_join(c[i],null);

Printf(“\n”); }

FIFO PAGE REPLACEMENT ALGORITHM

#include < stdio.h >

int main()

{

    int incomingStream[] = {4 , 1 , 2 , 4 , 5};

    int pageFaults = 0;

    int frames = 3;

    int m, n, s, pages;

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

    printf(" 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];

        }

        printf("\n");

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

        for(n = 0; n < frames; n++)

        {

            if(temp[n] != -1)

                printf(" %d\t\t\t", temp[n]);

            else

                printf(" - \t\t\t");

        }

    }

    printf("\nTotal Page Faults:\t%d\n", pageFaults);

    return 0;

}

OPTIMAL PAGE REPLACEMENT ALGORITHM

|  |
| --- |
| #include<stdio.h> |
|  |  |
|  | void optimal(int string[20],int n,int size) |
|  | { |
|  | //Creating array for block storage |
|  | int frames[n]; |
|  | //Initializing each block with -1 |
|  | for (int i=0;i<n;i++) |
|  | frames[i]=-1; |
|  |  |
|  | //Index to insert element |
|  | int index=-1; |
|  |  |
|  | //Counters |
|  | int page\_miss=0; |
|  | int page\_hits=0; |
|  |  |
|  | //Pointer to indicate initially frames filled or not |
|  | int full=0; |
|  |  |
|  | //Traversing each symbol in fifo |
|  | for (int i=0;i<size;i++) |
|  | { |
|  | int symbol=string[i]; |
|  | int flag=0; |
|  |  |
|  | for(int j=0;j<n;j++) |
|  | { |
|  | if (symbol==frames[j]) |
|  | { |
|  | flag=1; |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | if (flag==1) |
|  | { |
|  | printf("\nSymbol: %d Frame: ",symbol); |
|  | for (int j=0;j<n;j++) |
|  | printf("%d ",frames[j]); |
|  | page\_hits+=1; |
|  | } |
|  | else |
|  | { |
|  | //Frames are still empty |
|  | if (full==0) |
|  | { |
|  | index=(index+1)%n; |
|  | frames[index]=symbol; |
|  | page\_miss+=1; |
|  | printf("\nSymbol: %d Frame: ",symbol); |
|  | for (int j=0;j<n;j++) |
|  | printf("%d ",frames[j]); |
|  |  |
|  | //Frames filled or not |
|  | if (i==n-1) |
|  | full=1; |
|  | } |
|  |  |
|  | //Frames are full, now we can apply optimal page replacement |
|  | else |
|  | { |
|  | //First find the index to replace with |
|  | int pos=-1; |
|  | int index=-1; |
|  |  |
|  | //Traversing each symbol and checking their optimal possibility |
|  | for(int j=0;j<n;j++) |
|  | { |
|  | //Whether symbol in frame found or not in future cached frame |
|  | int found=0; |
|  | for (int k=i+1;k<size;k++) |
|  | { |
|  | //If symbol exists in cached string |
|  | if (frames[j]==string[k]) |
|  | { |
|  | found=1; |
|  | if (pos<k) |
|  | { |
|  | pos=k; |
|  | index=j; |
|  | } |
|  | break; |
|  | } |
|  | } |
|  | //Symbol does not exist in cached frame |
|  | if (found==0) |
|  | { |
|  | pos=size; |
|  | index=j; |
|  | } |
|  | } |
|  |  |
|  | //Now assign symbol in lru position |
|  | frames[index]=symbol; |
|  |  |
|  | printf("\nSymbol: %d Frame: ",symbol); |
|  | for (int j=0;j<n;j++) |
|  | printf("%d ",frames[j]); |
|  | } |
|  | } |
|  | } |
|  | printf("\nPage hits: %d",page\_hits); |
|  | printf("\nPage misses: %d",page\_miss); |
|  | } |
|  |  |
|  | //Main function |
|  | int main(void) |
|  | { |
|  | int string[]={2, 3, 4, 2, 1, 3, 7, 5, 4, 3}; |
|  | int no\_frames=3; |
|  | int size=sizeof(string)/sizeof(int); |
|  | optimal(string,no\_frames,size); |
|  | return 0; |
|  | } |