

Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Інститут комп'ютерних наук та інформаційних технологій
Кафедра Систем Штучного Інтелекту



Звіт
до лабораторної роботи № 9
з дисципліни
Операційні системи
на тему:
“Виконання задачі в декількох потоках в ОС Linux”

Виконав: студент КН-217

Ратушняк Денис

Прийняв: доцент каф. СШІ

Кривенчук Ю. П.

Львів – 2022

Мета роботи: Навчитись реалізовувати розпаралелювання алгоритмів за допомогою багатопоточності в ОС Linux з використанням пакету функцій pthread.

Завдання

1. [Макс. Складність 2] Модифікувати код лабораторної роботи №3 для виконання під OS Linux.
2. [Складність відповідно до лаб. №3] Модифікувати код лабораторної роботи №2, щоб він став крос-платформенним - міг бути скомпільованим як під ОС Windows, так і під OS Linux.

Код програми

```
#include <bits/stdc++.h>

#include <pthread.h>

using namespace std;
typedef long long ll;
typedef long double ld;
typedef vector< vector<ll> > matrix;

///HERE YOU CAN CHANGE INPUT
const int TOTAL_THREADS = 10;
string version = "10000_10000_10000";

ll type_of_task = 2;
ll low_priority_thread = 3;
ll high_priority_thread = 7;

///HERE YOU CAN CHANGE INPUT

typedef struct MyData
{
    int startx, starty, step, type;
    int threadNum;
} MYDATA, *PMYDATA;

MYDATA pDataArray[TOTAL_THREADS];

pthread_t dwThreadIdArray[TOTAL_THREADS];

matrix A;
ll n,m,k,max_number;
ll add;
vector < pair<ll,ll> > diag[TOTAL_THREADS];

vector< pair< pair<ll,ll>, pair<ll,ll> > > points_for_diag;

void solve(int sx, int sy, int step, int type, int num)
{
    if(type == 1)
    {
        int i = sx;
        int j = sy;
        while(step--){
            diag[num][i+j].first += A[i][j];
            diag[num][i+j].second ++;
        }
    }
}
```

```

        diag[num][i-j+add].first += A[i][j];
        diag[num][i-j+add].second ++;
        //cout << i << " " << j << endl;
        //cout << num << " " << step << endl;
        j++;
        if(j == m){
            j = 0;
            i ++;
            if(i == n) break;
        }
    }
}
else
{
    int now = sx * n + sy;

    while(now < n * m){
        int i = now / m;
        int j = now % m;

        diag[num][i+j].first += A[i][j];
        diag[num][i+j].second ++;

        diag[num][i-j+add].first += A[i][j];
        diag[num][i-j+add].second ++;
        //cout << num << " " << now << " " << step << endl;
        now += step;
    }
}

}
ld threadTime[TOTAL_THREADS];

void* MyThreadFunction(void *lpParam){
    PMYDATA pDataVar;

    pDataVar = (PMYDATA)lpParam;
    MYDATA DataVar = *pDataVar;

    std::chrono::time_point<std::chrono::system_clock> start, end;
    start = std::chrono::system_clock::now();

    solve(DataVar.startx, DataVar.starty, DataVar.step, DataVar.type, DataVar.threadNum);

    end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end - start;
    ld currtime = elapsed_seconds.count();

    threadTime[pDataVar->threadNum] = currtime;
    return NULL;
}

void print(matrix &a)
{
    for(int i = 0; i < a.size(); ++ i)
    {
        for(int j = 0; j < a[i].size(); ++ j)
        {
            cout << a[i][j] << " ";
        }
        cout << "\n";
    }
}

```

```

        cout << "\n";
    }

void print(matrix &a, ofstream &output)
{
    for(int i = 0; i < a.size(); ++ i)
    {
        for(int j = 0; j < a[i].size(); ++ j)
        {
            output << a[i][j] << " ";
        }
        output << "\n";
    }
    output << "\n";
}

void read(matrix &a, ifstream &input)
{
    for(int i = 0; i < a.size(); ++ i)
    {
        for(int j = 0; j < a[i].size(); ++ j)
        {
            input >> a[i][j];
        }
    }
}

void calc_points()
{
    points_for_diag.resize(2 * m + 2 * n - 2);
    ll x1,y1,x2,y2;
    x1 = y1 = x2 = y2 = 0;

    for(int i = 0; i < (2 * m + 2 * n - 2)/2; ++i)
    {
        points_for_diag[i] = {{x1, y1}, {x2, y2}};
        if(x1 == n - 1) y1++;
        else x1++;

        if(y2 == m - 1) x2++;
        else y2++;
    }

    x1 = x2 = 0;
    y1 = y2 = m-1;

    for(int i = (2 * m + 2 * n - 2)/2; i < (2 * m + 2 * n - 2); ++i)
    {
        points_for_diag[i] = {{x1, y1}, {x2, y2}};
        if(y1 == 0) x1++;
        else y1--;

        if(x2 == n - 1) y2--;
        else x2++;
    }
}

int main()
{
    #ifdef _WIN32
    string test_path = "A:\\T\\3_term\\Operating_Systems\\OSLabs\\9lab\\tests\\" + version + "_in.txt";

```

```

#endif

#ifdef __linux__
string test_path = "/home/denisr2007/QT/OSLabs/9lab/tests/" + version + "_in.txt";
#endif

low_priority_thread %= TOTAL_THREADS;
high_priority_thread %= TOTAL_THREADS;
if(low_priority_thread == high_priority_thread) low_priority_thread--;
if(low_priority_thread == -1) low_priority_thread = 1;
if(low_priority_thread == TOTAL_THREADS) low_priority_thread = 0;

ifstream input(test_path);
cout << test_path << endl;
ld t0 = clock();
input >> n >> m >> max_number;
calc_points();
add = n + 2*m - 2;

A.resize(n);
for(int i = 0; i < n; ++i) A[i].resize(m,0);

read(A, input);
//print(A);

ll step1 = (n * m / TOTAL_THREADS);
if(!step1) step1++;

std::chrono::time_point<std::chrono::system_clock> start, end;
start = std::chrono::system_clock::now();

ld t1 = clock();
ll now1 = 0;
ll now2 = 0;

ll needThreads = min(ll(n * m), ll(TOTAL_THREADS));
ll step2 = needThreads;

for(int i = 0; i < needThreads; ++i)
{
    diag[i].resize(2 * n + 2 * m - 2, {0, 0});

    pDataArray[i].threadNum = i;
    pDataArray[i].type = type_of_task;

    if(type_of_task == 1)
    {
        pDataArray[i].startx = now1 / m;
        pDataArray[i].starty = now1 % m;
        pDataArray[i].step = step1;
        now1 += step1;
    }
    else
    {
        pDataArray[i].startx = now2 / m;
        pDataArray[i].starty = now2 % m;
        pDataArray[i].step = step2;
        now2 ++;
    }

    pthread_create(&dwThreadIdArray[i], NULL, &MyThreadFunction, &pDataArray[i]);
}

```

```

    }
    for(int i = 0; i < needThreads; ++i) pthread_join(dwThreadIdArray[i], NULL);

    for(int i = 1; i < needThreads; ++i)
    {
        for(int j = 0; j < diag[0].size(); ++j){
            diag[0][j].first += diag[i][j].first;
            diag[0][j].second += diag[i][j].second;
        }
    }

    ll ans = 0;
    for(int i = 1 ; i < diag[0].size(); ++i)
        if(diag[0][i].first * diag[0][ans].second > diag[0][i].second * diag[0][ans].first) ans = i;
    end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end - start;
    ld dectime = elapsed_seconds.count();

    ld result = (ld)diag[0][ans].first / (ld)diag[0][ans].second;

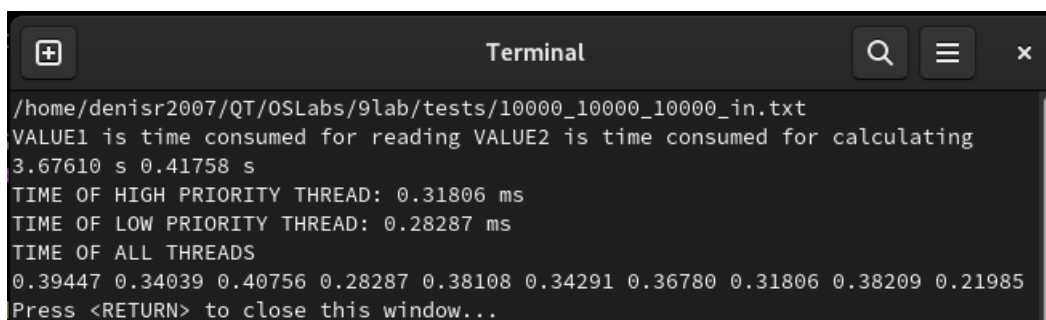
    #ifdef _WIN32
        string result_path = "A:\\T\\3_term\\Operating_Systems\\OSLabs\\9lab\\results_prob_right\\" +
to_string(TOTAL_THREADS);
    #endif // _WIN32

    #ifdef __linux__
        string result_path = "/home/denizr2007/QT/OSLabs/9lab/results_prob_right/" +
to_string(TOTAL_THREADS);
    #endif // __linux__

    result_path += "threads_" + to_string(type_of_task) + "typeOfTask_" + to_string(dectime) + "s_" + version
+ "_out.txt";
    ofstream output(result_path);
    output << "Number of diagonal where average element is the biggest = " << ans << " \n";
    output << fixed << setprecision(5) << "Average number = " << result << "\n";
    output << "2 points of this diagonal\n";
    output << points_for_diag[ans].first.first+1 << " " << points_for_diag[ans].first.second+1 << " " <<
points_for_diag[ans].second.first+1 << " " << points_for_diag[ans].second.second+1 << "\n";
    ld d1 = t1 - t0;
    cout << "VALUE1 is time consumed for reading VALUE2 is time consumed for calculating" << endl;
    cout << fixed << setprecision(5) << d1/CLOCKS_PER_SEC << " s " << dectime << " s" << endl;
    cout << "TIME OF HIGH PRIORITY THREAD: " << threadTime[high_priority_thread] << " ms" << endl;
    cout << "TIME OF LOW PRIORITY THREAD: " << threadTime[low_priority_thread] << " ms" << endl;
    cout << "TIME OF ALL THREADS" << endl;
    for(int i = 0; i < TOTAL_THREADS; ++i) cout << threadTime[i] << " ";
}

```

Результати



```

/home/denizr2007/QT/OSLabs/9lab/tests/10000_10000_10000_in.txt
VALUE1 is time consumed for reading VALUE2 is time consumed for calculating
3.67610 s 0.41758 s
TIME OF HIGH PRIORITY THREAD: 0.31806 ms
TIME OF LOW PRIORITY THREAD: 0.28287 ms
TIME OF ALL THREADS
0.39447 0.34039 0.40756 0.28287 0.38108 0.34291 0.36780 0.31806 0.38209 0.21985
Press <RETURN> to close this window...

```

```
Open ▾ + 10threads_2typeOfTask_0.417584s_10000_10000_10000_o...  
~/QT/OSLabs/9lab/results_prob_right  
Number of diagonal where average element is the biggest = 39996  
Average number = 8933.00000  
2 points of this diagonal  
9999 1 10000 2
```

Висновок: Я закріпив вміння та навички роботи з розпаралелюванням алгоритмів за допомогою багатопоточності в ОС Linux з використанням пакету функцій pthread.