Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Операционные системы

Студент: Лопатнюк П.В.

ФИТ 3 курс 1 группа

Преподаватель: Бернацкий П.В.

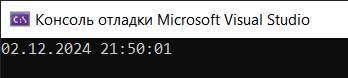
Минск 2024

**Лабораторная работа 07. Компьютерное время**

**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***.

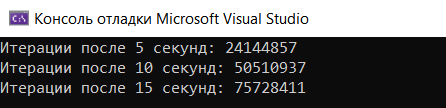
|  |
| --- |
| #include <iostream>  #include <ctime>  int main() {  std::time\_t now = std::time(nullptr);  std::tm local\_time;  localtime\_s(&local\_time, &now);  char buffer[20];  std::strftime(buffer, sizeof(buffer), "%d.%m.%Y %H:%M:%S", &local\_time);  std::cout << buffer << std::endl;  return 0;  } |



**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.

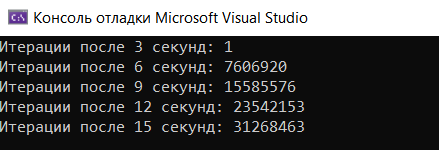
|  |
| --- |
| #include <Windows.h>  #include <iostream>  #include <ctime>  using namespace std;  int main()  {  setlocale(LC\_ALL, "rus");  clock\_t start = clock();  int counter = 0;  bool is\_5\_sec = true, is\_10\_sec = true;  while (true)  {  counter++;  if (is\_5\_sec && (clock() - start) / CLOCKS\_PER\_SEC == 5) {  cout << "Итерации после 5 секунд: " << counter << '\n';  is\_5\_sec = false;  }  if (is\_10\_sec && (clock() - start) / CLOCKS\_PER\_SEC == 10) {  cout << "Итерации после 10 секунд: " << counter << '\n';  is\_10\_sec = false;  }  if ((clock() - start) / CLOCKS\_PER\_SEC == 15) {  cout << "Итерации после 15 секунд: " << counter << '\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.

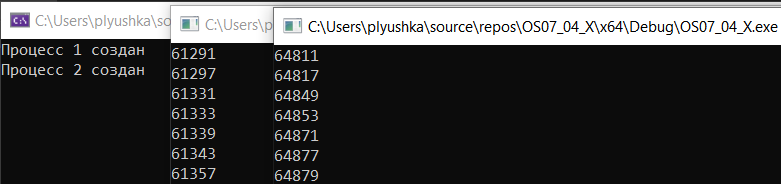
|  |
| --- |
| #include <Windows.h>  #include <iostream>  #include <ctime>  using namespace std;  int main()  {  setlocale(LC\_ALL, "rus");  HANDLE hTimer = CreateWaitableTimer(NULL, TRUE, NULL);  if (hTimer == NULL) {  cout << "Не удалось создать таймер." << endl;  return 1;  }  LARGE\_INTEGER liDueTime;  liDueTime.QuadPart = -30000000LL;  if (!SetWaitableTimer(hTimer, &liDueTime, 3000, NULL, NULL, FALSE)) {  cout << "Не удалось установить таймер." << endl;  CloseHandle(hTimer);  return 1;  }  int counter = 0;  clock\_t start = clock();  bool flag\_3\_sec = true, flag\_15\_sec = false;  while (!flag\_15\_sec)  {  DWORD dwWaitResult = WaitForSingleObject(hTimer, INFINITE);  if (dwWaitResult == WAIT\_OBJECT\_0) {  counter++;  clock\_t elapsed = clock() - start;  double elapsedSeconds = (double)elapsed / CLOCKS\_PER\_SEC;  if ((int)elapsedSeconds % 3 == 0 && flag\_3\_sec) {  cout << "Итерации после " << (int)elapsedSeconds << " секунд: " << counter << endl;  flag\_3\_sec = false;  }  if (elapsedSeconds >= 15) {  flag\_15\_sec = true;  }  }  if ((clock() - start) / CLOCKS\_PER\_SEC % 3 != 0) {  flag\_3\_sec = true;  }  }  CloseHandle(hTimer);  return 0;  } |

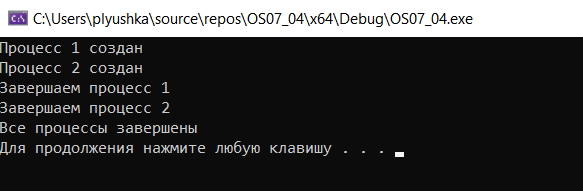


**Задание 04. Windows**

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 <iostream>  #include <Windows.h>  using namespace std;  HANDLE FirstProcessTimer, SecondProcessTimer;  HANDLE processes[2] = { NULL, NULL };  int main() {  setlocale(LC\_ALL, "Russian");  FirstProcessTimer = CreateWaitableTimer(NULL, FALSE, L"three\_sec");  SecondProcessTimer = CreateWaitableTimer(NULL, FALSE, L"fifteen\_sec");  if (!FirstProcessTimer || !SecondProcessTimer) {  cout << "[ОШИБКА] Не удалось создать таймеры\n";  return 1;  }  PROCESS\_INFORMATION processInfos[2];  LPCWSTR exePath = L"D:\\5sem\_BSTU\\OS\\lab7\\OS07\_04A\\x64\\Debug\\OS07\_04A.exe";  STARTUPINFO startupInfo = { sizeof(STARTUPINFO) };  for (int i = 0; i < 2; ++i) {  if (CreateProcessW(exePath, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &startupInfo, &processInfos[i])) {  cout << "Процесс " << i + 1 << " создан\n";  processes[i] = processInfos[i].hProcess;  }  else {  cout << "[ОШИБКА] Не удалось создать процесс " << i + 1 << endl;  }  }  LARGE\_INTEGER firstTimeout, secondTimeout;  firstTimeout.QuadPart = -600000000LL;  secondTimeout.QuadPart = -1200000000LL;  SetWaitableTimer(FirstProcessTimer, &firstTimeout, 0, NULL, NULL, FALSE);  SetWaitableTimer(SecondProcessTimer, &secondTimeout, 0, NULL, NULL, FALSE);  DWORD dwWaitResult;  while (processes[0] || processes[1]) {  dwWaitResult = WaitForMultipleObjects(2, new HANDLE[2]{ FirstProcessTimer, SecondProcessTimer }, FALSE, INFINITE);  if (dwWaitResult == WAIT\_OBJECT\_0) {  if (processes[0]) {  cout << "Завершаем процесс 1\n";  TerminateProcess(processes[0], 0);  CloseHandle(processes[0]);  processes[0] = NULL;  }  }  else if (dwWaitResult == WAIT\_OBJECT\_0 + 1) {  if (processes[1]) {  cout << "Завершаем процесс 2\n";  TerminateProcess(processes[1], 0);  CloseHandle(processes[1]);  processes[1] = NULL;  }  }  }  CloseHandle(FirstProcessTimer);  CloseHandle(SecondProcessTimer);  cout << "Все процессы завершены\n";  system("pause");  return 0;  } |
| #include <iostream>  #include <Windows.h>  using namespace std;  int main() {  int num = 2;  while (true) {  bool isPrime = true;  for (int i = 2; i \* i <= num; ++i) {  if (num % i == 0) {  isPrime = false;  break;  }  }  if (isPrime) {  cout << num << endl;  }  num++;  }  return 0;  } |

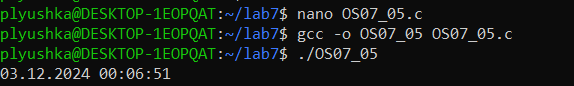




**Задание 05. Linux**

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***.

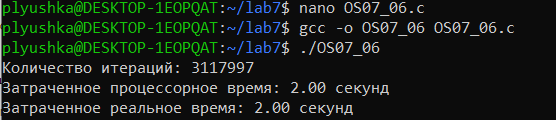
|  |
| --- |
| #include <stdio.h>  #include <time.h>  int main() {  time\_t raw\_time;  time(&raw\_time);  struct tm \*local\_time = localtime(&raw\_time);  printf("%02d.%02d.%04d %02d:%02d:%02d\n",  local\_time->tm\_mday,  local\_time->tm\_mon + 1,  local\_time->tm\_year + 1900,  local\_time->tm\_hour,  local\_time->tm\_min,  local\_time->tm\_sec);  return 0;  } |



**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.

|  |
| --- |
| #include <stdio.h>  #include <time.h>  int main() {  clock\_t start\_cpu\_time, end\_cpu\_time;  time\_t start\_real\_time, end\_real\_time;  double cpu\_time\_used, real\_time\_used;  long iterations = 0;  start\_cpu\_time = clock();  start\_real\_time = time(NULL);  while (1) {  iterations++;  end\_cpu\_time = clock();  end\_real\_time = time(NULL);  cpu\_time\_used = (double)(end\_cpu\_time - start\_cpu\_time) / CLOCKS\_PER\_SEC;  real\_time\_used = difftime(end\_real\_time, start\_real\_time);  if (cpu\_time\_used >= 2.0) {  printf("Количество итераций: %ld\n", iterations);  printf("Затраченное процессорное время: %.2f секунд\n", cpu\_time\_used);  printf("Затраченное реальное время: %.2f секунд\n", real\_time\_used);  break;  }  }  return 0;  } |



**Задание 07.** Ответьте на следующие вопросы

1. **Поясните понятие «социальное время» и почему оно не монотонное?**

**Социальное время** — это условное измерение времени, отражающее ритмы жизни общества, включая культурные, исторические и технологические аспекты. Оно не монотонное, поскольку его измерение изменялось в разные эпохи: корректировались календари (например, григорианская реформа 1582 года), а также из-за несовершенства физических единиц измерения, как увеличение длины солнечного года.

1. **Поясните понятие «эпоха Linux», назовите стартовую дату «эпохи Linux» и в каких единицах изменяется время?**

**Эпоха Linux (**POSIX-время) начинается с **1 января 1970 года в 00:00:00 UTC**. Время отсчитывается в секундах с этого момента, без учета високосных секунд. Для представления времени используется 32-битное целое число, из-за чего в 2038 году произойдет переполнение и счетчик уйдет в отрицательные значения. Коррекции времени (добавление координационных секунд) осуществляются Международной службой вращения Земли.

1. **Поясните понятие «Universal Coordinated Time (UCT)».**

**Universal Coordinated Time (UTC)** — это универсальное согласованное время, основанное на Гринвичском меридиане (ранее GMT). Оно определяется Международным бюро мер и весов (Париж) как усредненное значение атомного времени (TAI), рассчитываемое на основе данных более 50 лабораторий, оснащенных атомными часами (цезий-133). UTC отличается от солнечного времени на ~3 мс в сутки, с корректировкой при накопленной ошибке в 800 мс, включая добавление координационных секунд.

1. **Поясните понятия «относительное время» и «абсолютное время».**

**Относительное время** — это время, зависящее от наблюдателя и его системы отсчета. Например, в теории относительности время течет по-разному для объектов, движущихся с разной скоростью или находящихся в различных гравитационных полях.

**Абсолютное время** — это гипотетическое время, текущее равномерно и независимо от наблюдателя или внешних условий. Это концепция классической физики, предложенная Ньютоном, но опровергнутая теорией относительности Эйнштейна.

1. **Поясните понятие «тик».**

**Тик** — это минимальный дискретный интервал времени, который может быть измерен или обработан системой. В контексте вычислительной техники тик часто соответствует одному циклу работы процессора или единице времени, используемой для синхронизации системных часов. Например, в операционных системах тик определяет частоту обновления таймера.

1. **Поясните понятие «ожидающий таймер», перечислите типы таймеров, перечислите состояния, в которых может находится таймер.**

**Ожидающий таймер** (Waitable Timer) — это объект синхронизации в операционных системах Windows, который используется для выполнения задач через заданные интервалы времени или по наступлению определенного момента. Таймер может приостанавливать поток до наступления указанного времени или сигнализировать об этом другим потокам.

**Типы таймеров:**

- Однократный таймер (One-shot timer): Срабатывает один раз в указанное время.

- Периодический таймер (Periodic timer): Срабатывает через регулярные интервалы времени.

**Состояния таймера:**

- Неинициализированный (Uninitialized):Таймер только создан, но еще не запущен.

- Ожидание (Waiting): Таймер ожидает наступления заданного времени.

- Сигнальное состояние (Signaled): Таймер достиг указанного времени и уведомил об этом ожидающий поток.

- Перезагрузка (Resetting): При использовании периодического таймера он возвращается в состояние ожидания до следующего срабатывания.

1. **Перечислите типы часов, используемых в Linux, поясните их назначение.**

В Linux для работы со временем и датой используются различные типы часов и функции. Вот их основные типы и назначение:

**CLOCK\_REALTIME:** Представляет системное реальное время, которое соответствует текущему времени на часах. Может быть изменено администратором системы.

**CLOCK\_MONOTONIC:** Неизменяемое время, которое отсчитывается с момента запуска системы. Не зависит от изменений системных часов (например, корректировок вручную).

**CLOCK\_PROCESS\_CPUTIME\_ID:** Время, затраченное процессором на выполнение текущего процесса.

**CLOCK\_THREAD\_CPUTIME\_ID:** Время, затраченное процессором на выполнение текущего потока.

**CLOCK\_BOOTTIME:** Похож на CLOCK\_MONOTONIC, но включает время ожидания, например, при сне системы.

**CLOCK\_REALTIME\_ALARM** и **CLOCK\_BOOTTIME\_ALARM:** Используются для таймеров будильника.

Переменные и функции для работы со временем:

**time\_t:** Тип данных, представляющий время в секундах, прошедших с начала эпохи Unix (01.01.1970 00:00:00 UTC).

**Tm:** Структура для хранения разобранного времени: год, месяц, день, часы, минуты, секунды и т.д.

**time():** Возвращает текущее время как time\_t.

**ctime():** Преобразует time\_t в строку, представляющую локальное время. Не потокобезопасна.

**ctime\_r():** Потокобезопасная версия ctime().

**gmtime():** Преобразует time\_t в структуру tm для времени в UTC.

**gmtime\_r():** Потокобезопасная версия gmtime().

**localtime():** Преобразует time\_t в структуру tm для локального времени.

**clock():** Возвращает процессорное время, затраченное текущим процессом.

Эти функции и структуры позволяют эффективно работать с различными аспектами времени, включая преобразования между локальным и UTC временем, создание таймеров и измерение времени выполнения.

1. **Поясните назначение констант HZ, CLOCKS\_PER\_SEC.**

**HZ:** Константа, определяющая количество тиков системного таймера в секунду. Используется ядром Linux для определения временных интервалов.

Значение HZ зависит от архитектуры и конфигурации системы:

Обычно 100, 250, или 1000 (тик каждые 10, 4, или 1 мс соответственно).

Чем выше значение HZ, тем точнее временные интервалы, но тем больше нагрузка на процессор.

**CLOCKS\_PER\_SEC:** Константа из стандартной библиотеки C, задающая количество «тиков процессорного времени» в одной секунде. Всегда равна 1000000 (1 миллион), что соответствует микросекундам, независимо от системы. Используется для преобразования значений, возвращаемых функцией clock(), в секунды:

double cpu\_time = (double)clock() / CLOCKS\_PER\_SEC;

Различия и применение:

**HZ** управляет системой на уровне ядра (планировщик процессов, задержки).

**CLOCKS\_PER\_SEC** относится к пользовательским приложениям и времени выполнения процесса.

Обе константы используются для измерения и управления временем в различных уровнях операционной системы.