Reanna Laurell

Programming Assignment #2

Operating Systems

Code Files:

* Source.cpp

Description:

For this programming assignment, we were required to implement a program based on Banker’s Algorithm for avoiding deadlock, to check if a given sequence is safe or unsafe. We could have either coded it in C or C++ in Linux/Unix. Banker’s Algorithm is a deadlock avoidance algorithm that tests for a given set of data to see if it is safe or unsafe. It is called Banker’s Algorithm as it is used a lot in banks to help determine if they can give out a loan to a certain client or not.

We were given a system with five processes P0 through P4 and three resources of type A, B, C. Resource type A has ten instances, B has five instances and type C has seven instances, with the following information given:

Processes Allocation Max Available

A B C A B C A B C

P0 0 1 0 7 5 3 3 3 2

P1 2 0 0 3 2 2

P2 3 0 2 9 0 2

P3 2 1 1 2 2 2

P4 0 0 2 4 3 3

With this given information, we had to determine if the system was in a safe state or not and if it was what is that safe state. Through the code implementation that you will see in the results section below, the system does have a safe state.

Code:

#include <iostream>

using namespace std;

int main(){

int loop1var; //looping through the need array (2d) as array 1

int loop2var; //looping through the need array (2d) as array 2

int proLoop; //looping through the number of processes

const int numPro = 5; //number of processes

const int numReso = 3; //number of resources

int alloc[5][3] = { { 0, 1, 0 }, //P0 | allocation matrix

{ 2, 0, 0 }, //P1

{ 3, 0, 2 }, //P2

{ 2, 1, 1 }, //P3

{ 0, 0, 2 } }; //P4

int max[5][3] = { { 7, 5, 3 }, //P0 | max matrix

{ 3, 2, 2 }, //P1

{ 9, 0, 2 }, //P2

{ 2, 2, 2 }, //P3

{ 4, 3, 3 } }; //P4

int avail[3] = { 3, 3, 2 }; //available resources

//f is the result of if the algorithm is safe or not and should be 1 is safe

int result[numPro], ans[numPro], ind = 0; //setting everything in array to zero

for (proLoop = 0; proLoop < numPro; proLoop++) {

result[proLoop] = 0;

}

int need[numPro][numReso]; //calc the need

for (loop1var = 0; loop1var < numPro; loop1var++) {

for (loop2var = 0; loop2var < numReso; loop2var++)

need[loop1var][loop2var] = max[loop1var][loop2var] - alloc[loop1var][loop2var];

}

int index = 0; //index for each of the elements in the process

for (proLoop = 0; proLoop < 5; proLoop++) {

for (loop1var = 0; loop1var < numPro; loop1var++) {

if (result[loop1var] == 0) {

int flag = 0; //goes thorugh 3 times

for (loop2var = 0; loop2var < numReso; loop2var++) {

if (need[loop1var][loop2var] > avail[loop2var]) { //checking if the need is greater than the aviable space

flag = 1; //safe

break;

}

}

if (flag == 0) { //unsafe sequence

ans[ind++] = loop1var; //if index is safe it starts to fill the new one

for (index = 0; index < numReso; index++) //looping through all resources

avail[index] += alloc[loop1var][index]; //adding each of the resources for this process to the available array

result[loop1var] = 1; //process is now considered safe

}

}

}

}

int flag = 1;

// To check if sequence is safe or not

for (int loop1var = 0; loop1var < numPro; loop1var++){

if (result[loop1var] == 0){ //checking if the flag is 0 meaning the squence would not be safe

flag = 0;

cout << "The given sequence is not safe";

break;

}

}

if (flag == 1){ //checking if flag is 1 stating that the given squence would be sage

cout << "Following is the SAFE Sequence" << endl;

for (loop1var = 0; loop1var < numPro - 1; loop1var++)

cout << ans[loop1var] << " -> ";

cout << ans[numPro - 1] << endl;

}

return (0);

}

Results:

From programming Banker’s Algorithm, I was able to determine that the given sequence was considered a safe one and printed out the following result shown if Figure 1.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 1: Displaying the results of the safe sequence: 1 -> 3 -> 4 -> 0 -> 2

Conclusion:

From this programming assignment, I was able to gain a better understanding what the Banker’s Algorithm is and how to go about using it to figure out if a system is safe or not. By figuring out if a system is safe or not, you must go through each set of numbers in the array or matrix and compare the “need” to the “work”, determining which sets of data are true or false. Once that is done, you then want to start adding the work to the allocation data to check if all the results that were false change to true. Once this step is done you will be able to check if the finish result is true or false, or in the program written, 0 for false and 1 for true.