

**1,2:Find the Need Matrix And Safe sequence:**

#include<iostream>

using namespace std;

const int p = 5;

const int r = 3;

void needcalc(int need[p][r], int maxm[p][r], int allot[p][r]) {

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

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

need[i][j] = maxm[i][j] - allot[i][j];

}

bool safe(int avail[r], int maxm[p][r], int allot[p][r]) {

int need[p][r], work[r], seq[p], cnt = 0;

bool done[p] = {0};

needcalc(need, maxm, allot);

cout << "\nNeed Matrix:\n";

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

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

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

cout << endl;

}

for (int i = 0; i < r; i++) work[i] = avail[i];

while (cnt < p) {

bool ok = 0;

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

if (!done[i]) {

int j;

for (j = 0; j < r; j++)

if (need[i][j] > work[j]) break;

if (j == r) {

for (int k = 0; k < r; k++)

work[k] += allot[i][k];

seq[cnt++] = i;

done[i] = 1;

ok = 1;

}

}

}

if (!ok) {

cout << "\nSystem is not in a safe state.";

return 0;

}

}

cout << "\nSystem is in a safe state.\nSafe Sequence: ";

for (int i = 0; i < p; i++) cout << seq[i] << " ";

return 1;

int main() {

int avail[r], maxm[p][r], allot[p][r];

cout << "Enter Available Resources (" << r << " values):\n";

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

cin >> avail[i];

cout << "Enter Maximum Matrix row by row:\n";

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

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

cin >> maxm[i][j];

cout << "Enter Allocation Matrix row by row:\n";

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

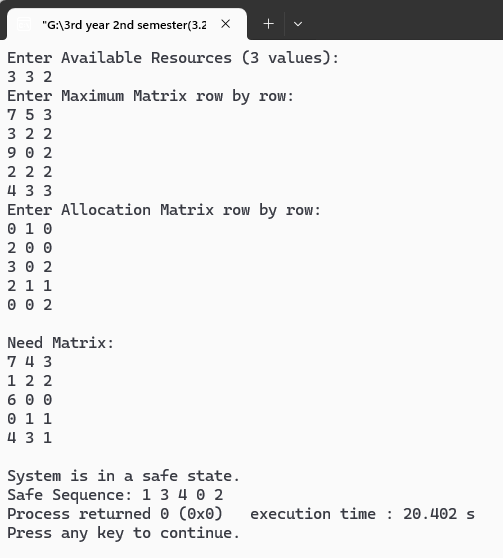
for (int j = 0; j < r; j++)

cin >> allot[i][j];

safe(avail, maxm, allot);

return 0;

}



**Task 03: Suppose now process P1 requests one additional instance of resource type A and two instances of resource type C, can the request be granted immediately? If granted, then print the sequence.**

#include<iostream>

using namespace std;

const int p = 5;

const int r = 3;

void needcalc(int need[p][r], int maxm[p][r], int allot[p][r]) {

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

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

need[i][j] = maxm[i][j] - allot[i][j];

}

bool safe(int avail[r], int maxm[p][r], int allot[p][r]) {

int need[p][r], work[r], seq[p], cnt = 0;

bool done[p] = {0};

needcalc(need, maxm, allot);

for (int i = 0; i < r; i++) work[i] = avail[i];

while (cnt < p) {

bool ok = 0;

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

if (!done[i]) {

int j;

for (j = 0; j < r; j++)

if (need[i][j] > work[j]) break;

if (j == r) {

for (int k = 0; k < r; k++)

work[k] += allot[i][k];

seq[cnt++] = i;

done[i] = 1;

ok = 1;

}

}

}

if (!ok) {

cout << "Not Safe";

return 0;

}

}

cout << "Safe\nSequence: ";

for (int i = 0; i < p; i++) cout << seq[i] << " ";

return 1;

}

int main() {

int avail[r] = {3, 3, 2};

int maxm[p][r] = {

{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}

};

int allot[p][r] = {

{0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2}

};

int req[r] = {1, 0, 2};

int need[p][r];

needcalc(need, maxm, allot);

cout << "Request from P1: ";

for (int i = 0; i < r; i++) cout << req[i] << " ";

cout << endl;

bool canGrant = true;

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

if (req[i] > need[1][i] || req[i] > avail[i]) {

canGrant = false;

break;

}

}

if (!canGrant) {

cout << "Cannot be granted immediately.";

return 0;

}

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

avail[i] -= req[i];

allot[1][i] += req[i];

need[1][i] -= req[i];

}

if (safe(avail, maxm, allot))

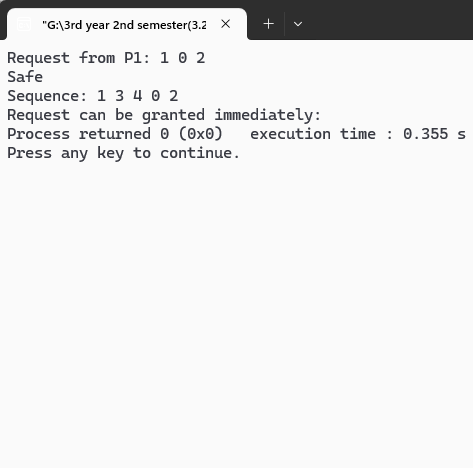
cout << "\nRequest can be granted immediately:";

else

cout << "\nRequest cannot be granted safely:";

return 0;

}



**Task 04: Again, a request for (3,3,0) by P4 happened. Now, can the request be granted immediately?**

#include<iostream>

using namespace std;

const int p = 5;

const int r = 3;

void needcalc(int need[p][r], int maxm[p][r], int allot[p][r]) {

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

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

need[i][j] = maxm[i][j] - allot[i][j];

}

bool safe(int avail[r], int maxm[p][r], int allot[p][r]) {

int need[p][r], work[r], seq[p], cnt = 0;

bool done[p] = {0};

needcalc(need, maxm, allot);

for (int i = 0; i < r; i++) work[i] = avail[i];

while (cnt < p) {

bool ok = 0;

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

if (!done[i]) {

int j;

for (j = 0; j < r; j++)

if (need[i][j] > work[j]) break;

if (j == r) {

for (int k = 0; k < r; k++)

work[k] += allot[i][k];

seq[cnt++] = i;

done[i] = 1;

ok = 1;

}

}

}

if (!ok) {

cout << "Not Safe";

return 0;

}

}

cout << "Safe\nSequence: ";

for (int i = 0; i < p; i++) cout << seq[i] << " ";

return 1;

}

int main() {

int avail[r] = {3, 3, 2};

int maxm[p][r] = {

{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}

};

int allot[p][r] = {

{0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2}

};

int req[r] = {3, 3, 0}; // Request by P4 (process index 4)

int need[p][r];

needcalc(need, maxm, allot);

cout << "Request from P4: ";

for (int i = 0; i < r; i++) cout << req[i] << " ";

cout << endl;

bool canGrant = true;

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

if (req[i] > need[4][i] || req[i] > avail[i]) {

canGrant = false;

break;

}

}

if (!canGrant) {

cout << "Cannot be granted immediately.";

return 0;

}

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

avail[i] -= req[i];

allot[4][i] += req[i];

need[4][i] -= req[i];

}

if (safe(avail, maxm, allot))

cout << "\nRequest can be granted immediately.\n";

else

cout << "\nRequest cannot be granted safely.\n";

return 0;

}

