WYŻSZA SZKOŁA HANDLOWA W RADOMIU

RADOM ACADEMY OF ECONOMICS

Wyższa Szkoła Handlowa w Radomiu

Programowanie równoległe
Laboratorium 2

Radom 2020/2021

1. Cel zadania

Celem laboratorium jest zapoznanie się z działaniem wątków w systemach Unix-like. Jako dodatkowy efekt będzie porównanie rozwiązań Unix-like z rozwiązaniami uniwersalnymi, jak wątki C++ czy Qthreads (wymaga dodatkowych bibliotek).

2. Potrzebne narzędzia.

- kompilator C++ minimum standardu 11 (optymalnie 14)
- system Linux lub Windows z subsystemem Linux i/lub środowisko Cygwin/MSys
- minimum dwuwątkowy procesor (optymalnie minimum dwurdzeniowy)

Opcjonalnie – TYLKO JEŻELI komputer posiada więcej niż dwa rdzenie:
- dowolne system wirtualizacji maszyn (VirtualBox, VMware, Hyper-V)
- dowolny system operacyjny wyposażony w kompilator z wymagań podstawowych
- biblioteki Qt

3. Informacje wstępne

W programowaniu nie ma ważniejszego kierunku niż tworzenie aplikacji wielowątkowych. Użytkownicy wymagają, by program odpowiadał na ich operacje i żądania, jednocześnie w jak najszybszym czasie ma kompletować dla nich odpowiednie dane.

Dzięki współbieżności/zrównolegleniu odpowiednich operacji można wykonywać bardziej rzeczywiste odwzorowania środowisk naturalnych np. innych planet, otwartych światów w grach komputerowych czy po prostu w programach z interfejsem graficznym, w których np. wczytywanie plików i/lub przetwarzanie danych (wysyłanie ich przez sieć, do lub z plików itp.)

Szczególnym przypadkiem są urządzenia, od których wymagamy by obsługiwały wiele aplikacji i wiele zapytań jednocześnie. Przykładem takich urządzeń są serwery przetwarzające jednoczenie wiele zapytań od klientów (np. żądania stron WWW, żądania pozyskania plików z ftp) przy jednoczesnej dostępności reszty swoich zasobów. Na tym polu nadal najlepiej sprawdzają się systemy Unix-like. Dzieje się tak za sprawą ich specyficznej budowy oraz wbudowanej w jądro obsłudze współbieżności zadań.

4. Przebieg.

Ćwiczenie należy rozpocząć od pozyskania i zainstalowania systemu Linux bądź innego systemu Unix-like – np. FreeBSD, OpenBSD czy macOS (jeżeli posiadamy sprzęt Apple to ten system jest już zainstalowany). Jeżeli na co dzień korzystamy z systemu Windows ćwiczenie można wykonać np. uruchamiając podsystem systemu Linux (jak tego dokonać można przeczytać tutaj – https://www.windowscentral.com/install-windows-subsystem-linux-windows-10).

Jeżeli wybierzemy Cygwin lub Msys – odpowiednio instalujemy oraz konfigurujemy Msys https://genome.sph.umich.edu/wiki/Installing_MinGW_%26_MSYS_on_Windows lub Cygwin https://cygwin.com/install.html.

Sam przebieg laboratorium należy wykonać w identyczny sposób jaki miał miejsce w laboratorium 1. Należy odpowiednio przebudować przykładowy kod tak, by działał przy użyciu biblioteki POSIX Threads oraz poprzez funkcję fork() (przykład fork - https://www.geeksforgeeks.org/fork-system-call/)

Na koniec należy porównać wyniki pozyskane z testu w laboratorium 1 z obecnym wykonaniem. Czy działanie biblioteki pthread oraz fork jest szybsze czy wolniejsze? Czy implementacja jest mniej lub bardziej skomplikowana? Czy możliwe jest rozproszenie kodu tak jak miało to miejsce w przypadku OpenMPI?

5. Zakończenie/uwagi do sprawozdania

Kod z dodatku (dla ułatwienia dodatek został zachowany) trzeba przerobić pod funkcje POSIX Threads. Dodatkowo kod ten należy przemienić dla funkcji fork(). Należy odpowiednio zanotować czy działanie funkcji fork() jako zrównoleglenia operacji ma sens, czy zastosowanie jej może być przydatne przy innych projektach.

Jeżeli będzie taka możliwość dobrze byłoby wypróbować działanie wielowątkowości w bibliotekach międzyplatformowych, takich jak Qt (instalacja i konfiguracja – https://doc.qt.io/qt-5/gettingstarted.html). Czy rozwiązanie tego typu jest wydajniejsze niż pozostałe implementacje? Ewentualnie czy braki w efektywności nie są nadrabiane znacznie wygodniejszą formą programowania i rozpropagowywania naszych programów?

Wykonanie wszystkich podpunktów, włączenie z bibliotekami Qt oraz poprawne wykonanie laboratorium 1 daje gwarancję otrzymania oceny dostateczny z egzaminu w pierwszym możliwym terminie (innymi słowy – student nie musi pisać egzaminu jeżeli zadowala go ocena dostateczny).

DODATEK

```
#include <iostream>
#include <chrono>

using namespace std;

int main()
{
    const unsigned int m=10;
    const unsigned int n=10;

    srand(static_cast<unsigned int>(static_cast< std::chrono::duration<double>
>(std::chrono::high_resolution_clock::now().time_since_epoch()).count()));

    double **matrixa;
    double **matrixb;
    double **matrixc;

    matrixa = new double*[m];
    matrixb = new double*[m];
    matrixc = new double*[m];

    unsigned int max = static_cast<unsigned int>(1u<<31);

    for (unsigned int i=0;i<m;i++)
        matrixa[i]=new double[n];

    for (unsigned int i=0;i<m;i++)
        matrixb[i]=new double[n];

    for (unsigned int i=0;i<m;i++)
        matrixc[i]=new double[n];

    for (unsigned int i=0;i<m;i++)
        for (unsigned int j=0;j<n;j++)
            matrixa[i]
[j]=static_cast<double>(static_cast<double>(rand())/max*10);

    for (unsigned int i=0;i<m;i++)
```

```
        for (unsigned int j=0;j<n;j++)
            matrixb[i]
[j]=static_cast<double>(static_cast<double>(rand())/max*10);

    auto start=std::chrono::high_resolution_clock::now();

    for (unsigned int i=0;i<m;i++)
        for (unsigned int j=0;j<n;j++)
           for (unsigned int k=0;k<m;k++)
               for (unsigned int l=0;l<m;l++)
                   matrixc[i][j]+=matrixa[k][l]*matrixb[l][k];

    auto stop=std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> time_diff=stop-start;

    cout << "Czas wykonania programu " << time_diff.count()<< " sekund." <<
endl;

    for (unsigned int i=0;i<m;i++)
        delete [] matrixa[i];

    for (unsigned int i=0;i<m;i++)
        delete [] matrixb[i];

    for (unsigned int i=0;i<m;i++)
        delete [] matrixc[i];

    delete [] matrixa;
    delete [] matrixb;
    delete [] matrixc;

    return 0;
}
```