**Задание 01. Windows**

1. Разработайте приложение **OS07\_01**.
2. Приложение **OS07\_01** выводит на консоль текущую локальную дату и время в формате ***дд.мм.ггг чч:мин:сек***.

#include <iostream>

int main()

{

time\_t t; //время UTC

tm tm; //преобразов.

time(&t);

localtime\_s(&tm, &t);

printf(

"%d.%d.%d %d:%d:%d",

tm.tm\_mday,

tm.tm\_mon + 1,

tm.tm\_year + 1900,

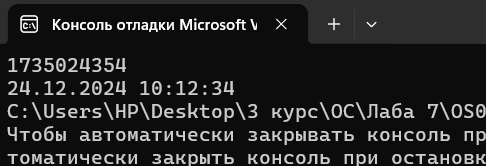
tm.tm\_hour,

tm.tm\_min,

tm.tm\_sec

);

}

****

**Задание 02. Windows**

1. Разработайте приложение **OS07\_02,** выполняющее бесконечный цикл.
2. В теле цикла подсчитывается количество итераций.
3. Выведите на консоль значения счетчика итераций через 5 сек. и 10 сек.
4. Корректно завершите работу цикла и приложения через 15 сек., выведите итоговое значение счетчика итераций.

#include <Windows.h>

#include <iostream>

#include <ctime>

using namespace std;

int main()

{

clock\_t start = clock();

int k = 0;

bool flag5 = true, flag10 = true;

while (true)

{

k++;

if ((clock() - start) / CLOCKS\_PER\_SEC == 5 && flag5) {

cout << " After 5s: " << k << '\n';

flag5 = false;

}

if ((clock() - start) / CLOCKS\_PER\_SEC == 10 && flag10) {

cout << "After 10s: " << k << '\n';

flag10 = false;

}

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

cout << "After 15s: " << k << '\n';

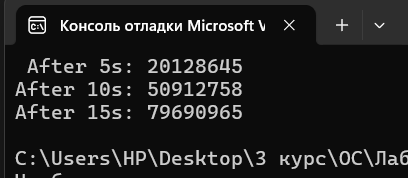
break;

}

}

return 0;

}



**Задание 03. Windows**

1. **Указание: самостоятельно освойте и примените периодический ожидающий таймер**
2. Разработайте приложение **OS07\_03,** выполняющее бесконечный цикл.
3. В теле цикла с задержкой подсчитывается количество итераций.
4. Выведите на консоль значения счетчика итераций каждые 3 сек.
5. Корректно завершите работу цикла и приложения через 15 сек., выведите итоговое значение счетчика итераций.

#include <iostream>

#include <Windows.h>

using namespace std;

HANDLE createThread(LPTHREAD\_START\_ROUTINE func);//создание потока

void close\_timer\_watcher();

void print\_timer\_watcher();

//переодические таймеры ожидания

HANDLE print\_timer = CreateWaitableTimer(NULL, FALSE, L"os07\_04v2\_print");

HANDLE close\_timer = CreateWaitableTimer(NULL, FALSE, L"os07\_04v2\_close");

bool print = false, iterate = true;

int main()

{//массив функция

LPTHREAD\_START\_ROUTINE funcs[] = { (LPTHREAD\_START\_ROUTINE)close\_timer\_watcher, (LPTHREAD\_START\_ROUTINE)print\_timer\_watcher };

const int size = sizeof(funcs) / sizeof(LPTHREAD\_START\_ROUTINE);

HANDLE threads[size];//храненение дескрипторов потоков

LARGE\_INTEGER close\_timeout;

close\_timeout.QuadPart = -150000000LL;//15 сек

LARGE\_INTEGER print\_timeout;

print\_timeout.QuadPart = -30000000LL;//3 сек

clock\_t current\_time, start\_time = clock();

unsigned long int i;

for (i = 0; i < size; i++)//потоки для двух функций

threads[i] = createThread(funcs[i]);

//установка таймеров

SetWaitableTimer(print\_timer, &print\_timeout, 3000, NULL, NULL, FALSE);

SetWaitableTimer(close\_timer, &close\_timeout, 0, NULL, NULL, 0);

for (i = 0; iterate; i++)

{

current\_time = ((clock() - start\_time) / CLOCKS\_PER\_SEC);

if (print)

{

cout << "Time: " << current\_time << "\tIterations: " << i << '\n';

print = false;

}

}

cout << "Final: " << i << '\n';

for (i = 0; i < size; i++)

CloseHandle(threads[i]);//закрытие дескрипторов

system("pause");

return 0;

}

HANDLE createThread(LPTHREAD\_START\_ROUTINE func)

{

DWORD thread\_id = NULL;

HANDLE thread = CreateThread(NULL, 0, func, NULL, 0, &thread\_id);

if (thread == NULL) {

throw "Error THREAD";

}

return thread;

}

void close\_timer\_watcher()

{

WaitForSingleObject(close\_timer, INFINITE);//ожидается срабатывание close\_timer

iterate = false;

CloseHandle(close\_timer);

}

void print\_timer\_watcher()

{

for (;;)

{

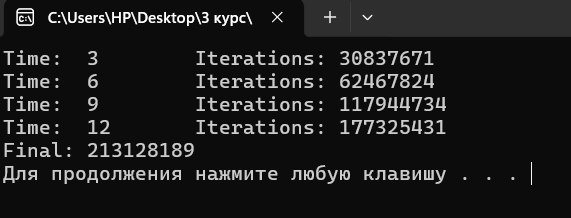
WaitForSingleObject(print\_timer, INFINITE);//ожидается срабатывание print\_timer

print = true;

}

CloseHandle(print\_timer);

}

****

**Задание 04. Windows**

1. Разработайте приложение **OS07\_04,** запускающее два одинаковых дочерних процесса **OS07\_04\_X**.
2. Процессы **OS07\_04\_X** вычисляют и выводят на консоль (каждый в свою) пронумерованный ряд простых положительных чисел (простое число делится нацело только на себя и 1).
3. Первый дочерний процесс должен выполняться 1 минуту и корректно завершаться.
4. Первый дочерний процесс должен выполняться 2 минуты и корректно завершаться.
5. Приложение **OS07\_04** завершается после завершения дочерних процессов.

#include <windows.h>

#include <iostream>

#include <string>

void runChildProcess(const std::wstring& duration, const std::wstring& processName) {

STARTUPINFO si = { sizeof(STARTUPINFO) };

PROCESS\_INFORMATION pi;

std::wstring command = processName + L" " + duration;

if (!CreateProcessW(

NULL,

&command[0],

NULL,

NULL,

FALSE,

0,

NULL,

NULL,

&si,

&pi)) {

std::wcerr << L"Failed to start the child process. Error code: " << GetLastError() << std::endl;

return;

}

std::wcout << L"Child process started: " << processName << L" with duration " << duration << L" seconds." << std::endl;

WaitForSingleObject(pi.hProcess, INFINITE);

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

int main() {

std::wcout << L"Launching two child processes OS07\_04\_X..." << std::endl;

runChildProcess(L"60", L"OS07\_04\_X.exe");

runChildProcess(L"120", L"OS07\_04\_X.exe");

std::wcout << L"Both child processes finished. Main process is exiting." << std::endl;

return 0;

}

#include <iostream>

#include <chrono>

#include <thread>

#include <vector>

#include <string>

#include <cstdlib>

bool isPrime(int number) {

if (number < 2) return false;

for (int i = 2; i \* i <= number; ++i) {

if (number % i == 0) return false;

}

return true;

}

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

if (argc < 2) {

std::cerr << "Error: duration in seconds must be provided." << std::endl;

return 1;

}

int duration = std::stoi(argv[1]);

auto startTime = std::chrono::steady\_clock::now();

int number = 2;

int counter = 1;

while (true) {

if (isPrime(number)) {

std::cout << "Prime #" << counter << ": " << number << std::endl;

++counter;

}

++number;

auto currentTime = std::chrono::steady\_clock::now();

auto elapsedSeconds = std::chrono::duration\_cast<std::chrono::seconds>(currentTime - startTime).count();

if (elapsedSeconds >= duration) {

std::cout << "Time is up. Process is exiting." << std::endl;

break;

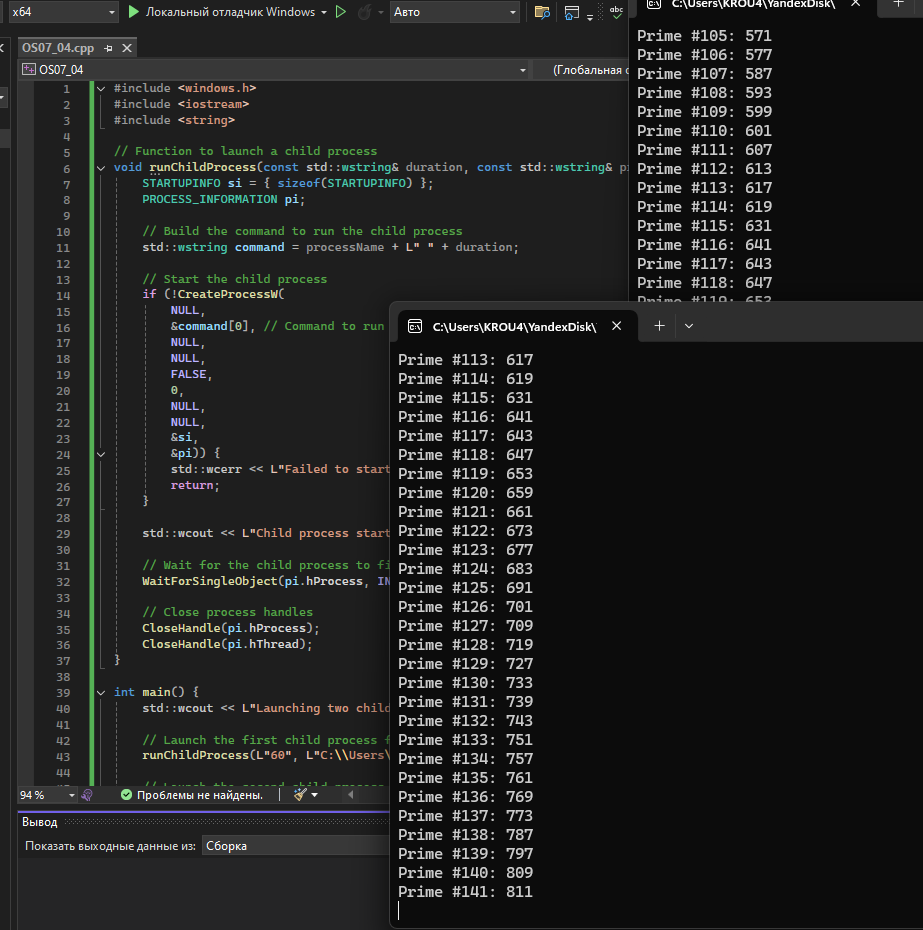
}

std::this\_thread::sleep\_for(std::chrono::milliseconds(1));

}

return 0;

}

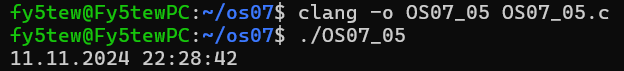


**Задание 05. Linux**

1. Разработайте приложение **OS07\_05**.
2. Приложение **OS07\_05** выводит на консоль текущую локальную дату и время в формате ***дд.мм.ггг чч:мин:сек***.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание



**Задание 06. Linux**

1. Разработайте приложение **OS07\_06,** выполняющее бесконечный цикл.
2. В теле цикла подсчитывается количество итераций.
3. Выведите на консоль значения счетчика итераций через 2 сек. **процессорного** времени и корректно завершите цикл.
4. Приложение **OS07\_06 должно** выполнять замер реального затраченного на работу цикла времени и выводить его значения на консоль.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Задание 07.** Ответьте на следующие вопросы

**23. Понятие «социальное время» и его не монотонность**

**Социальное время** — это условное измерение времени, отражающее коллективное восприятие и упорядочение событий в обществе. Оно связано с календарями, часовыми поясами, изменениями летнего и зимнего времени, что делает его зависимым от социальных решений.

**Почему не монотонное:**

* Переход на летнее/зимнее время изменяет ход времени.
* Корректировка высокосекунд (leap seconds) может добавлять или убирать секунды.
* Изменения границ часовых поясов приводят к скачкам времени.

**24. Понятие «эпоха Linux»**

**Эпоха Linux** — это точка отсчета времени в Linux-системах. Она начинается с **1 января 1970 года, 00:00:00 UTC**.

**Единицы измерения:**

* Время измеряется в секундах, прошедших с начала эпохи (Unix timestamp).

**25. Понятие «Universal Coordinated Time (UTC)»**

**Universal Coordinated Time (UTC)** — это международный стандарт времени.

* Основан на атомном времени, но синхронизируется с вращением Земли через добавление высокосекунд.
* Используется в вычислительных системах для унификации времени независимо от часовых поясов.

**26. Относительное время и абсолютное время**

1. **Относительное время**
   * Измеряется относительно какого-либо момента или события.
   * Пример: интервал в 5 секунд.
2. **Абсолютное время**
   * Определяется в фиксированной точке на временной шкале.
   * Пример: "5 декабря 2024 года, 14:00:00 UTC".

**27. Понятие «тик»**

**Тик** — минимальный промежуток времени, который может измерить операционная система.

* В Linux тик связан с частотой системного таймера.
* Частота определяется константой HZ, например, 100, 250, или 1000 тиков в секунду.

**28. Понятие «ожидающий таймер», типы и состояния таймеров**

1. **Ожидающий таймер**
   * Таймер, который "ждет" наступления определенного момента или события для выполнения задачи.
2. **Типы таймеров в Linux:**
   * **Однократные**: срабатывают один раз.
   * **Периодические**: запускаются с заданным интервалом.
3. **Состояния таймеров:**
   * **Активный**: работает и отсчитывает время.
   * **Остановленный**: не выполняет подсчета.
   * **Просроченный**: время вышло, ожидает обработчика события.

**29. Типы часов в Linux и их назначение**

1. **CLOCK\_REALTIME**
   * Отражает текущее реальное время (UTC).
   * Может изменяться вручную.
2. **CLOCK\_MONOTONIC**
   * Неизменяемое время от запуска системы.
   * Используется для измерения интервалов.
3. **CLOCK\_PROCESS\_CPUTIME\_ID**
   * Время работы текущего процесса на процессоре.
4. **CLOCK\_THREAD\_CPUTIME\_ID**
   * Время работы текущего потока на процессоре.
5. **CLOCK\_BOOTTIME**
   * Монолитное время с учетом времени сна системы.

**30. Константы HZ и CLOCKS\_PER\_SEC**

1. **HZ**
   * Определяет частоту системного таймера в тиках в секунду.
   * Зависит от конфигурации ядра: обычно 100, 250, или 1000.
2. **CLOCKS\_PER\_SEC**
   * Количество тиков процессора в секунду.
   * Обычно совпадает с HZ.
   * Используется для перевода времени из тиков в секунды.