Лабораторная работа 07

Компьютерное время

OC, ПОИТ-3

**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***.

|  |
| --- |
| 1. #include <iostream> 2. int main() 3. { 4. time\_t t; 5. tm tm; 6. time(&t); 7. localtime\_s(&tm, &t); 8. printf( 9. "%d.%d.%d %d:%d:%d", 10. tm.tm\_mday, 11. tm.tm\_mon + 1, 12. tm.tm\_year + 1900, 13. tm.tm\_hour, 14. tm.tm\_min, 15. tm.tm\_sec 16. ); 17. } |
|  |

**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.

|  |
| --- |
| 1. #include <Windows.h> 2. #include <iostream> 3. #include <ctime> 4. using namespace std; 5. int main() 6. { 7. clock\_t start = clock(); 8. int i = 0; 9. bool flag5 = true, flag10 = true; 10. while (true) 11. { 12. i++; 13. if ((clock() - start) / CLOCKS\_PER\_SEC == 5 && flag5) { 14. cout << "Iterations after 5s: " << i << '\n'; 15. flag5 = false; 16. } 17. if ((clock() - start) / CLOCKS\_PER\_SEC == 10 && flag10) { 18. cout << "Iterations after 10s: " << i << '\n'; 19. flag10 = false; 20. } 21. if ((clock() - start) / CLOCKS\_PER\_SEC == 15) { 22. cout << "Iterations after 15s: " << i << '\n'; 23. break; 24. } 25. } 26. return 0; 27. } |
|  |

**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.

|  |
| --- |
| 1. #include <iostream> 2. #include <Windows.h> 3. int main() { 4. HANDLE hTimer = CreateWaitableTimer(nullptr, TRUE, L"Timer"); 6. const LONG period = 1000; 7. const LONG outputInterval = 3000; 8. const LONG totalDuration = 15000; 9. LARGE\_INTEGER dueTime; //для хранения времени срабатывания таймера 10. dueTime.QuadPart = -10 \* 1000 \* period; 11. SetWaitableTimer(hTimer, &dueTime, period, nullptr, nullptr, FALSE); 12. int iteration = 0; 13. DWORD lastOutputTime = GetTickCount(); 14. DWORD startTime = GetTickCount(); 15. while (true) { 16. DWORD waitResult = WaitForSingleObject(hTimer, INFINITE); //ожидание срабатывания таймера 17. if (waitResult == WAIT\_OBJECT\_0) { 18. iteration++; 19. } 20. DWORD currentTime = GetTickCount(); 21. if (currentTime - lastOutputTime >= outputInterval) { 22. std::cout << "Iteration count: " << iteration << std::endl; 23. lastOutputTime = currentTime; 24. } 25. DWORD elapsedTime = currentTime - startTime; //вычисляет общее время выполнения программы 26. if (elapsedTime >= totalDuration) { 27. std::cout << "Total iteration count: " << iteration << std::endl; 28. break; 29. } 30. } 31. CancelWaitableTimer(hTimer); 32. CloseHandle(hTimer); 33. return 0; 34. } |
|  |

**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.

|  |
| --- |
| **OS07\_04**  #include <iostream>  #include <Windows.h>  int main() {  setlocale(LC\_ALL, "rus");  STARTUPINFO si1, si2; //параметры запуска  PROCESS\_INFORMATION pi1, pi2; //инфа  ZeroMemory(&si1, sizeof(si1));  si1.cb = sizeof(si1);  ZeroMemory(&pi1, sizeof(pi1));  BOOL bRes = CreateProcessW(L"C:\\Лабы\\Операционные системы\\Лабы\\Лаба7\\lab7\\x64\\Debug\\OS07\_04x.exe",  NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si1, &pi1);  if (!bRes) {  printf("Error %d\n", GetLastError());  }  ZeroMemory(&si2, sizeof(si2));  si2.cb = sizeof(si2);  ZeroMemory(&pi2, sizeof(pi2));  BOOL bRes1 = CreateProcessW(L"C:\\Лабы\\Операционные системы\\Лабы\\Лаба7\\lab7\\x64\\Debug\\OS07\_04x\_2.exe",  NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si2, &pi2);  if (!bRes1) {  printf("Error %d\n", GetLastError());  }  WaitForSingleObject(pi2.hProcess, INFINITE);  WaitForSingleObject(pi1.hProcess, INFINITE);  CloseHandle(pi1.hProcess);  CloseHandle(pi2.hProcess);  } |
| **OS07\_04x**  #include <iostream>  #include <ctime>  #include <Windows.h>  using namespace std;  bool isPrime(int num) {  if (num <= 1) return false;  for (int i = 2; i \* i <= num; i++) {  if (num % i == 0) return false;  }  return true;  }  int main() {  setlocale(LC\_ALL, "rus");  clock\_t start = clock();  int count = 0;  int number = 2;  cout << "Пронумерованный ряд простых чисел:\n";  while (true) {  int elapsedSeconds = (clock() - start) / CLOCKS\_PER\_SEC;  if (isPrime(number)) {  cout << count + 1 << ": " << number << ", Time elapsed: " << elapsedSeconds << " seconds\n";  count++;  }  number++;  if (elapsedSeconds == 60) {  break;  }  Sleep(100);  }  Sleep(1000);  cout << "Общее количество простых чисел: " << count << '\n';  } |
| **OS07\_04x\_2**  #include <iostream>  #include <ctime>  #include <Windows.h>  using namespace std;  bool isPrime(int num) {  if (num <= 1) return false;  for (int i = 2; i \* i <= num; i++) {  if (num % i == 0) return false;  }  return true;  }  int main() {  setlocale(LC\_ALL, "rus");  clock\_t start = clock(); //сохранение текущего времени  int iteration = 0;  int number = 2; //первое простое число  cout << "Ряд простых чисел:\n";  while (true) {  int elapsedSeconds = (clock() - start) / CLOCKS\_PER\_SEC; //подсчёт прошедшего времени с учётом перевода тактов в секунды  if (isPrime(number)) {  cout << iteration + 1 << ": " << number << ", Time: " << elapsedSeconds << " seconds\n";  iteration++;  }  number++;  if (elapsedSeconds == 120) {  break;  }  Sleep(100);  }  Sleep(1000);  cout << "Общее количество простых чисел: " << iteration << '\n';  } |
|  |

**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***.

|  |
| --- |
| 1. #include <stdio.h> 2. #include <time.h> 3. int main() { 4. // Получаем текущее время 5. time\_t currentTime; 6. struct tm \*localTimeInfo; 7. time(&currentTime);//инициализация; заполнение текущим временем 8. localTimeInfo = localtime(&currentTime); 9. // Форматируем и выводим дату и время 10. printf("%02d.%02d.%04d %02d:%02d:%02d\n", 11. localTimeInfo->tm\_mday, localTimeInfo->tm\_mon + 1, localTimeInfo->tm\_year + 1900, 12. localTimeInfo->tm\_hour, localTimeInfo->tm\_min, localTimeInfo->tm\_sec); 13. return 0; 14. } |
|  |

**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.

|  |
| --- |
| 1. #include <stdio.h> #include <time.h> #include <unistd.h>  int main() {  struct timespec start\_pTime, current\_pTime, start\_rTime, current\_rTime; //timespec (секунды и наносекунды)  long iteration = 0;  double all\_pTime = 0.0, all\_rTime = 0.0;   // Начало замера реального времени  clock\_gettime(CLOCK\_REALTIME, &start\_rTime);  // Начало замера процессорного времени  clock\_gettime(CLOCK\_PROCESS\_CPUTIME\_ID, &start\_pTime);   printf("Запуск бесконечного цикла...\n");   while (1) {  iteration++;   // Замер текущего процессорного времени  clock\_gettime(CLOCK\_PROCESS\_CPUTIME\_ID, &current\_pTime);  all\_pTime = (current\_pTime.tv\_sec - start\_pTime.tv\_sec) +  (current\_pTime.tv\_nsec - start\_pTime.tv\_nsec) / 1e9;   // Проверяем, прошло ли 2 секунды процессорного времени  if (all\_pTime >= 2.0) {  printf("Прошло 2 секунды процессорного времени.\n");  printf("Количество итераций: %ld\n", iteration);  break;  }  }   // Замер текущего реального времени  clock\_gettime(CLOCK\_REALTIME, &current\_rTime);  all\_rTime = (current\_rTime.tv\_sec - start\_rTime.tv\_sec) +  (current\_rTime.tv\_nsec - start\_rTime.tv\_nsec) / 1e9;   printf("Программа завершена.\n");  printf("Общее реальное время: %.2f сек\n", all\_rTime);  printf("Процессорное время: %.2f сек\n", all\_pTime);   return 0; } |
|  |

**Задание 07.** Ответьте на следующие вопросы

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

**Социальное время** – время, измеряемое не единицами продолжительности (минута, час, год), а абстрактными мерами (эпоха, поколение, жизнь). Социальное время отражает не то, сколько длится событие, а то, как ощущается его продолжительность.

**Немонотонность** социального времени означает, что оно не всегда движется в одном направлении с постоянной скоростью. Например, в некоторых случаях время может "тянуться" (например, во время ожидания), а в других – "лететь" (например, на празднике). Это приводит к субъективному восприятию времени, которое может меняться в зависимости от обстоятельств.

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

**Эпоха «Linux»** – временной интервал, с момента, когда в системе Linux была внедрена поддержка времени. Стартовая дата - (00:00:00 UTC) 1 января 1970 года. Отсчёт происходит в секундах.

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

**UCT (всемирное координированное время)** – стандартное время, используемое во всем мире. Оно основано на атомных часах и служит основой для определения времени по всей планете. UCT не зависит от часовых поясов и не изменяется на летнее/зимнее время. Это время используется в навигации, науке и других областях, требующих точной синхронизации.

1. Поясните понятия «относительное время» и «абсолютное время».  
   **Абсолютное** **время** – фиксированная мера времени.

**Относительное** **время** – измеряется относительно другого события.

1. Поясните понятие «тик».

**Тик** – равный интервал времени, через который инициируются прерывания. Базовая единица измерения времени.

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

**Ожидающий таймер** — это механизм, который позволяет программе "ожидать" определенное время, прежде чем продолжить выполнение.

**Два типа таймеров ожидания**: сброс вручную и синхронизация. Таймер любого типа также может быть периодическим.

**Таймер сброса вручную -** состояние которого остается сигнальным до вызова SetWaitableTimer, чтобы установить новое время выполнения.

**Таймер синхронизации -** состояние которого остается сигнальным до тех пор, пока поток не завершит операцию ожидания в объекте таймера.

**Периодический таймер** - который повторно активируется каждый раз, когда истечет указанный период, пока таймер не будет сброшен или отменен.

**Состояния таймера:**

* **Запущен**: таймер активен и отсчитывает время.
* **Ожидающий**: таймер завершил отсчет времени, но еще не сработал.
* **Сработал**: таймер завершил свою работу и вызывается соответствующий обработчик.

1. Перечислите типы часов, используемых в Linux, поясните их назначение.
   * **CLOCK\_REALTIME**: общесистемные часы реального времени, видимые для всех процессов, работающих в системе. Часы измеряют количество времени в секундах и наносекундах с начала эпохи. Так как это время базируются на времени настенных часов, оно может быть изменено.
   * **CLOCK\_MONOTONIC**: время непрерывной работы системы, видимое всем процессам в системе. В Linux оно измеряется как количество времени в секундах и наносекундах после загрузки системы. Это время не может быть изменено каким-либо процессом.
   * **CLOCK\_PROCESS\_CPUTIME\_ID**: часы, измеряющие время работы процесса. Время текущего процесса, потраченное на выполнение в системе, измеряется в секундах и наносекундах. Это время может быть изменено.
   * **CLOCK\_THREAD\_CPUTIME\_ID**: То же, что и CLOCK\_PROCESS\_CPUTIME\_ID, но для текущего потока.
2. Поясните назначение констант HZ, CLOCKS\_PER\_SEC.

**HZ** – частота системного таймера (обычно 100 или 1000), параметр ядра.

**CLOCKS\_PER\_SEC** – количество тактов процессора в секунду.