МИНИСТЕРСТВО ТРАНСПОРТА РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА (МИИТ)»

Кафедра ЦТУТП

**Отчёт**

**По лабораторной работе №3**  
по дисциплине «Операционные системы и системное программирование»

Тема: «СИНХРОНИЗАЦИЯ ПРОЦЕССОВ И ПОТОКОВ В ОС WINDOWS»

Выполнил: Кривоногов П. Н.

Группа: УИС-311

Преподаватель: доц. Варфоломеев В. А.

Борисенко Ф. А.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ВАРИАНТ ЗАДАНИЯ 2](#_Toc167645467)

[ТЕКСТ ПРОГРАММЫ 4](#_Toc167645468)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 15](#_Toc167645469)

[ВРЕМЕННЫЕ ДИАГРАММЫ ТРАСС ПОТОКОВ 16](#_Toc167645470)

[ВЫВОД 17](#_Toc167645471)

# ВАРИАНТ ЗАДАНИЯ

Таблица 1 – Варианты заданий

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Разделяемые переменные, (тип) | Операция 1 | | Операция 2 | |
| Клавиши (N) | Действия | Клавиши (N) | Действия |
| 9 | R (str) = “ABCDE” | ← | Циклический сдвиг символов влево | → | Циклический сдвиг символов вправо |

# ТЕКСТ ПРОГРАММЫ

#include <iostream>

#include <conio.h>

#include <windows.h>

#include <ctime>

#include <string>

#include <iomanip>

std::string R = "ABCDE";

clock\_t delayTime;

HANDLE hThread;

HANDLE semaphore;

DWORD threadID;

struct LogJournal

{

    int operationNumber;

    int pressedKeyCode;

    DWORD ThreadID;

    clock\_t enteringCriticalSectionTime;

    clock\_t leavingCriticalSectionTime;

    clock\_t operationStartTime;

    clock\_t operationEndTime;

    clock\_t waitingTime;

    std::string R;

    std::string Rc;

} logJournal[1000];

void delay(clock\_t delayTime)

{

    clock\_t start = clock();

    while (clock() - start < delayTime)

        ;

}

bool isKeyPressed()

{

    return (GetAsyncKeyState(VK\_LEFT) & 0x8000) || (GetAsyncKeyState(VK\_RIGHT) & 0x8000);

}

int getKeyCode()

{

    int currentKey = 0;

    if (GetAsyncKeyState(VK\_LEFT) & 0x8000)

    {

        currentKey = VK\_LEFT;

        while (GetAsyncKeyState(VK\_LEFT) & 0x8000)

        {

            Sleep(50);

        }

    }

    else if (GetAsyncKeyState(VK\_RIGHT) & 0x8000)

    {

        currentKey = VK\_RIGHT;

        while (GetAsyncKeyState(VK\_RIGHT) & 0x8000)

        {

            Sleep(50);

        }

    }

    return currentKey;

}

std::string shiftString(const std::string &str, bool isLeft)

{

    if (str.empty())

        return str;

    if (isLeft)

    {

        return str.substr(1) + str[0];

    }

    else

    {

        return str.back() + str.substr(0, str.size() - 1);

    }

}

DWORD WINAPI operationThreadProc(LPVOID param)

{

    WaitForSingleObject(semaphore, INFINITE);

    LogJournal \*logEntry = static\_cast<LogJournal \*>(param);

    logEntry->enteringCriticalSectionTime = clock();

    int currentKeyCode = logEntry->pressedKeyCode;

    bool isLeft = (currentKeyCode == VK\_LEFT);

    std::string buffer = R;

    delay(delayTime);

    buffer = shiftString(buffer, isLeft);

    delay(delayTime);

    R = buffer;

    logEntry->R = R;

    logEntry->operationEndTime = clock();

    logEntry->waitingTime = logEntry->operationEndTime - logEntry->operationStartTime;

    logEntry->leavingCriticalSectionTime = clock();

    ReleaseSemaphore(semaphore, 1, NULL);

    return 0;

}

void displayLogJournalHeader()

{

    std::cout << std::setw(5) << "No" << " | "

              << std::setw(4) << "Key" << " | "

              << std::setw(10) << "Thread ID" << " | "

              << std::setw(11) << "Start Time" << " | "

              << std::setw(9) << "End Time" << " | "

              << std::setw(10) << "Enter CS" << " | "

              << std::setw(10) << "Leave CS" << " | "

              << std::setw(10) << "Wait Time" << " | "

              << std::setw(7) << "R" << " | "

              << std::setw(7) << "Rc" << std::endl;

    std::cout << "--------------------------------------------------------------------------------------------------------------------" << std::endl;

}

void displayLogJournalEntry(const LogJournal &entry)

{

    std::cout << std::setw(5) << entry.operationNumber << " | "

              << std::setw(4) << entry.pressedKeyCode << " | "

              << std::setw(10) << entry.ThreadID << " | "

              << std::setw(11) << entry.operationStartTime << " | "

              << std::setw(9) << entry.operationEndTime << " | "

              << std::setw(10) << entry.enteringCriticalSectionTime << " | "

              << std::setw(10) << entry.leavingCriticalSectionTime << " | "

              << std::setw(10) << entry.waitingTime << " | "

              << std::setw(7) << entry.R << " | "

              << std::setw(7) << entry.Rc << std::endl;

}

int main(int argc, char \*argv[])

{

    SetConsoleCP(1251);

    SetConsoleOutputCP(1251);

    if (argc != 2)

    {

        printf("Работа программы: %s <время задержки>\n", argv[0]);

        return 1;

    }

    delayTime = atoi(argv[1]);

    semaphore = CreateSemaphore(NULL, 1, 1, NULL);

    int operationIndex = 0;

    int displayIndex = 1;

    bool running = true;

    std::string controlString = R;

    logJournal[1].operationEndTime = 0;

    displayLogJournalHeader();

    while (running && operationIndex < 1000)

    {

        if (isKeyPressed())

        {

            int currentKeyCode = getKeyCode();

            operationIndex++;

            logJournal[operationIndex].operationNumber = operationIndex;

            logJournal[operationIndex].pressedKeyCode = currentKeyCode;

            logJournal[operationIndex].operationEndTime = 0;

            logJournal[operationIndex].R = "";

            logJournal[operationIndex].operationStartTime = clock();

            hThread = CreateThread(NULL, 0, operationThreadProc, &logJournal[operationIndex], CREATE\_SUSPENDED, &threadID);

            bool isLeft = (currentKeyCode == VK\_LEFT);

            controlString = shiftString(controlString, isLeft);

            logJournal[operationIndex].ThreadID = threadID;

            logJournal[operationIndex].Rc = controlString;

            ResumeThread(hThread);

        }

        if (logJournal[displayIndex].operationEndTime != 0)

        {

            displayLogJournalEntry(logJournal[displayIndex]);

            displayIndex++;

        }

    }

    CloseHandle(semaphore);

    return 0;

}

# РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

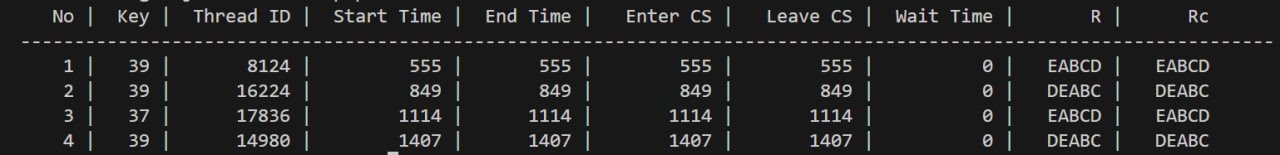


Рисунок 1 – Работа при задержке 0мс

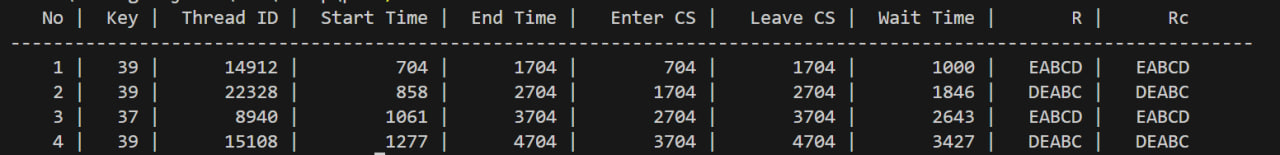
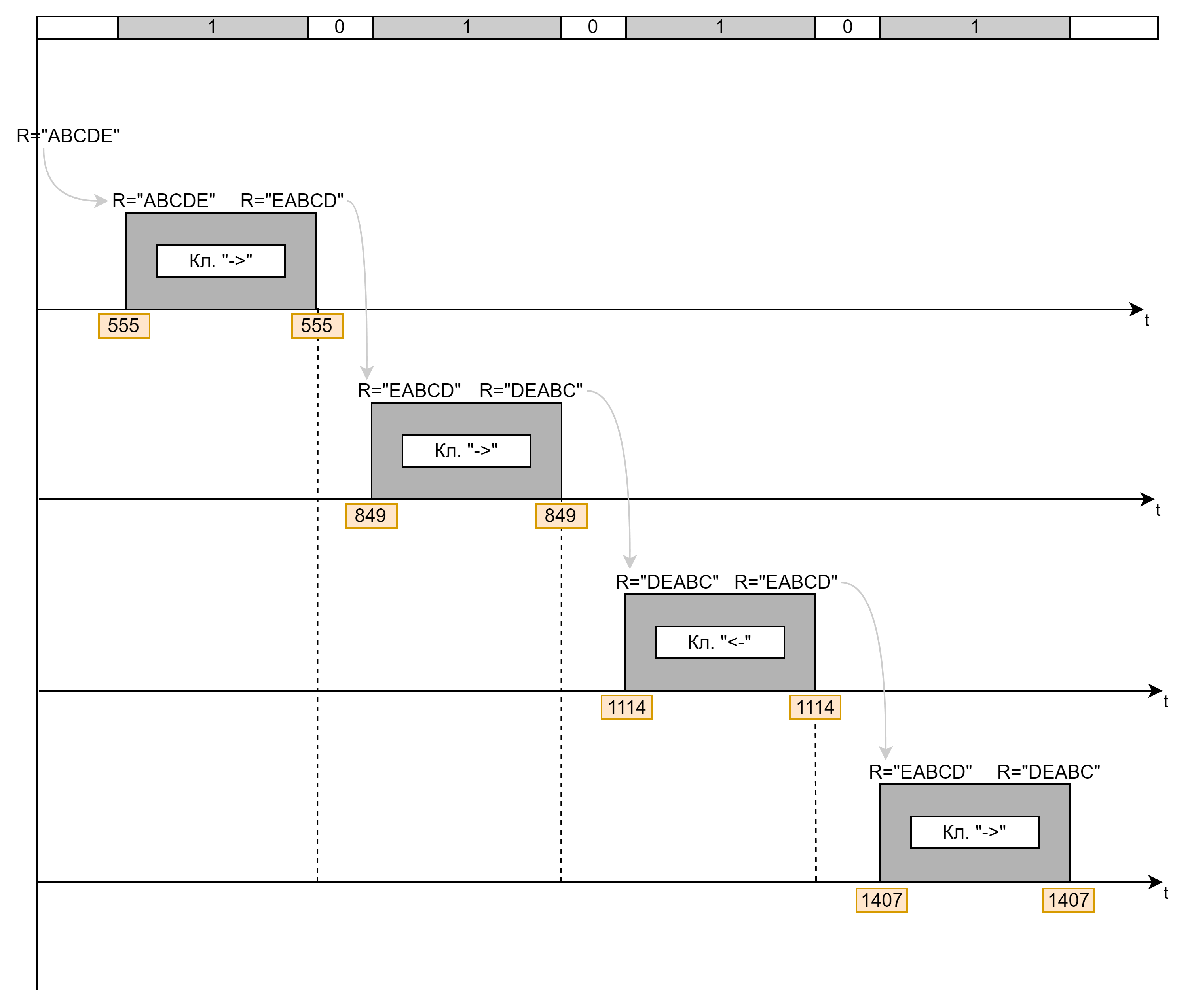
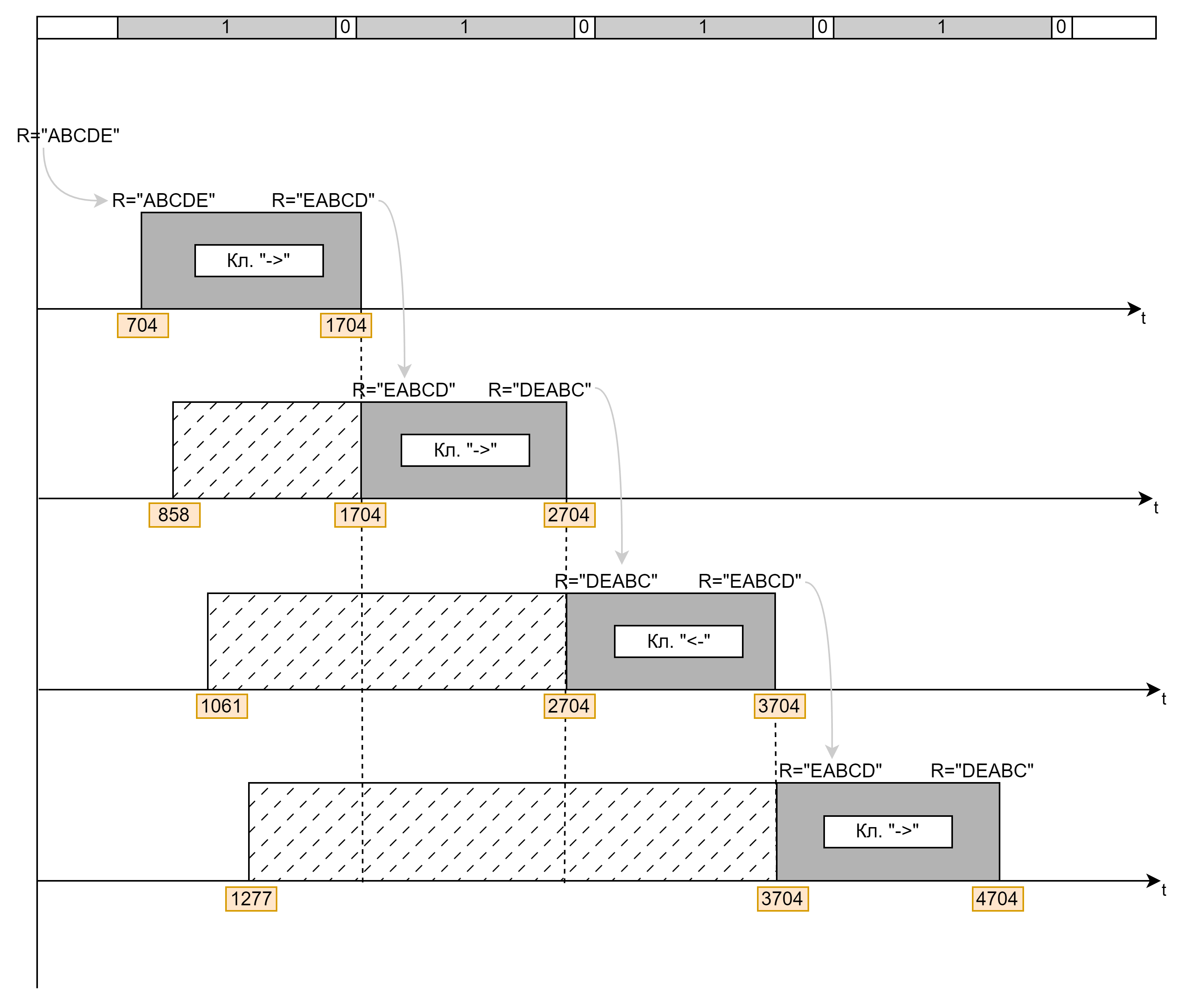


Рисунок 2 – Работа при задержке 500мс

# ВРЕМЕННЫЕ ДИАГРАММЫ ТРАСС ПОТОКОВ





# ВЫВОД

В ходе выполнения лабораторной работы была разработана и исследована программа на языке C++, демонстрирующая использование семафоров для синхронизации потоков. Целью работы было освоение методов синхронизации и управление критическими секциями для предотвращения конкурентного доступа к общим ресурсам.

Программа успешно использует семафоры для управления доступом к критической секции, обеспечивая, что только один поток имеет доступ к общей строке R в любое время. Это позволяет избежать условий гонки и гарантировать корректность изменения строки R в многопоточном окружении.

Создание семафора CreateSemaphore с начальным и максимальным значением 1 гарантирует, что только один поток может войти в критическую секцию. Ожидание семафора WaitForSingleObject и его освобождение ReleaseSemaphore реализуют механизм блокировки и разблокировки критической секции.

Потоки создаются функцией CreateThread и приостанавливаются до тех пор, пока не будут запущены с помощью ResumeThread. В каждом потоке осуществляется задержка для имитации времени выполнения операций, что демонстрирует влияние синхронизации на производительность приложения.

Программа ведет детальный журнал операций, фиксируя время начала и конца выполнения операций, время ожидания входа в критическую секцию, а также идентификаторы потоков и состояние строки R до и после изменения. Этот журнал позволяет анализировать производительность и эффективность синхронизации, а также выявлять потенциальные проблемы в многопоточном исполнении.

Для проверки корректности работы программы используется контрольная строка Rc, которая обновляется аналогично основной строке R, но без задержек и блокировок. Это позволяет сравнить результаты и убедиться в правильности выполнения операций.

Работа продемонстрировала важность и эффективность использования семафоров для синхронизации потоков в многопоточном приложении. Применение семафоров позволило избежать конфликтов при одновременном доступе к общим ресурсам и обеспечило корректное выполнение всех операций.

Проведение подобных лабораторных работ является важным этапом в обучении разработке многопоточных приложений, так как позволяет на практике освоить методы и инструменты, необходимые для создания надежного и эффективного программного обеспечения.