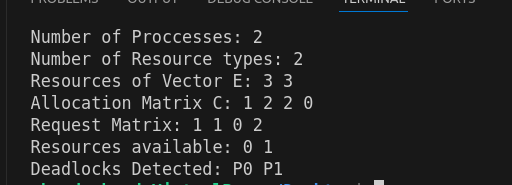
Task 7 Report

In this task we implemented a Deadlock detection where we have a file with the number of processes and the number of resource types and three vectors where they play all different roles.

|  |  |
| --- | --- |
| Number of Processes | Amount of processes that will occur (amount of columns added from vector C) |
| Number of Resource types | Amount of Resource types that are available( Example Memory ) (adds new values for each resource type in a matrix) |
| Resource Available (E) | Amount of Resources that are available for each resource type |
| Allocated Resources (C) | Currently allocated amount of resources for a process |
| Request Resources (R) | Additional amount of resources requested for each Process |

We detect deadlocks by checking if there are enough resources to accommodate all processes. If yes, then no deadlocks occur if not then there will be a deadlock.

GitHub: <https://github.com/yesso2004/Operating-sys>



#include <iostream>

#include <string>

#include <fstream>

#include <vector>

using namespace std;

void PrintMatrix(vector<vector<int>> &M, int row, int column)

{

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

{

for (int j = 0; j < column; j++)

{

cout << M[i][j] << " ";

}

cout << endl;

}

}

bool DeadLockDetection(vector<int> &E, vector<vector<int>> &C, vector<vector<int>> &R, int Processes, int ResourceTypes)

{

vector<int> Resources = E;

vector<bool> Done(Processes, false);

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

{

for (int j = 0; j < ResourceTypes; j++)

{

if (C[i][j] > E[j])

{

cout << "Error: Allocation exceeds total resources." << endl;

return false;

}

Resources[j] -= C[i][j];

}

}

cout << "Resources available: ";

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

{

cout << Resources[i] << " ";

}

cout << endl;

bool Progression = true;

while (Progression)

{

Progression = false;

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

{

if (!Done[i])

{

bool Runnable = true;

for (int j = 0; j < ResourceTypes; j++)

{

if (R[i][j] > Resources[j])

{

Runnable = false;

break;

}

}

if (Runnable)

{

for (int j = 0; j < ResourceTypes; j++)

{

Resources[j] += C[i][j];

}

Done[i] = true;

Progression = true;

}

}

}

}

cout << "Boolean Array: ";

for (int i = 0; i < Done.size(); i++)

{

cout << Done[i] << " ";

}

cout << endl;

vector<int> Deadlocks;

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

{

if (!Done[i])

{

Deadlocks.push\_back(i);

}

}

if (!Deadlocks.empty())

{

cout << "Deadlocks Detected: ";

for (int i = 0; i < Deadlocks.size(); i++)

{

cout << "P" << Deadlocks[i] << " ";

}

cout << endl;

return true;

}

cout << "No deadlocks detected" << endl;

return false;

}

void ProcessFile(const string &FileName)

{

ifstream File(FileName);

if (!File)

{

cout << "Failed to open file." << endl;

return;

}

int Processes, ResourceTypes;

File >> Processes >> ResourceTypes;

if (File.fail())

{

cout << "Error reading processes and resource types." << endl;

return;

}

cout << "Number of Processes: " << Processes << endl;

cout << "Number of Resource Types: " << ResourceTypes << endl;

vector<int> E(ResourceTypes);

cout << "Resources of Vector E: ";

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

{

File >> E[i];

if (File.fail())

{

cout << "Error reading E vector." << endl;

return;

}

cout << E[i] << " ";

}

cout << endl;

vector<vector<int>> C(Processes, vector<int>(ResourceTypes));

cout << "Allocation Matrix C: " << endl;

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

{

for (int j = 0; j < ResourceTypes; j++)

{

File >> C[i][j];

if (File.fail())

{

cout << "Error reading allocation matrix." << endl;

return;

}

}

}

PrintMatrix(C, Processes, ResourceTypes);

vector<vector<int>> Requests(Processes, vector<int>(ResourceTypes));

cout << "Request Matrix: " << endl;

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

{

for (int j = 0; j < ResourceTypes; j++)

{

File >> Requests[i][j];

if (File.fail())

{

cout << "Error reading request matrix." << endl;

return;

}

}

}

PrintMatrix(Requests, Processes, ResourceTypes);

File.close();

DeadLockDetection(E, C, Requests, Processes, ResourceTypes);

}

int main()

{

string FileName;

cout << "Enter a file name: ";

getline(cin, FileName);

ProcessFile(FileName);

}