**РУСЕНСКИ УНИВЕРСИТЕТ „АНГЕЛ КЪНЧЕВ“**

**КУРСОВА ЗАДАЧА**

ПО

ОПЕРАЦИОННИ СИСТЕМИ

Студент:

Фак. Номер:

Група:

Специалност:

Дата: Проверил:

**Съдържание**

[1. Задание 3](#_Toc438557451)

[2. Описание на използваните структури данни и алгоритми 3](#_Toc438557452)

[3. Код на програмата (листинг) 4](#_Toc438557453)

[4. Демо 10](#_Toc438557454)

# Задание

Да се разработи програмна система, симулираща работата на модула за **избягване на deadlock по алгоритъма на банкера** при следните изходни данни:

* ***Основни необходими структури:***
  + Max [1..P] максимални заявки на процесите за ресурси
  + State [1..P] текущо състояние на системата
* ***Основни обработки:***
  + GetResource(Proc) заявка за нов ресурс от процес Proc;
  + FreeResource(Proc) освобождаване на ресурс от процес;
  + Show показване на състоянието на основните структури

Разработеният проект да има подходящ потребителски интерфейс за обръщение към основните обработки. Обяснителната записка да съдържа заданието, описание на използваните структури данни и алгоритми и програмен листинг.

# Описание на използваните структури данни и алгоритми

Стратегията за избягване на взаимна блокировка гарантира, че макар и принципно да е възможно да настъпи взаимна блокировка, тя не възниква за конкретна група процеси и заявки, изпълнявани в даден момент. Избягването може да се разглежда като забрана за влизане в опасна зона.

Една от стратегиите за избягване на взаимна блокировка е т.н. алгоритъм на банкера, предложен от Дейкстра. Използва се аналогия на проблема с подобен в банките. Клиентите желаят да вземат на заем пари, съответстващи на процесите, а парите от определена валута – на типа на ресурсите. Банката има определен лимит от пари и списък от клиенти с определена кредитна линия. Всеки клиент указва максималната сума. Обикновено капиталът на банкера е по-малък, отколкото сумарните нужди на клиентите, т.к. банкерът предполага, че всички клиенти няма да вземат максималния заем.

При реализацията на алгоритъма са използвани следните:

Функции:

* void input() – функция за въвеждане на входните данни като брой процеси, ресурси и т.н.;
* void print() – функция за извеждане състоянието на основните структури;
* bool checkState() – функция за проверка дали състоянието е безопасно или не;
* bool checkRequest(int processNo, bool getResource) – функция за проверка дали заявката може да бъде изпълнена;
* void getResource(int processNo) – функция за заявяване на нов ресурс от процес processNo;
* void freeResource(int processNo) – функция за освобождаване на ресурс от процес processNo;
* int main() – главна програма;

Основни променливи:

* int maxclaim[MAX\_PROC][ MAX\_RES] – матрица максимални заявки на процесите за ресурси;
* int curr[MAX\_PROC][ MAX\_RES] – матрица разпределени ресурси;
* int need[MAX\_PROC][ MAX\_RES] – матрица нужни (искани) ресурси;
* int maxres[MAX\_RES] – максимален брой ресурси;
* int avl[MAX\_RES] – налични ресурси.

# Код на програмата (листинг)

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#define MAX\_PROC 5

#define MAX\_RES 5

int curr[MAX\_PROC][MAX\_RES], maxclaim[MAX\_PROC][MAX\_RES], need[MAX\_PROC][MAX\_RES], avl[MAX\_RES];

int request[MAX\_RES] = { 0 };

int alloc[MAX\_RES] = { 0 };

int maxres[MAX\_RES], running[MAX\_PROC], safe = 0;

int count = 0, i, j, exec, r, p;

void input() {

do {

printf("\nБрой процеси (max %d): ", MAX\_PROC);

scanf("%d", &p);

} while (p < 1 || p > MAX\_PROC);

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

running[i] = 1;

count++;

}

do {

printf("\nБрой ресурси (max %d): ", MAX\_RES);

scanf("%d", &r);

} while (r < 1 || r > MAX\_RES);

printf("\nРесурси: ");

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

scanf("%d", &maxres[i]);

}

printf("\nРазпределени ресурси:\n");

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

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

scanf("%d", &curr[i][j]);

}

}

printf("\nМаксималният брой заявки на процесите за ресурси:\n");

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

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

scanf("%d", &maxclaim[i][j]);

need[i][j] = maxclaim[i][j] - curr[i][j];

}

}

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

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

alloc[j] += curr[i][j];

}

}

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

avl[i] = maxres[i] - alloc[i];

}

}

void print() {

printf("\nРесурси: ");

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

printf("\t%d", maxres[i]);

}

printf("\nРазпределени ресурси:\n");

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

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

printf("\t%d", curr[i][j]);

}

printf("\n");

}

printf("\nМаксималният брой заявки на процесите за ресурси:\n");

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

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

printf("\t%d", maxclaim[i][j]);

}

printf("\n");

}

printf("\nНеобходимият брой заявки на процесите за ресурси:\n");

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

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

printf("\t%d", need[i][j]);

}

printf("\n");

}

printf("\nНалични ресурси:");

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

printf("\t%d", avl[i]);

}

printf("\n");

}

bool checkState() {

int currAvl[MAX\_RES];

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

currAvl[i] = avl[i];

}

while (count != 0) {

safe = 0;

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

if (running[i]) {

exec = 1;

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

if (need[i][j] - curr[i][j] > currAvl[j]) {

exec = 0;

break;

}

}

if (exec) {

printf("\nПроцес%d се изпълнява.", i + 1);

running[i] = 0;

count--;

safe = 1;

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

currAvl[j] += curr[i][j];

}

break;

}

}

}

if (!safe) {

printf("\nПроцесите не са в безопасно състояние.\n");

return false;

}

else {

printf("\nПроцесът е в безопасно състояние");

printf("\nНалични ресурси:");

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

printf("\t%d", currAvl[i]);

}

printf("\n");

}

}

return true;

}

bool checkRequest(int processNo, bool getResource) {

if (getResource) {

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

if (avl[i] < request[i]) {

printf("\nНяма достатъчно ресурси!\n");

return false;

}

}

}

else {

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

if (curr[processNo - 1][i] < request[i]) {

printf("\nПроцесът не разполага с толкова ресурси!\n");

return false;

}

}

}

return true;

}

void getResource(int processNo) {

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

avl[i] -= request[i];

curr[processNo - 1][i] += request[i];

need[processNo - 1][i] -= request[i];

}

count = 0;

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

running[i] = 1;

count++;

}

}

void freeResource(int processNo) {

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

avl[i] += request[i];

curr[processNo - 1][i] -= request[i];

need[processNo - 1][i] += request[i];

}

count = 0;

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

running[i] = 1;

count++;

}

}

int main()

{

setlocale(LC\_ALL, "bgr");

int currentProcessNo;

int choice;

input();

checkState();

do {

printf("%30sМЕНЮ\n", "");

printf("%15s1. Показване на състоянието на основните структури\n", "");

printf("%15s2. Заявяване на ресурс\n", "");

printf("%15s3. Освобождаване на ресурс\n", "");

printf("%18sПосочете операция или 0 за край: ", "");

scanf("%d", &choice); getchar();

system("cls");

switch (choice) {

case 1:

print();

break;

case 2:

printf("\nВъведете номера на процеса (1..%d): ", p);

scanf("%d", &currentProcessNo);

if (currentProcessNo < 1 || currentProcessNo > p) {

printf("\nНевалиден номер на процес!\n");

break;

}

printf("\nВъведете ресурсите: ");

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

scanf("%d", &request[i]);

}

if (checkRequest(currentProcessNo, true)) {

getResource(currentProcessNo);

if (!checkState()) {

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

avl[i] += request[i];

curr[currentProcessNo - 1][i] -= request[i];

need[currentProcessNo - 1][i] += request[i];

}

}

}

break;

case 3:

printf("\nВъведете номера на процеса (1..%d): ", p);

scanf("%d", &currentProcessNo);

if (currentProcessNo < 1 || currentProcessNo > p) {

printf("\nНевалиден номер на процес!\n");

break;

}

printf("\nВъведете ресурсите: ");

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

scanf("%d", &request[i]);

}

if (checkRequest(currentProcessNo, false)) {

freeResource(currentProcessNo);

if (!checkState()) {

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

avl[i] -= request[i];

curr[currentProcessNo - 1][i] += request[i];

need[currentProcessNo - 1][i] -= request[i];

}

}

}

break;

}

} while (choice != 0);

return 0;

}

# Демо





