

Laborator 6

Gavan Eduard

CR3.2A

Acet document descrie experienta mea in realizarea laboratorului 6 la disciplina Sisteme de Operare.

Tema a presupus implementarea comunicarii intre doua procese folosind memorie partajata si un mutex pentru sincronizare. Programul ruleaza pe Windows si utilizeaza API-uri precum **CreateFileMappingA**, **MapViewOfFile** si **CreateMutexA** pentru a crea o zona de memorie comună si a controla accesul la aceasta.

Primul proces creeaza memoria si mutex-ul, apoi lanseaza automat al doilea proces folosind **CreateProcessA**. Ambele procese executa aceeasi logica pe zona de memorie partajata, incrementand o valoare comună intr-un mod sincronizat.

Codul final al programului:

```
// Primul proces creeaza memoria+mutexul si lanseaza automat al doilea proces.  
// Al doilea proces se conecteaza si numara impreuna cu primul.  
#include <windows.h>  
#include <iostream>  
#include <string>  
#include <cstdlib>  
#include <ctime>  
#define MAX_N 1000  
using namespace std;  
struct SharedData {  
    int value;  
};  
static void die(const string& msg) {  
    cerr << msg << " (error=" << GetLastError() << ")\n";  
    ExitProcess(1);  
}  
static bool spawn_second_process() {  
    char exePath[MAX_PATH];  
    GetModuleFileNameA(NULL, exePath, MAX_PATH);  
    string cmd = string("") + exePath + "\" --child";  
    // buffer mutabil pentru CreateProcessA  
    char cmdMutable[512];  
    strcpy_s(cmdMutable, cmd.c_str());  
    STARTUPINFOA si{};  
    si.cb = sizeof(si);  
    PROCESS_INFORMATION pi{};  
    BOOL ok = CreateProcessA(  
        NULL,  
        cmdMutable,  
        NULL, NULL,
```

```

    FALSE,
    CREATE_NEW_CONSOLE,
    NULL, NULL,
    &si, &pi
);
if (!ok)
    return false;
CloseHandle(pi.hThread);
CloseHandle(pi.hProcess);
return true;
}

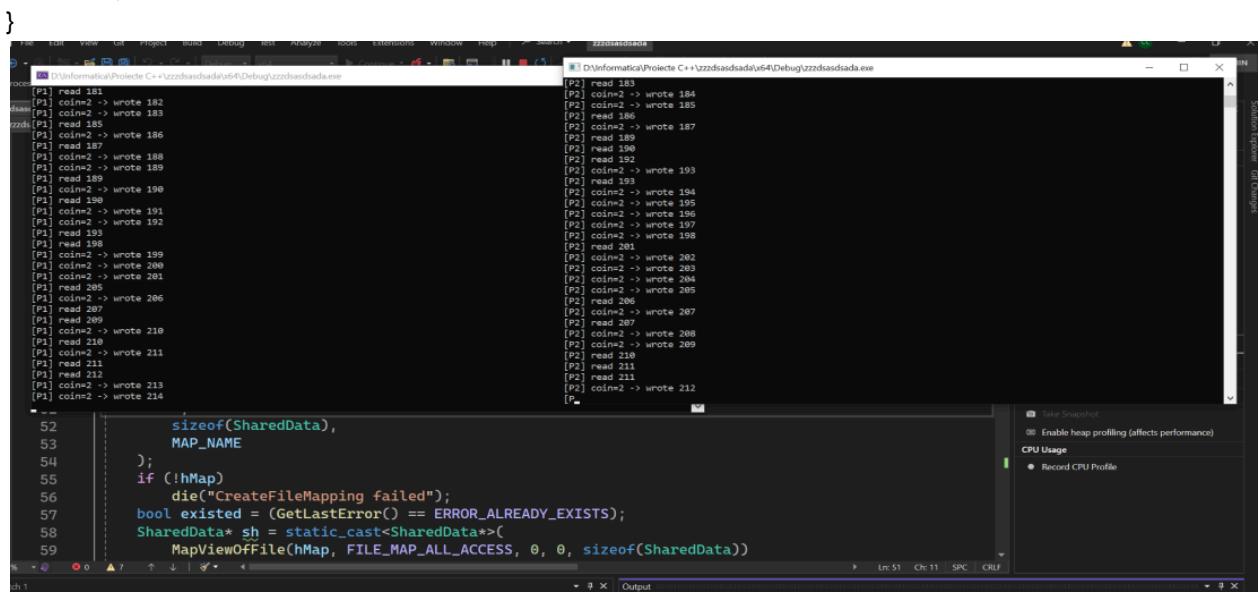
int main(int argc, char** argv) {
    bool isChild = (argc >= 2 && std::string(argv[1]) == "--child");
    const char* MAP_NAME = "coin_counter_map_cpp_simple"; // sau putem Global\coin_counter_map_cpp_simple
-> insa merge doar daca ii dam run ca administrator
    const char* MUTEX_NAME = "coin_counter_mutex_cpp_simple"; // sau Global\coin_counter_mutex_cpp_simple
-> (run as administrator ca sa mearga)
    HANDLE hMap = CreateFileMappingA(
        INVALID_HANDLE_VALUE,
        NULL,
        PAGE_READWRITE,
        0,
        sizeof(SharedData),
        MAP_NAME
    );
    if (!hMap)
        die("CreateFileMapping failed");
    bool existed = (GetLastError() == ERROR_ALREADY_EXISTS);
    SharedData* sh = static_cast<SharedData*>(
        MapViewOfFile(hMap, FILE_MAP_ALL_ACCESS, 0, 0, sizeof(SharedData))
    );
    if (!sh)
        die("MapViewOfFile failed");
    HANDLE hMutex = CreateMutexA(NULL, FALSE, MUTEX_NAME);
    if (!hMutex)
        die("CreateMutex failed");
    // daca NU exista mapping-ul, inseamna ca suntem primul proces
    int id = existed ? 2 : 1;
    // initializare doar in primul proces
    if (!existed) {
        WaitForSingleObject(hMutex, INFINITE);
        sh->value = 0;
        ReleaseMutex(hMutex);
        // primul proces porneste automat al doilea
        if (!spawn_second_process()) {
            cerr << "Warning: nu am putut porni al doilea proces automat.\n";
            cerr << "Poti rula manual acelasi exe inca o data.\n";
        }
    }
    // seed random diferit pe proces
    unsigned seed = (unsigned)time(NULL)
        ^ (unsigned)GetCurrentProcessId()
        ^ (unsigned)(id * 12345);
}

```

```

std::srand(seed);
while (true) {
    DWORD w = WaitForSingleObject(hMutex, INFINITE);
    if (w != WAIT_OBJECT_0) die("WaitForSingleObject failed");
    int elem_curent = sh->value;
    if (elem_curent >= MAX_N) {
        ReleaseMutex(hMutex);
        break;
    }
    cout << "[P" << id << "] read " << elem_curent << "\n";
    while (elem_curent < MAX_N) {
        int coin = (rand() % 2) + 1; // 1 sau 2
        if (coin != 2)
            break;
        elem_curent++;
        sh->value = elem_curent;
        cout << "[P" << id << "] coin=2 -> wrote " << elem_curent << "\n";
    }
    ReleaseMutex(hMutex);
    Sleep(50);
}
UnmapViewOfFile(sh);
CloseHandle(hMutex);
CloseHandle(hMap);
cout << "[P" << id << "] exit\n";
return 0;
}

```



Aceasta captura arata ambele procese in executie simultana. Se observa cum P1 si P2 citesc si scriu alternativ aceeasi valoare in memoria partajata, folosind mutex-ul pentru sincronizare.

The screenshot shows the Microsoft Visual Studio Debug Console window. The left pane displays a process dump for process 13612, showing multiple reads and writes between two processes, P1 and P2. Process P1 is reading from 974 to 999 and writing to 978 to 999. Process P2 is reading from 979 to 999 and writing to 978 to 999. The right pane shows the source code for a file named main.cpp, which includes code for creating shared memory and mapping it. The code uses shared pointers and shared data structures.

```
Microsoft Visual Studio Debug Console
Solution [P1] read 974
[P1] coin2 -> wrote 975
[P1] read 975
[P1] coin2 -> wrote 976
[P1] read 976
[P1] coin2 -> wrote 977
[P1] read 977
[P1] coin2 -> wrote 978
[P1] read 981
[P1] read 983
[P1] read 987
[P1] read 989
[P1] read 990
[P1] coin2 -> wrote 990
[P1] read 998
[P1] coin2 -> wrote 991
[P1] read 991
[P1] coin2 -> wrote 992
[P1] read 992
[P1] read 994
[P1] coin2 -> wrote 995
[P1] read 995
[P1] coin2 -> wrote 997
[P1] coin2 -> wrote 998
[P1] coin2 -> wrote 999
[P1] exit

D:\Informatica\Proiecte C++\zzzdsadsada\x64\Debug\zzzdsadsada.exe (process 13612) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
53     MAP_NAME
54 );
55     if (!hMap)
56         die("CreateFileMapping failed");
57     bool existed = (GetLastError() == ERROR_ALREADY_EXISTS);
58     SharedData* sh = static_cast<SharedData*>(
59         MapViewOfFile(hMap, FILE_MAP_ALL_ACCESS, 0, 0, sizeof(SharedData))
60     );
61     if (!sh)
```

In aceasta captura putem vedea finalizarea procesului P1 pana la valoarea 999, moment in care programul se opreste. Incrementarea sincronizata pana la 1000 confirma functionarea corecta.

Github: https://github.com/edigolan/Cerinta6_SO