Name : Dhruvanshu Parmar

Enrollment ID : AU1940166

Q1)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).

ANS)

2b) Describe the Buddy's Algorithm for Memory Allocation and Deallocation along with an example and implement it in C or C++.

i) Allocation of memory

```
#include<bits/stdc++.h>
using namespace std;


int size;


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


map<int, int> mp;

void initialize(int xy)
{
```

```
	int n = ceil(log(xy) / log(2));
	size = n + 1;

	for(int a = 0; a <= n; a++)
		free_list[a].clear();


	free_list[n].push_back(make_pair(0, xy - 1));
}

void allocate(int xy)
{


	int n = ceil(log(xy) / log(2));


	if (free_list[n].size() > 0)
	{
		pair<int, int> temp = free_list[n][0];


		free_list[n].erase(free_list[n].begin());
		cout << "Memory from " << temp.first
			<< " to " << temp.second << " allocated"
			<< "\n";


		mp[temp.first] = temp.second -
						temp.first + 1;
	}
	else
	{
		int a;
		for(a = n + 1; a < size; a++)
		{


			if(free_list[a].size() != 0)
				break;
		}


		if (a == size)
```

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


        else
        {
                pair<int, int> temp;
                temp = free_list[a][0];

                free_list[a].erase(free_list[a].begin());
                a--;

                for(; a >= n; a--)
                {


                        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);

                        free_list[a].push_back(pair1);


                        free_list[a].push_back(pair2);
                        temp = free_list[a][0];


                        free_list[a].erase(free_list[a].begin());
                }
                cout << "Memory from " << temp.first
                        << " to " << temp.second
                        << " allocated" << "\n";

                mp[temp.first] = temp.second -
                                                temp.first + 1;
        }
}
```

```
}


int main()
{


	int total,c,req;
	cin>>total;
	initialize(total);
	while(true)
	{
		cin>>req;
		if(req < 0)
			break;
		allocate(req);
	}

	initialize(128);
	allocate(32);
	allocate(7);
	allocate(64);
	allocate(56);

	return 0;
}
```

ii) Deallocation of memory

```
#include<bits/stdc++.h>
using namespace std;


int size;


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


map<int, int> mp;

void Buddy(int b)
{
```

```
        int n = ceil(log(b) / log(2));

        size = n + 1;
        for(int a = 0; a <= n; a++)
                arr[a].clear();


        arr[n].push_back(make_pair(0, b - 1));
}

void allocate(int b)
{


        int x = ceil(log(b) / 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 a;


                for(a = x + 1; a < size; a++)
                {


                        if (arr[a].size() != 0)
```

```
                    break;
}


if (a == size)
{
        cout << "Sorry, failed to allocate memory\n";
}


else
{
        pair<int, int> temp;
        temp = arr[a][0];


        arr[a].erase(arr[a].begin());
        a--;

        for(;a >= x; a--)
        {


                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[a].push_back(pair1);


                arr[a].push_back(pair2);
                temp = arr[a][0];


                arr[a].erase(arr[a].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 a, 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(a = 0; a < arr[n].size(); a++)
        {


                if (arr[n][a].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() + a);
                        arr[n].erase(arr[n].begin() +
                        arr[n].size() - 1);
                        break;
                }
        }


        mp.erase(id);
}


int main()
{



        int total,c,req;
        cout<<"Enter Total Memory Size (in Bytes) => ";
        cin>>total;
        initialize(total);
        label:
```

```
while(1)
{
	cout<<"\n1. Add Process into Memory\n
	2. Remove Process \n3. Allocation Map\n4. Exit\n=> ";
	cin>>c;
	switch(c)
	{
		case 1:
		cout<<"Enter Process Size (in Bytes) => ";
		cin>>req;
		cout<<"\n===>";
		allocate(req);
		break;

		case 2:
		cout<<"Enter Starting Address => ";
		cin>>req;
		cout<<"\n===>";
		deallocate(req);
		break;

		case 3:
		print();
		break;

		case 4:
		exit(0);
		break;

		default:
		goto label;
	}
}

Buddy(128);
allocate(16);
allocate(16);
allocate(16);
allocate(16);
deallocate(0);
deallocate(9);
deallocate(32);
deallocate(16);
```

```
	return 0;
}
```

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

```
#include <stdio.h>
#include <stdlib.h>

int mutex = 1;

int full = 0;

int empty = 10, x = 0;

void producer()
{
	--mutex;

	++full;

	--empty;

	x++;
	printf("\nProducer produces"
		"item %d",
		x);

	++mutex;
}

void consumer()
{
	--mutex;

	--full;
```

```
        ++empty;
        printf("\nConsumer consumes "
                "item %d",
                x);
        x--;

        ++mutex;
}

int main()
{
        int n, i;
        printf("\n1. Press 1 for Producer"
                "\n2. Press 2 for Consumer"
                "\n3. Press 3 for Exit");

#pragma omp critical

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

                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;
        }
    }
}
```