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 А.М. Закусило

Преподаватель В.А Тихомиров

2018

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

**Задание:**

Программа создает три потока, в которых делятся друг на друга случайные числа пока их число не станет 100000 или не произойдет деление на 0. Анализируется код возврата.

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 создает три потока, в которых делятся друг на друга случайные числа пока их число не станет 100000 или не произойдет деление на 0.

Анализируется код возврата. Результаты выводятся на экран. Так же на экран выводится метка достижения каждого потока середины своих вычислений приведен в листинге 1.

Листинг 1 – Текст файла prog1\_cpp

|  |
| --- |
| // compile with: /EHsc  #include <iostream>  #include <windows.h>  // 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;  double container = 0;  unsigned \_\_stdcall thread(void \*)  {  cout << "Поток запущен\n";  try  {  while (container < 100000)  {  //int max value  int randOne = rand() % 100000;  int randTwo = rand() % 1000;  if (randTwo == 0) throw -1;  container += (double)randOne / (double)randTwo;  if (container > 100000)  {  cout << "Контейнер заполнен\n";  }  Sleep(100);  }  }  catch (...)  {  cout << "Ошибка в потоке\n";  }  return 1;  }  int main()  {  setlocale(LC\_ALL, "rus");  system("color 0A ");  HANDLE hThread[3];  hThread[0] = (HANDLE)\_beginthreadex(NULL, 0, &thread, NULL, 0, NULL);  hThread[1] = (HANDLE)\_beginthreadex(NULL, 0, &thread, NULL, 0, NULL);  hThread[2] = (HANDLE)\_beginthreadex(NULL, 0, &thread, NULL, 0, NULL);  //WaitForSingleObject(hThread, INFINITE);  //WaitForMultipleObjects(3, hThread, TRUE, INFINITE);  //Analyze return code  DWORD exitcode;  while ((GetExitCodeThread(hThread[0], &exitcode) && exitcode == STILL\_ACTIVE) ||  (GetExitCodeThread(hThread[1], &exitcode) && exitcode == STILL\_ACTIVE) ||  (GetExitCodeThread(hThread[2], &exitcode) && exitcode == STILL\_ACTIVE)) {};  cout << "Все потоки завершили работу\n";  system("pause");  return 0;  } |

Результат работы программы показан на рисунке 1.

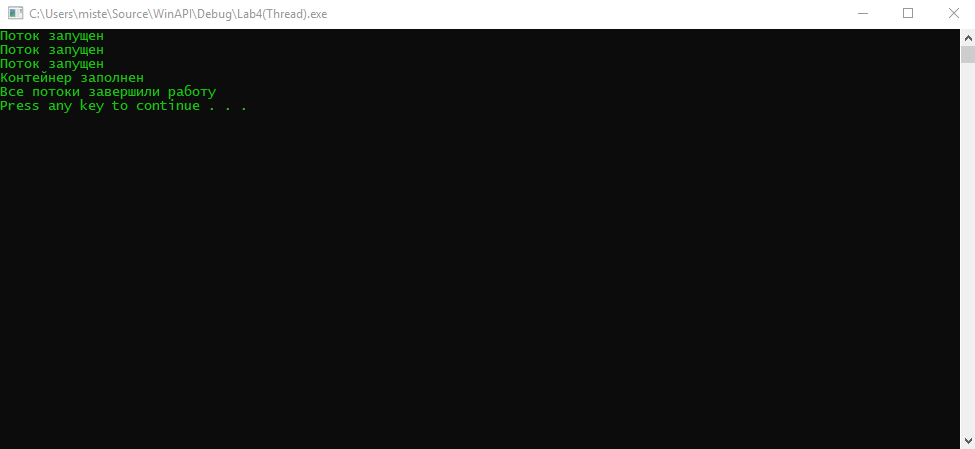


Рисунок 1 – Работа программы

**Список использованных источников**

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