Министерство науки и образования РФ

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

«Санкт-Петербургский государственный электротехнический

университет «ЛЭТИ» им. В. И. Ульянова (Ленина)»

(СПбГЭТУ «ЛЭТИ»)

Кафедра вычислительной техники

Отчёт

по лабораторной работе № 5

на тему:

“Процессы и потоки в Windows”

по дисциплине “Операционные системы”

Выполнил студент гр. 4306:

Табаков А.В.

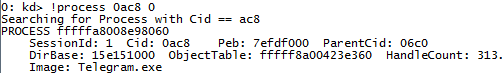
Принял: Тимофеев А.В.

Санкт-Петербург  
2016

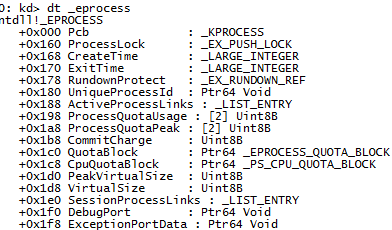
**Цель работы**: исследовать управление файловой системой с помощью Win32 API.

**Задание 5.1.** Исследовать структуры данных процессов и потоков.

Процесс для исследования – Telegram.exe



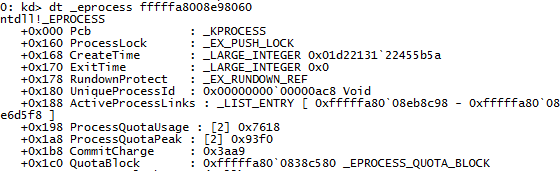
Список полей, составляющих блок EPROCESS:



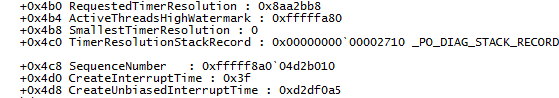
…



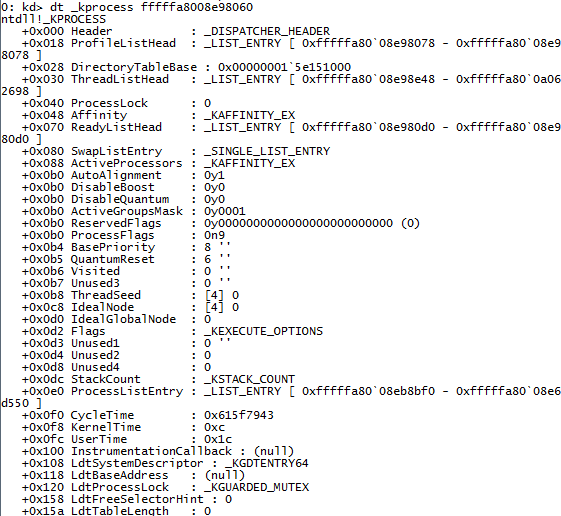
EPROCESS для Telegram.exe:



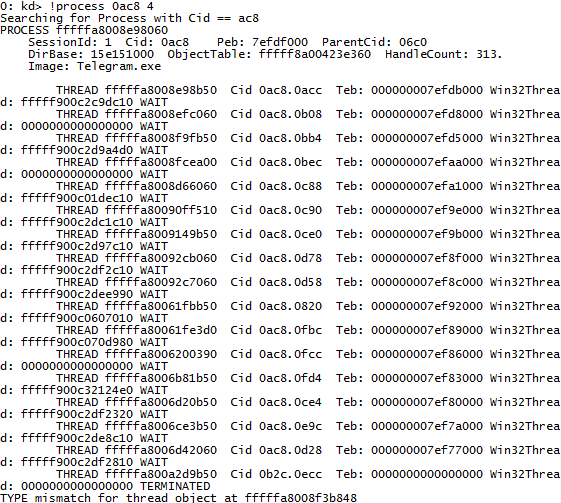
…



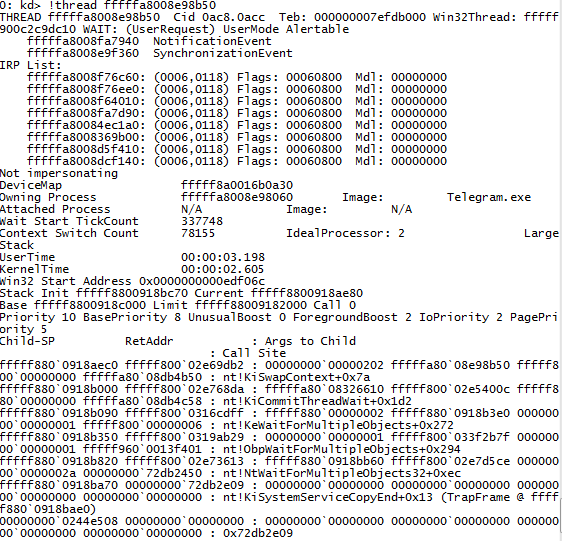
Структура KPROCESS для Telegram.exe



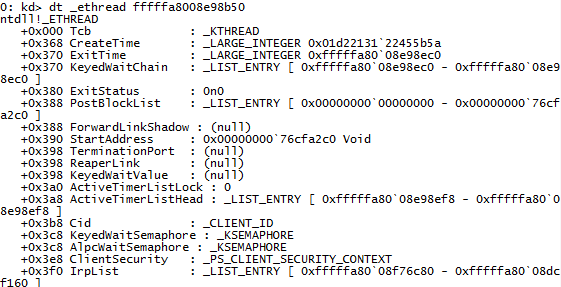
Потоки процесса Telegram.exe:



Подробная информация о первом потоке:



Значения полей ETHREAD первого потока Telegram.exe



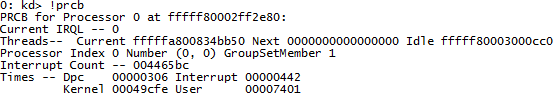
…



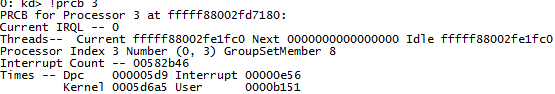
**Вывод**: при помощи livekd можно просматривать список EPROCESS, получать список потоков процесса и просматривать информацию о них.

**Задание 5.2.** Исследовать регистр контроля процессора и очередь потоков готовых для выполнения.

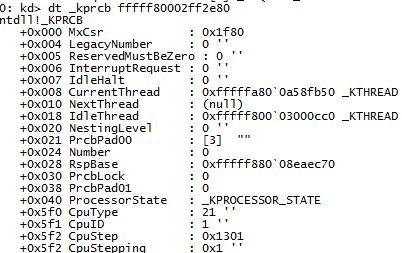
Регистр контроля ядер процессора:



…



Kernel Processor Register Control Block для ядра 0:



…

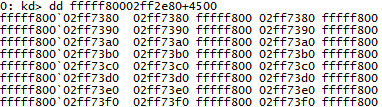


У данного ядра:

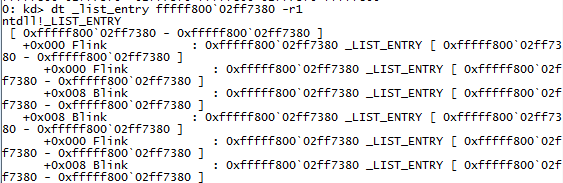




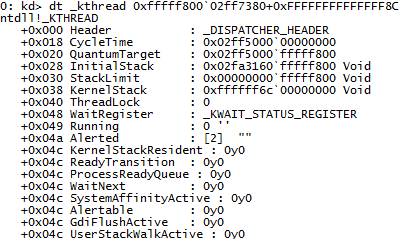
DispatcherReadtListHead в памяти:



Рекурсивный просмотр содержимого:



KTHREAD для 0 ядра:



…



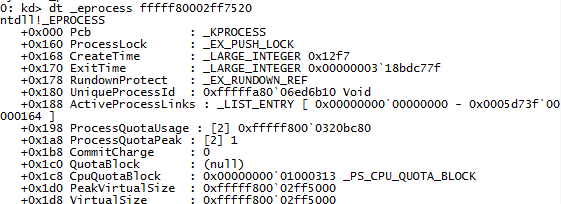
…



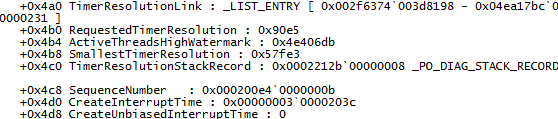
…



EPROCESS

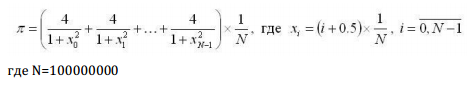


…

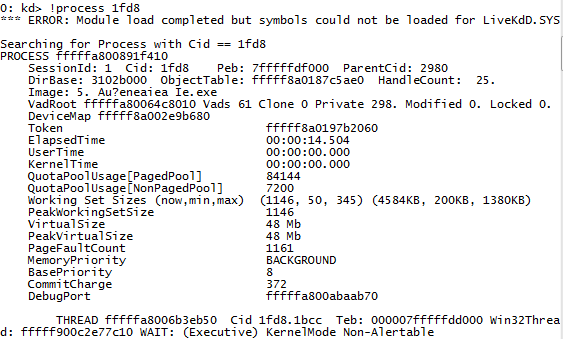


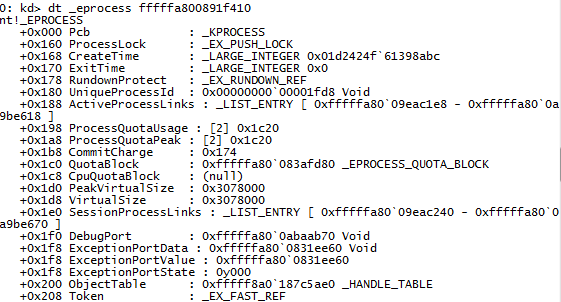
**Вывод**: при помощи livekd можно узнать какие приоритеты у потоков на выполнение и каким процессам они принадлежат.

**Задание 5.3.** Реализация многопоточного приложения с использованием функций Win32 API.



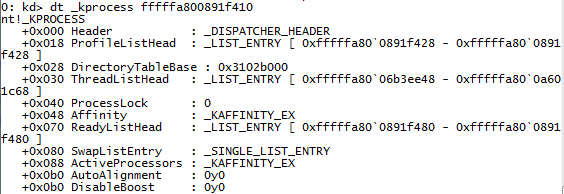
И размер блока 10\*500

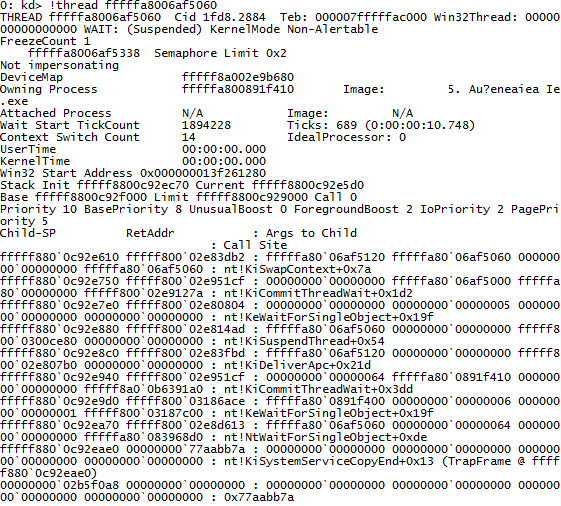




…

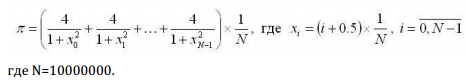


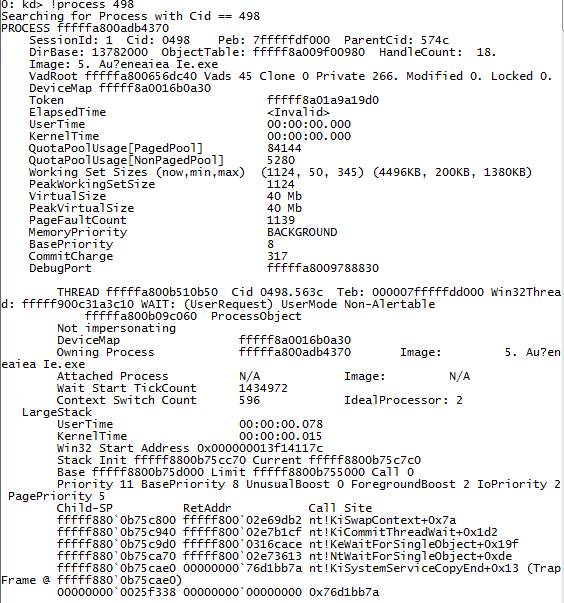


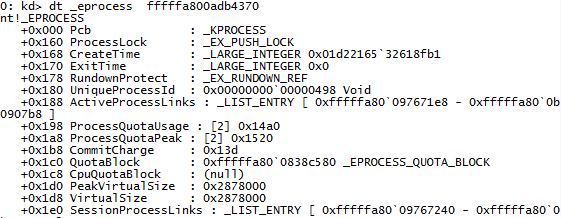


**Вывод:** WINAPI позволяет создавать многопоточные приложения, но для их разработки требуется большая внимательность, потому что очень легко ошибиться и всё будет работать неправильно.

**Задание 5.4.** Реализация многопоточного приложения с использованием технологии OpenMP.

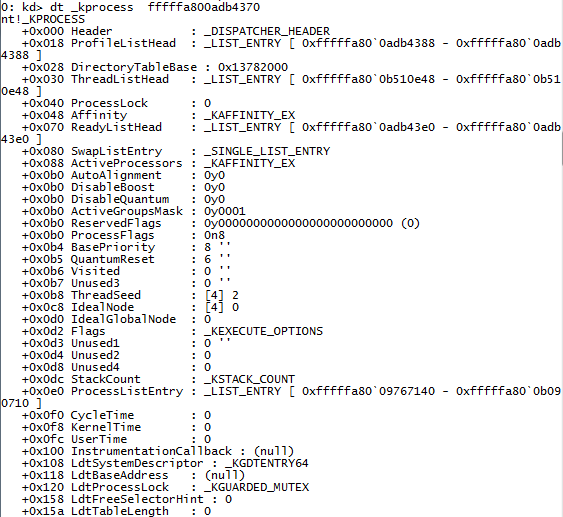


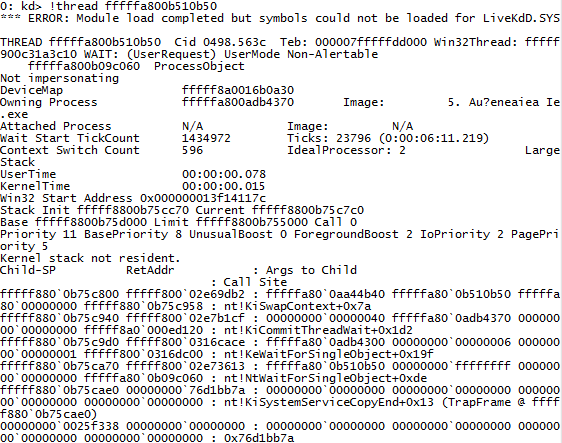




…







**Вывод:** Библиотека OpenMP позволяет значительно упростить жизнь разработчикам, за счёт своей простоты.

**Приложение**

**Текст программы**

#include <windows.h>

#include <iostream>

#include <iomanip>

#include <omp.h>

#include <AclAPI.h>

#include <AccCtrl.h>

using namespace std;

typedef struct \_Thread {

HANDLE hThread;

volatile unsigned int nextBlockIndex;

double threadPi;

volatile bool finished;

volatile bool calculating;

} Thread;

CRITICAL\_SECTION crit;

const int BLOCK\_SIZE = 10\*500;

const int N = 100000000;

const int TOTAL\_BLOCKS = N / BLOCK\_SIZE+(N % BLOCK\_SIZE ? 1 : 0);

Thread \*pThreads;

int menu();

void win32Processing();

DWORD WINAPI worker(LPVOID);

double OMP();

\_\_declspec(thread) DWORD dwTlsThreadIndex;

int main()

{

setlocale(0, ".1251");

int notExit;

switch (notExit = menu())

{

case 1:

win32Processing();

break;

case 2:

OMP();

break;

case 0:

break;

default:

if (notExit)

cout << "Такого варианта нет" << endl;

}

system("pause");

return 0;

}

int menu()

{

system("cls");

int point;

do {

cin.clear();

cin.sync();

cout << "Выберите пункт меню" << endl;

cout << "1 - Win32 API" << endl;

cout << "2 - Open Multi-Processing" << endl;

cout << "0 - Выход" << endl;

cout << ">";

cin >> point;

if (cin.fail())

cout << "Что-то пошло не так, выберите пункт меню повторно" << endl;

} while (cin.fail());

system("cls");

return point;

}

volatile LONG nextBlock = 0;

int numOfThreads = 1;

//int\* iterationsPerThread;

void win32Processing() {

srand(time(NULL));

double pi = 0, start = 0, end = 0;

cout << "Всего блоков: " << TOTAL\_BLOCKS << "\n";

cout << "Потоков : ";

cin >> numOfThreads;

//iterationsPerThread = new int[numOfThreads];

pThreads = new Thread[numOfThreads];

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

//iterationsPerThread [i] = 0;

pThreads[i].nextBlockIndex = nextBlock++;

pThreads[i].threadPi = 0;

pThreads[i].finished = false;

pThreads[i].calculating = false;

pThreads[i].hThread = CreateThread(NULL, 0, worker, (LPVOID)i, CREATE\_SUSPENDED, NULL);

}

HANDLE\* handlesArray = new HANDLE[numOfThreads];

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

handlesArray[i] = pThreads[i].hThread;

}

start = GetTickCount();

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

ResumeThread(pThreads[i].hThread);

}

while (nextBlock <= TOTAL\_BLOCKS) {

int i;

//rand() % numOfThreads

for (i = 0; ; i = (i + 1) % numOfThreads) {

SwitchToThread();

if (!pThreads[i].calculating || nextBlock > TOTAL\_BLOCKS)

break;

}

ResumeThread(pThreads[i].hThread);

}

cout << nextBlock << endl;

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

ResumeThread(pThreads[i].hThread);

cout << i << " " << a[i] << endl;

}

DWORD check = WaitForMultipleObjects(numOfThreads, handlesArray, true, INFINITE);

end = GetTickCount();

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

pi += pThreads[i].threadPi;

//cout << “i thread” << iterationsPerThread[i] << endl;

}

pi /= N;

cout << setprecision(70) << "Пи = " << pi << endl;

cout << "Время потрачено: " << (end - start) << " мс" << endl;

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

CloseHandle(pThreads[i].hThread);

}

}

DWORD WINAPI worker(LPVOID lpParameter) {

dwTlsThreadIndex = (DWORD)lpParameter;

pThreads[dwTlsThreadIndex].calculating = true;

unsigned long int beginIndex = 0;

unsigned long int endIndex = 0;

while (nextBlock <= TOTAL\_BLOCKS) {

double intermediatePi = 0;

beginIndex = pThreads[dwTlsThreadIndex].nextBlockIndex\*BLOCK\_SIZE;

endIndex = (pThreads[dwTlsThreadIndex].nextBlockIndex + 1)\*BLOCK\_SIZE;

if (endIndex > N)

endIndex = N;

for (unsigned long int i = beginIndex; i < endIndex; i++) {

double xi = (i + 0.5) / N;

intermediatePi += 4 / (1 + xi\*xi);

}

pThreads[dwTlsThreadIndex].threadPi += intermediatePi;

pThreads[dwTlsThreadIndex].calculating = false;

SuspendThread(pThreads[dwTlsThreadIndex].hThread);

pThreads[dwTlsThreadIndex].nextBlockIndex = InterlockedExchangeAdd(&nextBlock, 1);//nextBlock++;

//++ iterationsPerThread [dwTlsThreadIndex];

pThreads[dwTlsThreadIndex].calculating = true;

}

pThreads[dwTlsThreadIndex].finished = true;

return 0;

}

double OMP() {

int maxThreads;

cout << "Максимум потоков: ";

cin >> maxThreads;

omp\_set\_dynamic(0);

omp\_set\_num\_threads(maxThreads);

double start = GetTickCount();

double pi = 0;

#pragma omp parallel for schedule(dynamic, BLOCK\_SIZE) reduction(+:pi)

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

double xi = (i + 0.5) / N;

pi += 4 / (1 + xi\*xi);

}

double end = GetTickCount();

pi /= N;

cout << setprecision(60) << "Пи = " << pi << endl;

cout << "Время потрачено: " << (end - start) << " мс" << endl;

return pi;

}