Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

«Комсомольский-на-Амуре государственный университет»

Факультет компьютерных технологий

Кафедра «МОП ЭВМ»

ЛАБОРАТОРНАЯ РАБОТА 4

по дисциплине «Операционные системы и оболочки»

## Процессы, потоки и нити в операционной системе Windows

Студент группы 6ИСб-1 И. Нозимзода

Преподаватель В.Я Столяров

2019

**Цель работы**: Изучить теоретические вопросы управления потоками в ОС Windows и освоить приемы практической реализации этого управления с использованием системных функций.

**Задание:**

Пять потоков генерируют случайные целые числа в диапазоне от 1 до 49. Шестой поток – анализирует эти числа. Если появляется пара одинаковых чисел – выводится сообщение ПАРА и номера потоков этой пары. Если три числа одинаковы – ТРИ, Четыре – ЧЕТЫРЕ и пять – ПЯТЬ. Через 60 секунд работы программа завершается. Если не было ни одного совпадения – выводится об этом сообщение.

Выполнение:

При создании процесса в системе появляется новый программный поток, принадлежащий этому процессу. Вначале любой только что созданный процесс обладает лишь одним потоком. Этот поток может создавать новые потоки, а эти новые потоки, в свою очередь, могут создавать другие новые потоки. Процесс продолжает свое существование до тех пор, пока в его владении находится по крайней мере один программный поток (или до тех пор, пока не произойдет что-либо, в результате чего весь процесс или все задание не прекратит работу, напри­мер, обращение к функции **TerminateProcess**).

Зачем процессу несколько потоков? Потоки могут выполнять какие-либо действия в фоновом режиме относительно вашей основной программы. Например, вы можете создать новый программный поток, который будет в фоновом режи­ме осуществлять вывод информации на принтер. Потоки удобно использовать также в случае, если блокирование или подвисание какой-либо процедуры не должно стать причиной нарушений функционирования основной программы. Например, в то время как основная программа выполняет сложные математичес­кие вычисления, отдельный программный поток может осуществлять обмен дан­ными через асинхронный последовательный канал связи (например, через модем). В случае замедления передачи данных через канал или в случае подвисания модема функционирование основной программы не будет нарушено.

Базовый системный вызов, предназначенный для создания потока, — это **CreateThread**. Однако на практике программисты фактически никогда его не использу­ют. Почему? Потому что этот вызов создает абсолютно пустой поток, который в изначальном виде не может использоваться вашей библиотекой (С или MFC).

Программа запускается из командной строки с параметром, содержащим путь к файлу, который необходимо запустить.

Программа запускает пять потоков генерации чисел, один потом в котором работает таймер и один поток для нахождения одинаковых пар чисел и вывода этой информации на экран.

Код программы приведен в листинге 1.

Листинг 1 – Текст файла prog1\_cpp

|  |
| --- |
| // compile with: /EHsc  #include <iostream>  #include <windows.h>  #include <string>  #include <random>  // WinAPI function to create Thread (\_beginthread, \_beginthreadex, CreateThread, etc.)  #include <process.h>  // C++11 standard function to create portable Thread  // #include <thread>  using namespace std;  ////// GLOBAL VARIABLE //////  // Таймер, потоки генерирующие новые случайные значения будут ждать завершение таймера  HANDLE hTimer;  DWORD exitcode;  // Массив, куда потоки будут записывать значения  size\_t arrRandomValue[5];  // Обьекты рандомизации  default\_random\_engine generator;  uniform\_int\_distribution<int> distribution(1, 49);  ////////////////////////////  // Прототипы функций  string getMaxCountEqual();  // Таймер  unsigned \_\_stdcall timer(void\*)  {  cout << "Таймер запущен. Осталось 60 секунд." << endl;  Sleep(30000);  cout << "Осталось 30 секунд." << endl;  Sleep(30000);  cout << "Время вышло. Таймер выключен." << endl;  return 1;  }  // Потоки генирирующие случайные целые числа в диапазоне от 1 до 49  unsigned \_\_stdcall thread1(void\*)  {  cout << "Поток 1 запущен" << endl;  while ((GetExitCodeThread(hTimer, &exitcode) && exitcode == STILL\_ACTIVE))  {  arrRandomValue[0] = distribution(generator);  Sleep(50);  }  return 1;  }  unsigned \_\_stdcall thread2(void\*)  {  cout << "Поток 2 запущен" << endl;  while ((GetExitCodeThread(hTimer, &exitcode) && exitcode == STILL\_ACTIVE))  {  arrRandomValue[1] = distribution(generator);  Sleep(50);  }  return 1;  }  unsigned \_\_stdcall thread3(void\*)  {  cout << "Поток 3 запущен" << endl;  while ((GetExitCodeThread(hTimer, &exitcode) && exitcode == STILL\_ACTIVE))  {  arrRandomValue[2] = distribution(generator);  Sleep(50);  }  return 1;  }  unsigned \_\_stdcall thread4(void\*)  {  cout << "Поток 4 запущен" << endl;  while ((GetExitCodeThread(hTimer, &exitcode) && exitcode == STILL\_ACTIVE))  {  arrRandomValue[3] = distribution(generator);  Sleep(50);  }  return 1;  }  unsigned \_\_stdcall thread5(void\*)  {  cout << "Поток 5 запущен" << endl;  while ((GetExitCodeThread(hTimer, &exitcode) && exitcode == STILL\_ACTIVE))  {  arrRandomValue[4] = distribution(generator);  Sleep(50);  }  return 1;  }  // Анализ чисел  unsigned \_\_stdcall comparison(void\*)  {  cout << "Поток анализа чисел запущен" << endl;  while ((GetExitCodeThread(hTimer, &exitcode) && exitcode == STILL\_ACTIVE))  {  string count = getMaxCountEqual();  if (count != "") cout << count << endl;  Sleep(200);  }  return 1;  }  // Количество одинаковых чисел в массиве  string getMaxCountEqual()  {  // Делаем копию массива. Мы не можем работать с оригиналом, тк он постоянно изменяться  size\_t arrRandomValue\_Temp[5];  memcpy(arrRandomValue\_Temp, arrRandomValue, sizeof(size\_t) \* 5);  string res = "";  string res\_temp = "";  // Количество повторений числа  size\_t maxCount = 1;  size\_t count = 0;  for (size\_t i = 0; i < 5; i++, count = 0, res\_temp = "")  {  // Нахождение количество повторений для числа  for (size\_t j = 0; j < 5; j++)  {  if (arrRandomValue\_Temp[i] == arrRandomValue\_Temp[j]) {  count++;  res\_temp += " " + to\_string(j + 1);  }  }  if (count > maxCount) {  maxCount = count;  res = res\_temp;  };  };  if (maxCount == 2) res = "ПАРА Потоки:" + res;  else if (maxCount == 3) res = "ТРИ Потоки:" + res;  else if (maxCount == 4) res = "ЧЕТЫРЕ Потоки:" + res;  else if (maxCount == 5) res = "ПЯТЬ Потоки:" + res;  return res;  }  int main()  {  setlocale(LC\_ALL, "rus");  system("color 0A");  hTimer = (HANDLE)\_beginthreadex(NULL, 0, &timer, NULL, 0, NULL);  Sleep(1000);  HANDLE hThread[6];  hThread[0] = (HANDLE)\_beginthreadex(NULL, 0, &thread1, NULL, 0, NULL);  hThread[1] = (HANDLE)\_beginthreadex(NULL, 0, &thread2, NULL, 0, NULL);  hThread[2] = (HANDLE)\_beginthreadex(NULL, 0, &thread3, NULL, 0, NULL);  hThread[3] = (HANDLE)\_beginthreadex(NULL, 0, &thread4, NULL, 0, NULL);  hThread[4] = (HANDLE)\_beginthreadex(NULL, 0, &thread5, NULL, 0, NULL);  Sleep(1000);  hThread[5] = (HANDLE)\_beginthreadex(NULL, 0, &comparison, NULL, 0, NULL);  // Ожидание завершения всех потоков  WaitForMultipleObjects(6, hThread, TRUE, INFINITE);  cout << "Все потоки завершили работу." << endl;  system("pause");  return 0;  } |

Результат работы программы показан на рисунках 1,2.

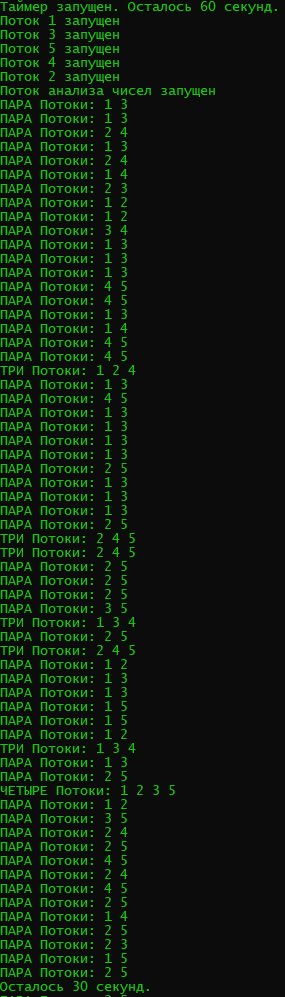


Рисунок 1 – Работа программы

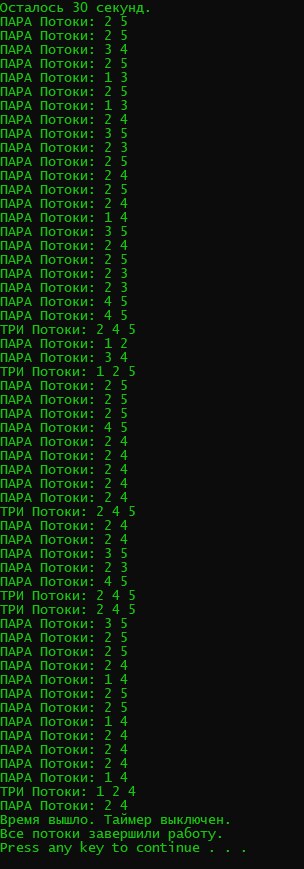


Рисунок 2 – Завершение программы

**Список использованных источников**

1. Саймон, Р. Windows 2000 API Энцеклопедия программиста / Р. Саймон. - М: DiaSoft, 2002