

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



Звіт

до лабораторної роботи № 12

з дисципліни

Операційні системи

на тему:

“ Файли, що відображаються в пам'ять в ОС Linux”

Виконав: студент КН-217

Ратушняк Денис

Прийняв: доцент каф. СШІ

Кривенчук Ю. П.

Мета роботи: Ознайомитися з відображенням файлів в оперативну пам'ять в ОС Linux. Навчитися реалізовувати відображення файлів в оперативну пам'ять.

Завдання

1. [Макс. Складність 2] Модифікувати код лабораторної роботи №6 для виконання під OS Linux.
2. [Складність відповідно до лаб. №6] Модифікувати код лабораторної роботи №6 так, щоб він став крос-платформенним - міг бути скомпільованим як під ОС Windows, так і під OS Linux.

Код програми

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

#ifdef _WIN32
#include <windows.h>
#endif

#include <pthread.h>
#include <semaphore.h>
#ifdef __linux__
#include <fcntl.h>
#include <sys/stat.h>
#include <unistd.h>
#include <cstring>
#include <sys/mman.h>
#endif
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 = 100;
const int MAX_SEM_COUNT = 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;

ld threadTime[TOTAL_THREADS];
pthread_mutex_t mymutex;
sem_t mysemaphore;
ll done_threads;
ll needThreads;

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;
        }
    }
    ll ans = 0;
    for(int i = 1 ; i < diag[num].size(); ++i)
        if(diag[num][i].first * diag[num][ans].second > diag[num][i].second * diag[num][ans].first ||
        diag[num][ans].second == 0) ans = i;

    ld result = (ld)diag[num][ans].first / (ld)diag[num][ans].second;
    pthread_mutex_lock(&mymutex);
    cout << "\033[F";
    cout << "\033[F";
    cout << "\n";
    cout << "Tread " << num << " MaxAverage = " << result << "\n";
    pthread_mutex_unlock(&mymutex);
}

void* MyThreadFunction(void *lpParam){

```

```

sem_wait(&mysemaphore);
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;

pthread_mutex_lock(&mymutex);

done_threads++;
cout << "\033[F";
cout << "\033[F";
cout << fixed << setprecision(3) << 100.0 * double(done_threads) / double(needThreads) << "%\n";
pthread_mutex_unlock(&mymutex);

sem_post(&mysemaphore);
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";
}

#ifdef __linux
unsigned char* pBuf;
#endif

#ifdef _WIN32
const char* pBuf;
#endif

```

```

ll pos;
void read(matrix &a)
{
    for(int i = 0; i < int(a.size()); ++i)
    {
        for(int j = 0; j < a[i].size(); ++j)
        {
            a[i][j] = 0;
            while(!(pBuf[pos]>='0' && pBuf[pos]<='9'))
            {
                pos++;
            }
            while(pBuf[pos]>='0' && pBuf[pos]<='9')
            {
                a[i][j] = a[i][j] * 10 + (int(pBuf[pos]) - 48);
                pos++;
            }
            //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++;
    }
}

struct FileMapping {
    int fd;
    size_t fsize;
    unsigned char* dataPtr;
};

int main()
{

```

```

#ifdef _WIN32
string test_path = "A:\\T\\3_term\\Operating_Systems\\OSLabs\\10lab\\tests\\" + version + "_in.txt";
#endif

#ifdef __linux__
string test_path = "/home/denisr2007/QT/OSLabs/10lab/tests/" + version + "_in.txt";
#endif

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

pthread_mutex_init(&mymutex, NULL);
sem_init(&mysemaphore, 0, MAX_SEM_COUNT);
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;

#ifdef _WIN32
HANDLE hMapFile = CreateFile("B:\\Codeblocks\\Projects\\2131231\\test.txt", GENERIC_READ, 0,
NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
HANDLE hMapFileW = CreateFile("B:\\Codeblocks\\Projects\\2131231\\resulttest.txt",
GENERIC_READ|GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL,
NULL);
HANDLE hMapping = CreateFileMapping(hMapFile, NULL, PAGE_READONLY, 0, 0, NULL);
pBuf = (const char*)MapViewOfFile(hMapping, FILE_MAP_READ, 0, 0, 0);
#endif

#ifdef __linux__
int fd = open("test.txt", O_RDONLY, 0);

struct stat st;

if (fstat(fd, &st) < 0) {
    cout<< "Error" << endl;
    close(fd);
}

size_t fsize = (size_t)st.st_size;
pBuf = (unsigned char*)mmap(nullptr, fsize, PROT_READ, MAP_PRIVATE, fd, 0);

FileMapping* mapping = (FileMapping*)malloc(sizeof(FileMapping));

mapping->fd = fd;
mapping->fsize = fsize;
mapping->dataPtr = pBuf;
#endif

ld t0 = clock();

while(pBuf[pos]>='0' && pBuf[pos]<='9')
{
    n = n * 10 + (int(pBuf[pos]) - 48);
    pos++;
}
pos++;

while(pBuf[pos]>='0' && pBuf[pos]<='9')
{

```

```

        m = m * 10 + (int(pBuf[pos]) - 48);
        pos++;
    }
    pos++;
    while(pBuf[pos]>='0' && pBuf[pos]<='9')
    {
        max_number = max_number * 10 + (int(pBuf[pos]) - 48);
        pos++;
    }
    while(!(pBuf[pos]>='0' && pBuf[pos]<='9'))
    {
        pos++;
    }

//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);
#ifdef _WIN32
    UnmapViewOfFile(pBuf);
    CloseHandle(hMapFile);
#endif
    //print(A);

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

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

    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);
cout << "\n";
pthread_mutex_destroy(&mymutex);
sem_destroy(&mysemaphore);

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 __linux__
    munmap(mapping->dataPtr, mapping->fsize);
    close(mapping->fd);
    free(mapping);
    int fdFile = open("resulttest.txt", O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
    char* file;
    string RES = to_string(result);

    file = (char*)mmap(0, RES.size(), PROT_READ | PROT_WRITE, MAP_SHARED, fdFile,
0);
    lseek(fdFile, 1, SEEK_SET);
    write(fdFile, "", RES.size() - 1);
    lseek(fdFile, 0, SEEK_SET);
    sprintf(file, "%s", RES.c_str());
#endif

#ifdef _WIN32
    string RES = to_string(result);
    HANDLE hMappingW= CreateFileMapping(hMapFileW, NULL, PAGE_READWRITE, 0, sizeof(RES),
NULL);
    unsigned char* pBuf1 = (unsigned char*)MapViewOfFile(hMappingW, FILE_MAP_WRITE, 0, 0, 0);
    for(int i = 0; i < RES.size(); ++ i) pBuf1[i] = RES[i];

    CloseHandle(hMapFile);
    CloseHandle(hMapFileW);
    CloseHandle(hMapping);
    CloseHandle(hMappingW);
#endif
    cout << RES << "\n Time: " << dectime << " s\n";
//    #ifdef __linux__

```



```

//  string result_path = "/home/denisr2007/QT/OSLabs/10lab/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";

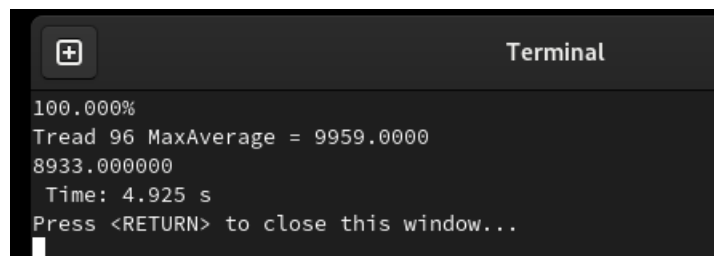
//  ld d1 = t1 - t0;

//  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



```

Terminal
100.000%
Tread 96 MaxAverage = 9959.0000
8933.000000
Time: 4.925 s
Press <RETURN> to close this window...

```



```

Open ▾ + resulttest.txt
~/QT/OSLabs/12lab/build-12lab-Desktop-Debug
8933.000000

```

Висновок: Я закріпив вміння та навички роботи з відображенням файлів в оперативну пам'ять в ОС Linux. Навчився реалізовувати відображення файлів в оперативну пам'ять.