Лабораторная работа 05

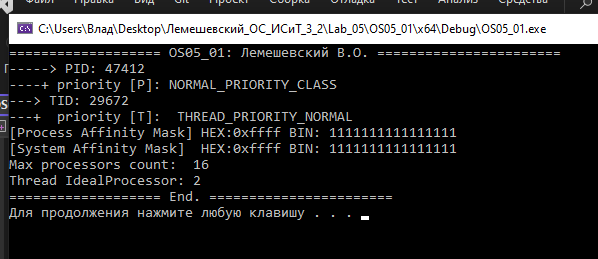
OC, ПОИТ-3

**Задание 01**

1. Разработайте консольное Windows-приложение **OS05\_01** на языке С++, выводящее на консоль следующую информации:

* идентификатор текущего процесса;
* идентификатор текущего (main) потока;
* приоритет (приоритетный класс) текущего процесса;
* приоритет текущего потока;
* маску (affinity mask) доступных процессу процессоров в двоичном виде;
* количество процессоров доступных процессу;
* процессор, назначенный текущему потоку.

|  |
| --- |
| * #include <Windows.h> * #include <iostream> * #include <bitset> * using namespace std; * void printProcessPrty(HANDLE h) { * DWORD prty = GetPriorityClass(h); * cout << "-----> PID: " << GetCurrentProcessId() << endl; * switch (prty) { * case IDLE\_PRIORITY\_CLASS: cout << "----+ priority [P]: IDLE\_PRIORITY\_CLASS\n"; break; * case BELOW\_NORMAL\_PRIORITY\_CLASS: cout << "----+ priority [P]: BELOW\_NORMAL\_PRIORITY\_CLASS\n"; break; * case NORMAL\_PRIORITY\_CLASS: cout << "----+ priority [P]: NORMAL\_PRIORITY\_CLASS\n"; break; * case ABOVE\_NORMAL\_PRIORITY\_CLASS: cout << "----+ priority [P]: ABOVE\_NORMAL\_PRIORITY\_CLASS\n"; break; * case HIGH\_PRIORITY\_CLASS: cout << "----+ priority [P]: HIGH\_PRIORITY\_CLASS\n"; break; * case REALTIME\_PRIORITY\_CLASS: cout << "----+ priority [P]: REALTIME\_PRIORITY\_CLASS\n"; break; * default: cout << "----+ priority [P]: [NULL] \n"; break; * } * } * void printThreadPrty(HANDLE h) { * DWORD prty = GetThreadPriority(h); * cout << "---> TID: " << GetCurrentThreadId() << endl; * switch (prty) { * case THREAD\_PRIORITY\_LOWEST: cout << "---+ priority [T]: THREAD\_PRIORITY\_LOWEST\n"; break; * case THREAD\_PRIORITY\_BELOW\_NORMAL: cout << "---+ priority [T]: THREAD\_PRIORITY\_BELOW\_NORMAL\n"; break; * case THREAD\_PRIORITY\_NORMAL: cout << "---+ priority [T]: THREAD\_PRIORITY\_NORMAL\n"; break; * case THREAD\_PRIORITY\_ABOVE\_NORMAL: cout << "---+ priority [T]: THREAD\_PRIORITY\_ABOVE\_NORMAL\n"; break; * case THREAD\_PRIORITY\_HIGHEST: cout << "---+ priority [T]: THREAD\_PRIORITY\_HIGHEST\n"; break; * case THREAD\_PRIORITY\_IDLE: cout << "---+ priority [T]: THREAD\_PRIORITY\_IDLE\n"; break; * case THREAD\_PRIORITY\_TIME\_CRITICAL: cout << "---+ priority [T]: THREAD\_PRIORITY\_TIME\_CRITICAL\n"; break; * default: cout << "---+ priority [T]: [NULL] \n"; break; * } * } * void PrintAffinityMask(HANDLE h) { * DWORD\_PTR pa = NULL, sa = NULL, icpu = -1; * char buf[10]; * if (!GetProcessAffinityMask(h, &pa, &sa)) throw "GetProcessAffinityMask [Exception]"; * cout << "[Process Affinity Mask] HEX:" << showbase << hex << pa << " BIN: " << bitset<sizeof(sa) \* 2>(pa) << endl; * cout << "[System Affinity Mask] HEX:" << showbase << hex << sa << " BIN: " << bitset<sizeof(sa) \* 2>(sa) << endl; * } * void PrintTotalProcessors(HANDLE h1, HANDLE h2) { * DWORD\_PTR pa = NULL, sa = NULL, icpu = -1; * SYSTEM\_INFO sys\_info; * if (!GetProcessAffinityMask(h1, &pa, &sa)) throw "GetProcessAffinityMask"; * GetSystemInfo(&sys\_info); * cout << "Max processors count: " << dec << sys\_info.dwNumberOfProcessors << "\n"; * icpu = SetThreadIdealProcessor(h2, MAXIMUM\_PROCESSORS); * cout << "Thread IdealProcessor: " << dec << icpu << "\n"; * } * int main() { * SetConsoleCP(1251); * SetConsoleOutputCP(1251); * cout << "=================== OS05\_01: Лемешевский В.О. =======================\n"; * HANDLE hp = GetCurrentProcess(); // текущий процесс * HANDLE ht = GetCurrentThread(); // текущий поток * printProcessPrty(hp); * printThreadPrty(ht); * PrintAffinityMask(hp); * PrintTotalProcessors(hp, ht); * cout << "=================== End. =======================\n"; * system("pause"); * return 0; * } |



**Задание 02**

1. Разработайте консольное Windows-приложение **OS05\_02x,** выполняющее цикл в 1млн итераций.
2. Каждая итерация осуществляет задержку на 200 мс через каждые 1тыс итераций и выводит следующую информацию:

* номер итерации;
* идентификатор процесса;
* идентификатор потока;
* класс приоритета процесса;
* приоритет потока:
* номер назначенного процессора.

|  |
| --- |
| * int main() { * SetConsoleCP(1251); * SetConsoleOutputCP(1251); * HANDLE hp = GetCurrentProcess(); // текущий процесс * HANDLE ht = GetCurrentThread(); // текущий поток * for (int i = 0; i < 1000000; i++) { * if (i % 1000 == 0) { * cout << "=================== OS05\_02x: Лемешевский В.О. =======================\n"; * cout << "I: " << dec << i << endl; * printProcessPrty(hp); * printThreadPrty(ht); * PrintAffinityMask(hp); * cout << "Номер назначенного процессора: " << GetCurrentProcessorNumber() << endl; * cout << "=================== <> =======================\n"; * Sleep(200); * } * } * cout << "=================== End. =======================\n"; * system("pause"); * return 0; * } |

1. Разработайте консольное Windows-приложение **OS05\_02,** принимающее следующие параметры:

* P1: целое число, задающее маску доступности процессоров (affinity mask);
* P2: целое число, задающее класс приоритета первого дочернего процесса;
* P3: целое число, задающее класс приоритета второго дочернего процесса.

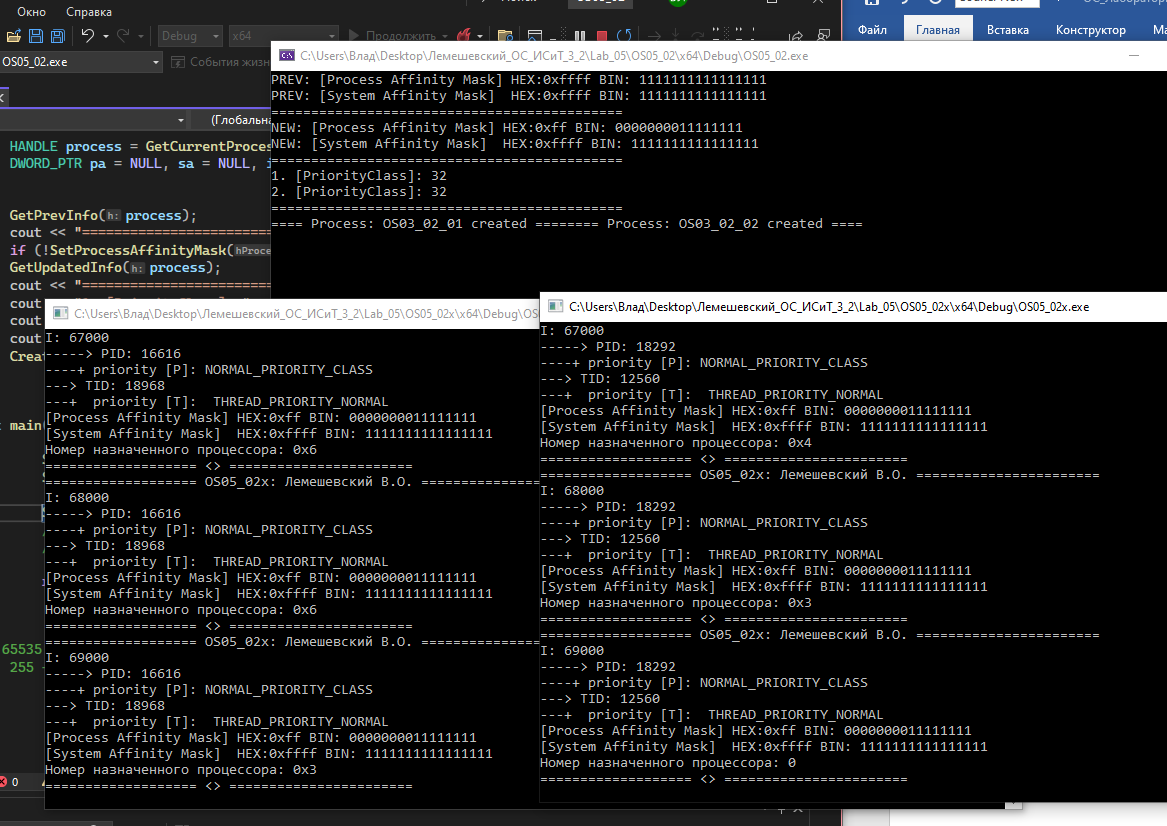
1. Приложение **OS05\_02** должно вывести в свое консольное окно заданные параметры и запустить два одинаковых дочерних процесса **OS05\_02x,** осуществляющих вывод в отдельные консольные окна и имеющих заданные в параметрах приоритеты.

|  |
| --- |
| 1. DWORD getProcessPrty(int p) { 2. switch (p) { 3. case 1: return IDLE\_PRIORITY\_CLASS; 4. case 2: return BELOW\_NORMAL\_PRIORITY\_CLASS; 5. case 3: return NORMAL\_PRIORITY\_CLASS; 6. case 4: return ABOVE\_NORMAL\_PRIORITY\_CLASS; 7. case 5: return HIGH\_PRIORITY\_CLASS; 8. case 6: return REALTIME\_PRIORITY\_CLASS; 9. default: throw "----+ priority [P]: [NULL] \n"; 10. } 11. } 12. void GetPrevInfo(HANDLE h) { 13. DWORD\_PTR pa = NULL, sa = NULL, icpu = -1; 14. if (!GetProcessAffinityMask(h, &pa, &sa)) throw "GetProcessAffinityMask [Exception]"; 15. cout << "PREV: [Process Affinity Mask] HEX:" << showbase << hex << pa << " BIN: " << bitset<sizeof(sa) \* 2>(pa) << endl; 16. cout << "PREV: [System Affinity Mask] HEX:" << showbase << hex << sa << " BIN: " << bitset<sizeof(sa) \* 2>(sa) << endl; 17. } 18. void GetUpdatedInfo(HANDLE h) { 19. DWORD\_PTR pa = NULL, sa = NULL, icpu = -1; 20. if (!GetProcessAffinityMask(h, &pa, &sa)) throw "GetProcessAffinityMask [Exception]"; 21. cout << "NEW: [Process Affinity Mask] HEX:" << showbase << hex << pa << " BIN: " << bitset<sizeof(sa) \* 2>(pa) << endl; 22. cout << "NEW: [System Affinity Mask] HEX:" << showbase << hex << sa << " BIN: " << bitset<sizeof(sa) \* 2>(sa) << endl; 23. } 24. void CreateXProcesses(int prty1 , int prty2) { 25. LPCWSTR f1 = L"C:\\Users\\Влад\\Desktop\\Лемешевский\_ОС\_ИСиТ\_3\_2\\Lab\_05\\OS05\_02x\\x64\\Debug\\OS05\_02x.exe"; // ссыкла на исполняемые модули 26. LPCWSTR f2 = L"C:\\Users\\Влад\\Desktop\\Лемешевский\_ОС\_ИСиТ\_3\_2\\Lab\_05\\OS05\_02x\\x64\\Debug\\OS05\_02x.exe"; 27. STARTUPINFO si1, si2; // структура для определения внешнего вида окна для нового процесса 28. PROCESS\_INFORMATION pi1, pi2; // структура с информацией о процессе и его потоке 29. ZeroMemory(&si1, sizeof(STARTUPINFO)); 30. si1.cb = sizeof(STARTUPINFO); // size 31. ZeroMemory(&si2, sizeof(STARTUPINFO)); 32. si2.cb = sizeof(STARTUPINFO); 33. if ( // Создание дочернего процесса 34. CreateProcess( 35. f1, // имя исполняемого модуля 36. NULL,// командная строка параметры 37. NULL, // может ли возвращаемый дескриптор быть унаследован дочерними процессами 38. NULL, // может ли возвращаемый дескриптор быть унаследован дочерними потоками 39. FALSE, // каждый возвращаемый дескриптор наследуется дочерним процессом 40. CREATE\_NEW\_CONSOLE | getProcessPrty(prty1), // флаги создания, управляющие приоритетом и параметрами процесса; конкретно этот создает новый инстанс консоли 41. NULL, // блок конфигурации нового процесса 42. NULL, // олный путь дочернего процесса 43. &si1, // внешний вид окна 44. &pi1 // дескрипторы процесса и первичного потока 45. ) 46. ) { 47. cout << "==== Process: OS03\_02\_01 created ===="; 48. } 49. else { 50. cout << "==== Process: OS03\_02\_01 not created ===="; 51. } 52. if ( 53. CreateProcess( 54. f2, 55. NULL, 56. NULL, 57. NULL, 58. FALSE, 59. CREATE\_NEW\_CONSOLE | getProcessPrty(prty2), 60. NULL, 61. NULL, 62. &si2, 63. &pi2 64. ) 65. ) { 66. cout << "==== Process: OS03\_02\_02 created ====\n" << endl; 67. } 68. else { 69. cout << "==== Process: OS03\_02\_02 not created ====\n" << endl; 70. } 71. WaitForSingleObject(pi1.hProcess, INFINITE); // Бесконечное ожидание завершения дочернего процесса пока оба не завершаться 72. WaitForSingleObject(pi2.hProcess, INFINITE); 73. CloseHandle(pi1.hProcess); 74. CloseHandle(pi2.hProcess); // Закритие происходит вручную/явно 75. } 76. void ShowResult(string par1 , string par2 ,string par3) { 77. int p1 = stoi(par1); 78. int p2 = stoi(par2); 79. int p3 = stoi(par3); 80. HANDLE process = GetCurrentProcess(); 81. DWORD\_PTR pa = NULL, sa = NULL, icpu = -1; 82. GetPrevInfo(process); 83. cout << "============================================\n"; 84. if (!SetProcessAffinityMask(process, p1)) throw "SetProcessAffinityMask [Exception]"; 85. GetUpdatedInfo(process); 86. cout << "============================================\n"; 87. cout << "1. [PriorityClass]: " << to\_string(getProcessPrty(p2)) << endl; 88. cout << "2. [PriorityClass]: " << to\_string(getProcessPrty(p3)) << endl; 89. cout << "============================================\n"; 90. CreateXProcesses(p2, p3); 91. } 92. int main(int argc,char\* argv[]) { 93. SetConsoleCP(1251); 94. SetConsoleOutputCP(1251); 96. //ShowResult("255","3","3"); 97. //ShowResult("255", "2", "5"); 98. ShowResult("1", "2", "5"); 99. return 0; 100. } |

1. Запустите приложение **OS05\_02**,принимающее следующие значения параметров:

* P1: доступны все процессоры;
* P2: Normal;
* P3: Normal.

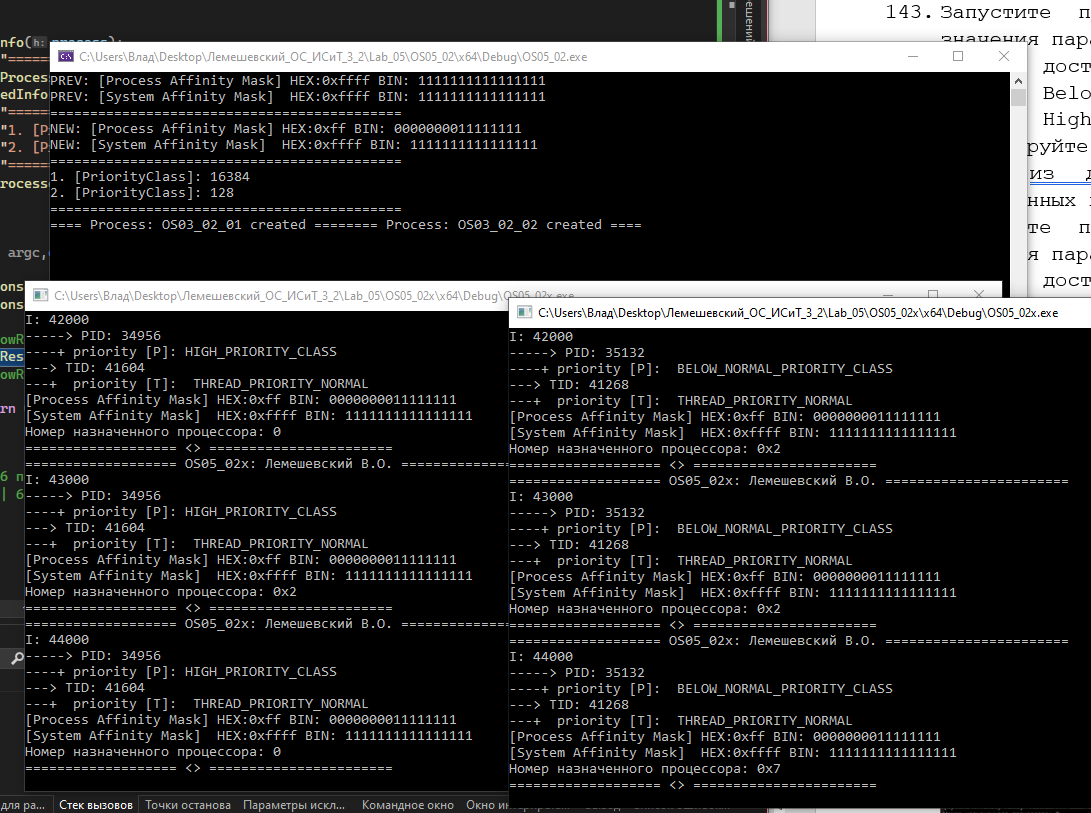
Зафиксируйте (скриншот) в момент первого окончания одного из дочерних процессов расхождение в количестве выполненных процессами итераций.



1. Запустите приложение **OS05\_02**,принимающее следующие значения параметров:

* P1: доступны все процессоры;
* P2: Below Normal;
* P3: High.

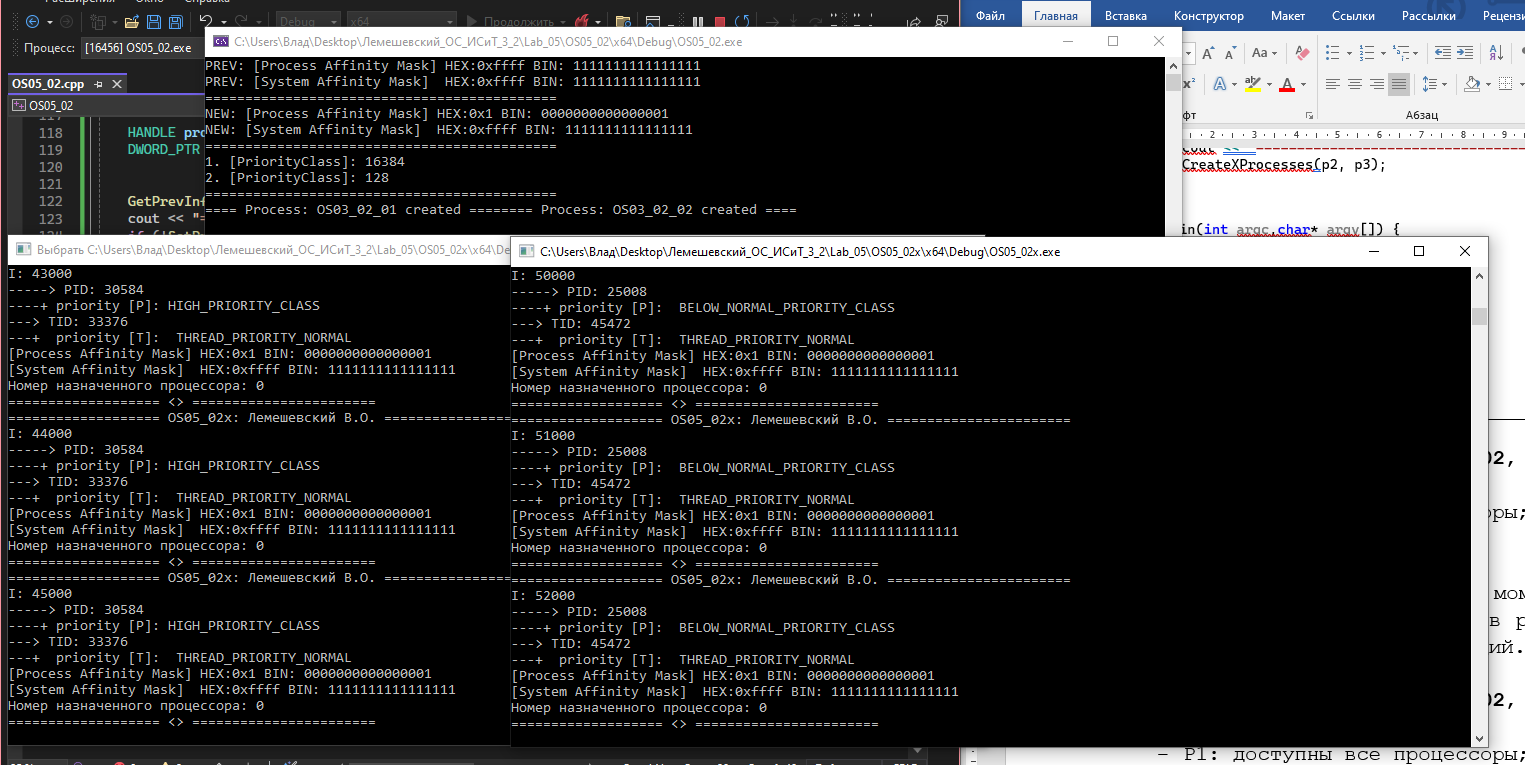
Зафиксируйте (скриншот) в момент первого окончания одного из дочерних процессов расхождение в количестве выполненных процессами итераций.



1. Запустите приложение **OS05\_02**,принимающее следующие значения параметров:

* P1: доступен один процессор;
* P2: Below Normal;
* P3: High.

Зафиксируйте (скриншот) в момент первого окончания одного из дочерних процессов расхождение в количестве выполненных процессами итераций.



1. По зафиксированным скриншотам, объясните полученные результаты.

**Задание 03**

1. Разработайте консольное Windows-приложение **OS05\_03,** принимающее следующие параметры:

* P1: целое число, задающее маску доступности процессоров (affinity mask);
* P2: целое число, задающее класс приоритет процесса;
* P3: целое число, задающее приоритет первого дочернего потока;
* P4: целое число, задающее приоритет второго дочернего потока.

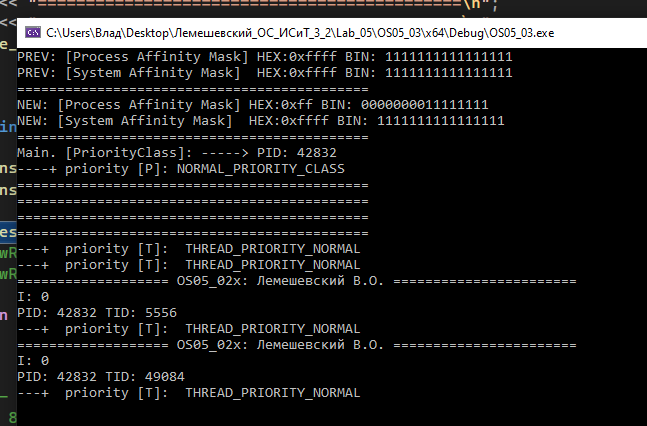
1. Приложение **OS05\_03**  включает в себя потоковую функцию **TA**, выполняющую цикл в 1млн итераций, аналогичный циклу в задании 02.
2. Приложение **OS05\_03** должно вывести в свое консольное окно заданные параметры и запустить два одинаковых дочерних потока (потоковая функция **TA**)**,** осуществляющих вывод консольное окно и имеющих заданные в параметрах приоритеты.

|  |
| --- |
| 1. void printProcessPrty(HANDLE h) { 2. DWORD prty = GetPriorityClass(h); 3. cout << "-----> PID: " << dec << GetCurrentProcessId() << endl; 4. switch (prty) { 5. case IDLE\_PRIORITY\_CLASS: cout << "----+ priority [P]: IDLE\_PRIORITY\_CLASS\n"; break; 6. case BELOW\_NORMAL\_PRIORITY\_CLASS: cout << "----+ priority [P]: BELOW\_NORMAL\_PRIORITY\_CLASS\n"; break; 7. case NORMAL\_PRIORITY\_CLASS: cout << "----+ priority [P]: NORMAL\_PRIORITY\_CLASS\n"; break; 8. case ABOVE\_NORMAL\_PRIORITY\_CLASS: cout << "----+ priority [P]: ABOVE\_NORMAL\_PRIORITY\_CLASS\n"; break; 9. case HIGH\_PRIORITY\_CLASS: cout << "----+ priority [P]: HIGH\_PRIORITY\_CLASS\n"; break; 10. case REALTIME\_PRIORITY\_CLASS: cout << "----+ priority [P]: REALTIME\_PRIORITY\_CLASS\n"; break; 11. default: cout << "----+ priority [P]: [NULL] \n"; break; 12. } 13. } 14. void printThreadPrty(HANDLE h) { 15. DWORD prty = GetThreadPriority(h); 16. switch (prty) { 17. case THREAD\_PRIORITY\_LOWEST: cout << "---+ priority [T]: THREAD\_PRIORITY\_LOWEST\n"; break; 18. case THREAD\_PRIORITY\_BELOW\_NORMAL: cout << "---+ priority [T]: THREAD\_PRIORITY\_BELOW\_NORMAL\n"; break; 19. case THREAD\_PRIORITY\_NORMAL: cout << "---+ priority [T]: THREAD\_PRIORITY\_NORMAL\n"; break; 20. case THREAD\_PRIORITY\_ABOVE\_NORMAL: cout << "---+ priority [T]: THREAD\_PRIORITY\_ABOVE\_NORMAL\n"; break; 21. case THREAD\_PRIORITY\_HIGHEST: cout << "---+ priority [T]: THREAD\_PRIORITY\_HIGHEST\n"; break; 22. case THREAD\_PRIORITY\_IDLE: cout << "---+ priority [T]: THREAD\_PRIORITY\_IDLE\n"; break; 23. case THREAD\_PRIORITY\_TIME\_CRITICAL: cout << "---+ priority [T]: THREAD\_PRIORITY\_TIME\_CRITICAL\n"; break; 24. default: cout << "---+ priority [T]: [NULL] \n"; break; 25. } 26. } 27. void PrintAffinityMask(HANDLE h) { 28. DWORD\_PTR pa = NULL, sa = NULL, icpu = -1; 29. char buf[10]; 30. if (!GetProcessAffinityMask(h, &pa, &sa)) throw "GetProcessAffinityMask [Exception]"; 31. cout << "[Process Affinity Mask] HEX:" << showbase << hex << pa << " BIN: " << bitset<sizeof(sa) \* 2>(pa) << endl; 32. cout << "[System Affinity Mask] HEX:" << showbase << hex << sa << " BIN: " << bitset<sizeof(sa) \* 2>(sa) << endl; 33. } 34. void PrintTotalProcessors(HANDLE h1, HANDLE h2) { 35. DWORD\_PTR pa = NULL, sa = NULL, icpu = -1; 36. SYSTEM\_INFO sys\_info; 37. if (!GetProcessAffinityMask(h1, &pa, &sa)) throw "GetProcessAffinityMask"; 38. GetSystemInfo(&sys\_info); 39. cout << "Max processors count: " << dec << sys\_info.dwNumberOfProcessors << "\n"; 40. icpu = SetThreadIdealProcessor(h2, MAXIMUM\_PROCESSORS); 41. cout << "Thread IdealProcessor: " << dec << icpu << "\n"; 42. } 43. DWORD WINAPI TA() { 44. HANDLE hp = GetCurrentProcess(); // текущий процесс 45. HANDLE ht = GetCurrentThread(); // текущий поток 46. DWORD pid = GetCurrentProcessId(); 47. DWORD tid = GetCurrentThreadId(); 48. for (int i = 0; i < 1000000; i++) { 49. if (i % 1000 == 0) { 50. cout << "=================== OS05\_02x: Лемешевский В.О. =======================\n"; 51. cout << "I: " << dec << i << endl; 52. cout << "PID: "<< dec << pid << dec << " TID: " << tid << endl; 53. //printProcessPrty(hp); 54. printThreadPrty(ht); 55. //PrintAffinityMask(hp); 56. Sleep(2000); 57. } 58. } 59. return 0; 60. } 61. DWORD getProcessPrty(int p) { 62. switch (p) { 63. case 1: return IDLE\_PRIORITY\_CLASS; 64. case 2: return BELOW\_NORMAL\_PRIORITY\_CLASS; 65. case 3: return NORMAL\_PRIORITY\_CLASS; 66. case 4: return ABOVE\_NORMAL\_PRIORITY\_CLASS; 67. case 5: return HIGH\_PRIORITY\_CLASS; 68. case 6: return REALTIME\_PRIORITY\_CLASS; 69. default: throw "----+ priority [P]: [NULL] \n"; 70. } 71. } 72. DWORD getThreadPrty(int i) 73. { 74. switch (i) 75. { 76. case 1: return THREAD\_PRIORITY\_LOWEST; 77. case 2: return THREAD\_PRIORITY\_BELOW\_NORMAL; 78. case 3: return THREAD\_PRIORITY\_NORMAL; 79. case 4: return THREAD\_PRIORITY\_ABOVE\_NORMAL; 80. case 5: return THREAD\_PRIORITY\_HIGHEST; 81. case 6: return THREAD\_PRIORITY\_IDLE; 82. default: throw "----+ priority [T]: [NULL] \n"; 83. } 84. } 85. void GetPrevInfo(HANDLE h) { 86. DWORD\_PTR pa = NULL, sa = NULL, icpu = -1; 87. if (!GetProcessAffinityMask(h, &pa, &sa)) throw "GetProcessAffinityMask [Exception]"; 88. cout << "PREV: [Process Affinity Mask] HEX:" << showbase << hex << pa << " BIN: " << bitset<sizeof(sa) \* 2>(pa) << endl; 89. cout << "PREV: [System Affinity Mask] HEX:" << showbase << hex << sa << " BIN: " << bitset<sizeof(sa) \* 2>(sa) << endl; 90. } 91. void GetUpdatedInfo(HANDLE h) { 92. DWORD\_PTR pa = NULL, sa = NULL, icpu = -1; 93. if (!GetProcessAffinityMask(h, &pa, &sa)) throw "GetProcessAffinityMask [Exception]"; 94. cout << "NEW: [Process Affinity Mask] HEX:" << showbase << hex << pa << " BIN: " << bitset<sizeof(sa) \* 2>(pa) << endl; 95. cout << "NEW: [System Affinity Mask] HEX:" << showbase << hex << sa << " BIN: " << bitset<sizeof(sa) \* 2>(sa) << endl; 96. } 97. void Create\_TA\_Thread(int prty1, int prty2) { 98. DWORD pid = GetCurrentProcessId(); 99. DWORD tid = GetCurrentThreadId(); // хранилище инициализатора описатель типа 100. DWORD Child1Id = NULL, 101. Child2Id = NULL; 102. HANDLE t1Child = CreateThread( // дескриптор, т.е. число, с помощью которого можно идентифицировать ресурс. С помощью дескприторов можно ссылаться на окна, объекты ядра, графические объекты и т.п. 103. NULL, // Параметры безопастности 104. 0, // Размер стека 105. (LPTHREAD\_START\_ROUTINE)TA// функция потока 106. , NULL // Параметры для функции 107. , 0 // Флаги создания 108. , &Child1Id // Ид потока 109. ); 110. HANDLE t2Child = CreateThread( 111. NULL, 112. 0, 113. (LPTHREAD\_START\_ROUTINE)TA 114. , NULL 115. , 0 116. , &Child2Id 117. ); 118. SetThreadPriority(t1Child, getThreadPrty(prty1)); 119. SetThreadPriority(t2Child, getThreadPrty(prty2)); 120. printThreadPrty(t1Child); 121. printThreadPrty(t2Child); 122. WaitForSingleObject(t1Child, INFINITE); // ожидает завершения указанного объекта 123. WaitForSingleObject(t2Child, INFINITE); 124. CloseHandle(t1Child); // закрывает дескрпиторы обьекта 125. CloseHandle(t2Child); 126. } 127. void ShowResult(int par1, int par2, int par3,int par4) { 128. HANDLE process = GetCurrentProcess(); 129. DWORD\_PTR pa = NULL, sa = NULL, icpu = -1; 130. GetPrevInfo(process); 131. cout << "============================================\n"; 132. if (!SetProcessAffinityMask(process, par1)) throw "SetProcessAffinityMask [Exception]"; 133. GetUpdatedInfo(process); 134. SetPriorityClass(process,getProcessPrty(par2)); 135. cout << "============================================\n"; 136. cout << "Main. [PriorityClass]: " ; 137. printProcessPrty(process); 138. cout << "============================================\n"; 139. cout << "============================================\n"; 140. cout << "============================================\n"; 141. cout << "============================================\n"; 142. Create\_TA\_Thread(par3,par4); 143. } 144. int main(int argc, char\* argv[]) { 145. SetConsoleCP(1251); 146. SetConsoleOutputCP(1251); 147. //ShowResult(255,3,3,3); 148. //ShowResult(255,3,1,5); 149. ShowResult(1,3,1,5); 150. return 0; 151. } |

1. Запустите приложение **OS05\_03**,принимающее следующие значения параметров:

* P1: доступны все процессоры;
* P2: Normal;
* P3: Normal;
* P4: Normal;

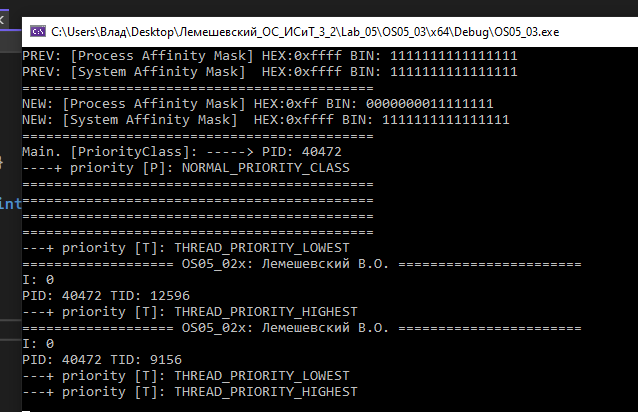
Зафиксируйте (скриншот) в момент первого окончания одного из дочерних потоков расхождение в количестве выполненных потоками итераций.



1. Запустите приложение **OS05\_03**,принимающее следующие значения параметров:

* P1: доступны все процессоры;
* P2: Normal;
* P3: Lowest;
* P4: Highest.

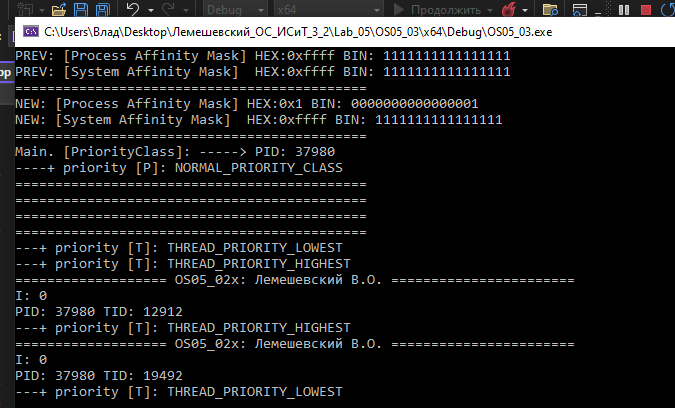
Зафиксируйте (скриншот) в момент первого окончания одного из дочерних потоков расхождение в количестве выполненных потоками итераций.



1. Запустите приложение **OS05\_03**,принимающее следующие значения параметров:

* P1: доступен один процессор;
* P2: Normal;
* P3: Lowest;
* P4: Highest.

Зафиксируйте (скриншот) в момент первого окончания одного из дочерних потоков расхождение в количестве выполненных потоками итераций.



1. По зафиксированным скриншотам объясните полученные результаты.

**Задание 04**

1. Разработайте консольное Linux-приложение **OS05\_04** на языке С++, выводящее на консоль следующую информации:

* идентификатор текущего процесса;
* идентификатор текущего (main) потока;
* приоритет (nice) текущего потока;
* номера доступных процессоров.

**Задание 05**

1. Разработайте консольное Linux-приложение **OS05\_05** на языке С, выполняющее длинный цикл.
2. Запустите приложение **OS05\_05.**
3. Зафиксируйте (скриншот) текущее значение **nicе**, полученное с помощью команды **top**.
4. Увеличьте приоритет для **OS05\_05** до максимального значения (самого привилегированного). Зафиксируйте (скриншот) текущее значение **nicе**, полученное с помощью команды **top**.
5. Уменьшите приоритет для **OS05\_05** до минимального значения (самого ничтожного). Зафиксируйте (скриншот) текущее значение **nicе**, полученное с помощью команды **top**

**Задание 06**

1. Разработайте консольное Linux-приложение **OS05\_06** на языке С, выполняющее длинный цикл с задержкой в 1сек в каждой итерации.
2. Продемонстрируйте запуск нескольких приложения **OS05\_06** в фоновом режиме, и команды bg, fg, jobs, Ctrl+Z, kill -9

**Задание 07**

1. Разработайте консольное Linux-приложение **OS05\_07** на языке С, выполняющее длинный цикл с задержкой в 1сек в каждой итерации. Приложение с помощью **OS05\_07** системного вызова **fork** вызывает дочерний поток который понижает свой приоритет на 10.
2. С помощью команды **watch ps** продемонстрируйте работу этих потоков и их значение **nice**.

**Задание 08.**Ответьте на следующие вопросы

1. Поясните понятие «мультизадачная OS с вытеснением».
2. Поясните понятие «циклическое планирование».
3. Поясните понятие «приоритетное планирование».
4. Поясните понятие «кооперативное планирование».
5. Поясните понятие «OS реального времени».
6. Поясните понятие «приоритет процесса».
7. Поясните выражение «поток уступает процессор другому потоку».
8. Windows: как поток может уступить процессор?
9. Windows: что такое базовый приоритет потока, как он вычисляется и диапазон его изменения?
10. Windows: поясните назначение и принцип применения системного вызова SetThreadIdealProcessor.
11. Windows: поясните назначение и принцип применения системного вызова ResumeThread.
12. Windows: поясните назначение и принцип применения системного вызова WaitForSingleObject.
13. Windows: поясните назначение и принцип применения системных вызовов GetProcessPriorityBoost, GetThreadPriorityBoost, SetProcessPriorityBoost, SetThreadPriorityBoost.
14. Linux: поясните принцип идентификации процессов и потоков и поясните почему он такой.
15. Linux: Поясните понятие «планировщик потоков».
16. Linux: поясните принцип использования значения nice –процесса, диапазон его изменения, для какого режима работы планировщика это значение применяется?
17. Linux: перечислите политики планирования, какая действует по умолчанию?
18. Linux: как выяснить действующую политику планирования для процесса с помощью файловой системы proc?
19. Linux: с помощью какого системного вызова поток может уступить процессор.
20. Linux: чем отличается системный вызов **nice** от вызова **setpriority**.
21. Linux: поясните понятие «планировщик ввода вывода», каким образом можно выяснить какие планировщики ввода/ вывода доступны?
22. Linux: перечислите известные вам планировщики ввода/ вывода, кратко охарактеризуйте их.
23. Linux: каким образом можно выяснить тип планировщика действующего для блокового устройства?