Report

* The finished array tracks which processes have successfully completed.
* finished\_count helps determine when all processes are done.
* available holds the current availability of resources

A white square with red writing on it

AI-generated content may be incorrect.

This matrix represents our allocation and request matrices. Where each processes has a row and each column is the amount of resources of the respective column. The numbers represent the way we are moving (iterations)

The first nested for loop:

help us iterate over the matrix and subtracts the resources already allocated to processes from the total system resources moving column by column, so for example we finish all the A’s them move over to B’s etc.

for (int i = 0; i < resource\_types; i++)

{

available[i] = num\_resources[i];

for (int j = 0; j < num\_process; j++) {

available[i] -= matrix\_allocation[j][i]; }

}

Second Nested for Loop :

* This loop iterates through all the unfinished processes.
* Checks if the process’s resource request can be satisfied using the currently available resources.

for (int i = 0; i < num\_process; i++) {

if (!finished[i]) {

bool have\_enough\_resources = true;

// Check if the process can be satisfied

for (int j = 0; j < resource\_types; j++) {

if (matrix\_requests[i][j] > available[j]) {

have\_enough\_resources = false;

break;

}

}

Allocating Requested Resources and Marking done:

* If a process's request can be fulfilled, it is marked as complete.
* The resources allocated to it are released back to the system.

if (have\_enough\_resources) {

for (int j = 0; j < resource\_types; j++)

{

available[j] += matrix\_allocation[i][j]; // return the allocated resources

}

finished[i] = true;

finished\_count++;

any\_process\_completed = true;

}

}

Deadlock Detection Check

* If no process could proceed in the current iteration, a deadlock is detected.
* The function prints "Deadlock detected!" and exits.
* If all processes finish successfully, the system is free of deadlocks.

if (!any\_process\_completed) {

cout << "Deadlock detected!";

return;}

cout << "No Deadlock detected!";

The Main Function

* num\_process and resource\_types store the number of processes and resource types.
* num\_resources[10] holds the total available resources for each type.
* matrix\_allocation[10][10] represents the resources allocated to each process.
* matrix\_requests[10][10] contains the resource requests made by processes
* Initialize the path to the file and then we call on the functions .

int num\_process, resource\_types;

int num\_resources[10], matrix\_allocation[10][10], matrix\_requests[10][10]; // Intialize the arrays with random sizes (not the most efficient)

string filename = "C:\\Users\\salla\\OneDrive\\Desktop\\DeadLock\_Detection\\Deadlock\_Detection\\Deadlock.txt";

read\_parameters(filename, num\_process, resource\_types, num\_resources, matrix\_allocation, matrix\_requests);

determine\_deadlock(num\_process, resource\_types, num\_resources, matrix\_allocation, matrix\_requests);

Source Code

#include <iostream>

#include <fstream>

#include <sstream>

using namespace std;

// Ai Generated

void read\_parameters(const string& filename, int& num\_process, int& resource\_types, int num\_resources[], int matrix\_allocation[][10], int matrix\_requests[][10]) {

ifstream file(filename);

if (!file) {

cerr << "Error: Unable to open file." << endl;

return;

}

string line;

getline(file, line);

stringstream ss1(line);

ss1 >> num\_process >> resource\_types;

getline(file, line);

stringstream ss2(line);

for (int i = 0; i < resource\_types; i++) // fills the amount of resources we have

{

ss2 >> num\_resources[i];

}

for (int i = 0; i < num\_process; i++) // fills the allocation matrix

{

getline(file, line);

stringstream ss3(line);

for (int j = 0; j < resource\_types; j++) {

ss3 >> matrix\_allocation[i][j];

}

}

for (int i = 0; i < num\_process; i++) // fills the matrix requests

{

getline(file, line);

stringstream ss4(line);

for (int j = 0; j < resource\_types; j++) {

ss4 >> matrix\_requests[i][j];

}

}

file.close();

}

//

void determine\_deadlock(int num\_process, int resource\_types, int num\_resources[], int matrix\_allocation[][10], int matrix\_requests[][10]) {

bool finished[10] = { false }; // Tracks completed processes

int finished\_count = 0;

int available[10];

// Calculate available resources correctly

for (int i = 0; i < resource\_types; i++) {

available[i] = num\_resources[i]; // make a copy of the amount of resources that we can directly change

for (int j = 0; j < num\_process; j++) {

available[i] -= matrix\_allocation[j][i]; // we move column by column,deducing resource a from all the proccesses then moving to the next resource

}

}

while (finished\_count < num\_process)

{

bool any\_process\_completed = false;

for (int i = 0; i < num\_process; i++) {

if (!finished[i]) {

bool have\_enough\_resources = true;

// Check if the process can be satisfied

for (int j = 0; j < resource\_types; j++) {

if (matrix\_requests[i][j] > available[j]) {

have\_enough\_resources = false;

break;

}

}

// If we have enough resources , mark process as finished

if (have\_enough\_resources) {

for (int j = 0; j < resource\_types; j++)

{

available[j] += matrix\_allocation[i][j]; // return the allocated resources

}

finished[i] = true;

finished\_count++;

any\_process\_completed = true;

}

}

}

if (!any\_process\_completed) {

cout << "Deadlock detected!";

return;

}

}

cout << "No Deadlock detected!";

}

int main() {

int num\_process, resource\_types;

int num\_resources[10], matrix\_allocation[10][10], matrix\_requests[10][10]; // Intialize the arrays with random sizes (not the most efficient)

string filename = "C:\\Users\\salla\\OneDrive\\Desktop\\DeadLock\_Detection\\Deadlock\_Detection\\Deadlock.txt";

read\_parameters(filename, num\_process, resource\_types, num\_resources, matrix\_allocation, matrix\_requests);

determine\_deadlock(num\_process, resource\_types, num\_resources, matrix\_allocation, matrix\_requests);

return 0;

}