**Задание 1**

#include <iostream>

#include <ctime>

using namespace std;

int main() {

time\_t t1 = time(&t1);

tm ttm;

localtime\_s(&ttm, &t1);

cout << ttm.tm\_mday << "."

<< ttm.tm\_mon + 1 << "."

<< ttm.tm\_year << " "

<< ttm.tm\_hour << ":"

<< ttm.tm\_min << ":"

<< ttm.tm\_sec << endl;

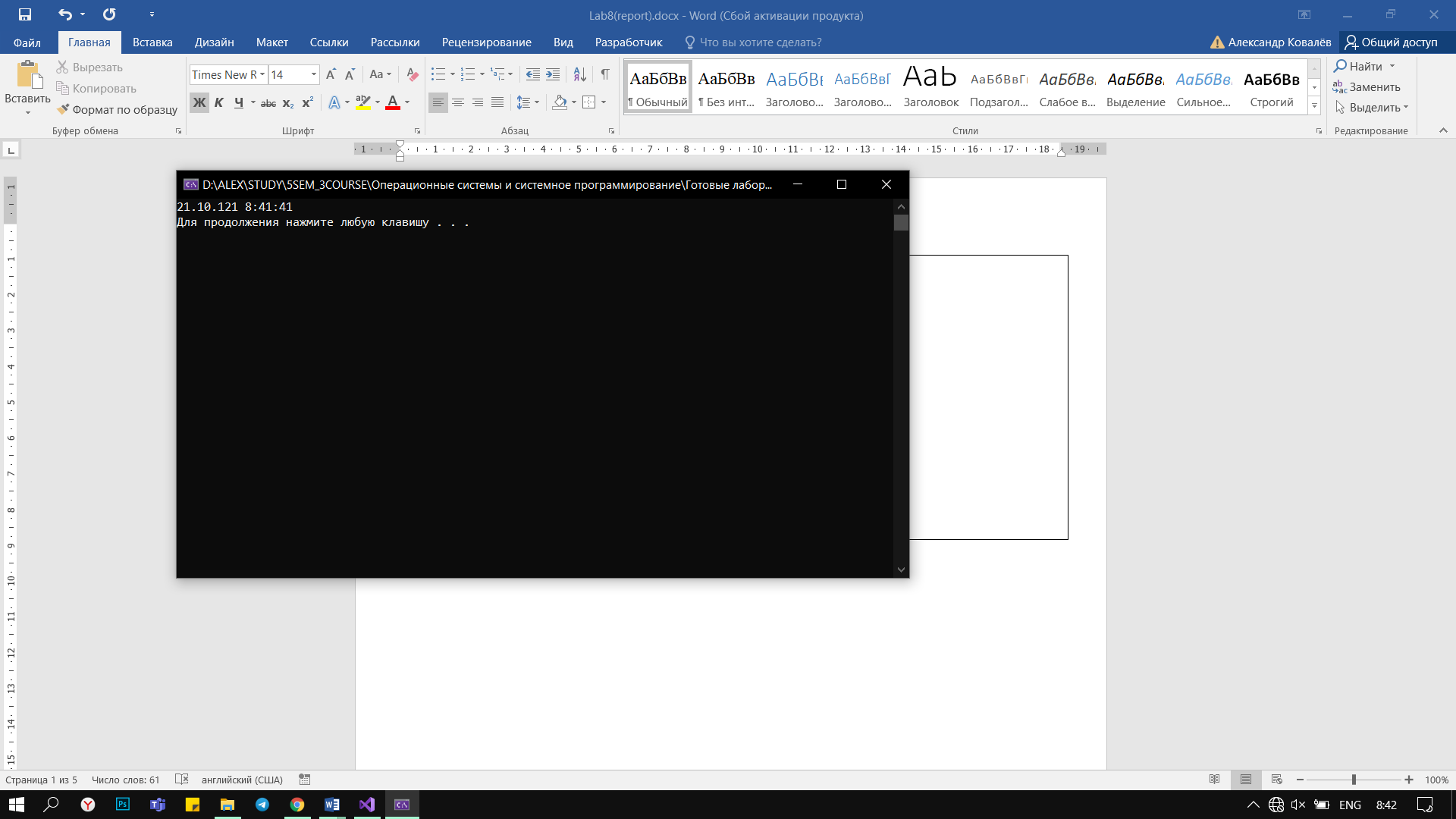
system("pause");

return 0;

}

**OS08\_01.cpp**

Результат выполнения программы:



Функция time() возвращает текущее календарное время системы. Если аргумент не является нулевым указателем, ей передается значение времени типа time\_t.

localtime\_s(const tm\* tm, const time\_t \* time) - функция преобразовывает текущее значение времени, передаваемое как аргумент, через указатель time на time\_t в структуру tm. Время указанное в time используется для заполнения структуры tm значениями текущей даты и времени.

**Задание 2**

#include <iostream>

#include <ctime>

using namespace std;

int main() {

clock\_t start = clock();

bool f5 = true, f10 = true;

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

if ((clock() - start) / CLOCKS\_PER\_SEC == 5 && f5) {

cout << "5 c - " << i << endl;

f5 = false;

}

else if ((clock() - start) / CLOCKS\_PER\_SEC == 10 && f10) {

cout << "10 c - " << i << endl;

f10 = false;

}

else if ((clock() - start) / CLOCKS\_PER\_SEC == 15) {

cout << "15 c - " << i << endl;

break;

}

}

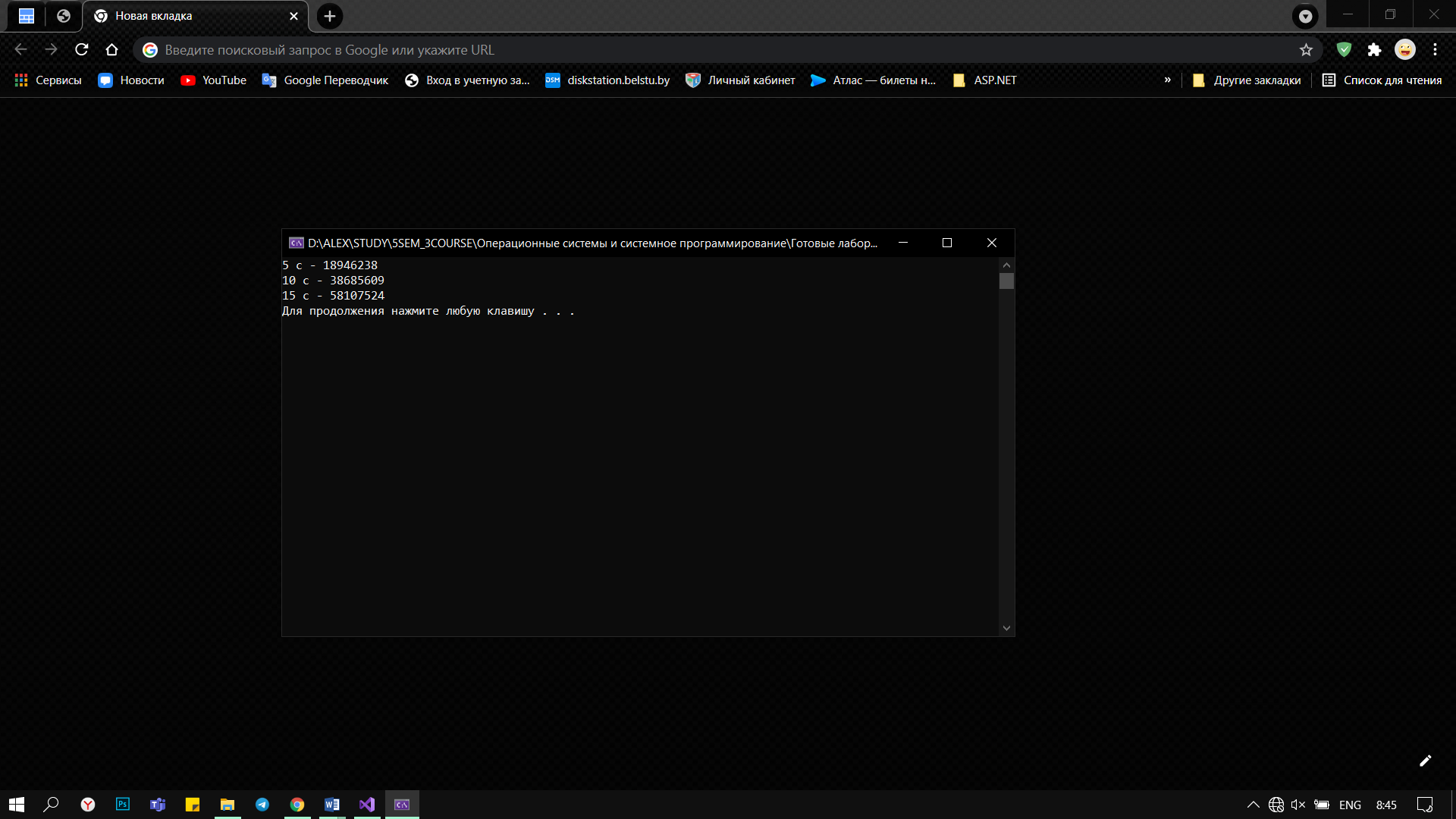
system("pause");

return 0;

}

**OS08\_02.cpp**

Результат выполнения программы:



Функция clock возвращает количество временных тактов, прошедших с начала запуска программы. С помощью [макроса CLOCKS\_PER\_SEC](http://cppstudio.com/spravochnik/standartnye-zagolovochnye-fajly-iz-si-v-s/zagolovochnyj-fajl-ctime-time-h/makros-clocks_per_sec/) функция получает количество пройденных тактов за 1 секунду. Таким образом, зная сколько выполняется тактов в секунду, зная время запуска программы можно посчитать время работы всей программы или отдельного её фрагмента, что и делает данная функция.

**Задание 3**

#include <iostream>

#include <ctime>

#include <Windows.h>

#define SECOND 10000000

using namespace std;

int main() {

HANDLE htimer = CreateWaitableTimer(NULL, FALSE, NULL);

long long it = -3 \* SECOND;

clock\_t start = clock();

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

if (!SetWaitableTimer(htimer, (LARGE\_INTEGER\*)&it, 3000, NULL, NULL, FALSE))

{

throw "Error SetWaitableTimer";

}

if ((clock() - start)/ CLOCKS\_PER\_SEC == 15)

{

cout << "- " << i << ", c - " << clock() - start << endl;

break;

}

cout << "- " << i << ", c - " << clock() - start << endl;

WaitForSingleObject(htimer, INFINITE);

}

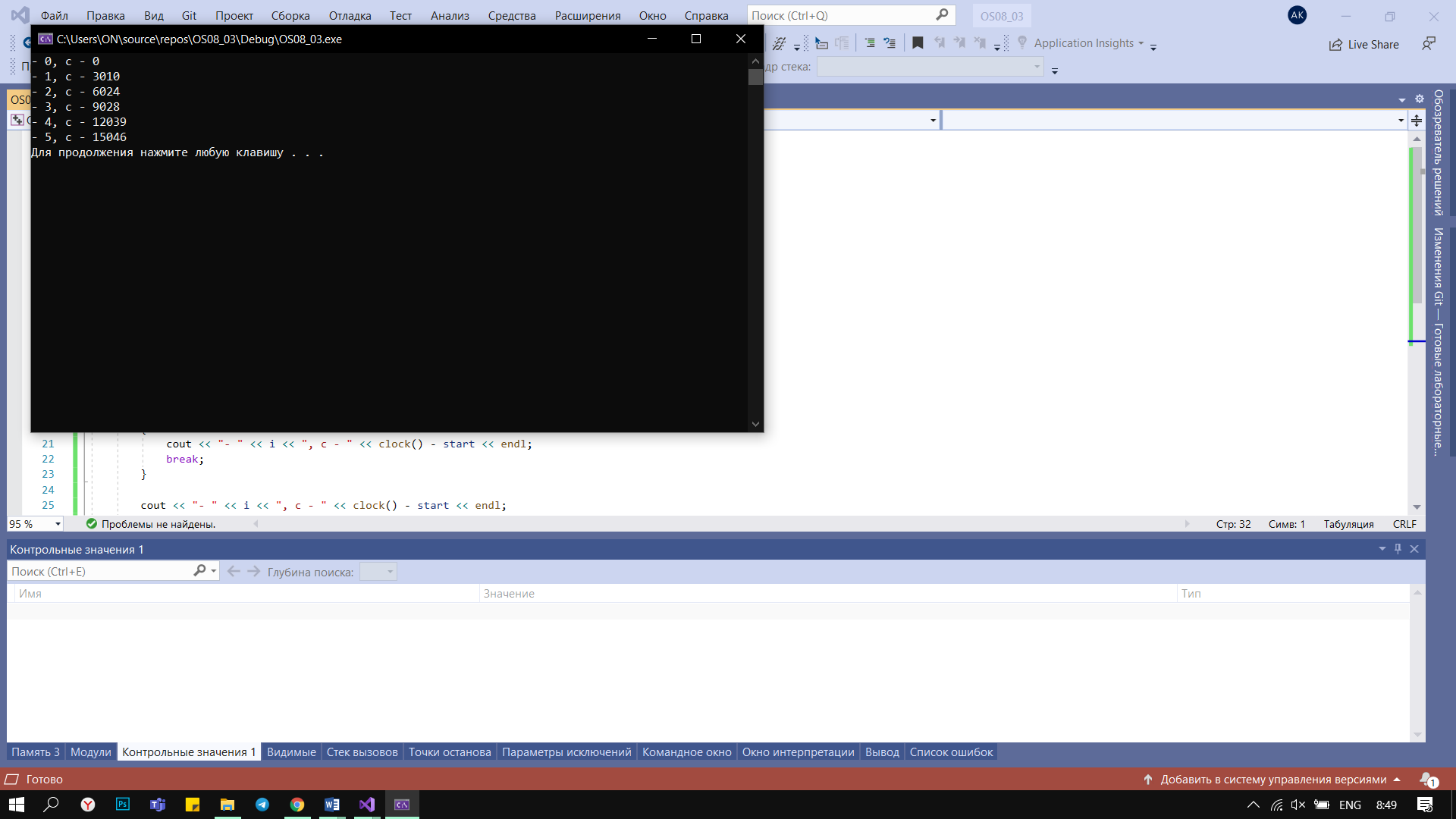
system("pause");

return 0;

}

**OS08\_03.cpp**

Результат выполнения программы:



function CreateWaitableTimer(

  lpTimerAttributes: PSecurityAttributes;     // Адрес структуры

                                              // TSecurityAttributes

  bManualReset: BOOL;  // Задает, будет ли таймер переходить в

                       // сигнальное состояние по завершении функции

                       // ожидания

  lpTimerName: PChar   // Имя объекта

) – создание таймера ожидания

Когда параметр bManualReset равен TRUE, то таймер после срабатывания функции ожидания остается в сигнальном состоянии до явного вызова SetWaitableTimer, если FALSE-таймер автоматически переходит в несигнальное состояние.

Если lpTimerName совпадает с именем уже существующего в системе таймера, то  функция возвращает его идентификатор, позволяя использовать объект для синхронизации между процессами. Имя таймера не должно совпадать с именем уже существующих объектов типов event, semaphore, mutex, job или file-mapping.

BOOL SetWaitableTimer(

HANDLE hTimer, - дескриптор таймера

const LARGE\_INTEGER \*lpDueTime, определяет время перехода таймера в свободное состояние (signaled) или время срабатывания таймера

LONG lPeriod, определяет режим работы и одновременно период повторения срабатываний ожидающего таймера

PTIMERAPCROUTINE pfnCompletionRoutine, определяет указатель на необязательную функцию асинхронного вызова (APC), которая помещается в очередь функций APC в момент срабатывания таймера и затем вызывается

LPVOID lpArgToCompletionRoutine, параметр передает в функцию асинхронного вызова (APC) произвольный аргумент, например указатель на объект или структуру

BOOL fResume - если он не нуль, выводит машину из спящего состояния по срабатыванию таймера

); - Активирует указанный таймер ожидания. Когда приходит должное время, таймер сигнализируется, и поток, который установил таймер, вызывает необязательную процедуру завершения.

**Задание 4**

#include <iostream>

#include <ctime>

#include <Windows.h>

#define SECOND 10000000

using namespace std;

int main() {

clock\_t start = clock();

DWORD pid = GetCurrentProcessId();

cout << "Main PID: " << pid << endl;

long long it = -60 \* SECOND;

HANDLE htimer = CreateWaitableTimer(NULL, FALSE, L"OS08\_04");

if (!SetWaitableTimer(htimer, (LARGE\_INTEGER\*)&it, 60000, NULL, NULL, FALSE)) {

throw "Error SrtWaitableTimer";

}

LPCWSTR an = L".\\OS08\_04\_X.exe";

PROCESS\_INFORMATION pi1, pi2;

pi1.dwThreadId = 1; pi2.dwThreadId = 2;

{

STARTUPINFO si; ZeroMemory(&si, sizeof(STARTUPINFO)); si.cb = sizeof(STARTUPINFO);

CreateProcess(an, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi1) ?

cout << "Process OS08\_04xA created \n" : cout << "Process OS08\_04xA not created \n";

}

{

STARTUPINFO si; ZeroMemory(&si, sizeof(STARTUPINFO)); si.cb = sizeof(STARTUPINFO);

CreateProcess(an, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi2) ?

cout << "Process OS08\_04xB created \n" : cout << "Process OS08\_04xB not created \n";

}

WaitForSingleObject(pi1.hProcess, INFINITE); WaitForSingleObject(pi2.hProcess, INFINITE);

CancelWaitableTimer(htimer);

cout << "time: " << clock() - start << endl;

system("pause");

return 0;

}

**OS08\_04.cpp**

#include <iostream>

#include <ctime>

#include <Windows.h>

#include <tlhelp32.h>

#include <stdio.h>

using namespace std;

DWORD ChildThread();

DWORD GetParentPid();

int check(int n);

bool stop = false;

int main() {

clock\_t start = clock();

cout << "Parent Pid: " << GetParentPid() << endl;

HANDLE htimer = OpenWaitableTimer(TIMER\_ALL\_ACCESS, FALSE, L"OS08\_04");

DWORD ChildId2 = NULL;

HANDLE hChild2 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)ChildThread, NULL, 0, &ChildId2);

WaitForSingleObject(htimer, INFINITE);

stop = true;

CloseHandle(hChild2);

cout << "time: " << clock() - start << endl;

system("pause");

return 0;

}

DWORD GetParentPid() {

HANDLE hSnapshot;

PROCESSENTRY32 pe32;

DWORD ppid = 0, pid = GetCurrentProcessId();

hSnapshot = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);

\_\_try {

if (hSnapshot == INVALID\_HANDLE\_VALUE) \_\_leave;

ZeroMemory(&pe32, sizeof(pe32));

pe32.dwSize = sizeof(pe32);

if (!Process32First(hSnapshot, &pe32)) \_\_leave;

do {

if (pe32.th32ProcessID == pid) {

ppid = pe32.th32ParentProcessID;

break;

}

} while (Process32Next(hSnapshot, &pe32));

}

\_\_finally {

if (hSnapshot != INVALID\_HANDLE\_VALUE) CloseHandle(hSnapshot);

}

return ppid;

}

DWORD ChildThread() {

DWORD tid = GetCurrentThreadId();

for (long long i = 2; ; i++) {

if (stop) {

break;

}

if (check(i)) {

cout << i << endl;

}

}

return 0;

}

int check(int n) {

for (int i = 2; i <= n / 2; i++) {

if ((n % i) == 0) {

return 0;

}

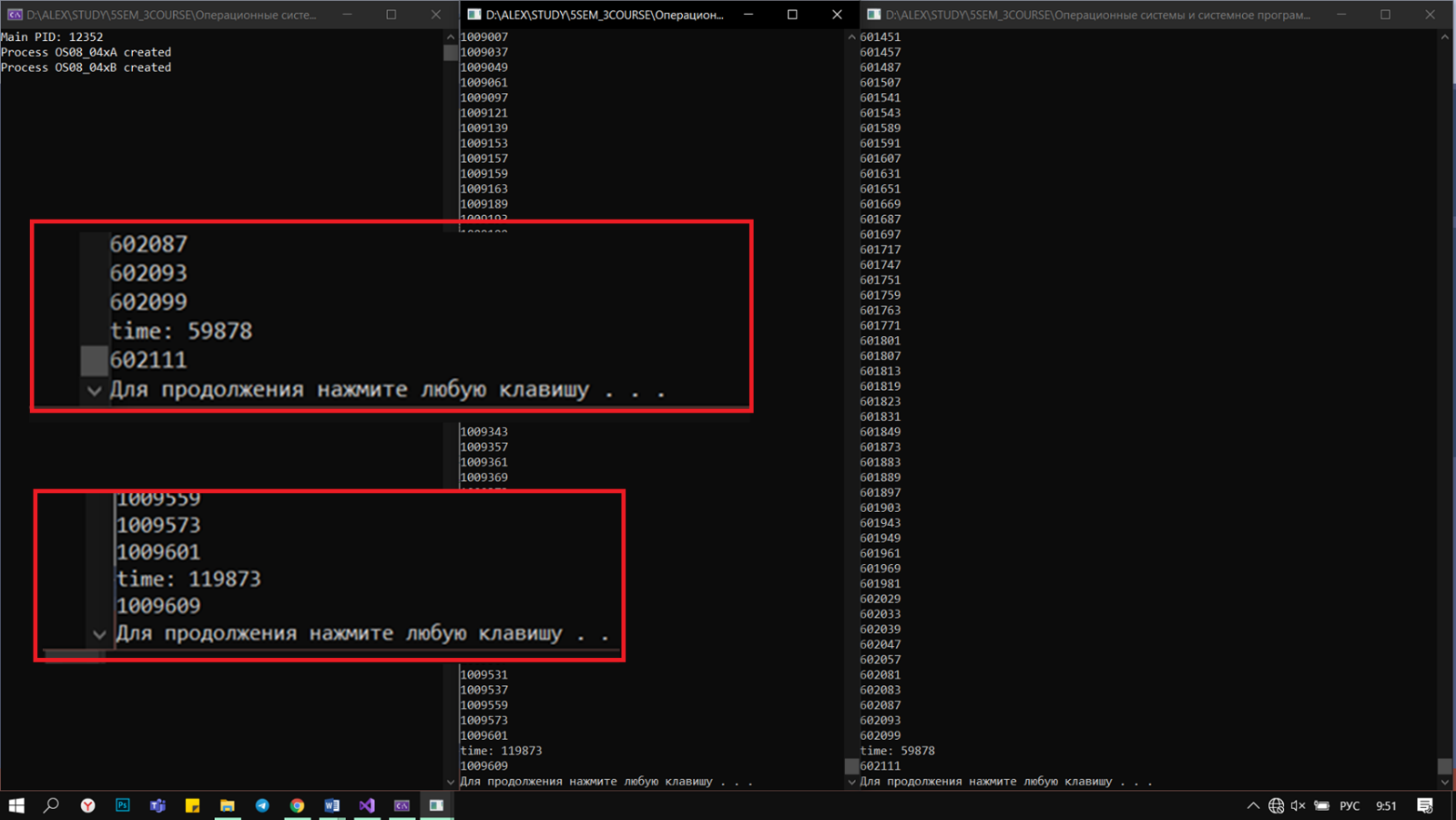
}

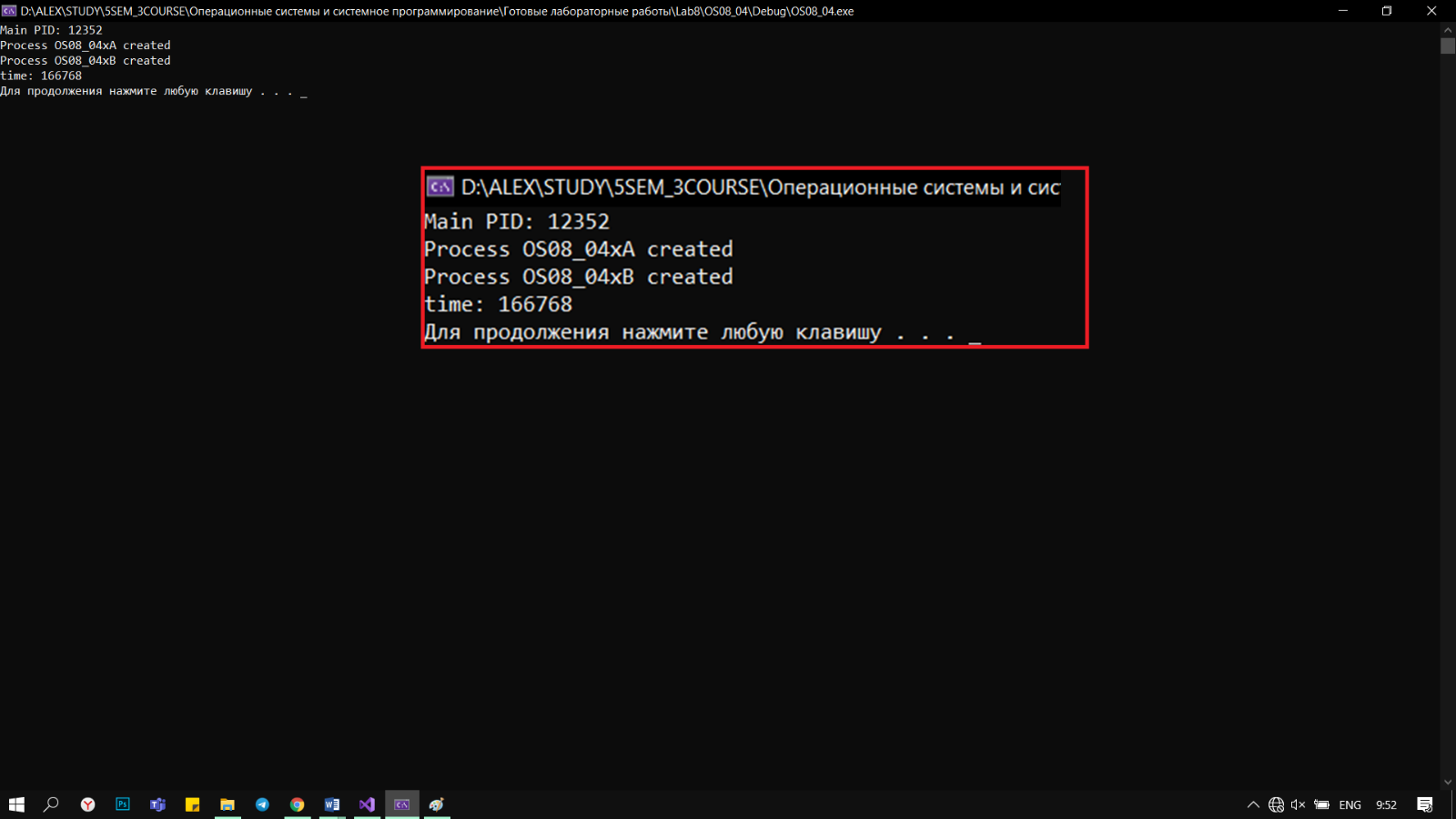
return 1;

}

**OS08\_04\_X.cpp**

Результат выполнения программы:



****

**Задание 5**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <pthread.h>

#include <time.h>

int main() {

time\_t t = time(&t);

struct tm \* tmptr;

tmptr = gmtime(&t);

printf("%d.", tmptr->tm\_mday);

printf("%d.", tmptr->tm\_mon + 1);

printf("%d.", tmptr->tm\_year);

printf(" ");

printf("%d.", tmptr->tm\_hour);

printf("%d.", tmptr->tm\_min);

printf("%d\n", tmptr->tm\_sec);

exit(0);

}

**OS08\_05.c**

Результат выполнения программы:

****

time(time\_t \*second) – эта функция возвращает время с 00:00:00 UTC 1 января 1970 г. (временная метка Unix) в секундах. Если second не является нулевым указателем, возвращаемое значение также сохраняется в объекте, на который указывает second.

Библиотечная функция struct tm \* gmtime (const time\_t \* timer) использует значение, указанное таймером, для заполнения структуры tm значениями, которые представляют соответствующее время, выраженное в универсальном координированном времени (UTC) или часовом поясе GMT.

**Задание 6**

#define \_POSIX\_C\_SOURCE 199309L

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <pthread.h>

#include <time.h>

int main() {

int count = 0;

struct timespec start, end, proc;

clock\_gettime(CLOCK\_REALTIME, &start);

for(;;) {

count++;

clock\_gettime(CLOCK\_PROCESS\_CPUTIME\_ID, &proc);

if(proc.tv\_sec == 2) break;

}

clock\_gettime(CLOCK\_REALTIME, &end);

printf("2 c: %d\n", count);

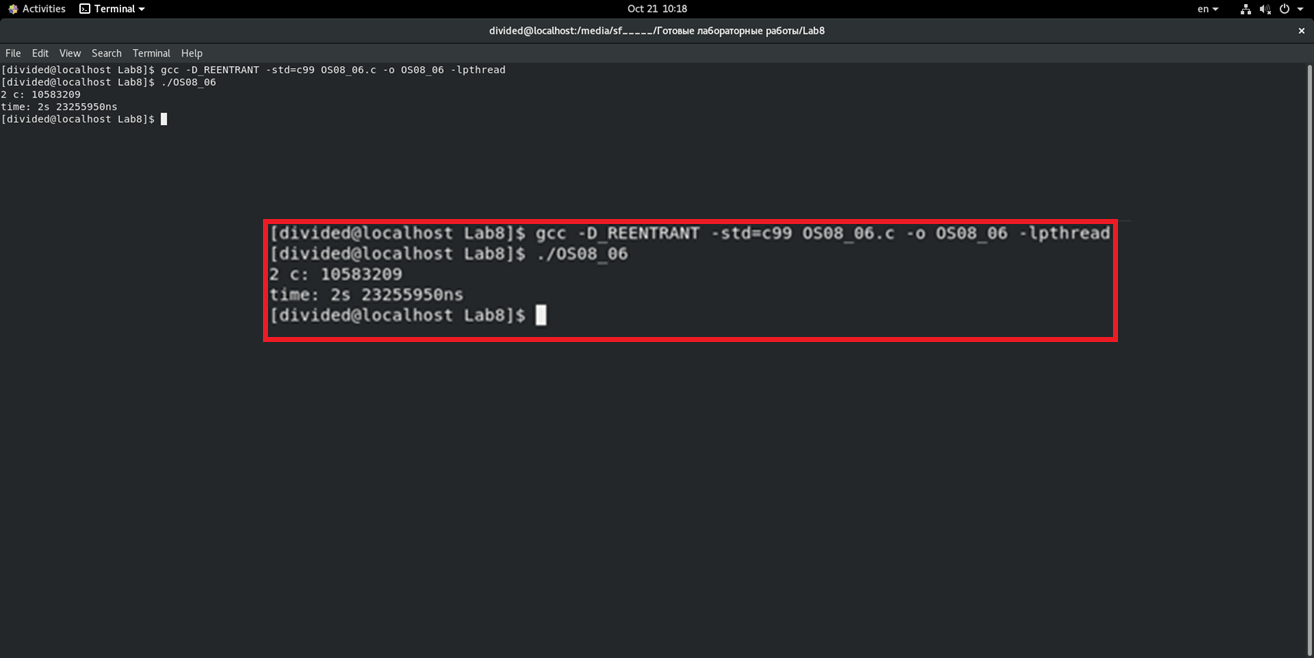
printf("time: %lds %ldns\n", end.tv\_sec - start.tv\_sec, end.tv\_nsec - start.tv\_nsec);

exit(0);

}

**OS08\_06.c**

Результат выполнения программы:

****

Функция int clock\_gettime (clockid\_t clock\_id, struct timespec \*t\_time) - получает текущее время по часам, идентифицированным часами, сохраняя его в секундах и наносекундах в \* t\_time.

clock\_id – идентификатор часов.

t\_time – структура, в которую будет записано определенное время.