|  |  |
| --- | --- |
| **Lab 7:** | **Program to implement Producer Consumer Problem in C** |

#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);

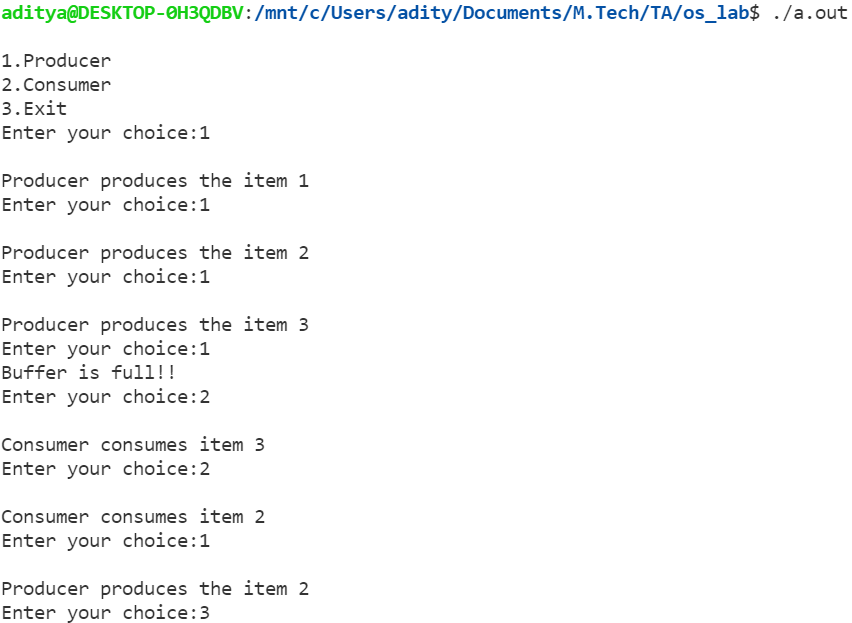
    empty=signal(empty);

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

    x--;

    mutex=signal(mutex);

}



|  |  |
| --- | --- |
| **Lab 8:** | **Program for page replacement policy using: FIFO** |

#include<bits/stdc++.h>

using namespace std;

int pageFaults(int pages[], int n, int capacity)

{

unordered\_set<int> s;

queue<int> indexes;

int page\_faults = 0;

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

{

if (s.size() < capacity)

{

if (s.find(pages[i])==s.end())

{

   s.insert(pages[i]);

page\_faults++;

indexes.push(pages[i]);

}

}

else

{

if (s.find(pages[i]) == s.end())

{

int val = indexes.front();

indexes.pop();

s.erase(val);

s.insert(pages[i]);

indexes.push(pages[i]);

page\_faults++;

}

}

}

return page\_faults;

}

// Driver code

int main()

{

int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

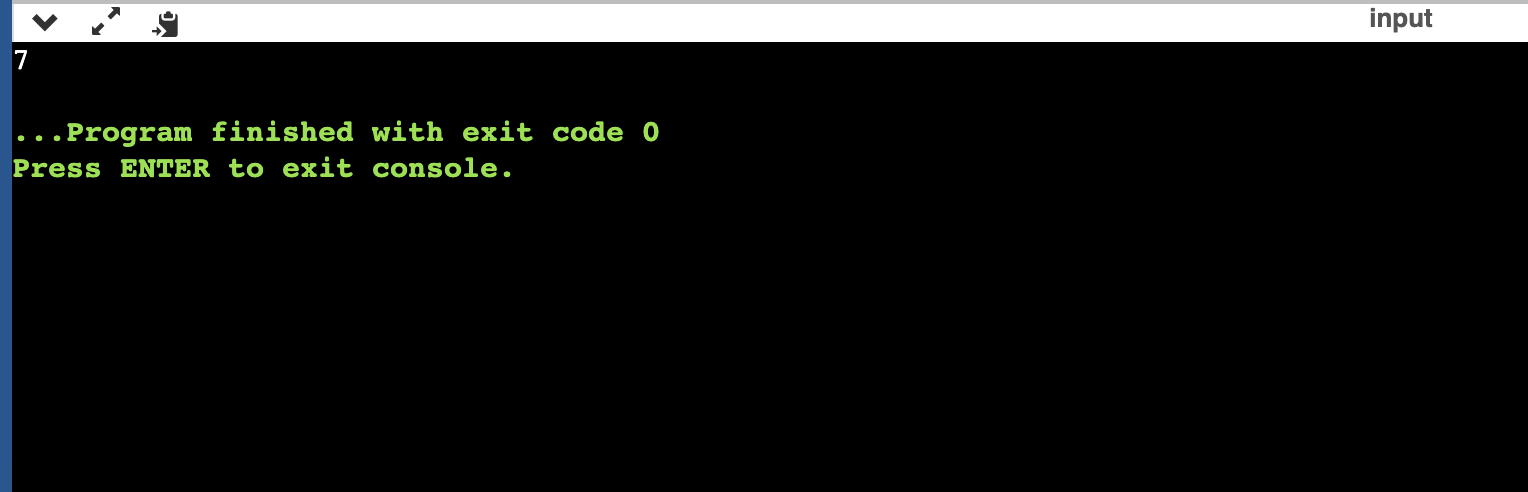
int n = sizeof(pages)/sizeof(pages[0]);

int capacity = 4;

cout << pageFaults(pages, n, capacity);

return 0;

}



|  |  |
| --- | --- |
| **Lab 9:** | **Program for page replacement policy using: LRU** |

#include<bits/stdc++.h>

using namespace std;

int main()

{

int capacity = 4;

int arr[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};

deque<int> q(capacity);

int count=0;

int page\_faults=0;

deque<int>::iterator itr;

q.clear();

for(int i:arr)

{

// Insert it into set if not present

// already which represents page fault

itr = find(q.begin(),q.end(),i);

if((itr == q.end()))

{

++page\_faults;

// Check if the set can hold equal pages

if(q.size() == capacity)

{

q.erase(q.begin());

q.push\_back(i);

}

else{

q.push\_back(i);

}

}

else

{

// Remove the indexes page

q.erase(itr);

// insert the current page

q.push\_back(i);

}

}

cout<<page\_faults;

}

