**1. Print the following Pattern**

**A 1 a B 2 b C 3 c ... Y 25 y Z 26 z**

**Using any one of the following concepts**

**a. Multiprocesses (Hint: using 3 child processes)**

**b. Multithreads (Hint: using 3 Threads)**

**(PS: Process Synchronization or Thread Synchronization is key thing for pattern printing) [25]**

**Descripition**

Here to solve this program we have use multi-threading involves two terms a process and a thread. Program being executed is a process. Threads are independent units of process.

**Code**

#include <stdio.h>

#include <thread.h>

using namespace std;

void fo(int K)

{

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

printf("%c" , Alp);

Alp++;

}

}

class thread\_obj {

public:

void operator()(int z)

{

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

}

};

int Num = 0

int Alp = 65

int lowerAlp = 97

int main()

{

printf("%c" , lowerAlp);

lowerAlp++;

thread th1(fo, 3);

thread th2(thread\_obj(), 3);

auto f = [](int x) {

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

Num ++

printf("%d" , Num);

};

thread th3(f, 3);

th1.join();

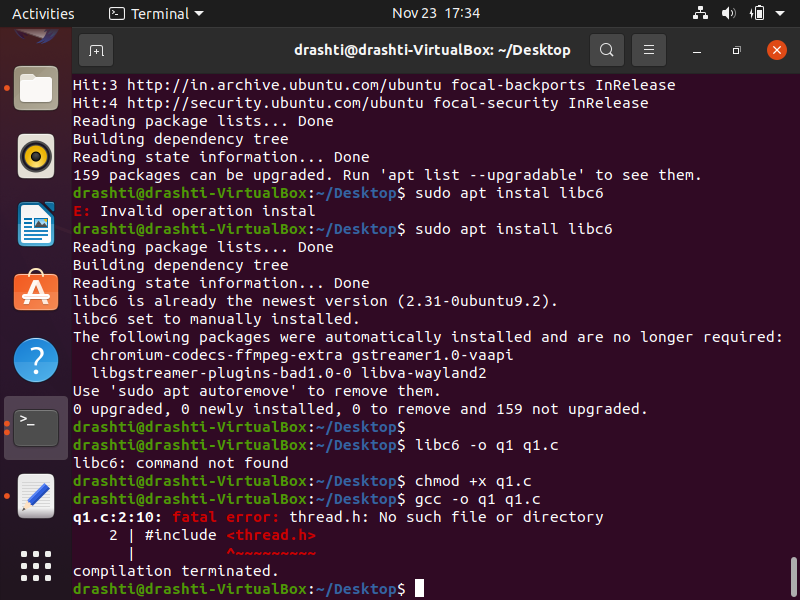
th2.join();

th3.join();

return 0;

}

Screen Shots



**2. Describe and implement any one of the following [20]**

**b. Describe the Buddy’s Algorithm for Memory Allocation and Deallocation along with an example and implement it in C or C++.**

Buddy’s system for memory allocation, sizes of free blocks of integral power of 2 eg 2, 4, 8, 16, 32 etc up the last size of the memory. Memory(whole block) will only be allocated if and only required size is more than 2^U-1 and less than 2^U.

**Example**

Total memory size be 512 KB and let a Process Pr1, requires 70 KB to be swapped in. The hole lists are only for powers of 2, 128 KB will be big enough. Initially no 128KB is there, nor are blocks 256KB. Thus 512KB block is split into two buddies of 256KB each, one is further split into two 128KB blocks and one of them is allocated to the process. Next P2 requires 35KB. Rounding 35KB up to a power of 2, a 64KB block is required.

**C**ode

#include<bits/stdc++.h>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

int size;

vector<pair <int, int> > arr[100000];

map<int, int> mp;

void buddy(int s)

{

int a = ceil(log(s) / log(2));

size = a + 1;

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

arr[i].clear();

arr[a].push\_back(make\_pair(0, s - 1));

}

void allocate(int s)

{

int x = ceil(log(s) / log(2));

if (arr[x].size() > 0)

{

pair<int, int> temp = arr[x][0];

arr[x].erase(arr[x].begin());

cout << "Memory from " << temp.first

<< " to " << temp.second

<< " allocated" << "\n";

mp[temp.first] = temp.second -

temp.first + 1;

}

else

{

int i;

for(i = x + 1; i < size; i++)

{

if (arr[i].size() != 0)

break;

}

if (i == size)

{

cout << "Sorry, failed to allocate memory\n";

}

else

{

pair<int, int> temp;

temp = arr[i][0];

arr[i].erase(arr[i].begin());

i--;

for(;i >= x; i--)

{

pair<int, int> pair1, pair2;

pair1 = make\_pair(temp.first,

temp.first +

(temp.second -

temp.first) / 2);

pair2 = make\_pair(temp.first +

(temp.second -

temp.first + 1) / 2,

temp.second);

arr[i].push\_back(pair1);

arr[i].push\_back(pair2);

temp = arr[i][0];

arr[i].erase(arr[i].begin());

}

cout << "Memory from " << temp.first

<< " to " << temp.second

<< " allocate" << "\n";

mp[temp.first] = temp.second -

temp.first + 1;

}

}

}

void deallocate(int id)

{

if(mp.find(id) == mp.end())

{

cout << "Sorry, invalid free request\n";

return;

}

int n = ceil(log(mp[id]) / log(2));

int i, buddyNumber, buddyAddress;

arr[n].push\_back(make\_pair(id,

id + pow(2, n) - 1));

cout << "Memory block from " << id

<< " to "<< id + pow(2, n) - 1

<< " freed\n";

buddyNumber = id / mp[id];

if (buddyNumber % 2 != 0)

buddyAddress = id - pow(2, n);

else

buddyAddress = id + pow(2, n);

for(i = 0; i < arr[n].size(); i++)

{

if (arr[n][i].first == buddyAddress)

{

if (buddyNumber % 2 == 0)

{

arr[n + 1].push\_back(make\_pair(id,

id + 2 \* (pow(2, n) - 1)));

cout << "Coalescing of blocks starting at "

<< id << " and " << buddyAddress

<< " was done" << "\n";

}

else

{

arr[n + 1].push\_back(make\_pair(

buddyAddress, buddyAddress +

2 \* (pow(2, n))));

cout << "Coalescing of blocks starting at "

<< buddyAddress << " and "

<< id << " was done" << "\n";

}

arr[n].erase(arr[n].begin() + i);

arr[n].erase(arr[n].begin() +

arr[n].size() - 1);

break;

}

}

mp.erase(id);

}

int main()

{

buddy(128);

allocate(16);

allocate(16);

allocate(16);

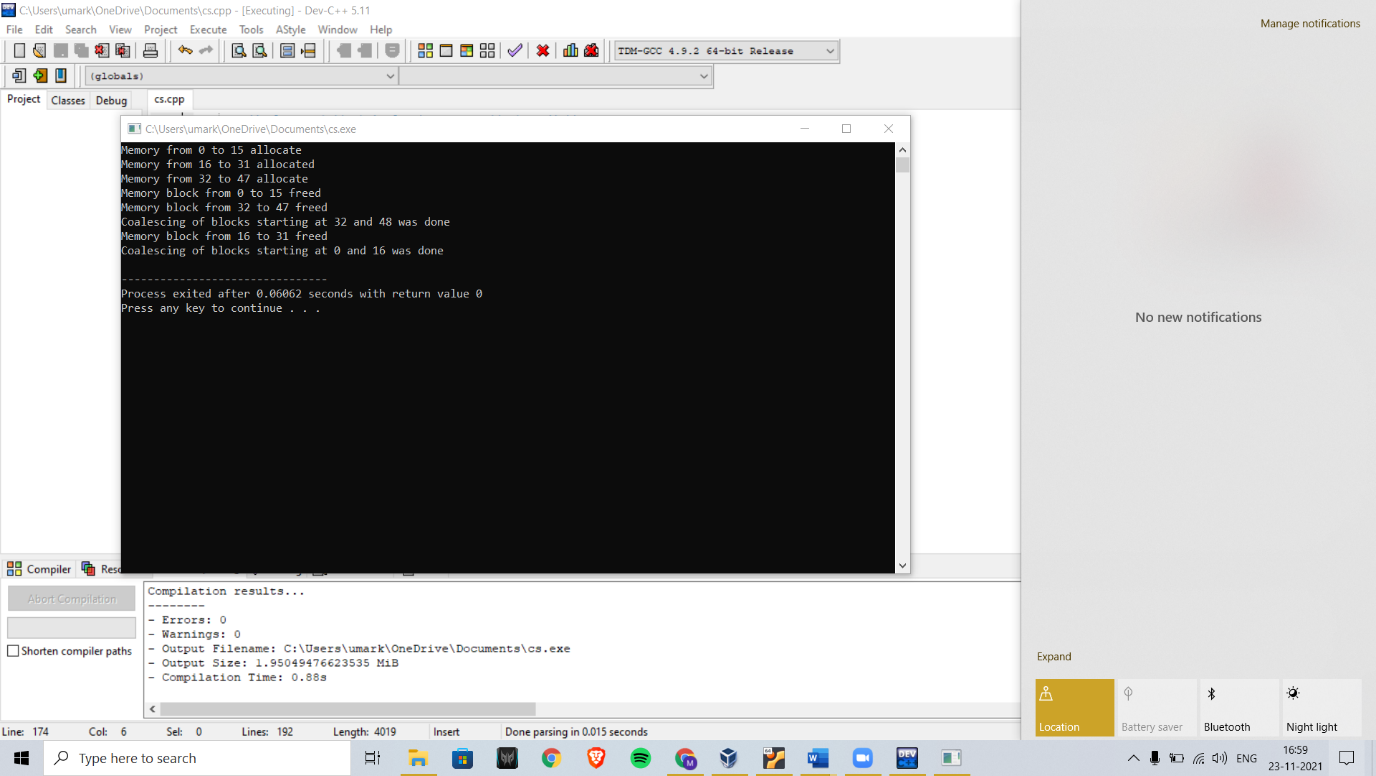
deallocate(0);

deallocate(32);

deallocate(16);

return 0;

}ode

****

**3. [Bonus] Describe what is Producer Consumer Problem and its solution in detail using Semaphores and Mutex and implement it in C.**

The producer- consumer problem is a problem that is used for multi- process synchronization. Which is synchronization between more than one processes.

Here in this problem, there is one producer who is producing something and there is one Consumer who is consuming the goods produced by that producer. Here our producer and consumer share the same memory buffer that is of fixed-size.

Producer generates the data, then puts it into buffer, and again start generating data. On the other hand, Consumer is to consume that data from the buffer

But Here is a problem is when the producer produces data only when our buffer is not full. If buffer is full, then producer shouldn’t be allowed to keep any data in the buffer. If the buffer is empty, then the consumer shouldn’t be allowed to take any data from the buffer. In short consumer should consume data only when the buffer is not empty.

Also, Producer and Consumer should not access the buffer at the same time. Note we have a fixed size buffer.

Solution : Producer has to either discard or go for sleep if the buffer is full. For the next time the consumer removes an item from the buffer, it notifies the producer, who starts to fill the buffer again. Consumer can go to sleep or discard if it finds buffer empty. Producer puts the data into buffer which will make sleeping buffer wake up.

Code:

#include <stdio.h>

#include <stdlib.h>

int mut = 1;

int empt = 10, y = 0;

int fu = 0;

void producer()

{

--mut;

++fu;

--empt;

y++;

printf("\nProducer produces"

"item %d",

y);

++mut;

}

void consumer()

{

--mut;

--fu;

++empt;

printf("\nConsumer consumes "

"item %d",

y);

y--;

++mut;

}

int main()

{

int n, i;

printf("\n1. Please Press 1 for Producer"

"\n2. Please Press 2 for Consumer"

"\n3. Please Press 3 for Exit");

#pragma omp critical

for (i = 1; i > 0; i++) {

printf("\n Please Enter your choice:");

scanf("%d", &n);

switch (n) {

case 1:

if ((mut == 1)

&& (empt != 0)) {

producer();

}

else {

printf("Ops !Buffer is full!");

}

break;

case 2:

if ((mut == 1)

&& (fu != 0)) {

consumer();

}

else {

printf("Your buffer is empty!");

}

break;

case 3:

exit(0);

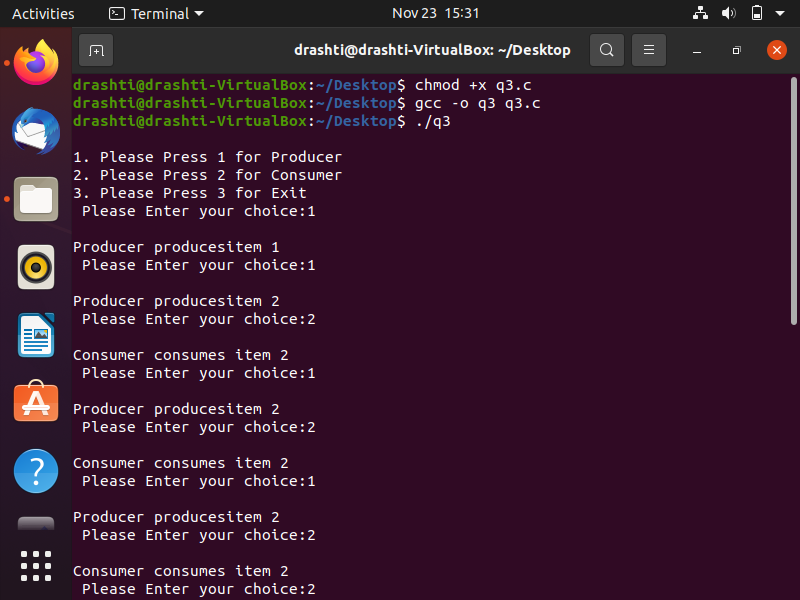
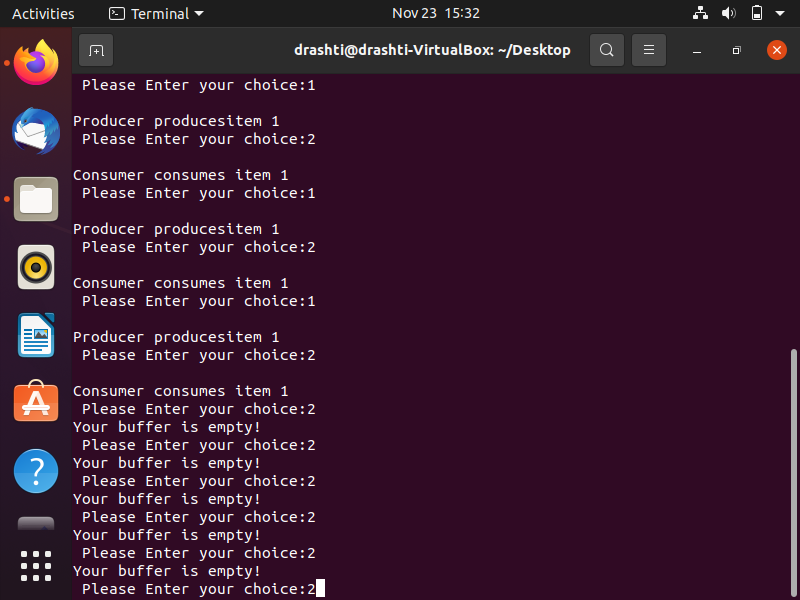
break;

}

}

}

**Screen shots**

****