**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ НАЦІОНАЛЬНОМУ УНІВЕРСИТЕТІ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»**

Кафедра систем штучного інтелекту

Лабораторна робота №3

З дисципліни

# «Операційні системи»

**Виконав:**

Студент групи КН-215

Огарь Арсеній Олександрович

**Викладач:**

Пшеничний Олександр Юрійович

Львів - 2022 р.

**Тема.** Виконання задачі в декількох потоках в ОС Windows

**Мета.** Навчитись реалізовувати розпаралелювання алгоритмів за допомогою багатопоточності в ОС Windows з використанням функцій WinAPI.

**15. [Складність 2]** Знайти **цифру, яка зустрічається найчастіше** у елементахмасиву, та кількість виявлень.

Завдання.

1. Підготувати файли з тестовими даними різного розміру: від малих прикладів для відлагоджування – і до десятків мегабайт для тестування швидкодії.

Тестові дані згенерувати випадковим чином за допомогою вбудованих функцій, а у випадку роботи зі зображенями – вручну.

1. Вирішити поставлену задачу в окремому потоці. Результат зберегти у файл.
2. Виконати розпаралелювання заданого алгоритму на 2, 4, 8, 20, 100, 1000 потоків. Переконатись, що результат співпадає із одно-поточною імплементацією.  
   Проаналізувати вплив розпаралелювання задачі на час виконання.
3. [Окрім задач, у яких потрібно працювати зі словами та послідовностями] Модифікуйте програму так, щоб потоки працювали не із послідовними секціями даних (наприклад, 1й потік – елементи масиву 0..500 000, а 2й потік – 500 001..1000 000), а змішано (1й потік – парні елементи, а 2й потік – непарні). Проведіть аналогічні тести із тими ж кількостями потоків, що й у попередньому пункті. Проаналізуйте вплив «перестановки доданків» на сумарний час виконання та поясніть результати.
4. Реалізувати можливість зміни пріоритету окремого потоку та проаналізувати вплив на тривалість виконання його частини задачі.
5. Результати виконання роботи відобразити у звіті.
   1. Текст програми-генератора тестових даних.
   2. Текст програми-рішення задачі.

Якщо рішення містить блоки запозиченого коду (stackoverflow та ін.) – виділити його курсивом та у коментарях вказати джерело.

Розуміння скопійованого коду – обов’язкове!

1. Вхідні-вихідні дані (для великих файлів – зразки та посилання на повні файли).
2. Час виконання в залежності від кількості потоків при розбитті задачі на суміжні інтервали. Пояснення часових характеристик.
3. Час виконання в залежності від кількості потоків при розбитті задачі на не-суміжні елементи у межах кожного потоку (парні – 1й потік / непарні – 2й потік). Пояснення часових характеристик.
4. Час виконання окремого потоку із пониженим/підвищеним пріоритетом у порівнянні із його «братами» стандартного пріоритету. Пояснення результату.

***Хід роботи:***

#include <iostream>

#include "Command.cpp"

#include "FileExplorer.cpp"

#include <fstream>

#include <windows.h>

#include <map>

#include <mutex>

#include <chrono>

#include <algorithm>

using namespace std::chrono;

CommandExtractor extractor = CommandExtractor();

FileExplorer fileExplorer = FileExplorer();

std::mutex mtx;

HANDLE hThreadArray[1000];

bool threadExecuting[1000];

int currentThreads = 0;

std::map<long long, long long> o;

typedef struct ThreadData1{

vector<long long> l;

long long initial;

long long threads;

ThreadData1(vector<long long> const &l, int initial, int threads) {

this->l = l;

this->initial = initial;

this->threads = threads;

}

} TD, \*PTD;

typedef struct ThreadData2{

int i; int n;

vector<long long> l;

ThreadData2(vector<long long> const& l, int i, int n) {

this->l = l;

this->i = i;

this->n = n;

}

} TD2, \*PTD2;

typedef struct OccurenceResult {

pair<long long, long long> occurence;

milliseconds alg\_time;

public:

OccurenceResult(pair<long long, long long> occurence, milliseconds alg\_time) {

this->alg\_time = alg\_time;

this->occurence = occurence;

}

} OR, \*POR;

OccurenceResult\* previousResult;

Command getCommand() {

cout << fileExplorer.path.string() << ">";

string command;

getline(cin, command);

Command cmd = extractor.extract(command);

return cmd;

}

void resetThreadExecuting() {

int threads = currentThreads == 0 ? 1000 : currentThreads;

for (int i = 0; i < threads; i++) {

threadExecuting[i] = false;

}

}

vector<long long> generateArray(int n) {

srand((unsigned)time(NULL));

vector<long long> result(n);

for (int i = 0; i < n; i++) {

result[i] = rand();

}

return result;

}

vector<long long> readArray(string path){

ifstream s(path);

vector<long long> r;

if (s.is\_open()) {

while (!s.eof()) {

long long el;

s >> el;

r.push\_back(el);

}

return r;

}

else {

cout << "File has not been opened, try again" << endl;

return r;

}

}

void writeArray(vector<long long> r, string path){

ofstream s(path);

if (s.is\_open()){

for (auto& e : r) {

s << e << " ";

}

s.close();

}

else {

s.close();

throw new invalid\_argument("Path should be valid");

}

}

DWORD popular(LPVOID param) {

PTD d = (PTD)param;

auto v = d->l;

long long initial, threads;

initial = d->initial;

threads = d->threads;

delete d;

for (int i = initial; i < v.size(); i += threads) {

//mtx.lock();

o[v[i]]++;

//mtx.unlock();

}

return 0;

}

DWORD popular\_default(LPVOID param) {

PTD2 data = (PTD2)param;

int i, n;

vector<long long> l = data->l;

i = data->i;

n = data->n;

for (; i < n && i < l.size(); i++) {

o[l[i]]++;

}

return 0;

}

void parallelPopular(vector<long long> &r, int th\_n) {

o.clear();

delete previousResult;

previousResult = NULL;

currentThreads = th\_n;

for (int i = 0; i < th\_n; i++) {

auto td = new ThreadData1(r, i, th\_n);

hThreadArray[i] = CreateThread(NULL, 0, popular, td, 0, NULL);

SuspendThread(hThreadArray[i]);

threadExecuting[i] = false;

}

}

void parallelPopular2(vector<long long>& r, int th\_n) {

o.clear();

long long d = max(r.size() / th\_n, 1);

int j = 0;

currentThreads = th\_n;

for (int i = 0; i < th\_n; i++) {

auto td = new ThreadData2(r, j, j + d);

hThreadArray[i] = CreateThread(NULL, 0, popular\_default, td, 0, NULL);

SuspendThread(hThreadArray[i]);

threadExecuting[i] = false;

j += d;

}

}

void resume(int i) {

if (threadExecuting[i] == false) {

ResumeThread(hThreadArray[i]);

threadExecuting[i] = true;

}

}

void rush() {

auto start = high\_resolution\_clock::now();

for (int i = 0; i < currentThreads; i++) {

ResumeThread(hThreadArray[i]);

}

for (int i = 0; i < currentThreads; i++) {

WaitForSingleObject(hThreadArray[i], INFINITE);

}

pair<long long, long long> max = \*o.begin();

for (auto &e : o) {

if (e.second > max.second)

max = e;

}

auto stop = high\_resolution\_clock::now();

previousResult = new OccurenceResult(max, duration\_cast<milliseconds>(stop - start));

}

void assortey(string path) {

vector<long long> v;

v = readArray(path);

vector<int> ths{ 1,2,4,10,20,100,1000 };

for (auto& e : ths) {

parallelPopular(v, e);

rush();

if (previousResult != NULL) {

cout << "Num of Threads: " << e << endl << previousResult->occurence.first << ":" << previousResult->occurence.second << endl;

cout << "Running time " << previousResult->alg\_time.count() << " ms" << endl;

}

}

}

void assortey2(string path) {

vector<long long> v;

v = readArray(path);

vector<int> ths{ 1,2,4,10,20,100,1000 };

for (auto& e : ths) {

parallelPopular2(v, e);

rush();

if (previousResult != NULL) {

cout << "Num of Threads: " << e << endl << previousResult->occurence.first << ":" << previousResult->occurence.second << endl;

cout << "Running time " << previousResult->alg\_time.count() << " ms" << endl;

}

}

}

void fileSystemCommands(Command command) {

if (!command.compare("dir")) {

cout << "\t" << "Type";

cout << "\t" << "Name" << endl;

cout << "\t" << "----" << "\t" << "---------" << endl;

for (std::filesystem::directory\_iterator next(fileExplorer.path), end; next != end; ++next) {

if (std::filesystem::is\_directory(next->path())) {

cout << "\t" << "d";

}

else {

cout << "\t" << "f";

}

cout << "\t" + (next->path()).filename().string() << endl;

}

}

if (!command.compare("cd")) {

string s\_path = command.arguments.back();

if (s\_path != ".") {

std::filesystem::path cdpath;

if (s\_path == "..") {

cdpath = fileExplorer.path.parent\_path();

}

else {

cdpath = std::filesystem::path(s\_path);

}

if (is\_directory(cdpath)) {

fileExplorer.goTo(absolute(cdpath));

}

else {

// give it a second chance

cdpath = std::filesystem::path(fileExplorer.path.string() + "\\" + cdpath.string());

if (std::filesystem::is\_directory(cdpath)) {

fileExplorer.goTo(std::filesystem::absolute(cdpath));

}

}

}

}

}

void threadsCommand(Command command) {

if (!command.compare("threads")) {

if (currentThreads == 0)

cout << "No threads have been created!" << endl;

else {

for (int i = 0; i < currentThreads; i++) {

cout << "Thread " << i << " ";

DWORD exitCode;

GetExitCodeThread(hThreadArray[i], &exitCode);

cout << exitCode << " " << (threadExecuting[i] ? "Not suspended" : "Suspended")

<< " " << GetThreadPriority(hThreadArray[i]) << " priority" << endl;

}

}

}

}

void occsCommand(Command command) {

if (!command.compare("occs")) {

if (command.arguments.size() == 1) {

string path = command.arguments.front();

cout << "Running on 1, 2, 4, 10, 100, 1000 threads in a row: " << endl;

assortey(path);

}

else if (command.arguments.size() == 2) {

string path = command.arguments.front();

int th\_n = stoi(command.arguments.back());

auto v = readArray(path);

parallelPopular(v, th\_n);

}

}

if (!command.compare("occs2")) {

if (command.arguments.size() == 1) {

string path = command.arguments.front();

cout << "Running on 1, 2, 4, 10, 100, 1000 threads in a row: " << endl;

assortey2(path);

}

else if (command.arguments.size() == 2) {

string path = command.arguments.front();

int th\_n = stoi(command.arguments.back());

auto v = readArray(path);

parallelPopular2(v, th\_n);

}

}

if (!command.compare("result")) {

if (previousResult == NULL) {

cout << "We haven't finished yet or we haven't even started!" << endl;

}

else {

cout << "Result is " << previousResult->occurence.first << "(number of occurences is "

<< previousResult->occurence.second << ")" << endl;

cout << "Running time: " << previousResult->alg\_time.count() << " ms" << endl;

ofstream s("D:\\o.txt");

if (s.is\_open()) {

s << previousResult->occurence.first << endl;

s << previousResult->occurence.second << endl;

}

else {

cout << "File cannot be opened." << endl;

}

}

}

if (!command.compare("resall")) {

for (int i = 0; i < currentThreads; i++) {

resume(i);

}

rush();

}

if (!command.compare("resume")) {

if (command.arguments.size() == 1) {

int i = stoi(command.arguments.front());

if (i < currentThreads && i >= 0) {

resume(i);

}

}

}

}

void generationCommand(Command command) {

if (!command.compare("generate")) {

if (command.arguments.size() == 2) {

string path = command.arguments.front();

int n = stoi(command.arguments.back());

writeArray(generateArray(n), path);

}

else {

cout << "Invalid arguments" << endl;

}

}

}

void priorityCommand(Command command) {

if (!command.compare("priority")) {

if (command.arguments.front() == "-h") {

cout << "List threads first with 'threads' command first. Then use: priority <num\_of\_thread> <1-5>" << endl;

}

else if (command.arguments.size() == 2) {

int i = stoi(command.arguments.front());

int priority = stoi(command.arguments.back());

if (i >= 0 && i < currentThreads) {

switch (priority) {

case 1:

priority = THREAD\_PRIORITY\_IDLE;

break;

case 2:

priority = THREAD\_PRIORITY\_BELOW\_NORMAL;

break;

case 3:

priority = THREAD\_PRIORITY\_NORMAL;

break;

case 4:

priority = THREAD\_PRIORITY\_ABOVE\_NORMAL;

break;

default:

priority = THREAD\_PRIORITY\_HIGHEST;

break;

}

SetThreadPriority(hThreadArray[i], priority);

cout << "Priority of Thread #" << i << " has been changed to priority " << priority << endl;

}

}

}

}

int main() {

Command command = getCommand();

while (command.compare("exit")) {

fileSystemCommands(command);

threadsCommand(command);

occsCommand(command);

generationCommand(command);

priorityCommand(command);

command = getCommand();

}

}

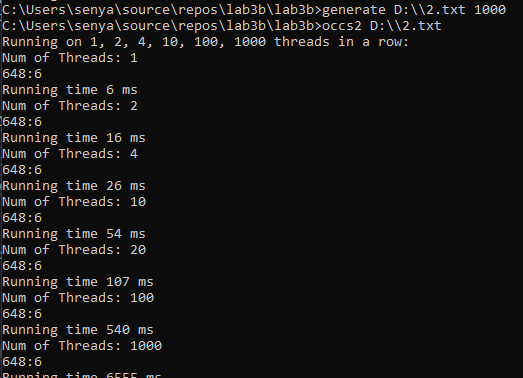
**Вхідні файли:**

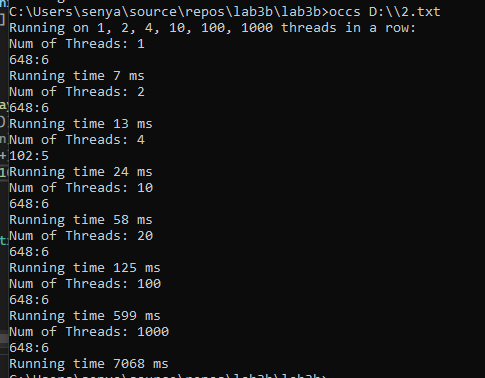
**184 498 855 205 636 728 25 690 752 293 37 911 195 410 828 426 937 381 476 317 732 172 213 674 138 311 564 939 520 807 968 91 183 225 491 241 318 816 171 935 648 106 241 85 328 474 361 795 514 438 772 431 852 233 186 44 605 579 132 604 52 215 65 822 508 846 600 795 726 347 566 614 672 918 991 755 29 429 578 890 285 576 456 643 671 75 797 475 28 226 283 288 365 975 555 554 376 368 349 627 614 879 650 992 617 850 352 920 724 196 214 997 275 219 449 456 478 404 198 938 320 939 448 584 732 928 197 430 314 398 644 680 734 726 521 277 281 852 892 605 533 911 591 348 283 623 407 95 648 834 661 111 471 704 565 179 120 968 102 235 930 610 123 227 228 343 857 852 315 477 379 857 967 623 379 280 422 412 954 343 93 967 41 778 20 872 536 250 121 137 807 392 840 275 395 331 634 473 228 438 908 877 389 749 2 898 139 73 346 507 217 9 813 386 851 900 276 627 770 765 560 225 896 502 741 497 876 776 255 774 320 466 30 501 161 912 542 218 87 878 562 426 61 157 577 123 187 76 401 448 179 715 420 938 376 269 640 212 414 908 252 117 174 937 728 163 744 163 419 582 718 488 811 290 622 874 241 122 801 316 98 489 642 349 613 498 609 340 858 956 360 93 248 358 949 330 362 745 154 182 858 133 722 102 687 516 4 41 891 383 260 729 848 125 313 938 748 275 537 235 954 97 572 187 27 801 927 998 69 725 375 136 315 769 369 329 93 423 279 561 313 282 558 141 978 173 310 315 824 924 761 883 640 299 408 607 70 198 274 165 832 185 430 217 636 833 820 416 941 90 327 827 682 179 269 480 592 207 490 32 305 13 1 84 868 130 429 793 551 517 975 462 411 499 545 107 135 653 819 20 991 735 691 558 102 186 90 82 338 940 981 200 816 558 561 349 903 686 235 672 272 478 438 697 342 413 214 579 515 469 997 669 780 405 724 427 142 559 837 458 945 380 646 187 675 95 633 527 320 199 393 501 265 177 655 319 778 502 134 900 945 273 876 834 692 212 978 201 529 74 678 224 452 672 551 154 901 304 296 873 8 430 977 487 489 999 970 648 879 406 809 820 984 505 790 323 117 867 277 370 920 188 436 173 675 718 566 517 363 617 679 588 165 273 257 935 528 774 150 412 651 133 943 187 424 821 732 889 761 916 645 747 897 571 520 505 743 95 996 715 703 914 102 208 769 458 314 868 198 248 548 513 344 523 975 802 449 325 814 692 407 734 913 102 742 345 459 523 863 908 417 213 952 291 469 444 108 239 428 871 12 683 100 962 821 712 83 25 404 105 500 996 616 297 347 531 127 380 434 476 19 628 677 34 534 61 804 184 548 546 382 783 105 25 166 709 490 204 249 7 632 698 605 240 26 414 713 385 954 462 912 251 740 22 287 155 642 982 250 422 235 66 698 702 309 200 690 347 768 871 354 621 333 318 603 272 823 955 51 809 753 125 84 609 169 996 337 648 555 263 72 323 392 390 630 281 921 50 503 803 686 345 853 66 172 703 780 813 735 161 603 435 52 587 653 485 922 981 345 304 553 865 886 698 788 640 707 202 627 934 22 253 53 395 636 31 740 499 849 759 801 568 648 686 233 279 284 456 232 81 336 707 541 658 428 225 966 535 425 865 814 123 88 443 219 619 629 995 557 108 874 877 141 617 665 10 146 617 689 738 873 624 374 235 886 36 90 296 550 163 16 229 791 643 818 58 618 451 961 927 840 608 734 111 883 506 407 880 566 697 236 722 400 377 186 749 884 302 311 408 12 908 215 47 800 197 295 661 458 153 32 891 196 529 503 511 236 805 594 792 966 367 877 904 210 639 309 908 992 743 725 929 529 522 32 327 8 473 414 986 25 823 210 902 929 918 299 67 719 448 450 265 977 619 987 726 713 303 480 158 129 329 10 706 196 53 137 100 280 607 190 641 153 155 135 768 808 735 435 411 628 133 702 844 391 733 991 730 396 330 705 203 607 547 271 387 46 685 779 886 870 951 760 875 986 679 414 859 456 387 271 816 875 227 48 328 464 133 969 636 59 816 961 4 434 29 196 870 30 666 638 534 501 306 148 648 210 986 161 260 265 804 54 194 845 857 686 78 381 763 121 341 47 385 727 313 547 159 360 616 572 541 665 148 50 686 730 719 59 36 222 622 251 825 690 864 538 57 553 174 253 479 999 308 247 566 989 905 577 537 156 268 944 524 318 855 863 92 465 97**

**Вихідні:**

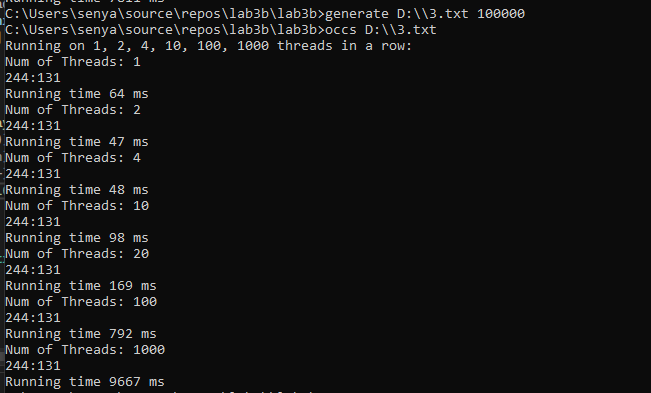
**648**

**6**





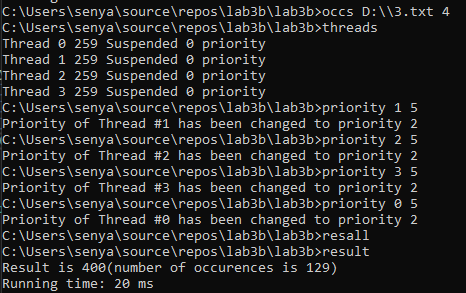
**При достатньо великій кількості даних ми бачимо значний прирост до ефективності алгоритму з використанням потоків:**



**Швидкість зростає на 30% на моєму комп’ютері, при використанні всіх ядер комп’ютера.**

**Звісно, використання потоків більше ніж 4 на моєму ком’ютері призводить до значних втрат швидкості алгоритму. Зв’язано це з тим, що багато часу застрачається на створення threadів, але при цьому вони не використовуються водночас та відбувається багато context switchів.**

**При зміні пріоритету на вищий, ми бачимо зменшення часу виконання в майже 2 рази:**



**Без пріоритетів HIGH: 48 ms,**

**З пріоритетами HIGH: 20 ms.**

**Висновок:** у ході виконання цієї лабораторної роботи ми ознайомилися з процесами та потоками в ОС Windows. Навчилися працювати із системними утилітами, що дають можливість отримувати інформацію про процеси, потоки, використовувану ними пам'ять, та іншу необхідну інформацію.