Sisteme de Operare

**Nicoleta Radu – Tema 4**

# Exercitiul 1

DWORD WINAPI factorial(LPVOID n)

{

int copyN = (int)n;

int tempCopy{0};

//char temp[100];

if (copyN == 1 || copyN == 0)

{

return 1;

}

else

{

printf("%d\n", copyN);

tempCopy = copyN - 1;

return copyN \* factorial((LPVOID)tempCopy);

}

}

// nu am inteles cerinta legata de folosirea a mai multor threaduri pentru calculul recursiv al functiei factorial.

// Am efectuat o testare asupra functiei „factorial” iar functia este totusi corecta

// DWORD dExitCode[1];

int userInput{ 0 }, rezultat{0};

char buffer[100];

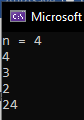
printf("n = "); scanf\_s("%d", &userInput);

dExitCode[0] = factorial((LPVOID)userInput);

sprintf\_s(buffer, "%lu", (unsigned int)dExitCode[0]);

rezultat = atoi(buffer);

printf("%d\n", rezultat);



# Exercitiul 2

#undef UNICODE

#include <windows.h>

#include "resource.h"

#include <stdio.h>

#define \_WIN32\_WINNT \_WIN32\_WINNT\_WINXP

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

static BOOL CALLBACK DlgProc(HWND hDlg, UINT iMsg,

WPARAM wParam, LPARAM lParam);

BOOL dlgActive = FALSE; //variabila pentru caseta de dialog

HWND hwndMain; //variabila handle a ferestrei principale

DWORD WINAPI ThreadProc(LPVOID lpParam);

#define MAX\_THREADS 3

typedef struct

{

int Start, Final;

int idTh;

}info;

info infoTh[3];

HANDLE hThreads[3];

DWORD idT[3];

DWORD dwExitCode[3];

DWORD waiter;

LPDWORD ExitCodeTest[3];

DWORD WINAPI ThreadFunc(LPVOID n)

{

int suma=0;

int id = int(n); // conversie id thread

for (size\_t i = infoTh[id].Start; i <= infoTh[id].Final; i++)

{

suma = suma + i;

}

return (DWORD)suma;

}

LRESULT APIENTRY WndProc(HWND, UINT, WPARAM, LPARAM);

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,

PSTR szCmdLine, int iCmdShow)

{

static char szAppName[] = "Dialog";

HWND hwnd;

MSG msg;

WNDCLASSEX wndclass;

wndclass.cbSize = sizeof(wndclass);

wndclass.style = CS\_HREDRAW | CS\_VREDRAW; //optiuni pentru stilul clasei

wndclass.lpfnWndProc = WndProc;

wndclass.cbClsExtra = 0;

wndclass.cbWndExtra = 0;

wndclass.hInstance = hInstance;

wndclass.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

// incarca pictograma fisierelor de tip aplicatie

wndclass.hCursor = LoadCursor(NULL, IDC\_ARROW); // incarca cursorul sageata

wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);

// fondul ferstrei de culoare alba

wndclass.lpszMenuName = NULL;

wndclass.lpszClassName = szAppName;

wndclass.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

RegisterClassEx(&wndclass);

hwnd = CreateWindow(szAppName, //numele clasei inregistrat cu RegisterClass

"Un program simplu", // text pentru bara de titlu a ferestrei

WS\_OVERLAPPEDWINDOW, // stilul ferestrei

CW\_USEDEFAULT, // pozitia orizontala implicit�

CW\_USEDEFAULT, // pozitia verticala implicita

CW\_USEDEFAULT, // latimea implicita

CW\_USEDEFAULT, // inaltimea implicita

NULL, // handle-ul ferestrei parinte

NULL, // handle-ul meniului ferestrei

hInstance, // proprietara ferestrei

NULL);

SetWindowPos(hwnd, HWND\_BOTTOM, 0, 0, 0, 0, SWP\_NOACTIVATE);

// schimba dimensiunea, pozitia si ordinea z a ferestrei copil, a ferestrei pop-up

ShowWindow(hwnd, SW\_HIDE);

UpdateWindow(hwnd);

hwndMain = hwnd;

while (GetMessage(&msg, NULL, 0, 0))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)

{

static HINSTANCE hInstance;

switch (iMsg)

{

case WM\_CREATE: // operatiile ce se executa la crearea ferestrei

// se creaza caseta de dialog

hInstance = ((LPCREATESTRUCT)lParam)->hInstance;

if (!dlgActive) {

DialogBox(hInstance, MAKEINTRESOURCE(IDD\_DIALOG),

hwnd, (DLGPROC)DlgProc);

PostMessage(hwnd, WM\_CLOSE, 0, 0);

// insereaza un nou mesaj nou in coada de asteptare

}

return 0;

case WM\_DESTROY:

PostQuitMessage(0); // insereaza un mesaj de incheiere

return 0;

}

return DefWindowProc(hwnd, iMsg, wParam, lParam);

}

template <typename T>

void Swap(T& a, T &b )

{

T aux = a;

a = b;

b = aux;

}

static BOOL CALLBACK DlgProc(HWND hDlg, UINT iMsg, WPARAM wParam, LPARAM lParam)

{

char text[100], buffer[100], suma\_thread\_1[100], suma\_thread\_2[100], suma\_thread\_3[100];

bool isReadyForExecution = true;

LPSECURITY\_ATTRIBUTES THREAD\_ALL\_ACCESS;

switch (iMsg) {

case WM\_INITDIALOG:

// setari initiale

return TRUE;

case WM\_CLOSE:

dlgActive = FALSE;

EndDialog(hDlg, 0);

return TRUE;

case WM\_COMMAND:

switch (LOWORD(wParam)) {

case IDC\_CALCULATE:

// salvarea datelor in array

infoTh[0].idTh = 0;

GetDlgItemText(hDlg, IDC\_EDIT\_THREAD\_1\_1, text, 100);

infoTh[0].Start = atoi(text);

GetDlgItemText(hDlg, IDC\_EDIT\_THREAD\_1\_2, text, 100);

infoTh[0].Final = atoi(text);

infoTh[1].idTh = 1;

GetDlgItemText(hDlg, IDC\_EDIT\_THREAD\_2\_1, text, 100);

infoTh[1].Start = atoi(text);

GetDlgItemText(hDlg, IDC\_EDIT\_THREAD\_2\_2, text, 100);

infoTh[1].Final = atoi(text);

infoTh[2].idTh = 2;

GetDlgItemText(hDlg, IDC\_EDIT\_THREAD\_3\_1, text, 100);

infoTh[2].Start = atoi(text);

GetDlgItemText(hDlg, IDC\_EDIT\_THREAD\_3\_2, text, 100);

infoTh[2].Final = atoi(text);

// verificare interval corect [0,10.000]

if (infoTh[0].Start < 0 || infoTh[0].Final > 10000)

{

MessageBox(hDlg, "Eroare!", "Interval Gresit (Thread 1)!", MB\_OK);

isReadyForExecution = false;

}

if (infoTh[1].Start < 0 || infoTh[1].Final > 10000)

{

MessageBox(hDlg, "Eroare!", "Interval Gresit (Thread 1)!", MB\_OK);

isReadyForExecution = false;

}

if (infoTh[2].Start < 0 || infoTh[2].Final > 10000)

{

MessageBox(hDlg, "Eroare!", "Interval Gresit (Thread 1)!", MB\_OK);

isReadyForExecution = false;

}

// caz in care variabila "final" este mai mica decat cea "start"

// vom inversa cele doua valori pentru a minimiza erorile in program

if (infoTh[0].Start > infoTh[0].Final)

{

Swap(infoTh[0].Start, infoTh[0].Final);

}

if (infoTh[1].Start > infoTh[1].Final)

{

Swap(infoTh[1].Start, infoTh[1].Final);

}

if (infoTh[2].Start > infoTh[2].Final)

{

Swap(infoTh[2].Start, infoTh[2].Final);

}

if (isReadyForExecution)

{

//creare thread

hThreads[0] = CreateThread(NULL, 0, ThreadFunc, (LPVOID)0, NULL, &idT[0]);

hThreads[1] = CreateThread(NULL, 0, ThreadFunc, (LPVOID)1, NULL, &idT[1]);

hThreads[2] = CreateThread(NULL, 0, ThreadFunc, (LPVOID)2, NULL, &idT[2]);

// asteptare

waiter = WaitForMultipleObjects(MAX\_THREADS, hThreads,

TRUE, INFINITE);

// get exit code

for (size\_t i = 0; i < MAX\_THREADS; i++)

{

GetExitCodeThread(hThreads[i], (LPDWORD)dwExitCode[i]);

}

//inchidere

for (int i = 0; i < MAX\_THREADS; i++) {

CloseHandle(hThreads[i]);

}

// afisare rezultat final

sprintf\_s(buffer, "%lu", (unsigned int)dwExitCode[0] + (unsigned int)dwExitCode[1] + (unsigned int)dwExitCode[2]);

sprintf\_s(suma\_thread\_1, "%lu", (unsigned int)dwExitCode[0]);

sprintf\_s(suma\_thread\_2, "%lu", (unsigned int)dwExitCode[1]);

sprintf\_s(suma\_thread\_3, "%lu", (unsigned int)dwExitCode[2]);

SetDlgItemText(hDlg, IDC\_EDIT\_THREAD\_1\_SUMA, suma\_thread\_1);

SetDlgItemText(hDlg, IDC\_EDIT\_THREAD\_2\_SUMA, suma\_thread\_2);

SetDlgItemText(hDlg, IDC\_EDIT\_THREAD\_3\_SUMA, suma\_thread\_3);

SetDlgItemText(hDlg, IDC\_EDIT\_SUMA\_TOTALA, buffer);

}

else

{

// programul nu poate fi executat

MessageBox(hDlg, "Eroare!", "Programul nu poate executa!", MB\_OK);

return TRUE;

}

return TRUE;

case IDC\_CANCEL: // operatii ce se executa la actiunea apasarii but. CANCEL

MessageBox(hDlg, "Abandon!", "Ati selectat CANCEL", MB\_OK);

dlgActive = FALSE;

EndDialog(hDlg, 0);

return TRUE;

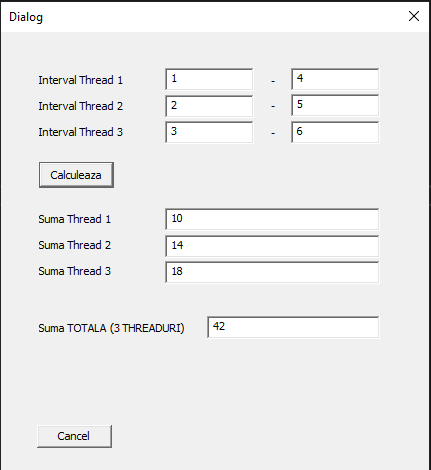
}

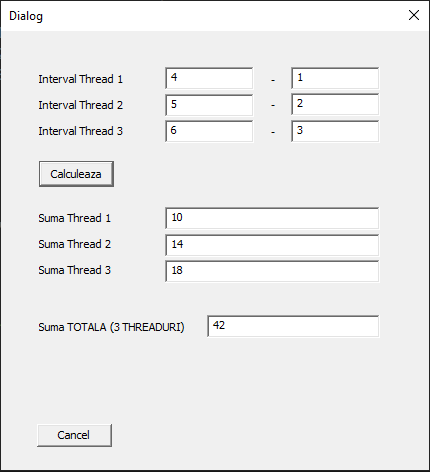
break;

}

return FALSE;

}



 // inversarea valorilor pentru minimalizarea erorilor in program

