**Задание 1**

#include <iostream>

#include <ctime>

#include <Windows.h>

using namespace std;

DWORD WINAPI A();

DWORD WINAPI B();

void EnterCriticalSectionAssem();

void LeaveCriticalSectionAssem();

int check = 0;

int main() {

DWORD AId = NULL, BId = NULL;

HANDLE hA = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)A, NULL, 0, &AId);

HANDLE hB = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)B, NULL, 0, &BId);

WaitForSingleObject(hA, INFINITY); WaitForSingleObject(hB, INFINITY);

CloseHandle(hA); CloseHandle(hB);

system("pause");

return 0;

}

DWORD WINAPI A() {

int start = clock();

EnterCriticalSectionAssem();

for (int i = 0; i < 5; i++) {

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

}

LeaveCriticalSectionAssem();

return 0;

}

DWORD WINAPI B() {

int start = clock();

EnterCriticalSectionAssem();

for (int i = 0; i < 5; i++) {

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

}

LeaveCriticalSectionAssem();

return 0;

}

void EnterCriticalSectionAssem() {

//Префикс lock, помещенный перед командой, устанавливает сигнал на линии LOCK системной шины и запрещает доступ к шине другим процессорам на время выполнения данной команды.

//Команда BTS сохраняет значение бита, из первого операнда со смещением, указанным вторым операндом, во флаге CF, а затем устанавливает этот бит в 1.

\_asm {

CriticalSection:

lock bts check, 0;

jc CriticalSection

}

}

void LeaveCriticalSectionAssem() {

//Команда BTR сохраняет значение бита, из первого операнда со смещением, указанным вторым операндом, во флаге CF, а затем обнуляет этот бит.

\_asm lock btr check, 0

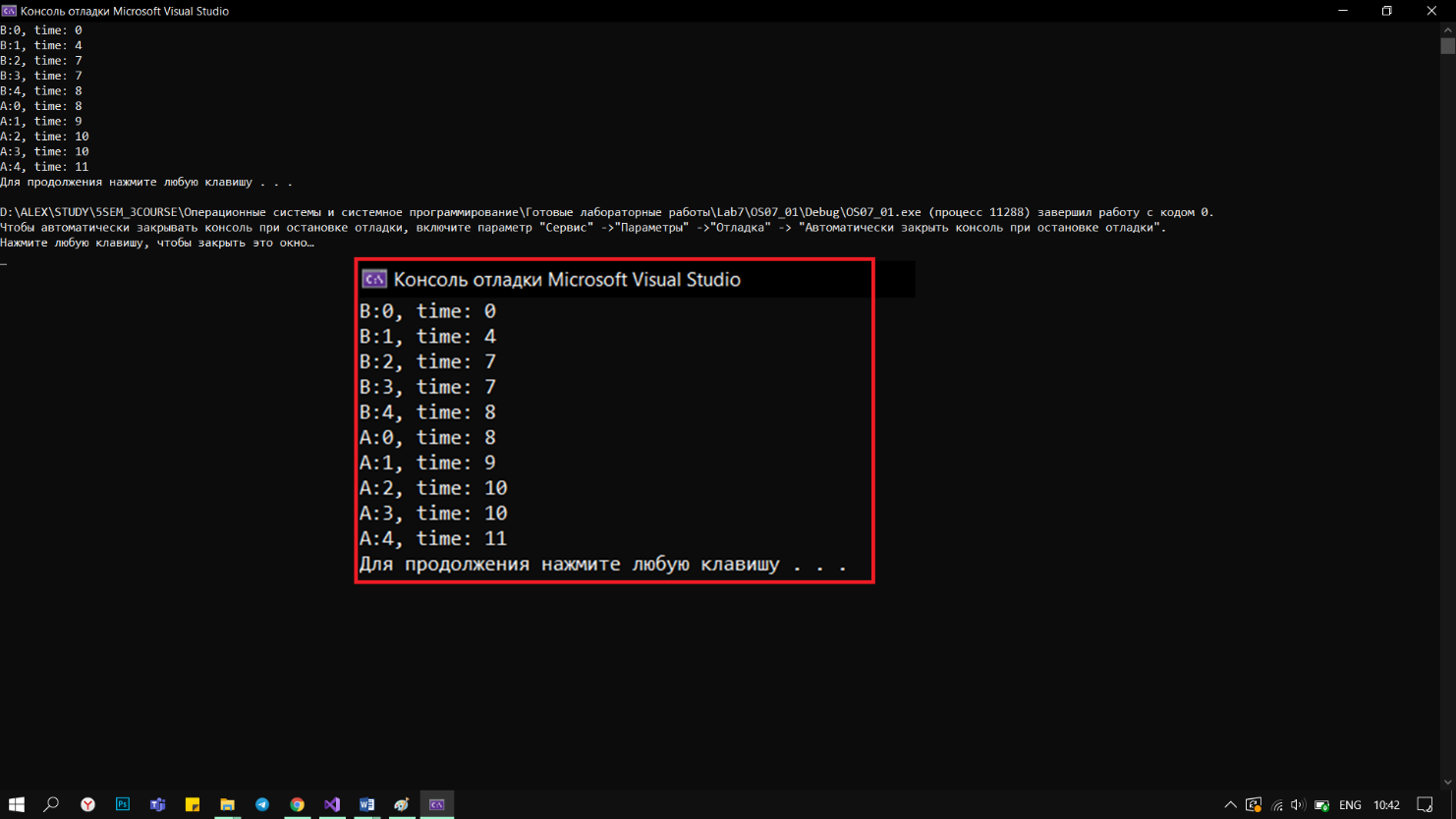
}

**OS07\_01.cpp**

Результат выполнения программы без применения команд BTS или BTR:

****

Результат выполнения программы с применением команд BTS или BTR:

****

При запуске потоков вызывается функция EnterCriticalSectionAssem(), которая содержит вставку ассемблерного кода. Команда BTS сохраняет значение бита, из первого операнда со смещением, указанным вторым операндом, во флаге CF, а затем устанавливает этот бит в 1. Команда jc выполняет ветвление (цикл) по адресу, если флаг переноса равен "1", в противном случае выполняется следующая команда. Этот переход будет осуществляться до тех пор, пока не будет вызвана функция LeaveCriticalSectionAssem, в которой находится команда BTR, которая сохраняет значение бита, из первого операнда со смещением, указанным вторым операндом, во флаге CF, а затем обнуляет этот бит. То есть поток А установит 1 и поток Б будет ходить по этому циклу, в то время, как А будет выполнять свою работу. Потом А сбросит и поток Б сможет задать 1 и продолжить свою работу.

**Задание 2**

#include <iostream>

#include <ctime>

#include <Windows.h>

using namespace std;

DWORD pid = NULL;

DWORD WINAPI A();

DWORD WINAPI B();

CRITICAL\_SECTION cs;

int main() {

pid = GetCurrentProcessId();

InitializeCriticalSection(&cs);

DWORD tid = GetCurrentThreadId();

DWORD AId = NULL, BId = NULL;

HANDLE hA = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)A, NULL, 0, &AId);

HANDLE hB = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)B, NULL, 0, &BId);

int start = clock();

for (int i = 0; i < 90; i++) {

if (i == 30) {

EnterCriticalSection(&cs);

}

if (i == 60) {

LeaveCriticalSection(&cs);

}

cout << i << " pid = " << pid << ", main tid = " << tid << ", time: " << clock() - start << endl;

Sleep(100);

}

WaitForSingleObject(hA, INFINITY); WaitForSingleObject(hB, INFINITY);

DeleteCriticalSection(&cs);

CloseHandle(hA); CloseHandle(hB);

system("pause");

return 0;

}

DWORD WINAPI A() {

DWORD tid = GetCurrentThreadId();

int start = clock();

for (int i = 0; i < 90; i++) {

if (i == 30) {

EnterCriticalSection(&cs);

}

if (i == 60) {

LeaveCriticalSection(&cs);

}

cout << i << " pid = " << pid << ", A tid = " << tid << ", time: " << clock() - start << endl;

Sleep(100);

}

return 0;

}

DWORD WINAPI B() {

DWORD tid = GetCurrentThreadId();

int start = clock();

for (int i = 0; i < 90; i++) {

if (i == 30) {

EnterCriticalSection(&cs);

}

if (i == 60) {

LeaveCriticalSection(&cs);

}

cout << i << " pid = " << pid << ", B tid = " << tid << ", time: " << clock() - start << endl;

Sleep(100);

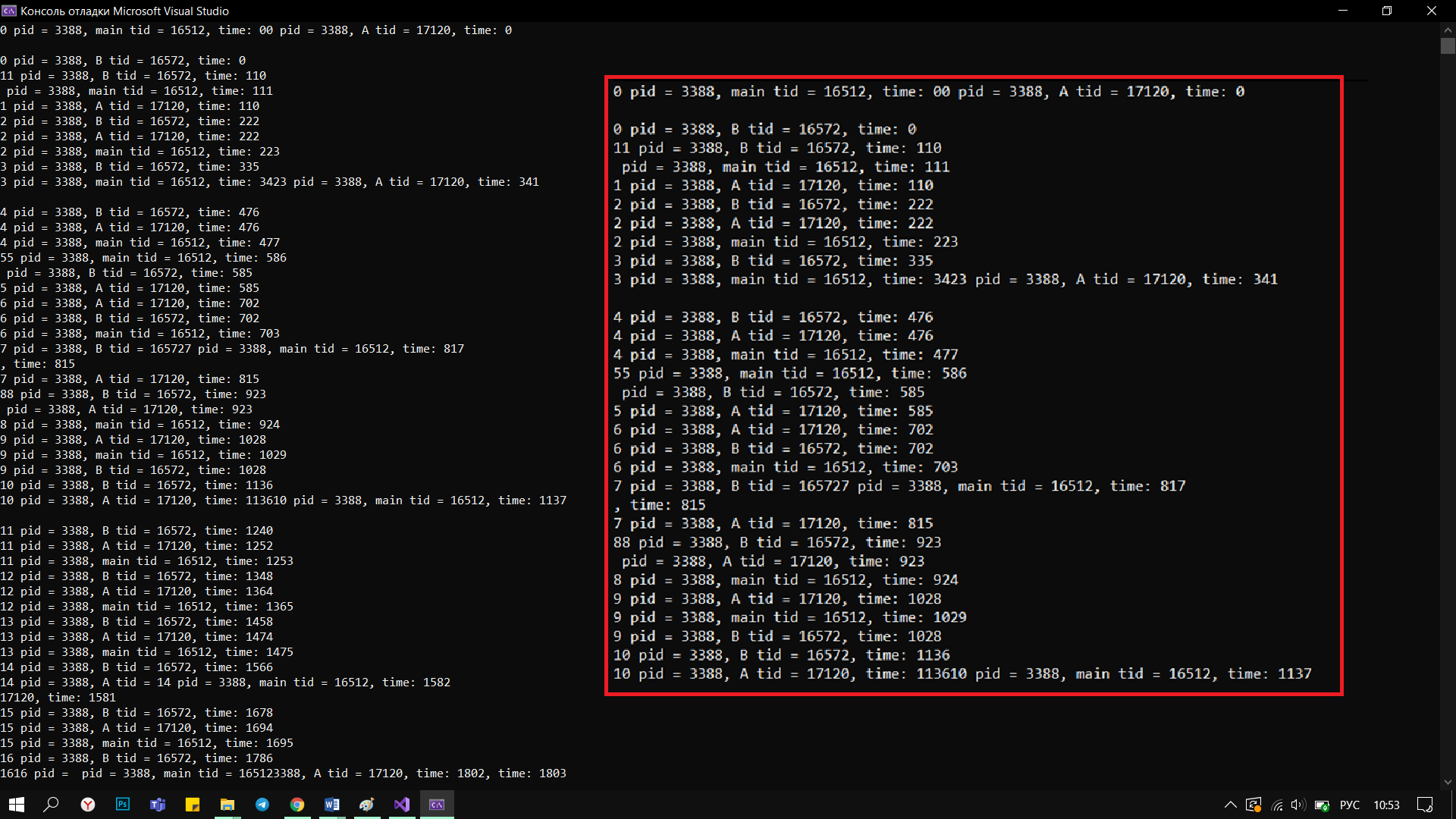
}

return 0;

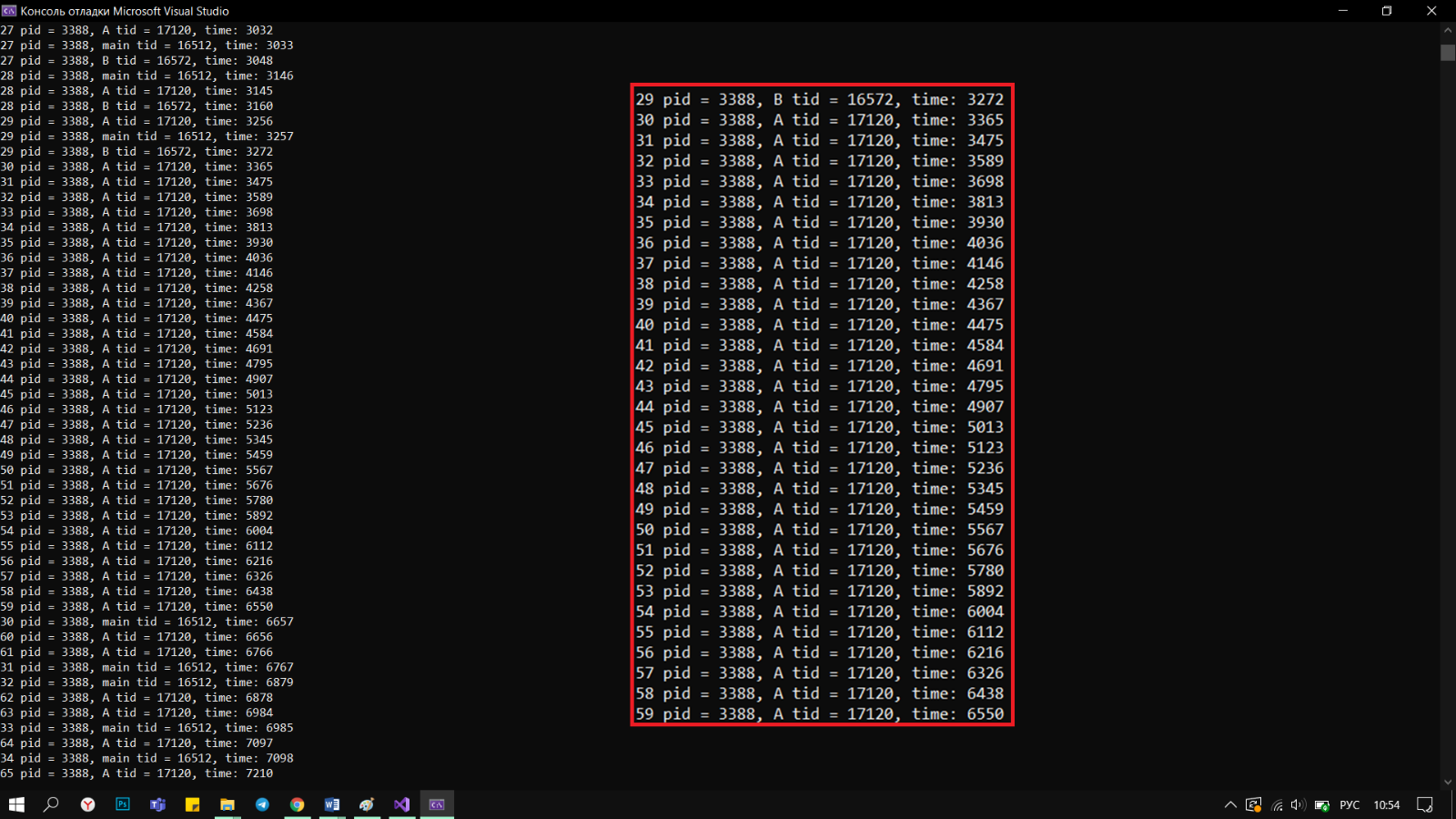
}

**OS07\_02.cpp**

До 30 итерации цикла:

****

С 30 по 60 итерации цикла:

****

С 60 итерации цикла:

****

**Задание 3**

#include <Windows.h>

#include <iostream>

#include <ctime>

using namespace std;

int main() {

int start = clock();

DWORD pid = GetCurrentProcessId();

HANDLE hm = CreateMutex(NULL, FALSE, L"Mutex");

LPCWSTR an1 = L"D:\\ALEX\\STUDY\\5SEM\_3COURSE\\Операционные системы и системное программирование\\Готовые лабораторные работы\\Lab7\\OS07\_03\\Debug\\OS07\_03A.exe";

LPCWSTR an2 = L"D:\\ALEX\\STUDY\\5SEM\_3COURSE\\Операционные системы и системное программирование\\Готовые лабораторные работы\\Lab7\\OS07\_03\\Debug\\OS07\_03B.exe";

STARTUPINFO si1; STARTUPINFO si2;

PROCESS\_INFORMATION pi1; PROCESS\_INFORMATION pi2;

ZeroMemory(&si1, sizeof(STARTUPINFO)); ZeroMemory(&si2, sizeof(STARTUPINFO));

si1.cb = sizeof(STARTUPINFO); si2.cb = sizeof(STARTUPINFO);

if (CreateProcess(an1, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si1, &pi1)) {

cout << "--Process OS07\_03A created\n";

}

else {

cout << "--Process OS07\_03A not created\n";

}

if (CreateProcess(an2, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si2, &pi2)) {

cout << "--Process OS07\_03B created\n";

}

else {

cout << "--Process OS07\_03B not created\n";

}

for (int i = 0; i < 90; i++) {

if (i == 30) {

WaitForSingleObject(hm, INFINITE);

}

if (i == 60) {

ReleaseMutex(hm);

}

cout << i << " OS07\_03 pid = " << pid << ", time: " << clock() - start << endl;

Sleep(100);

}

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

CloseHandle(hm);

CloseHandle(pi1.hProcess); CloseHandle(pi2.hProcess);

system("pause");

return 0;

}

**OS07\_03.cpp**

#include <Windows.h>

#include <iostream>

#include <ctime>

using namespace std;

int start;

int main() {

start = clock();

DWORD pid = GetCurrentProcessId();

HANDLE hm = OpenMutex(SYNCHRONIZE, FALSE, L"Mutex");

hm == NULL ? cout << "OS07\_03A: Open Error Mutex\n" : cout << "OS07\_03A: Open Mutex\n";

for (int i = 0; i < 90; i++) {

if (i == 30) {

WaitForSingleObject(hm, INFINITE);

}

if (i == 60) {

ReleaseMutex(hm);

}

cout << i << " OS07\_03A pid = " << pid << ", time: " << clock() - start << endl;

Sleep(100);

}

CloseHandle(hm);

system("pause");

}

**OS07\_03A.cpp**

#include <Windows.h>

#include <iostream>

#include <ctime>

using namespace std;

int main() {

int start = clock();

DWORD pid = GetCurrentProcessId();

HANDLE hm = OpenMutex(SYNCHRONIZE, FALSE, L"Mutex");

hm == NULL ? cout << "OS07\_03B: Open Error Mutex\n" : cout << "OS07\_03B: Open Mutex\n";

for (int i = 0; i < 90; i++) {

if (i == 30) {

WaitForSingleObject(hm, INFINITE);

}

if (i == 60) {

ReleaseMutex(hm);

}

cout << i << " OS07\_03B pid = " << pid << ", time: " << clock() - start << endl;

Sleep(100);

}

CloseHandle(hm);

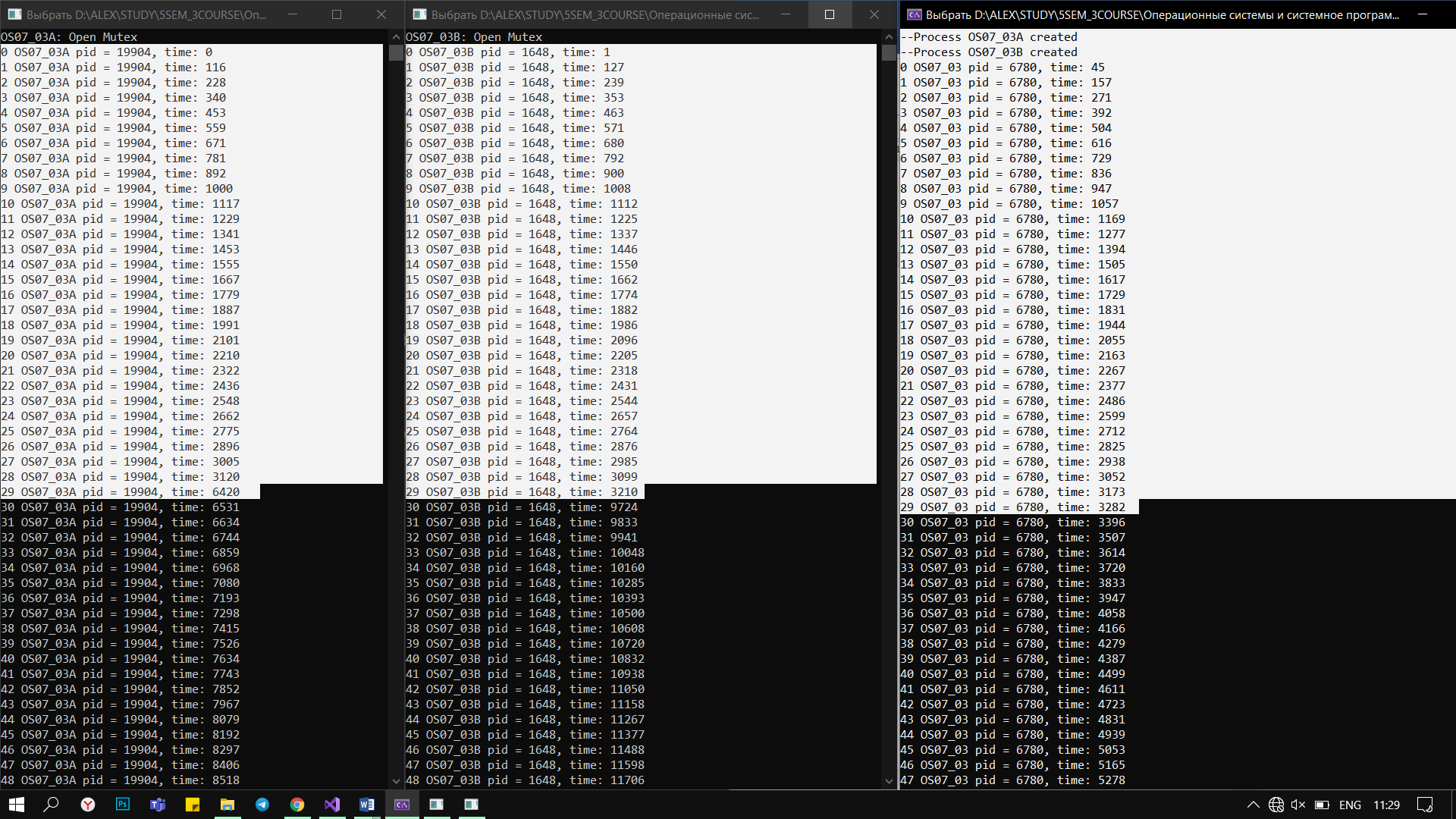
system("pause");

return 0;

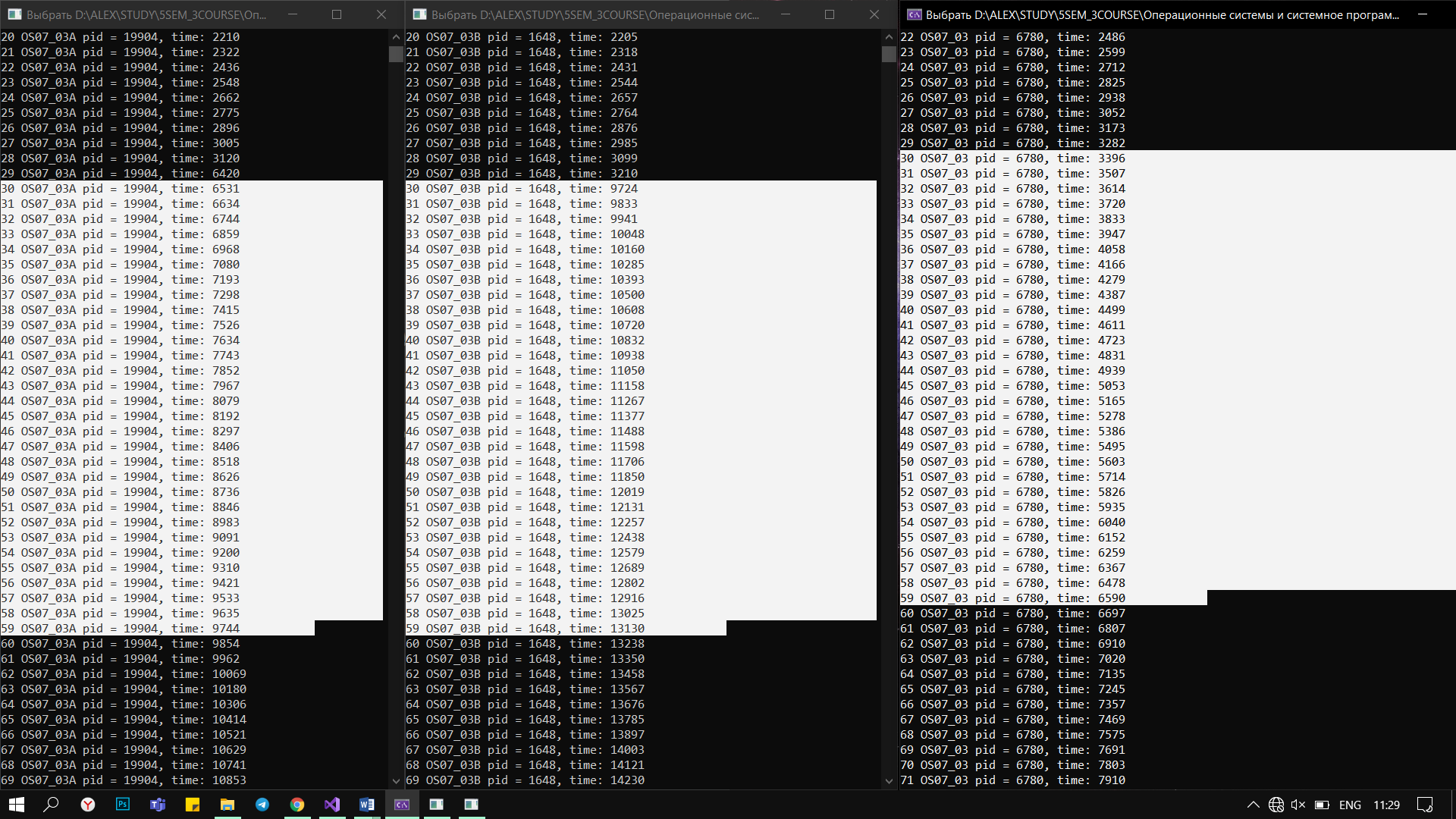
}

**OS07\_03B.cpp**

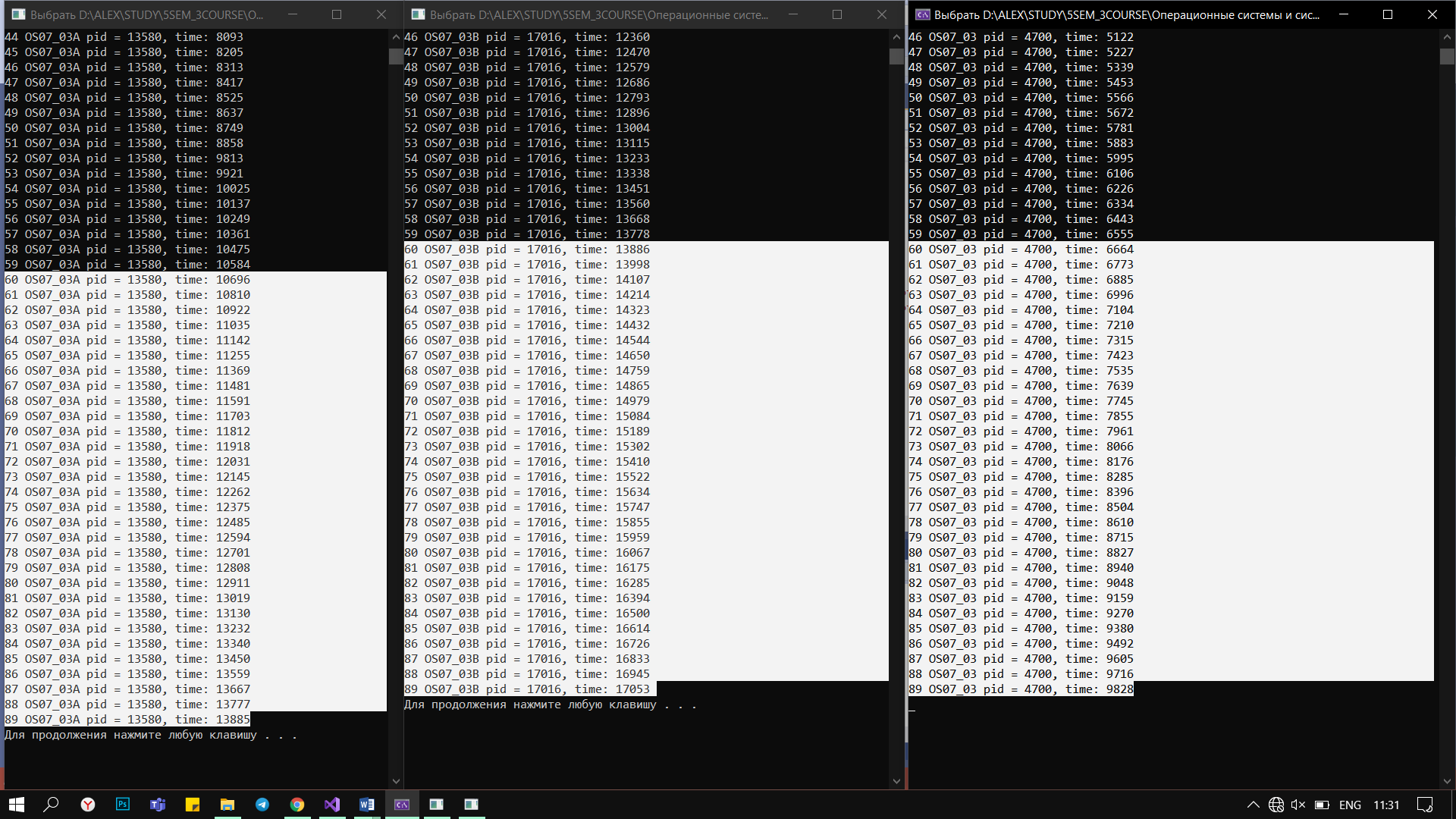
Выполнение программы до 30 итерации:



Выполнение программы с 30 до 60 итерации:



Выполнение программы с 60 итерации:



**Задание 4**

#include <Windows.h>

#include <iostream>

#include <ctime>

using namespace std;

int main() {

int start = clock();

DWORD pid = GetCurrentProcessId();

HANDLE hs = CreateSemaphore(NULL, 2, 3, L"Semaphore");

LPCWSTR an1 = L"D:\\ALEX\\STUDY\\5SEM\_3COURSE\\Операционные системы и системное программирование\\Готовые лабораторные работы\\Lab7\\OS07\_04\\Debug\\OS07\_04A.exe";

LPCWSTR an2 = L"D:\\ALEX\\STUDY\\5SEM\_3COURSE\\Операционные системы и системное программирование\\Готовые лабораторные работы\\Lab7\\OS07\_04\\Debug\\OS07\_04B.exe";

STARTUPINFO si1; STARTUPINFO si2;

PROCESS\_INFORMATION pi1; PROCESS\_INFORMATION pi2;

ZeroMemory(&si1, sizeof(STARTUPINFO)); ZeroMemory(&si2, sizeof(STARTUPINFO));

si1.cb = sizeof(STARTUPINFO); si2.cb = sizeof(STARTUPINFO);

if (CreateProcess(an1, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si1, &pi1)) {

cout << "--Process OS07\_04A created\n";

}

else {

cout << "--Process OS07\_04A not created\n";

}

if (CreateProcess(an2, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si2, &pi2)) {

cout << "--Process OS07\_04B created\n";

}

else {

cout << "--Process OS07\_04B not created\n";

}

LONG prevcount = 0;

for (int i = 0; i < 90; i++) {

if (i == 30) {

WaitForSingleObject(hs, INFINITE);

}

if (i == 60) {

ReleaseSemaphore(hs, 1, &prevcount);

}

cout << i << " OS07\_04 pid = " << pid << ", time: " << clock() - start << endl;

Sleep(100);

}

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

CloseHandle(hs);

CloseHandle(pi1.hProcess); CloseHandle(pi2.hProcess);

system("pause");

return 0;

}

**OS07\_04.cpp**

#include <Windows.h>

#include <iostream>

#include <ctime>

using namespace std;

int main() {

int start = clock();

DWORD pid = GetCurrentProcessId();

HANDLE hs = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, L"Semaphore");

hs == NULL ? cout << "OS07\_04A: Open Error Semaphore\n" : cout << "OS07\_04A: Open Semaphore\n";

for (int i = 0; i < 90; i++) {

if (i == 30) {

WaitForSingleObject(hs, INFINITE);

}

if (i == 60) {

ReleaseMutex(hs);

}

cout << i << " OS07\_04A pid = " << pid << ", time: " << clock() - start << endl;

Sleep(100);

}

CloseHandle(hs);

system("pause");

return 0;

}

**OS07\_04A.cpp**

#include <Windows.h>

#include <iostream>

#include <ctime>

using namespace std;

int main() {

int start = clock();

DWORD pid = GetCurrentProcessId();

HANDLE hs = OpenSemaphore(SEMAPHORE\_ALL\_ACCESS, FALSE, L"Semaphore");

hs == NULL ? cout << "OS07\_04B: Open Error Semaphore\n" : cout << "OS07\_04B: Open Semaphore\n";

for (int i = 0; i < 90; i++) {

if (i == 30) {

WaitForSingleObject(hs, INFINITE);

}

if (i == 60) {

ReleaseMutex(hs);

}

cout << i << " OS07\_04B pid = " << pid << ", time: " << clock() - start << endl;

Sleep(100);

}

CloseHandle(hs);

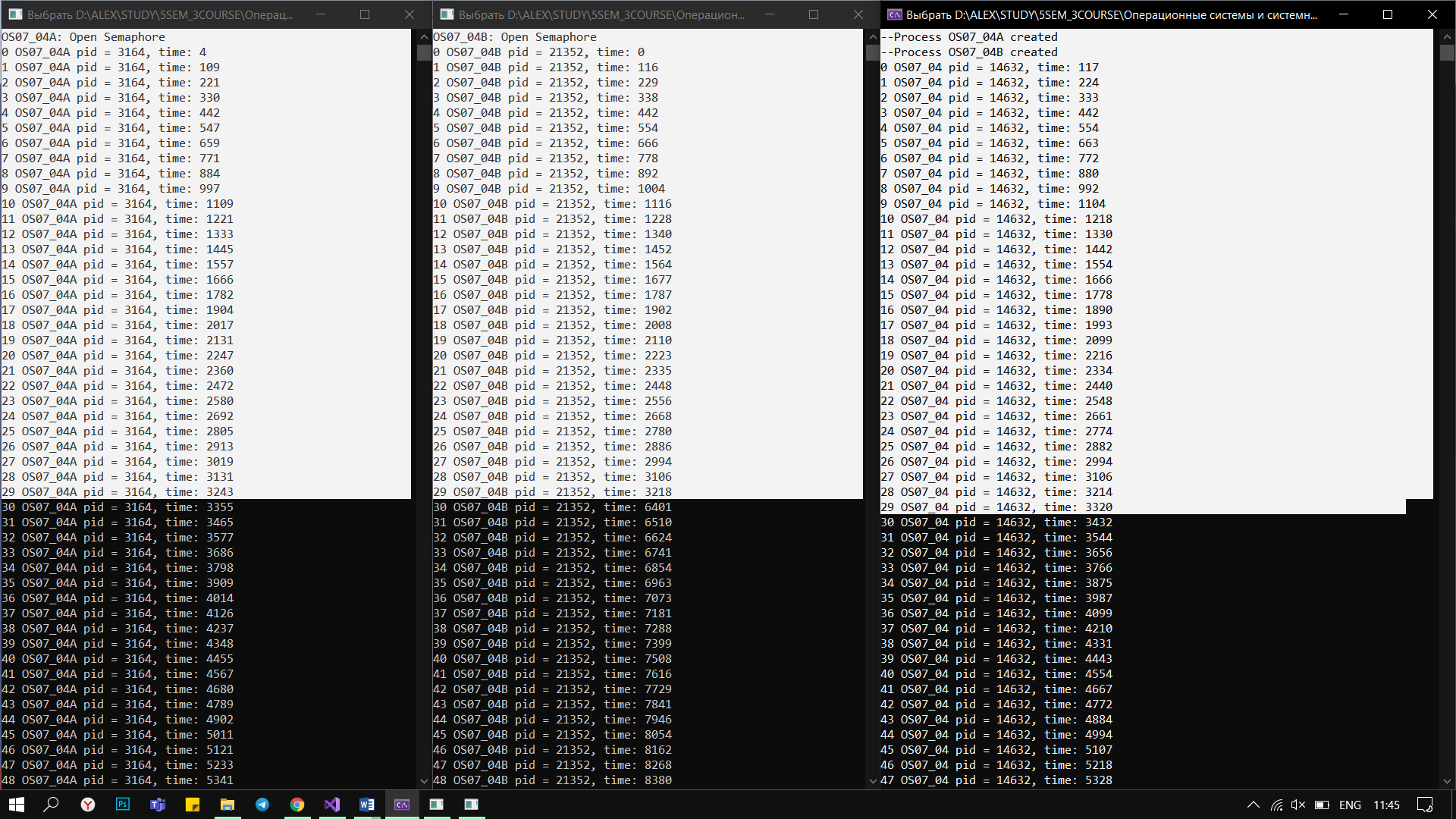
system("pause");

return 0;

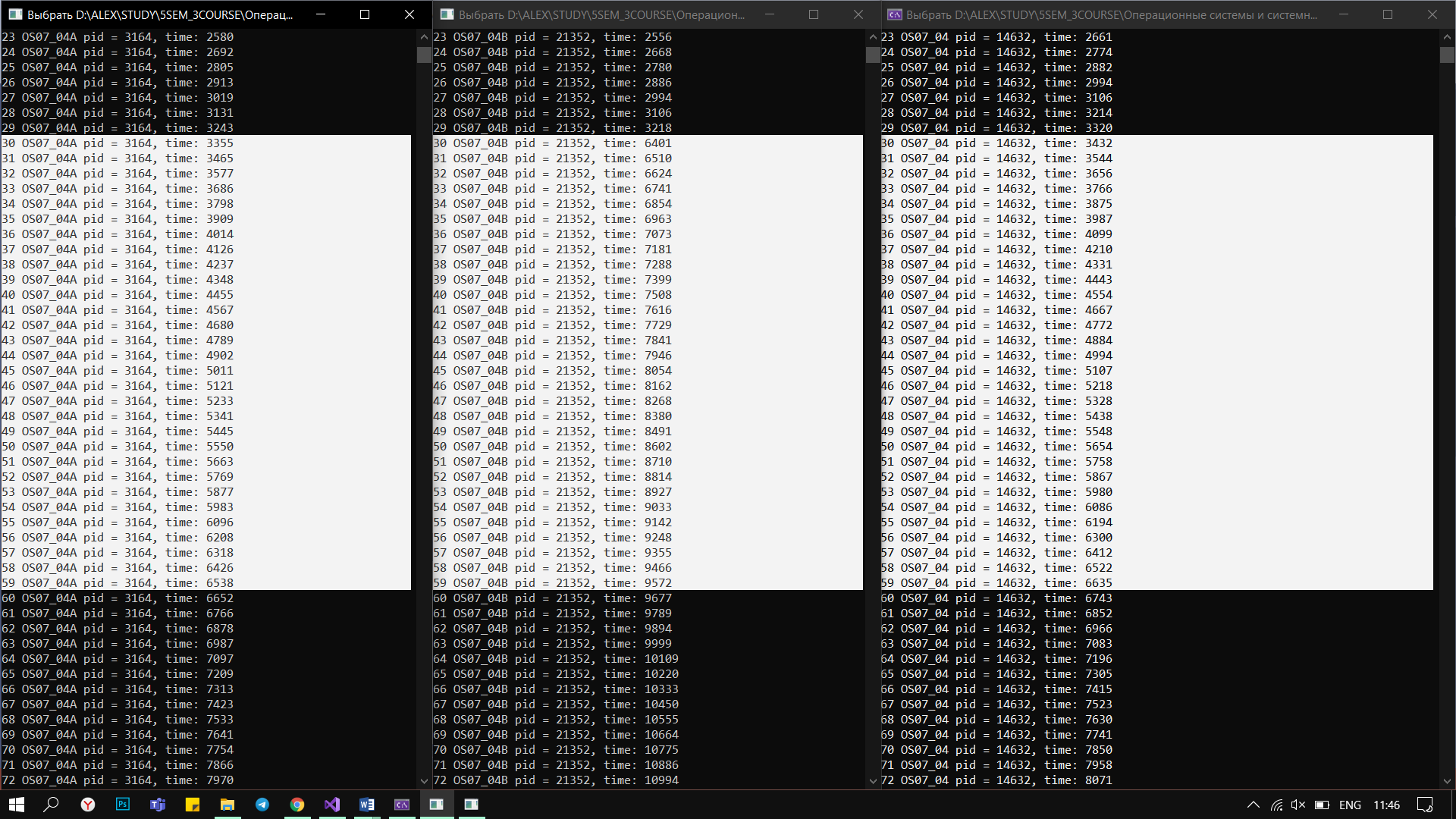
}

**OS07\_04B.cpp**

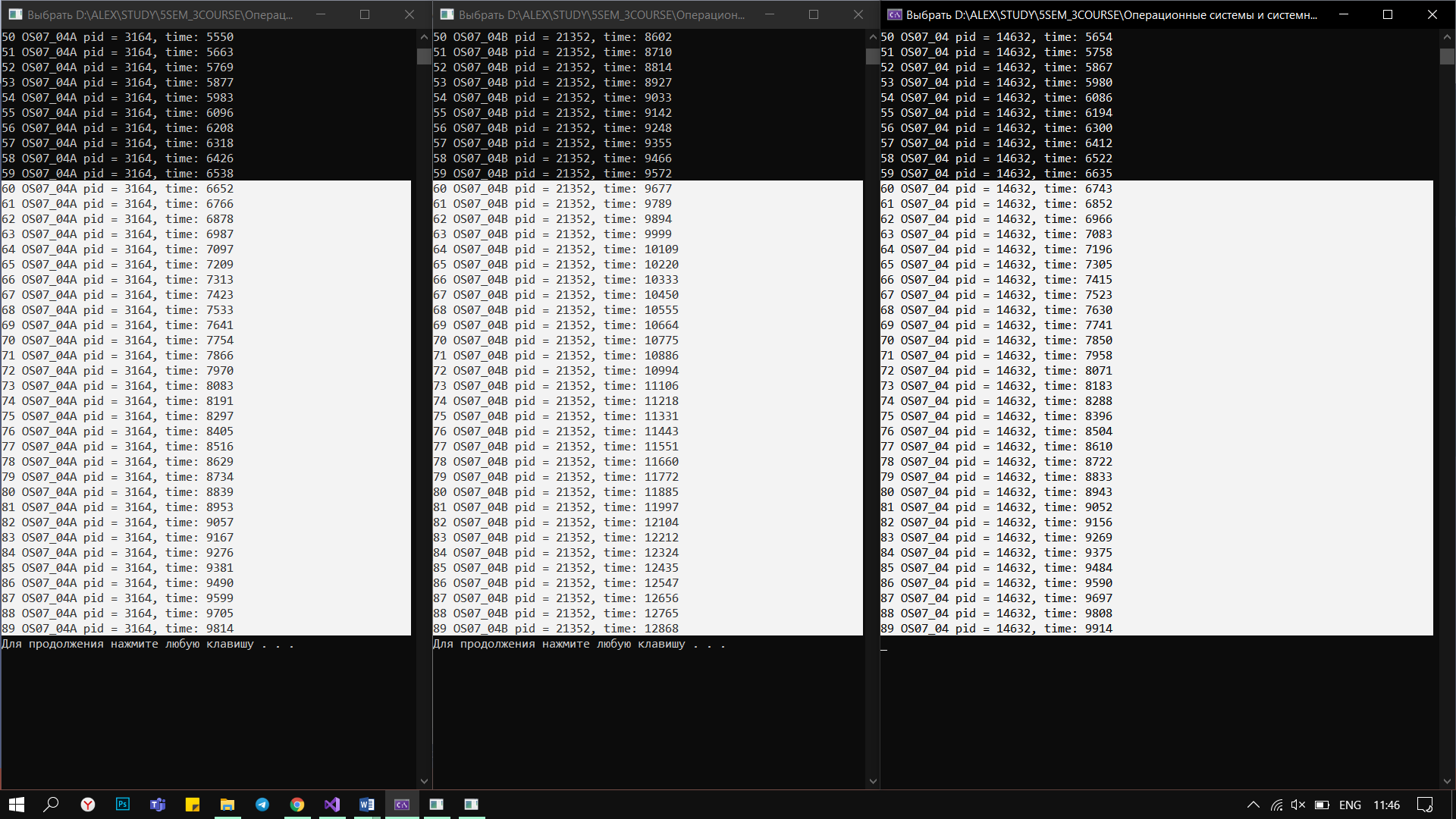
Выполнение программы до 30 итерации:



Выполнение программы с 30 до 60 итерации:



Выполнение программы с 60 итерации:



**Задание 5**

#include <Windows.h>

#include <iostream>

#include <ctime>

using namespace std;

int main() {

int start = clock();

DWORD pid = GetCurrentProcessId();

HANDLE he = CreateEvent(NULL, TRUE, FALSE, L"Event");

LPCWSTR an1 = L"D:\\ALEX\\STUDY\\5SEM\_3COURSE\\Операционные системы и системное программирование\\Готовые лабораторные работы\\Lab7\\OS07\_05\\Debug\\OS07\_05A.exe";

LPCWSTR an2 = L"D:\\ALEX\\STUDY\\5SEM\_3COURSE\\Операционные системы и системное программирование\\Готовые лабораторные работы\\Lab7\\OS07\_05\\Debug\\OS07\_05B.exe";

STARTUPINFO si1; STARTUPINFO si2;

PROCESS\_INFORMATION pi1; PROCESS\_INFORMATION pi2;

ZeroMemory(&si1, sizeof(STARTUPINFO)); ZeroMemory(&si2, sizeof(STARTUPINFO));

si1.cb = sizeof(STARTUPINFO); si2.cb = sizeof(STARTUPINFO);

if (CreateProcess(an1, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si1, &pi1)) {

cout << "--Process OS07\_05B created\n";

}

else {

cout << "--Process OS07\_05B not created\n";

}

if (CreateProcess(an2, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si2, &pi2)) {

cout << "--Process OS07\_05B created\n";

}

else {

cout << "--Process OS07\_05B not created\n";

}

LONG prevcount = 0;

for (int i = 0; i < 90; i++) {

if (i == 14) {

PulseEvent(he);

}

cout << i << " OS07\_05 pid = " << pid << ", time: " << clock() - start << endl;

Sleep(100);

}

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

CloseHandle(he);

CloseHandle(pi1.hProcess); CloseHandle(pi2.hProcess);

system("pause");

return 0;

}

**OS07\_05.cpp**

#include <Windows.h>

#include <iostream>

#include <ctime>

using namespace std;

int main() {

int start = clock();

DWORD pid = GetCurrentProcessId();

HANDLE he = OpenEvent(EVENT\_ALL\_ACCESS, FALSE, L"Event");

he == NULL ? cout << "OS07\_05A: Open Error Event\n" : cout << "OS07\_05A: Open Event\n";

WaitForSingleObject(he, INFINITE);

for (int i = 0; i < 90; i++) {

cout << i << " OS07\_05A pid = " << pid << ", time: " << clock() - start << endl;

Sleep(100);

}

CloseHandle(he);

system("pause");

return 0;

}

**OS07\_05A.cpp**

#include <Windows.h>

#include <iostream>

#include <ctime>

using namespace std;

int main() {

int start = clock();

DWORD pid = GetCurrentProcessId();

HANDLE he = OpenEvent(EVENT\_ALL\_ACCESS, FALSE, L"Event");

he == NULL ? cout << "OS07\_05B: Open Error Event\n" : cout << "OS07\_05B: Open Event\n";

WaitForSingleObject(he, INFINITE);

for (int i = 0; i < 90; i++) {

cout << i << " OS07\_05B pid = " << pid << ", time: " << clock() - start << endl;

Sleep(100);

}

CloseHandle(he);

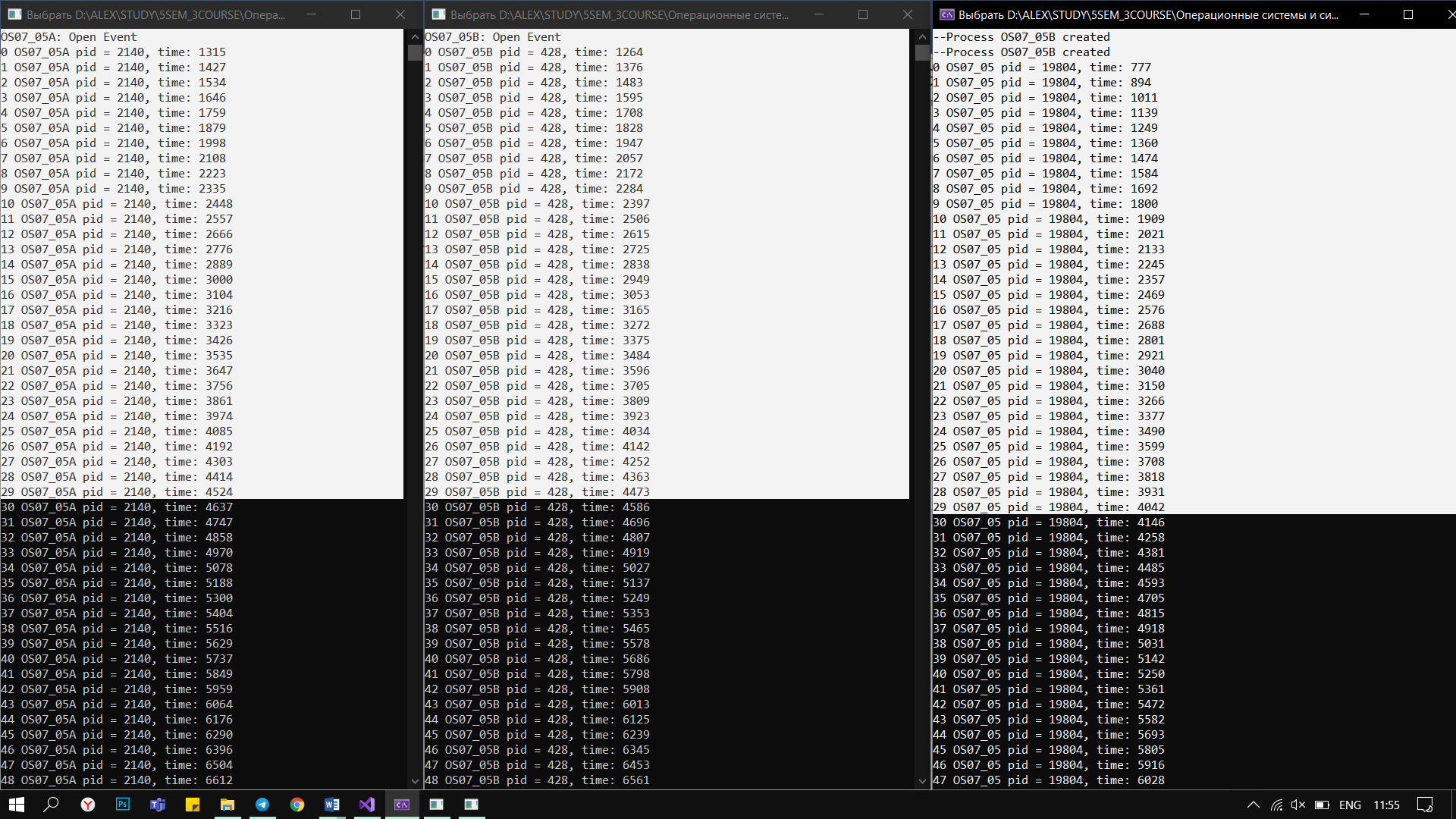
system("pause");

return 0;

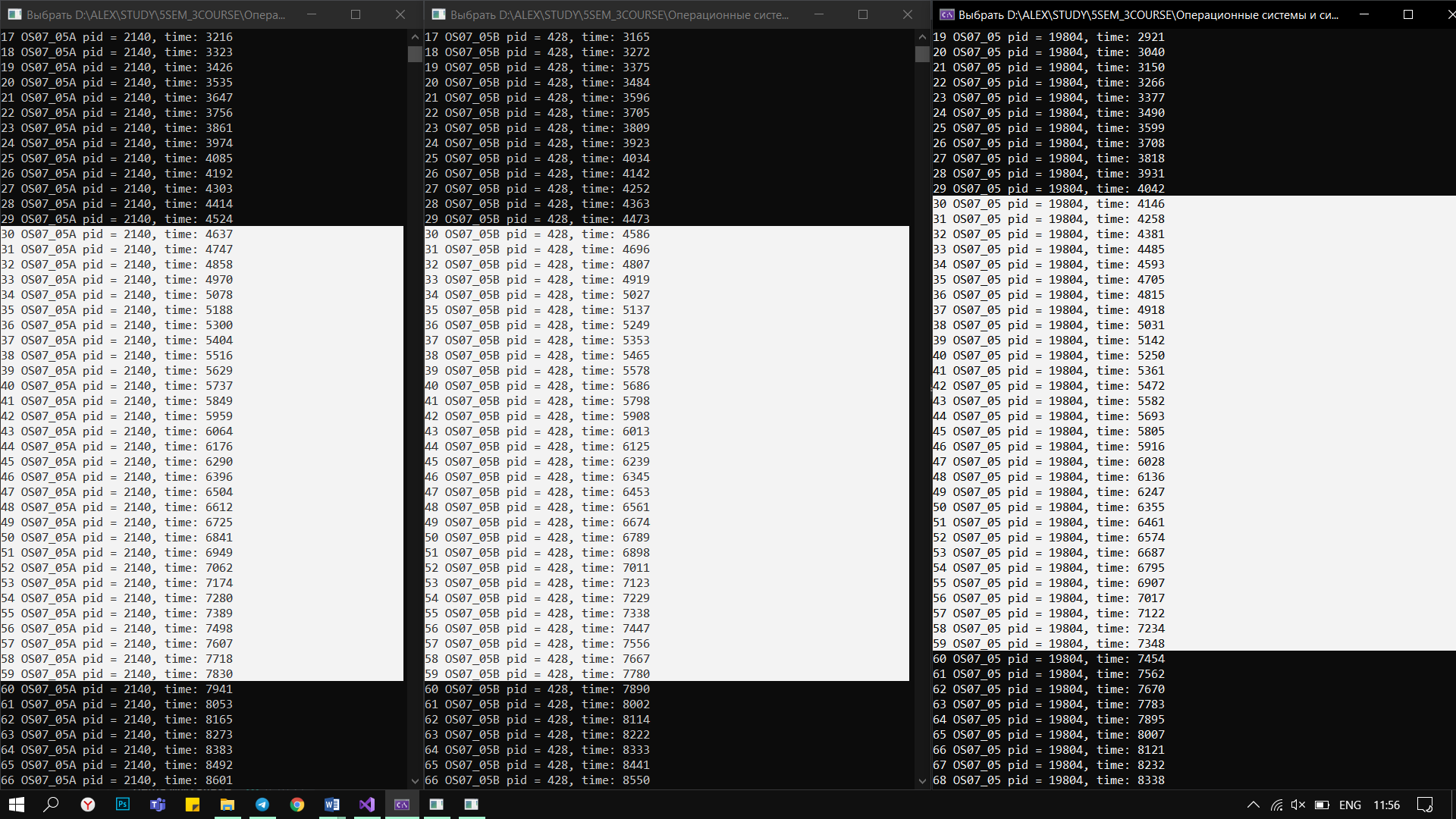
}

**OS07\_05B.cpp**

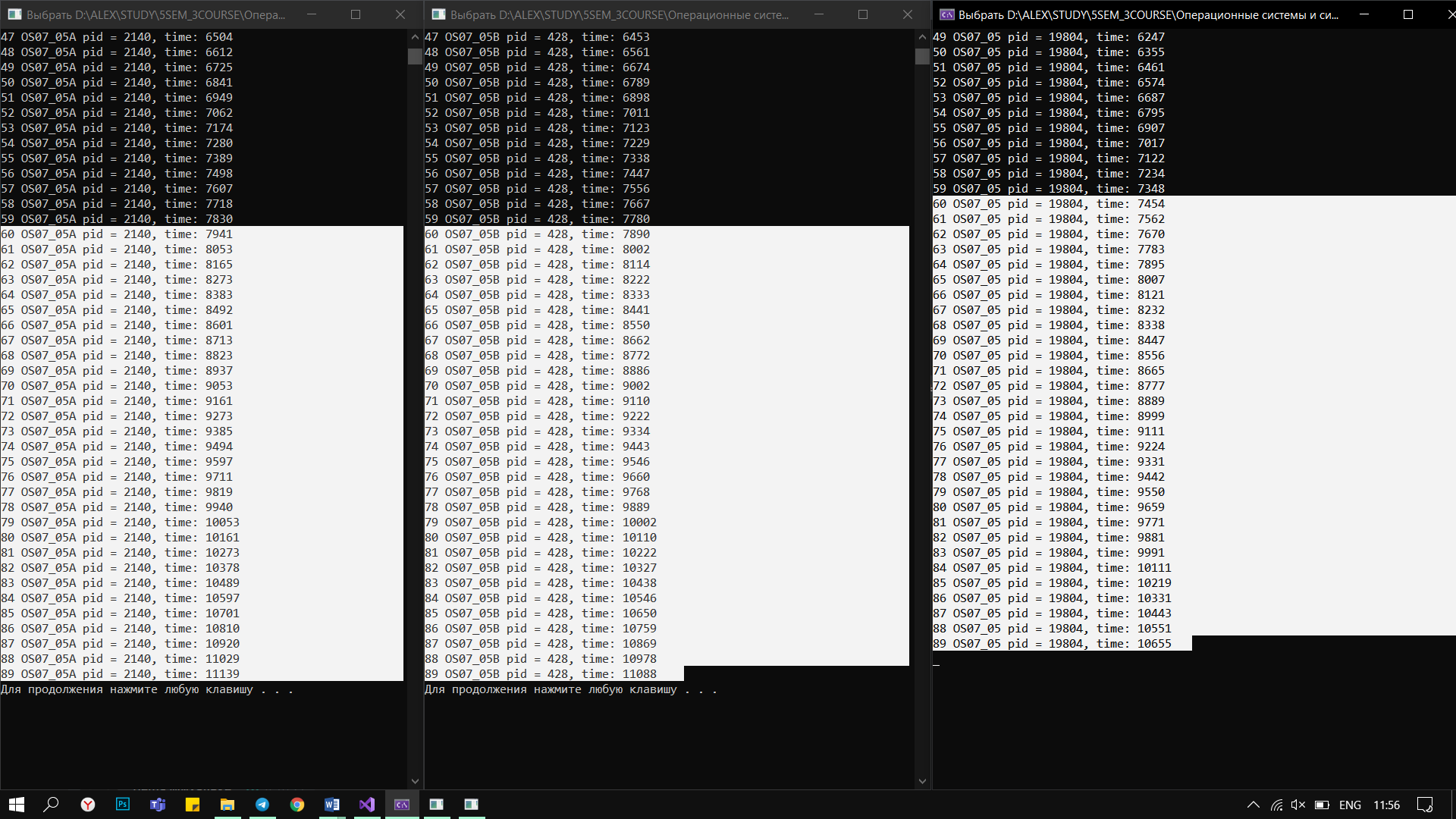
Выполнение программы до 15 итерации:



Выполнение программы с 30 до 60 итерации:



Выполнение программы с 60 итерации:

 **Задание 6**

#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

#include <pthread.h>

pthread\_mutex\_t mx;

void\* A(void\* arg)

{

printf("------------Thread A is started-----------\n");

for (int i = 1; i < 90; i++)

{

sleep(1); printf("Thread A: %d\n", i);

if (i == 30)

{

pthread\_mutex\_lock(&mx);

}

if (i == 60)

{

pthread\_mutex\_unlock(&mx);

}

}

printf("------------Thread A is finished-----------\n");

pthread\_exit("Child thread A");

}

void\* B(void\* arg)

{

printf("------------Thread B is started-----------\n");

for (int i = 1; i < 90; i++)

{

sleep(1); printf("Thread B: %d\n", i);

if (i == 30)

{

pthread\_mutex\_lock(&mx);

}

if (i == 60)

{

pthread\_mutex\_unlock(&mx);

}

}

printf("------------Thread B is finished-----------\n");

pthread\_exit("Child thread B");

}

int main()

{

pthread\_mutex\_init(&mx, NULL);

pthread\_t a\_th1, a\_th2;

void\* r\_th1, \*r\_th2;

pid\_t pid = getpid();

printf("Main: pid = %d \n", pid)

int res1 = pthread\_create(&a\_th1, NULL, A, NULL);

int res2 = pthread\_create(&a\_th2, NULL, B, NULL);

for (int i = 1; i < 90; i++)

{

sleep(1);

printf("Main: %d \n", i);

if (i == 30)

{

pthread\_mutex\_lock(&mx);

}

if (i == 60)

{

pthread\_mutex\_unlock(&mx);

}

}

int status1 = pthread\_join(&a\_th1, (void\*\*)&r\_th1);

int status2 = pthread\_join(&a\_th2, (void\*\*)&r\_th2);

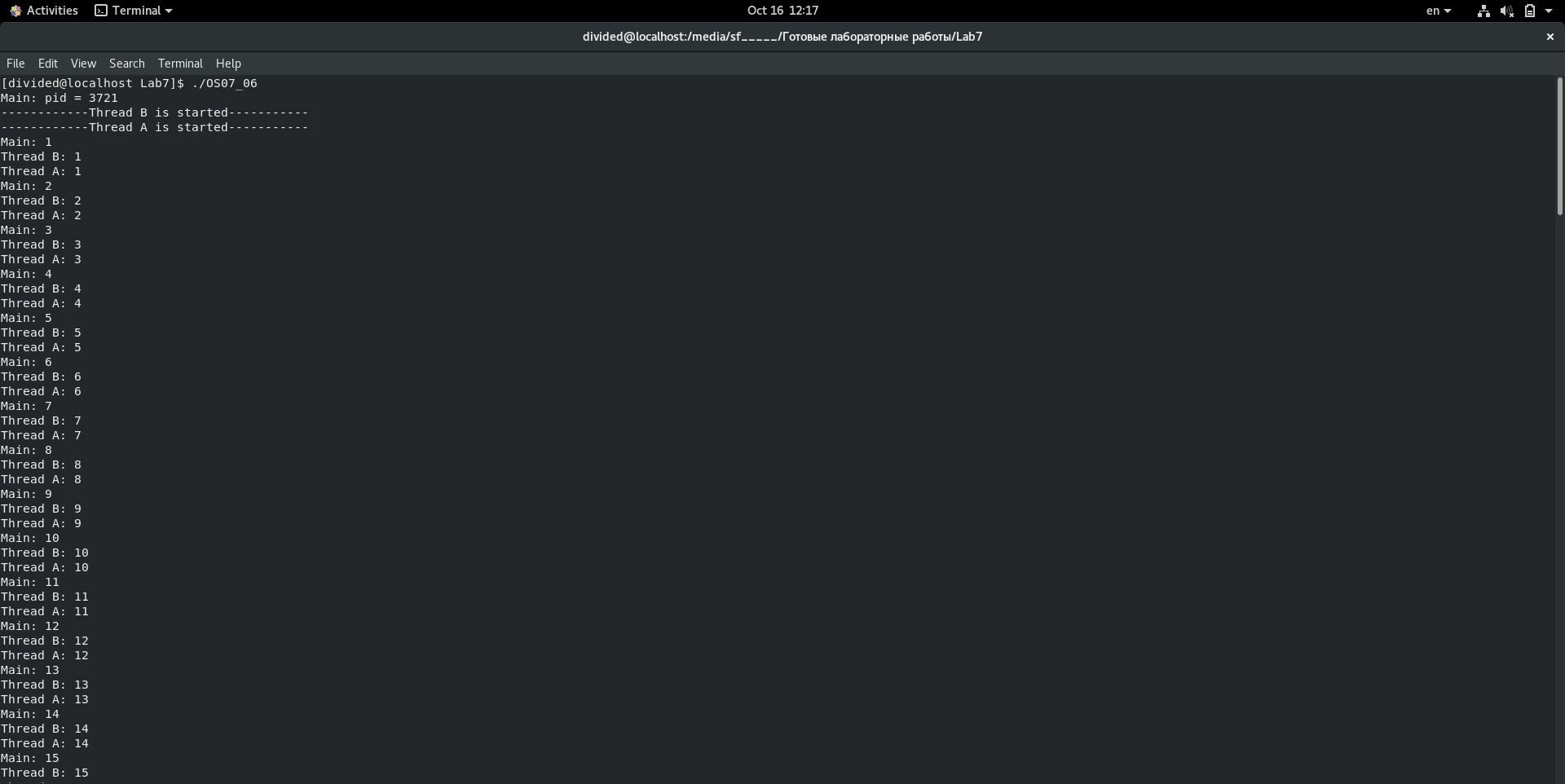
pthread\_mutex\_destroy(&mx);

exit(0);

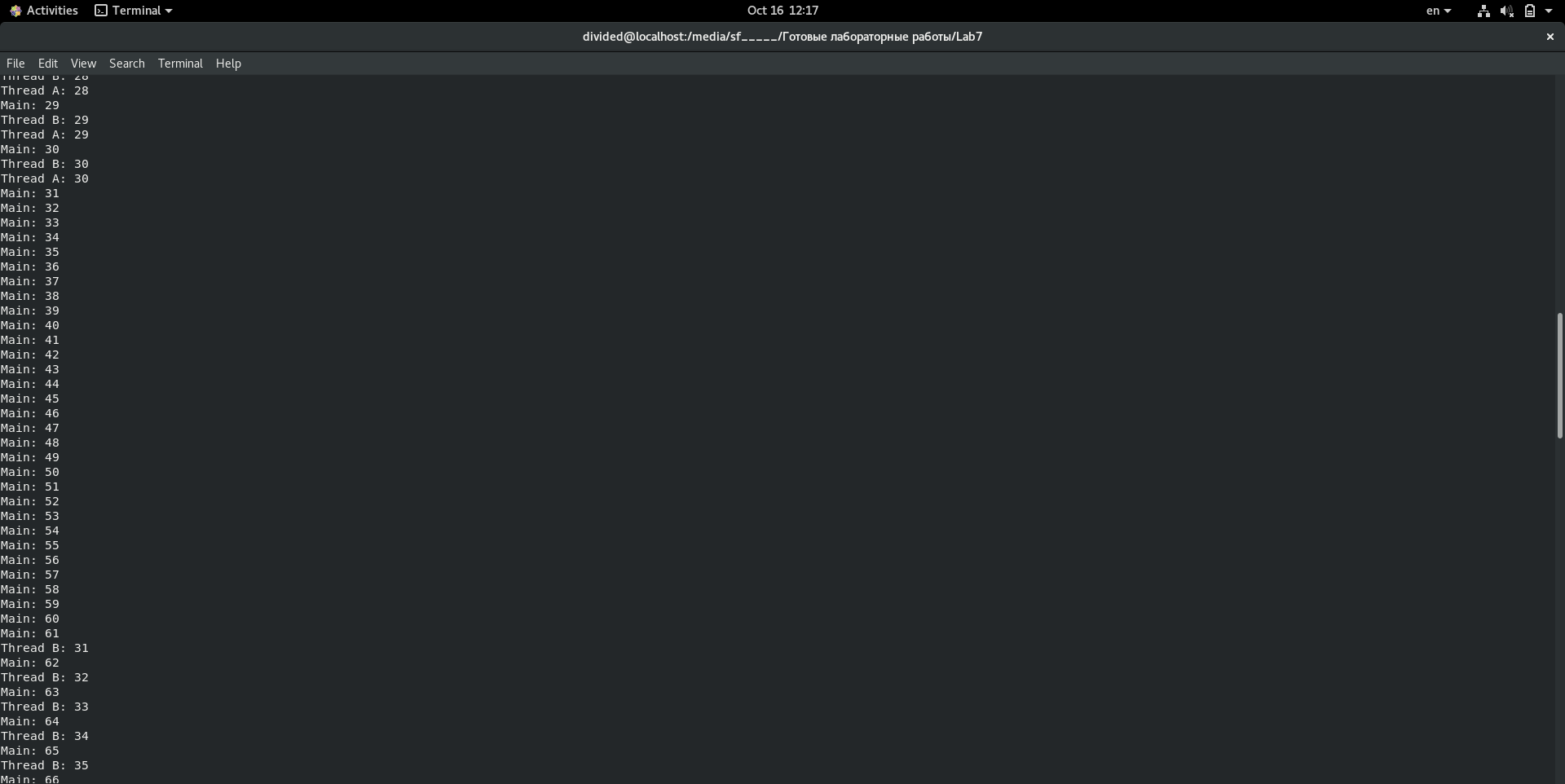
}

**OS07\_06.c**

Выполнение программы до 30 итерации:



Выполнение программы с 30 до 60 итерации:



Выполнение программы с 60 итерации:

