Белорусский государственный технологический университет
Факультет информационных технологий
Кафедра программной инженерии

Лабораторная работа 7

По дисциплине «Операционные системы»

На тему «Компьютерное время»

Выполнил:
Студент 3 курса 9 группы
Павлович Ян Андреевич
Преподаватель: Савельева М.Г.

Минск, 2025

Введение

Целью данной лабораторной работы является освоение практических приемов работы с компьютерным временем в операционных системах Windows и Linux, а также изучение механизмов измерения временных интервалов, использования таймеров и управления процессами на уровне системных API.

В ходе выполнения работы требуется разработать набор консольных приложений для платформ Windows и Linux, демонстрирующих основные подходы к получению и форматированию системного времени, измерению времени выполнения вычислительных циклов, применению ожидающих таймеров и организации взаимодействия между процессами. Реализуемые приложения должны наглядно показывать использование системных вызовов и специализированных интерфейсов операционных систем для работы со временем и процессами.

Для операционной системы Windows необходимо реализовать получение и представление локального времени с использованием функций WinAPI, измерение длительности работы циклов с применением высокоточных счетчиков, настройку ожидающих таймеров на основе объектов ядра, а также запуск дочерних процессов и контроль их жизненного цикла с ожиданием завершения. В среде Linux аналогичные задачи должны быть решены с использованием POSIX-совместимых функций, включая получение и форматирование времени, измерение процессорного и реального времени, реализацию ожидающих таймеров, создание дочерних процессов и синхронизацию с их завершением.

Используемые инструменты:

- Parallels Desktop
- Командная оболочка *cmd*
- Process Explorer
- компилятор *g++*
- среда разработки *Visual Studio*
- справочная документация по WinAPI и POSIX

1 Windows

1.1 Приложение Lab-07a

Данное приложение предназначено для вывода в консоль текущих значений локальной даты и времени в формате YYYY-MM-DDThh:mm:ss±hh (например, 2005-08-09T18:31:42+03). Особенностью задания является требование самостоятельного вычисления смещения часового пояса относительно универсального времени без использования специализированных функций для определения часового пояса.

Для получения временных данных применяются функции работы с системным и локальным временем, предоставляемые WinAPI. Использование типов данных и функций стандартных библиотек языка C для работы со временем в рамках данного задания не допускается.

```
C:\Lab-07\x64\Debug>Lab-07a.exe
2025-12-26T17:42:47+03
```

Рисунок 1.1 – Результат работы Lab-07a

На первом этапе было создано консольное приложение, в которое был добавлен программный код, приведенный в листинге А. После компиляции и запуска программы был получен результат, представленный на рисунке 1.1.

В процессе работы приложения были получены значения системного и локального времени, на основе которых вычислено смещение часового пояса. Далее результаты были отформатированы в соответствии с заданным шаблоном и выведены в консоль.

1.2 Приложение Lab-07b

Приложение реализует непрерывный вычислительный цикл без использования искусственных задержек, в рамках которого осуществляется подсчет количества выполненных итераций. Основной задачей программы является измерение производительности цикла во времени с использованием средств работы с системным временем.

В процессе выполнения приложение выводит текущее значение счетчика итераций по истечении 5 и 10 секунд с момента запуска. По достижении 15 секунд работы программа корректно завершает выполнение, предварительно выводя итоговое количество выполненных итераций.

```
C:\Lab-07\x64\Debug>Lab-07b.exe
Iterations after 5 seconds: 244270211
Iterations after 10 seconds: 499088915
Total iterations after 15 seconds: 760519203
```

Рисунок 1.2 – Результат работы Lab-07b

Для выполнения задания было создано консольное приложение, программный код которого приведен в листинге Б. После компиляции и запуска приложения результат его работы представлен на рисунке 1.2.

В ходе работы программы используются таймеры для фиксации момента старта и определения прошедшего времени. Через заданные интервалы осуществляется вывод текущего значения счетчика, а по достижении контрольного времени выполняется вывод итогового результата и завершение приложения.

1.3 Приложение Lab-07c

В данном задании реализуется приложение, выполняющее цикл подсчета количества итераций с периодическим выводом промежуточных результатов. Значение счетчика выводится в консоль через каждые 3 секунды работы программы. По истечении 15 секунд приложение завершает выполнение и отображает итоговое количество выполненных итераций.

Ключевым ограничением является использование исключительно ожидающих таймеров для организации временных интервалов. Получаемые значения счетчика не должны быть малыми (1–5), так как корректная реализация предполагает интенсивное выполнение цикла, при котором количество итераций измеряется десятками или сотнями миллионов, аналогично предыдущему заданию.

```
C:\Lab-07\x64\Debug>Lab-07c.exe
Elapsed 3 seconds, iterations in period: 6874279
Elapsed 6 seconds, iterations in period: 3922275
Elapsed 9 seconds, iterations in period: 3318412
Elapsed 12 seconds, iterations in period: 3362014
Elapsed 15 seconds, iterations in period: 3283055
Total accumulated iterations (15 seconds): 20760035
```

Рисунок 1.3 – Результат работы Lab-07c

Для выполнения работы было создано консольное приложение, программный код которого приведен в листинге В. После запуска программы результат её работы представлен на рисунке 1.3.

В ходе выполнения приложение с использованием ожидающих таймеров осуществляет периодический вывод количества выполненных итераций с интервалом в 3 секунды, а по завершении заданного времени формирует и выводит итоговое значение счётчика.

1.4 Приложение Lab-07x

Данное приложение предназначено для вычисления и последовательного вывода в консоль ряда простых чисел с их нумерацией. По завершении работы программа дополнительно выводит общее время своего выполнения.

В рамках выполнения задания было создано консольное приложение, исходный код которого приведен в листинге Г. После запуска программы результат её работы представлен на рисунках 1.4-1.5.

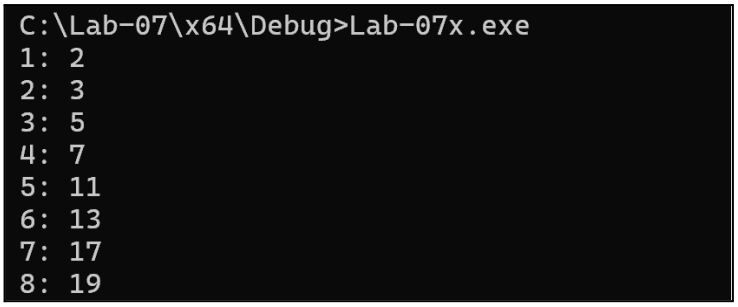

Рисунок 1.4 – Результат работы Lab-07x

В приложении реализована функция проверки чисел на простоту, которая используется в основном вычислительном цикле. Для нумерации найденных простых чисел применяется отдельный счётчик. Вывод простых чисел осуществляется в течение заданного интервала времени, который передаётся в программу через параметры командной строки.

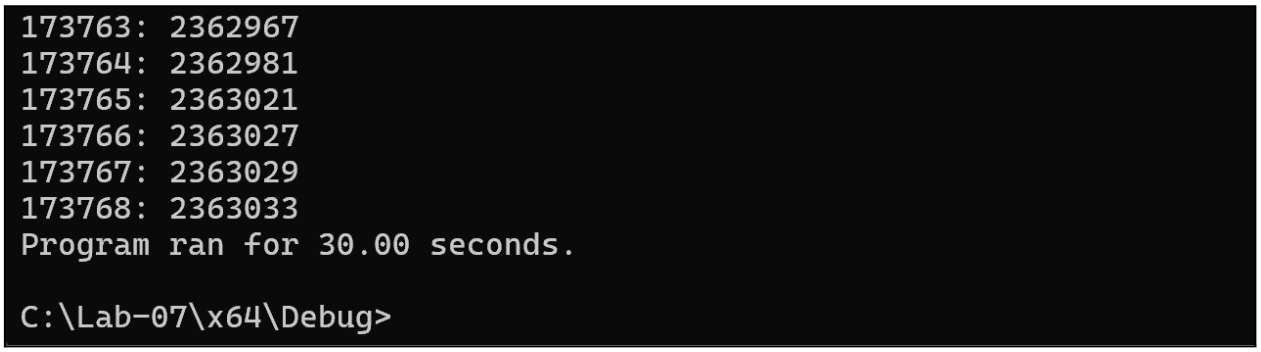

Рисунок 1.5 – Результат работы Lab-07x

При отсутствии входных параметров используется значение по умолчанию, равное 30 секундам. Контроль продолжительности работы приложения, как и в предыдущих заданиях, осуществляется с помощью таймеров, позволяющих определить фактическое время выполнения программы.

1.5 Приложение Lab-07d

Данное приложение реализует управление несколькими процессами и демонстрирует механизм ожидания завершения дочерних процессов. Родительский процесс запускает два дочерних приложения, созданных на основе программы Lab-07x, каждое из которых выполняется в течение заданного промежутка времени.

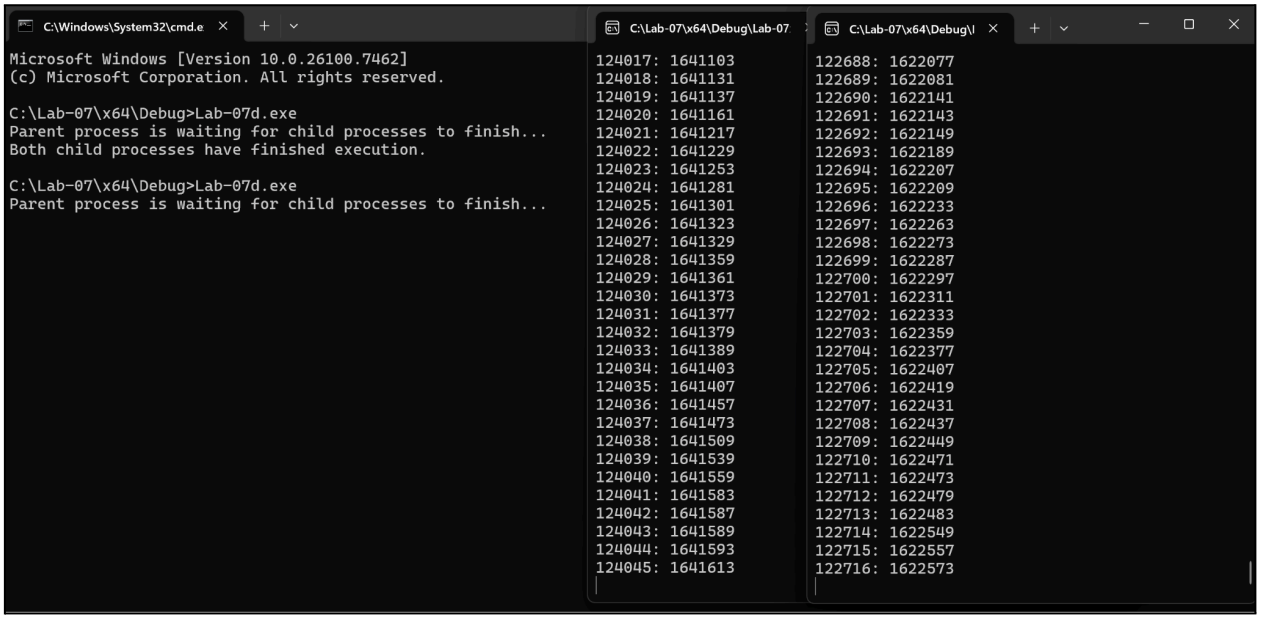

Рисунок 1.6 – Результат работы Lab-07d

Первый дочерний процесс работает одну минуту и корректно завершает выполнение по истечении установленного времени. Второй дочерний процесс функционирует в течение двух минут, после чего также завершается штатным образом. Родительский процесс при этом переходит в режим ожидания и продолжает выполнение только после завершения обоих дочерних процессов.

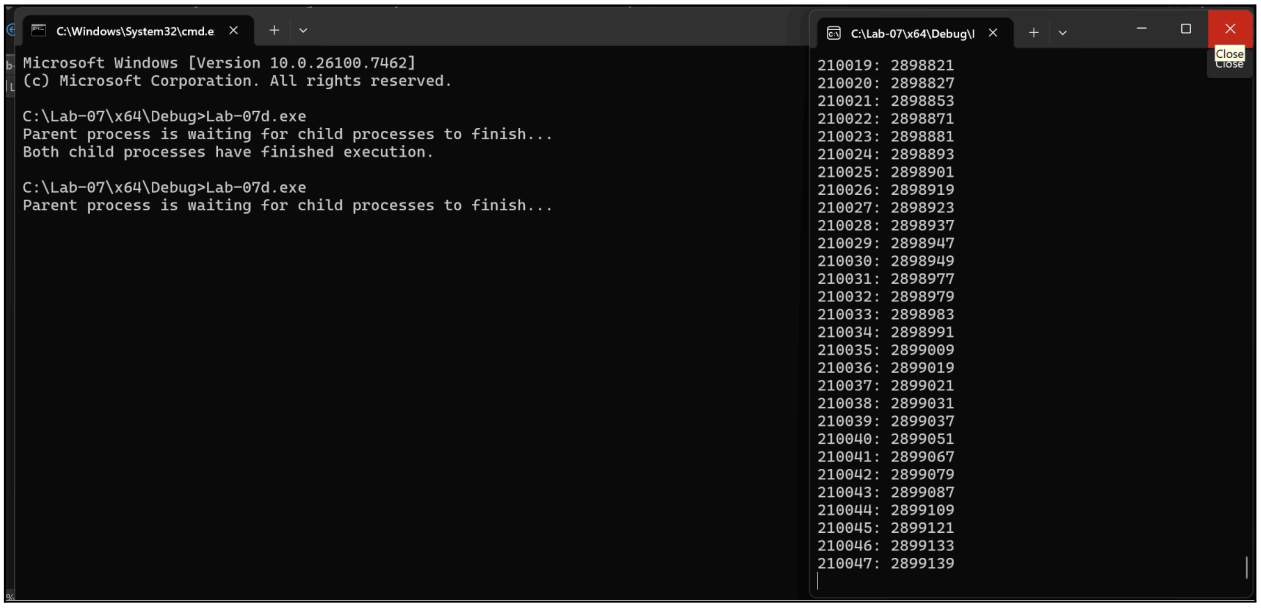

Рисунок 1.7 – Результат работы Lab-07d

Для выполнения задания было создано консольное приложение, программный код которого приведен в листинге Д. После запуска приложения на рисунке 1.6 показана одновременная работа двух дочерних процессов, в то время как родительский процесс ожидает их завершения. На рисунке 1.7 отображен момент, когда первый дочерний процесс уже завершил

работу, однако родительский процесс продолжает ожидание окончания выполнения второго дочернего процесса. Окончательное завершение работы приложения, после завершения обоих дочерних процессов и самого родительского процесса, представлено на рисунке 1.8.

```
C:\Lab-07\x64\Debug>Lab-07d.exe
Parent process is waiting for child processes to finish...
Both child processes have finished execution.
```

Рисунок 1.8 – Результат работы Lab-07d

Таким образом, разработанное приложение демонстрирует запуск нескольких дочерних процессов с различным временем выполнения, передачу параметров через командную строку, а также корректную синхронизацию и ожидание их завершения со стороны родительского процесса.

2 Linux

2.1 Приложение Lab-07a

Данное приложение является функциональным аналогом версии для операционной системы Windows, однако при его реализации используются функции, определённые стандартом POSIX. Программа обеспечивает получение и вывод текущей локальной даты и времени в требуемом формате с самостоятельным вычислением смещения часового пояса.

Для выполнения задания был разработан и скомпилирован программный код, приведенный в листинге Е. После запуска приложения результат его работы представлен на рисунке 2.1.

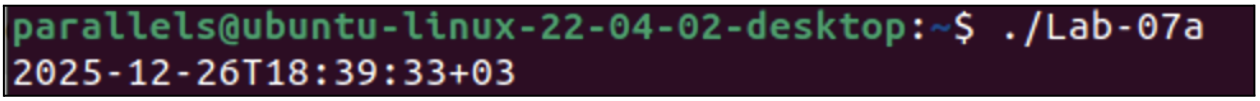

Рисунок 2.1 – Результат работы Lab-07a

Приложение под управлением Linux реализует тот же набор возможностей, что и соответствующая версия для Windows. Отличие заключается исключительно в применяемых программных интерфейсах: в среде Windows используются структуры и функции WinAPI, тогда как в Linux работа с временем осуществляется с использованием POSIX-совместимых функций.

2.2 Приложение Lab-07b

Данное приложение является функциональным аналогом версии для Windows и реализует подсчёт количества итераций в непрерывном цикле. Отличительной особенностью Linux-варианта является то, что ограничение времени работы цикла в 15 секунд определяется по чистому процессорному времени, затраченному процессом, а не по реальному (астрономическому) времени. По завершении выполнения программа дополнительно выводит фактическое реальное время работы приложения.

Для выполнения задания был разработан и скомпилирован программный код, представленный в листинге Ж. После запуска приложения результат его работы показан на рисунке 2.2.

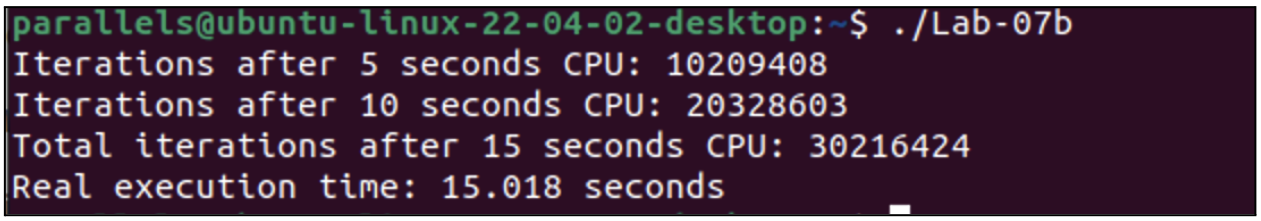

Рисунок 2.2 – Результат работы Lab-07b

Таким образом, основное отличие данной реализации от версии под Windows заключается в использовании процессорного времени для

измерения длительности выполнения цикла. При этом для наглядности в конце работы также выводится значение реального времени, прошедшего с момента запуска программы.

2.3 Приложение Lab-07c

Данное приложение является функциональным аналогом соответствующей реализации для операционной системы Windows и выполняет те же задачи по подсчету количества итераций с периодическим выводом промежуточных результатов и последующим завершением работы через заданный интервал времени.

Для выполнения задания был разработан и скомпилирован программный код, приведенный в листинге 3. После запуска приложения результат его работы представлен на рисунке 2.3.

```
parallels@ubuntu-linux-22-04-02-desktop:~$ ./Lab-07c
Elapsed 3 seconds, counter value: 90377245
Elapsed 6 seconds, counter value: 181390526
Elapsed 9 seconds, counter value: 276027874
Elapsed 12 seconds, counter value: 372238901
Elapsed 15 seconds, counter value: 464904321
Final counter value: 464904321
```

Рисунок 2.3 – Результат работы Lab-07c

В отличие от реализации под Windows, в версии для Linux вместо ожидающих таймеров ядра используется функция приостановки выполнения, обеспечивающая «сон» текущего процесса или потока на заданное время. При измерении времени применяются часы, не зависящие от изменения системного времени. Кроме того, вместо объектов ядра Windows в данной реализации используются соответствующие POSIX-функции и механизмы.

2.4 Приложение Lab-07x

Приложение Lab-07x предназначено для вычисления и последовательного вывода в консоль ряда простых чисел с их нумерацией. Основная задача программы — демонстрация работы с вычислительными циклами высокой интенсивности и контроля времени выполнения. По завершении работы приложение дополнительно выводит общее время выполнения, что позволяет оценить эффективность алгоритма и затраченные ресурсы процессора. Результат работы на рисунке 2.4.

```
1252208: 19692121
1252209: 19692143
1252210: 19692157
1252211: 19692187
1252212: 19692191
1252213: 19692217
Program finished after 30 seconds.
parallels@ubuntu-linux-22-04-02-desktop:~$
```

Рисунок 2.4 – Результат работы Lab-07x

Для реализации используются функции проверки чисел на простоту и таймеры для фиксации времени работы программы, что обеспечивает точный учет прошедшего времени и позволяет корректно завершать выполнение после заданного интервала. Этот блок помогает закрепить навыки работы с циклами, обработкой времени и форматированием данных для вывода в консоль.

2.5 Приложение Lab-07d

Приложение Lab-07d реализует запуск нескольких дочерних процессов на основе Lab-07x с последующим ожиданием их завершения. Первый дочерний процесс работает одну минуту, второй — две минуты, после чего родительский процесс корректно завершает выполнение только после окончания всех дочерних. Результат работы на рисунке 2.5.

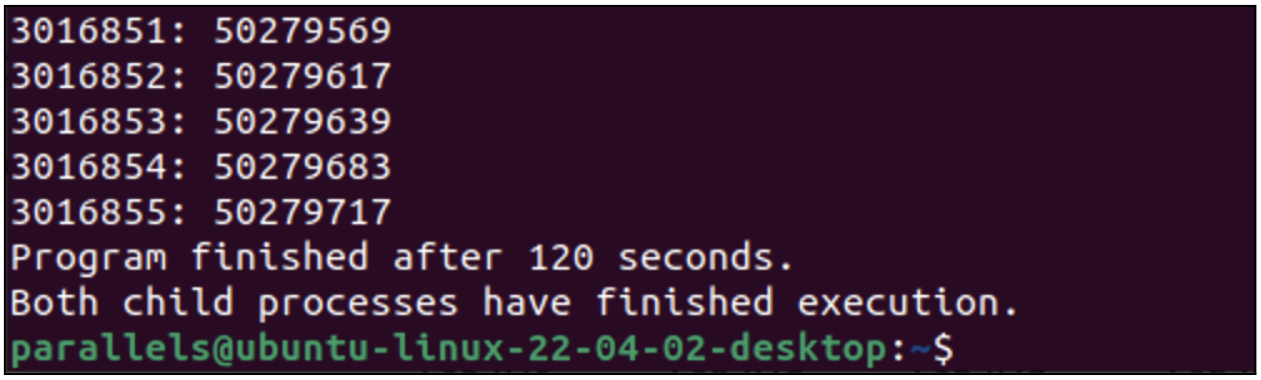

Рисунок 2.5 – Результат работы Lab-07d

В Linux дочерние процессы не могут запускаться в отдельных окнах консоли, поэтому их вывод осуществляется параллельно и одновременно в одном терминале, что требует внимательного контроля синхронизации и управления выводом данных. Такое построение программы позволяет наглядно изучить механизмы межпроцессного взаимодействия, особенности работы с процессами в POSIX-системах и организацию параллельного выполнения вычислительных задач. Дополнительно данный блок закрепляет понимание передачи параметров в дочерние процессы и контроля их времени выполнения.

Заключение

В ходе выполнения лабораторной работы №7 были освоены основные подходы к работе с системным временем, таймерами, процессами и межпроцессным взаимодействием, а также изучены различия в реализации этих механизмов в операционных системах Windows и Linux.

В среде Windows были созданы приложения Lab-07a, Lab-07b, Lab-07c и Lab-07d, демонстрирующие различные аспекты работы с временем и процессами: получение и форматирование текущего времени с использованием функций WinAPI, измерение длительности циклов с помощью высокоточных счётчиков QueryPerformanceCounter, настройку ожидающих таймеров через объекты ядра CreateWaitableTimer, а также создание и управление дочерними процессами с применением CreateProcessW и ожидание их завершения. Особое внимание уделялось точности измерений и организации параллельного выполнения процессов.

В Linux аналогичный функционал был реализован с применением POSIX-интерфейсов: функции time, gmtime, localtime и clock_gettime использовались для работы с временем, nanosleep обеспечивал задержки, а управление процессами выполнялось через fork и exec с последующим ожиданием завершения дочерних процессов с помощью waitpid. Высокоточные счётчики Windows были заменены соответствующими POSIX-механизмами, позволяющими измерять как реальное, так и процессорное время, что способствовало формированию навыков адаптации программных решений под разные платформы с учётом особенностей системных API.

Сравнительный анализ показал, что при идентичной логике работы существенные различия между Windows и Linux проявляются на уровне реализации. Windows предоставляет более высокоуровневые и интегрированные средства, такие как объекты ядра для таймеров и процессов, тогда как в Linux управление осуществляется через низкоуровневые POSIX-примитивы. Также особенности организации ввода-вывода приводят к различиям в отображении работы процессов: дочерние процессы в Windows могут запускаться в отдельных окнах консоли, тогда как в Linux вывод всех процессов по умолчанию отображается в одном терминале.

Приложение А – Листинг А.1

```
#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h> // Для abs
// 1 час в 100-наносекундных интервалах (1 час = 3600 сек * 10 000 000 (10^7)
100нс интервалов в сек)
#define ONE_HOUR_IN_100NS (3600LL * 10000000LL)
int main() {
    SYSTEMTIME utcTime;
    SYSTEMTIME localTime;
    FILETIME ftUtc, ftLocal;
    ULARGE_INTEGER uUtc, uLocal;
    LONGLONG diff;
    int offsetHours;
    char iso8601String[30]; // Буфер для строки времени
    int offsetSign;
    // 1. Получаем время
    GetSystemTime(&utcTime); // Время UTC
    GetLocalTime(&localTime); // Локальное время
    // 2. Переводим SYSTEMTIME в FILETIME
    SystemTimeToFileTime(&utcTime, &ftUtc);
    SystemTimeToFileTime(&localTime, &ftLocal);
    // 3. Преобразуем FILETIME в 64-битное значение
    uUtc.LowPart = ftUtc.dwLowDateTime;
    uUtc.HighPart = ftUtc.dwHighDateTime;
    uLocal.LowPart = ftLocal.dwLowDateTime;
    uLocal.HighPart = ftLocal.dwHighDateTime;
    // 4. Разница в 100-наносекундных интервалах
    // Разница между локальным и UTC дает смещение
    diff = (LONGLONG)(uLocal.QuadPart - uUtc.QuadPart);
    // 5. Переводим в часы
    offsetHours = (int)(diff / ONE_HOUR_IN_100NS);
    // 6. Формируем строку в формате ISO 8601 YYYY-MM-DDThh:mm:ss+HH
    offsetSign = (offsetHours >= 0) ? '+' : '-';
    // Форматирование с использованием sprintf. (Более безопасная версия
snprintf не является частью WinAPI,
    // но доступна через стандартную библиотеку C, включенную компилятором)
    sprintf(iso8601String,
        "%04d-%02d-%02dT%02d:%02d:%02d%c%02d",
        localTime.wYear,
        localTime.wMonth,
        localTime.wDay,
        localTime.wHour,
        localTime.wMinute,
        localTime.wSecond,
        offsetSign,
        abs(offsetHours) // Используем abs() из <stdlib.h> для получения
абсолютного значения
    );
    printf("%s\n", iso8601String);
    return 0;
}
```

Приложение Б – Листинг Б.1

```
#include <windows.h>
#include <stdio.h>
int main() {
    LARGE_INTEGER freq, start, now;
    if (!QueryPerformanceFrequency(&freq)) {
        // Output error to standard error stream
        fprintf(stderr, "Error: QueryPerformanceFrequency not supported.\n");
        return 1;
    }
    QueryPerformanceCounter(&start);
    unsigned long long counter = 0;
    int printed5 = 0;
    int printed10 = 0;
    while (1) {
        counter++;
        QueryPerformanceCounter(&now);
        double elapsed = (double)(now.QuadPart - start.QuadPart) /
(double)freq.QuadPart;
        if (!printed5 && elapsed >= 5.0) {
            // Output in English (ASCII)
            printf("Iterations after 5 seconds: %llu\n", counter);
            printed5 = 1;
        }
        if (!printed10 && elapsed >= 10.0) {
            printf("Iterations after 10 seconds: %llu\n", counter);
            printed10 = 1;
        }
        if (elapsed >= 15.0) {
            printf("Total iterations after 15 seconds: %llu\n", counter);
            break;
        }
    }
    return 0;
}
```

Приложение В – Листинг В.1

```
#include <windows.h>
#include <stdio.h>
int main() {
    // 64-битные переменные для подсчета итераций
    unsigned long long periodIterations = 0; // Итерации, накопленные за 3
секунды
    unsigned long long totalIterations = 0;  // Общая сумма всех итераций
    // Дескриптор ожидающего таймера
    HANDLE hTimer = NULL;
    // Время, через которое сработает таймер (отрицательное значение для
относительного времени)
    // 3 секунды = 3 * 10 000 000 (100нс интервалов в секунду) = 30 000 000
    // Отрицательное значение указывает на относительное время
    LARGE_INTEGER liDueTime;
    liDueTime.QuadPart = -30000000LL;
    int totalPeriods = 5; // 15 секунд / 3 секунды = 5 периодов
    int currentPeriod = 1;
    // --- 1. Создание ожидающего таймера ---
    // Требуется использовать его, чтобы выполнить требование задания
    hTimer = CreateWaitableTimer(
        NULL,  // Атрибуты безопасности по умолчанию
        FALSE, // Автоматический сброс (таймер сбрасывается после
срабатывания)
        NULL   // Имя таймера
    );
    if (!hTimer) {
        fprintf(stderr, "Error: Failed to create waitable timer.\n");
        return 1;
    }
    // --- 2. Основной цикл ---
    while (currentPeriod <= totalPeriods) {
        // Запускаем таймер на 3 секунды
        // Период = 0, т.е. сработает только один раз.
        if (!SetWaitableTimer(hTimer, &liDueTime, 0, NULL, NULL, FALSE)) {
            fprintf(stderr, "Error: Failed to set waitable timer.\n");
            CloseHandle(hTimer);
            return 1;
        }
        // --- Активный подсчет итераций, пока таймер не сработает ---
        // Это гарантирует, что мы получим миллионы итераций,
        // используя свободное время ЦПУ до срабатывания таймера.
        while (WaitForSingleObject(hTimer, 0) != WAIT_OBJECT_0) {
            periodIterations++;
        }
        // --- Вывод результатов периода ---
        totalIterations += periodIterations;
        printf("Elapsed %d seconds, iterations in period: %llu\n",
            currentPeriod * 3,
            periodIterations);
        currentPeriod++;
        periodIterations = 0; // Сбрасываем счетчик для следующего периода
    }
    printf("Total accumulated iterations (15 seconds): %llu\n",
```

```
totalIterations);
    CloseHandle(hTimer);
    return 0;
}
```

Приложение Г – Листинг Г.1

```
#include <windows.h>
#include <stdio.h>  // Для printf и fprintf
#include <stdlib.h> // Для atoi и exit
#define WIDE_STR(x) L##x
// Функция проверки простоты числа (чистый C)
int isPrime(unsigned long long n) {
    if (n < 2) return 0; // 0 = false
    if (n == 2) return 1; // 1 = true
    if (n % 2 == 0) return 0;
    // В C, i * i <= n может вызвать переполнение, если n близко к ULLONG_MAX
    // Но в контексте этого задания это безопасно.
    for (unsigned long long i = 3; i * i <= n; i += 2) {
        if (n % i == 0) return 0;
    }
    return 1;
}
int main(int argc, char* argv[]) {
    // --- 1. Обработка аргументов и настройка времени ---
    int runSeconds = 30; // По умолчанию 30 секунд
    // Обработка аргумента командной строки
    if (argc == 2) {
        // atoi() - функция из stdlib.h (чистый C)
        runSeconds = atoi(argv[1]);
        if (runSeconds <= 0) {
            fprintf(stderr, "Error: Invalid run duration specified.\n");
            return 1;
        }
    }
    // Инициализация WinAPI таймера
    LARGE_INTEGER freq, start, now;
    if (!QueryPerformanceFrequency(&freq)) {
        fprintf(stderr, "Error: QueryPerformanceFrequency not supported.\n");
        return 1;
    }
    QueryPerformanceCounter(&start);
    // --- 2. Настройка счетчиков ---
    unsigned long long num = 2;   // Текущее число для проверки
    unsigned long long index = 1; // Порядковый номер найденного простого
числа
    // --- 3. Основной цикл поиска простых чисел ---
    while (1) {
        if (isPrime(num)) {
            // Вывод пронумерованного простого числа (ASCII)
            printf("%llu: %llu\n", index, num);
            index++;
        }
        num++;
        // Проверка времени
        QueryPerformanceCounter(&now);
        double elapsed = (double)(now.QuadPart - start.QuadPart) /
(double)freq.QuadPart;
        if (elapsed >= runSeconds) {
            break;
```

```
        }
    }
    // --- 4. Завершение и вывод итогового времени ---
    QueryPerformanceCounter(&now);
    double realElapsed = (double)(now.QuadPart - start.QuadPart) /
(double)freq.QuadPart;
    printf("Program ran for %.2f seconds.\n", realElapsed);
    return 0;
}
```

Приложение Д – Листинг Д.1

```
#include <windows.h>
#include <stdio.h>
int main() {
    STARTUPINFOW si = { 0 };
    si.cb = sizeof(si);
    // PROCESS_INFORMATION - для получения информации о созданном процессе
(дескрипторы)
    PROCESS_INFORMATION pi1 = { 0 };
    PROCESS_INFORMATION pi2 = { 0 };
    // Командные строки (Unicode) для запуска дочерних процессов
    // Lab-07x.exe 60 (1 минута)
    // Lab-07x.exe 120 (2 минуты)
    wchar_t cmd1[] = L"Lab-07x.exe 15";
    wchar_t cmd2[] = L"Lab-07x.exe 30";
    // --- 1. Запуск первого дочернего процесса ---
    if (!CreateProcessW(
        NULL,           // lpApplicationName (NULL, если указано в
lpCommandLine)
        cmd1,           // lpCommandLine (команда и аргументы)
        NULL,           // lpProcessAttributes
        NULL,           // lpThreadAttributes
        FALSE,          // bInheritHandles (не наследовать дескрипторы)
        CREATE_NEW_CONSOLE, // dwCreationFlags (запуск в новом консольном
окне)
        NULL,           // lpEnvironment
        NULL,           // lpCurrentDirectory
        &si,            // lpStartupInfo
        &pi1            // lpProcessInformation (выходные данные)
    )) {
        // Использование fprintf(stderr) и GetLastError
        fprintf(stderr, "Error launching Process 1. Error code: %lu\n",
GetLastError());
        return 1;
    }
    // --- 2. Запуск второго дочернего процесса ---
    if (!CreateProcessW(
        NULL,
        cmd2,
        NULL,
        NULL,
        FALSE,
        CREATE_NEW_CONSOLE,
        NULL,
        NULL,
        &si,
        &pi2
    )) {
        fprintf(stderr, "Error launching Process 2. Error code: %lu\n",
GetLastError());
        // В случае ошибки, закрываем дескрипторы первого процесса, чтобы
избежать утечек
        CloseHandle(pi1.hProcess);
        CloseHandle(pi1.hThread);
```

```
        return 1;
    }
    // --- 3. Ожидание завершения дочерних процессов ---
    printf("Parent process is waiting for child processes to finish...\n");
    // Ожидание завершения первого процесса (INFINITE - бесконечное ожидание)
    //
    WaitForSingleObject(pi1.hProcess, INFINITE);
    // Ожидание завершения второго процесса
    WaitForSingleObject(pi2.hProcess, INFINITE);
    // Вывод сообщения о завершении (ASCII)
    printf("Both child processes have finished execution.\n");
    // --- 4. Закрытие дескрипторов ---
    // Важно закрыть дескрипторы, полученные от CreateProcess, чтобы ОС могла
освободить ресурсы
    CloseHandle(pi1.hProcess);
    CloseHandle(pi1.hThread);
    CloseHandle(pi2.hProcess);
    CloseHandle(pi2.hThread);
    return 0;
}
```

Приложение Е – Листинг Е.1

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

int main() {
    time_t rawtime;
    struct tm local_tm; // Храним копию локального времени
    struct tm gmt_tm;   // Храним копию UTC
    long diff_seconds;
    int offsetHours;
    char timeZoneString[6];
    char iso8601String[30];

    time(&rawtime);

    // Важно: сразу копируем содержимое структур, так как функции используют
один буфер
    local_tm = *localtime(&rawtime);
    gmt_tm = *gmtime(&rawtime);

    // Расчет разницы
    diff_seconds = (local_tm.tm_hour - gmt_tm.tm_hour) * 3600 +
                   (local_tm.tm_min - gmt_tm.tm_min) * 60;

    // Корректировка перехода через сутки
    if (local_tm.tm_yday > gmt_tm.tm_yday) {
        diff_seconds += 24 * 3600;
    } else if (local_tm.tm_yday < gmt_tm.tm_yday) {
        // Случай, если локальное время — конец года, а UTC — начало нового
        if (!(local_tm.tm_yday == 0 && gmt_tm.tm_yday > 300)) {
            diff_seconds -= 24 * 3600;
        } else {
            diff_seconds += 24 * 3600;
        }
    }
     offsetHours = (int)(diff_seconds / 3600);

    // Форматирование ISO 8601
    size_t len = strftime(iso8601String, sizeof(iso8601String),
"%Y-%m-%dT%H:%M:%S", &local_tm);

    // Добавляем смещение ±HH
    sprintf(iso8601String + len, "%+03d", offsetHours);

    printf("%s\n", iso8601String);

    return 0;
}
```

Приложение Ж – Листинг Ж.1

```
#include <stdio.h>    // Для printf, fprintf
#include <stdlib.h>   // Для exit
#include <time.h>     // Для clock_t, clock(), CLOCKS_PER_SEC, timespec,
clock_gettime

int main() {
    unsigned long long counter = 0;
 // Засекаем процессорное время (CPU time)
    clock_t startCPU = clock();

    // Засекаем реальное время (Wall Clock time)
    struct timespec startReal;
    // clock_gettime - функция POSIX
    if (clock_gettime(CLOCK_REALTIME, &startReal) != 0) {
        fprintf(stderr, "Error: clock_gettime failed for CLOCK_REALTIME.\n");
        return 1;
    }


    int printed5 = 0;
    int printed10 = 0;


    while (1) {
        counter++;
        clock_t nowCPU = clock();
        // Вычисление прошедшего времени CPU в секундах
        double elapsedCPU = (double)(nowCPU - startCPU) / CLOCKS_PER_SEC;
        if (!printed5 && elapsedCPU >= 5.0) {
            // Вывод в чистом ASCII
            printf("Iterations after 5 seconds CPU: %llu\n", counter);
            printed5 = 1;
        }
        if (!printed10 && elapsedCPU >= 10.0) {
            // Вывод в чистом ASCII

            printf("Iterations after 10 seconds CPU: %llu\n", counter);

            printed10 = 1;

        }

        if (elapsedCPU >= 15.0) {
            // Вывод в чистом ASCII
            printf("Total iterations after 15 seconds CPU: %llu\n", counter);
            break;
        }
    }

    // --- Вывод реального времени ---
    struct timespec endReal;
    if (clock_gettime(CLOCK_REALTIME, &endReal) != 0) {
        fprintf(stderr, "Error: clock_gettime failed for CLOCK_REALTIME on
end.\n");
        return 1;
    }
    // Расчет реального времени работы

    double elapsedReal = (double)(endReal.tv_sec - startReal.tv_sec)
                       + (double)(endReal.tv_nsec - startReal.tv_nsec) / 1e9;
    printf("Real execution time: %.3f seconds\n", elapsedReal);
```

```
    return 0;

}
```

Приложение 3 – Листинг 3.1

```
#include <stdio.h>    // Для printf, fprintf
#include <stdlib.h>   // Для exit
#include <time.h>     // Для timespec, clock_gettime, CLOCK_MONOTONIC

// Примечание: <unistd.h> не является частью стандартного C,
// но функции timespec/clock_gettime часто доступны в POSIX-совместимых средах
(MinGW/Linux/macOS).

// Функция для вычисления разницы в секундах между двумя timespec
double diff_timespec(const struct timespec *end, const struct timespec *start)
{
    return (double)(end->tv_sec - start->tv_sec) +
           (double)(end->tv_nsec - start->tv_nsec) / 1e9;
}

int main() {
    unsigned long long counter = 0;

    // Переменные для таймера
    struct timespec start, now;

    // Засекаем стартовое время (CLOCK_MONOTONIC - идеален для измерения
интервалов)
    if (clock_gettime(CLOCK_MONOTONIC, &start) != 0) {
        fprintf(stderr, "Error: clock_gettime failed.\n");
        return 1;
    }

    int elapsedPeriods = 0;

    while (1) {
        // Активное ожидание (Busy Waiting)
        while (1) {
            counter++;

            // Получаем текущее время
            if (clock_gettime(CLOCK_MONOTONIC, &now) != 0) {
                 fprintf(stderr, "Error: clock_gettime failed during
loop.\n");
                 return 1;
            }

            // Вычисляем прошедшее время
            double elapsed = diff_timespec(&now, &start);

            // Проверка, достигнут ли конец текущего 3-секундного интервала
            if (elapsed >= (elapsedPeriods + 1) * 3.0) {
                break;
            }
        }

        // --- Вывод результатов периода ---
        elapsedPeriods++;

        // Вывод в чистом ASCII (аналог C++)
        printf("Elapsed %d seconds, counter value: %llu\n",
               elapsedPeriods * 3,
               counter);

        // --- Проверка завершения ---
        if (elapsedPeriods * 3 >= 15) {
```

```
            printf("Final counter value: %llu\n", counter);
            break;
        }
    }

    return 0;
}
```

Приложение И – Листинг И.1

```
#include <stdio.h>    // Для printf, fprintf
#include <stdlib.h>   // Для atoi, exit
#include <time.h>     // Для timespec, clock_gettime, CLOCK_MONOTONIC
#include <unistd.h>   // Для POSIX-функций

// Функция проверки простоты числа (чистый C)
int isPrime(unsigned long long n) {
    if (n < 2) return 0;
    if (n == 2) return 1;
    if (n % 2 == 0) return 0;

    for (unsigned long long i = 3; i * i <= n; i += 2) {
        if (n % i == 0) return 0;
    }
    return 1;
}

// Функция для вычисления разницы в секундах между двумя timespec
double diff_timespec(const struct timespec *end, const struct timespec *start)
{
    return (double)(end->tv_sec - start->tv_sec) +
           (double)(end->tv_nsec - start->tv_nsec) / 1e9;
}


int main(int argc, char* argv[]) {
    // --- Обработка аргументов ---
    int runSeconds = 30; // Default 30 seconds
    if (argc == 2) {
        // atoi() из <stdlib.h>
        runSeconds = atoi(argv[1]);
        if (runSeconds <= 0) {
            fprintf(stderr, "Error: Invalid run duration specified.\n");
            return 1;
        }
    }

    // --- Инициализация таймера ---
    struct timespec start, now;
    if (clock_gettime(CLOCK_MONOTONIC, &start) != 0) {
        fprintf(stderr, "Error: clock_gettime failed.\n");
        return 1;
    }

    // --- Основной цикл поиска ---
    unsigned long long num = 2;
    unsigned long long index = 1;

    while (1) {
        if (isPrime(num)) {
            printf("%llu: %llu\n", index, num);
            index++;
        }
        num++;

        // Проверка времени
        if (clock_gettime(CLOCK_MONOTONIC, &now) != 0) {
             fprintf(stderr, "Error: clock_gettime failed during loop.\n");
             return 1;
        }
```

```
        double elapsed = diff_timespec(&now, &start);

        if (elapsed >= runSeconds) {
            break;
        }
    }

    // --- Итоговый вывод ---
    printf("Program finished after %d seconds.\n", runSeconds);

    return 0;
}
```

Приложение К – Листинг К.1

```
#include <stdio.h>    // Для printf, fprintf
#include <stdlib.h>   // Для exit
#include <unistd.h>   // Для fork, execl
#include <sys/wait.h> // Для waitpid

// Название исполняемого файла дочернего процесса
#define CHILD_EXEC_NAME "./Lab-07x"

int main() {

    // Создаем первый дочерний процесс
    pid_t pid1 = fork();

    if (pid1 == -1) {
        perror("Error creating child process 1");
        return 1;
    }

    if (pid1 == 0) {
        // Дочерний процесс 1: запускаем Lab-07x на 60 секунд
        // execl - функция POSIX, заменяет текущий процесс новым.
        // Аргументы: путь, имя программы, arg1, arg2, ..., NULL
        execl(CHILD_EXEC_NAME, CHILD_EXEC_NAME, "60", (char*)NULL);

        // Если execl вернула управление, значит произошла ошибка
        perror("Error launching child process 1");
        exit(1);
    }

    // Создаем второй дочерний процесс
    pid_t pid2 = fork();

    if (pid2 == -1) {
        perror("Error creating child process 2");

        // Если не удалось запустить pid2, нужно ждать pid1, чтобы не оставить
зомби
        waitpid(pid1, NULL, 0);
        return 1;
    }

    if (pid2 == 0) {
        // Дочерний процесс 2: запускаем Lab-07x на 120 секунд
        execl(CHILD_EXEC_NAME, CHILD_EXEC_NAME, "120", (char*)NULL);

        // Если execl вернула управление, значит произошла ошибка
        perror("Error launching child process 2");
        exit(1);
    }

    // --- Родительский процесс ---

    printf("Parent process is waiting for child processes to finish...\n");

    int status;

    // Ожидание завершения первого дочернего процесса
    waitpid(pid1, &status, 0);

    // Ожидание завершения второго дочернего процесса
    waitpid(pid2, &status, 0);
```

```
    printf("Both child processes have finished execution.\n");

    return 0;
}
```