Пузиков Алексей, ФИТ-6

**Лабораторная работа №5**

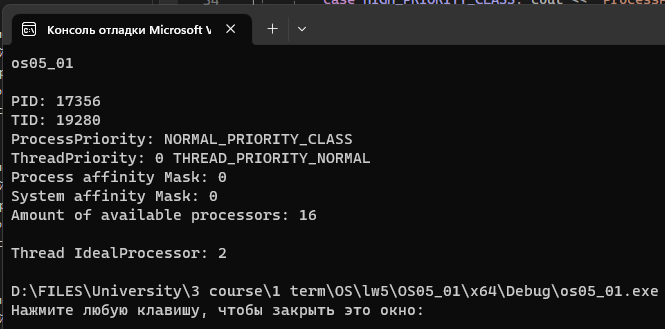
«Диспетчеризация»

**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 <iostream>  #include <cstdlib>  #include "Windows.h"    using namespace std;  void getProcessPriority(HANDLE hp); //приоритетный класс процесса  void getThreadPriority(HANDLE ht);  void getAffinityMask(HANDLE hp, HANDLE ht);  int main()  {  HANDLE hp = GetCurrentProcess();  HANDLE ht = GetCurrentThread();  DWORD pid = GetCurrentProcessId();  DWORD tid = GetCurrentThreadId();  cout << "OS05\_01\n\n";  cout << "PID: " << pid << endl << "TID: " << tid << endl;  getProcessPriority(hp);  getThreadPriority(ht);  getAffinityMask(hp, ht);  system("pause");  }  void getProcessPriority(HANDLE hp) {  DWORD prty = GetPriorityClass(hp);  switch (prty)  {  case IDLE\_PRIORITY\_CLASS: cout << "ProcessPriority: IDLE\_PRIORITY\_CLASS\n"; break;  case BELOW\_NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: BELOW\_NORMAL\_PRIORITY\_CLASS\n"; break;  case NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: NORMAL\_PRIORITY\_CLASS\n"; break;  case ABOVE\_NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: ABOVE\_NORMAL\_PRIORITY\_CLASS\n"; break;  case HIGH\_PRIORITY\_CLASS: cout << "ProcessPriority: HIGH\_PRIORITY\_CLASS\n"; break;  case REALTIME\_PRIORITY\_CLASS: cout << "ProcessPriority: REALTIME\_PRIORITY\_CLASS\n"; break;  default: cout << "?\n\n"; break;  }  }  void getThreadPriority(HANDLE ht) {  DWORD prty = GetPriorityClass(ht);  cout << "ThreadPriority: " << GetThreadPriority(ht) << " ";  switch (prty) {  case THREAD\_PRIORITY\_LOWEST: cout << "THREAD\_PRIORITY\_LOWEST\n" ; break;  case THREAD\_PRIORITY\_BELOW\_NORMAL: cout << "THREAD\_PRIORITY\_BELOW\_NORMAL\n" ; break;  case THREAD\_PRIORITY\_NORMAL: cout << "THREAD\_PRIORITY\_NORMAL\n" ; break;  case THREAD\_PRIORITY\_ABOVE\_NORMAL: cout << "THREAD\_PRIORITY\_ABOVE\_NORMAL\n" ; break;  case THREAD\_PRIORITY\_HIGHEST: cout << "THREAD\_PRIORITY\_HIGHEST\n" ; break;  case THREAD\_PRIORITY\_IDLE: cout << "THREAD\_PRIORITY\_IDLE\n" ; break;  default: cout << "?\n\n"; break;  }  }  void getAffinityMask(HANDLE hp, HANDLE ht) {  DWORD pa = NULL, sa = NULL, icpu = -1;  char buf[13];  if (!GetProcessAffinityMask(hp, &pa, &sa)) throw "GetProcessAffinityMask";  // преобразует pa в строку buf в 2-й СС  \_itoa\_s(pa, buf, 2);  cout << "Process affinity Mask: " << buf << endl;  \_itoa\_s(sa, buf, 2);  cout << "System affinity Mask: " << buf << endl;  SYSTEM\_INFO sys\_info;  GetSystemInfo(&sys\_info);  cout << "Amount of available processors: " << sys\_info.dwNumberOfProcessors << endl << endl;  icpu = SetThreadIdealProcessor(ht, MAXIMUM\_PROCESSORS);  cout << "Thread IdealProcessor: " << dec << icpu << endl;  } |



**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.

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.
2. 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:

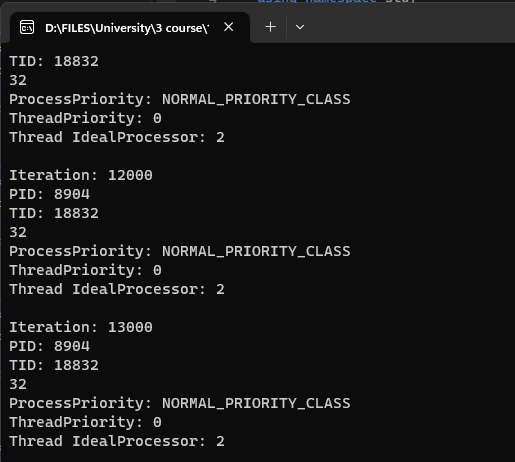
* P1: доступен один процессор;
* P2: Below Normal;
* P3: High.

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

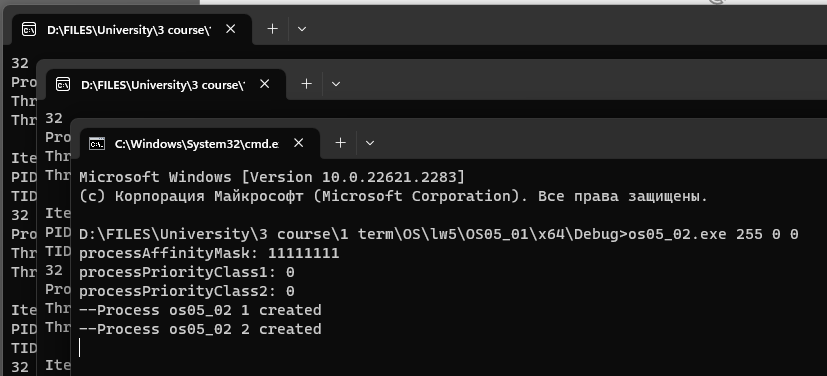
1. По зафиксированным скриншотам, объясните полученные результаты.

|  |
| --- |
| #include <iostream>  #include <cstdlib>  #include "Windows.h"  using namespace std;  int main(int argc, char\* argv[])  {  try {  if (argc == 4) {  char buf[10];  int p1 = atoi(argv[1]);  int p2 = atoi(argv[2]);  int p3 = atoi(argv[3]);  if (!SetProcessAffinityMask(GetCurrentProcess(), p1)) {  throw "SetProcessAffinityMask";  }  \_itoa\_s(p1, buf, 2);  cout << "processAffinityMask: " << buf << endl;  cout << "processPriorityClass1: " << p2 << endl;  cout << "processPriorityClass2: " << p3 << endl;  LPCWSTR an1 = L"os05\_02x.exe";  LPCWSTR an2 = L"os05\_02x.exe";  STARTUPINFO si1, si2;  PROCESS\_INFORMATION pi1, pi2;  ZeroMemory(&si1, sizeof(STARTUPINFO)); ZeroMemory(&si2, sizeof(STARTUPINFO));  si1.cb = sizeof(STARTUPINFO); si2.cb = sizeof(STARTUPINFO);  if (CreateProcess(an1, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE | p2, NULL, NULL, &si1, &pi1)) {  cout << "--Process os05\_02 1 created\n";  }  else {  cout << "--Process os05\_02 1 not created\n";  }  if (CreateProcess(an2, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE | p3, NULL, NULL, &si2, &pi2)) {  cout << "--Process os05\_02 2 created\n";  }  else {  cout << "--Process os05\_02 2 not created\n";  }  WaitForSingleObject(pi1.hProcess, INFINITE);  WaitForSingleObject(pi2.hProcess, INFINITE);  CloseHandle(pi1.hProcess);  CloseHandle(pi2.hProcess);  }  else {  cout << "No arguments" << endl;  }  }  catch (string err) {  cout << err << endl;  }  system("pause");  } |

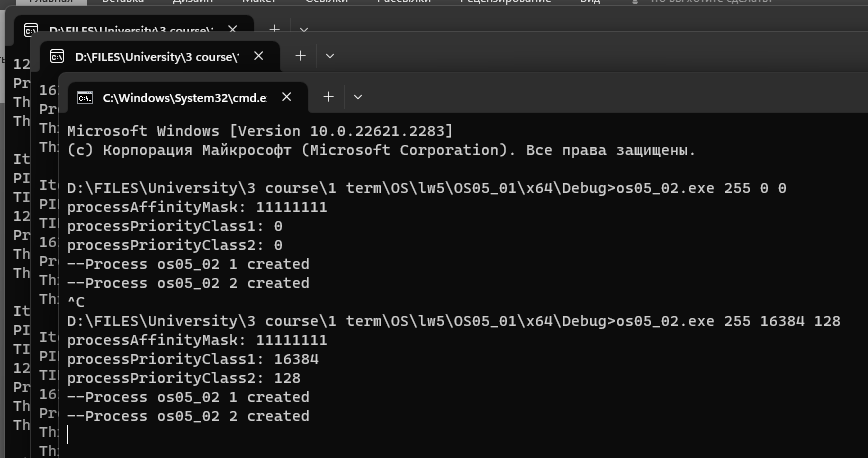
**Os05\_02x**



**os05\_02.exe 255 0 0**

****

**os05\_02.exe 255 16384 128**



**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.
3. 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:

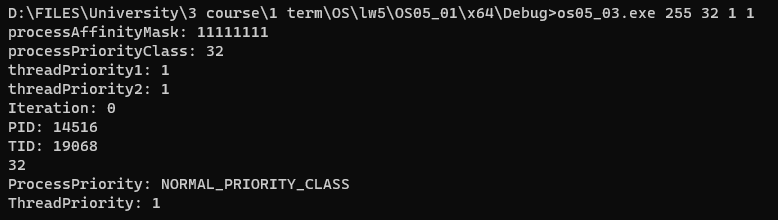
* P1: доступен один процессор;
* P2: Normal;
* P3: Lowest;
* P4: Highest.

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

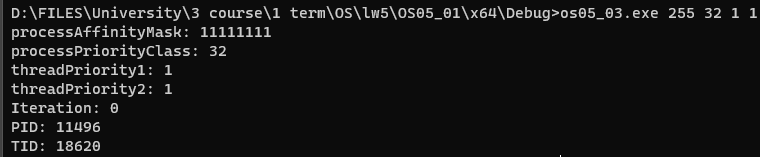
1. По зафиксированным скриншотам, объясните полученные результаты.

|  |
| --- |
| #include <iostream>  #include <cstdlib>  #include "Windows.h"  using namespace std;  HANDLE createThread(int threadPriority);  DWORD WINAPI os05\_03();  void getProcessPriority(HANDLE hp);  void getThreadPriority(HANDLE ht);  void getAffinityMask(HANDLE hp, HANDLE ht);  int main(int argc, char\* argv[])  {  if (argc == 5) {  char buf[10];  int p1 = atoi(argv[1]);  int p2 = atoi(argv[2]);  int p3 = atoi(argv[3]);  int p4 = atoi(argv[4]);  \_itoa\_s(p1, buf, 2);  cout << "processAffinityMask: " << buf << endl;  cout << "processPriorityClass: " << p2 << endl;  cout << "threadPriority1: " << p3 << endl;  cout << "threadPriority2: " << p4 << endl;  HANDLE h = GetCurrentProcess();  SetPriorityClass(h, p2);  SetProcessAffinityMask(h, p1);  HANDLE thread1 = createThread(p3);  HANDLE thread2 = createThread(p4);  ResumeThread(thread1);  ResumeThread(thread2);  HANDLE threads[] = { thread1, thread2 };  WaitForMultipleObjects(2, threads, TRUE, INFINITY);  CloseHandle(thread1);  CloseHandle(thread2);  }  else {  cout << "No arguments" << endl;  }  system("pause");  }  HANDLE createThread(int threadPriority) {  DWORD ChildId = NULL;  HANDLE h = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)os05\_03, NULL, CREATE\_SUSPENDED, &ChildId);  SetThreadPriority(h, threadPriority);  return h;  }  DWORD WINAPI os05\_03() {  HANDLE hp = GetCurrentProcess();  HANDLE ht = GetCurrentThread();  DWORD pid = GetCurrentProcessId();  DWORD tid = GetCurrentThreadId();  for (int i = 0; i < 1000000; i++) {  if (i % 1000 == 0) {  cout << "Iteration: " << i << endl;  cout << "PID: " << pid << endl << "TID: " << tid << endl;  getProcessPriority(hp);  getThreadPriority(ht);  getAffinityMask(hp, ht);  Sleep(200);  }  }  system("pause");  return 0;  }  void getProcessPriority(HANDLE hp) {  DWORD prty = GetPriorityClass(hp);  cout << prty << endl;  switch (prty) {  case IDLE\_PRIORITY\_CLASS: cout << "ProcessPriority: IDLE\_PRIORITY\_CLASS\n"; break;  case BELOW\_NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: BELOW\_NORMAL\_PRIORITY\_CLASS\n"; break;  case NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: NORMAL\_PRIORITY\_CLASS\n"; break;  case ABOVE\_NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: ABOVE\_NORMAL\_PRIORITY\_CLASS\n"; break;  case HIGH\_PRIORITY\_CLASS: cout << "ProcessPriority: HIGH\_PRIORITY\_CLASS\n"; break;  case REALTIME\_PRIORITY\_CLASS: cout << "ProcessPriority: REALTIME\_PRIORITY\_CLASS\n"; break;  default: cout << "?\n\n"; break;  }  }  void getThreadPriority(HANDLE ht) {  DWORD prty = GetPriorityClass(ht);  cout << "ThreadPriority: " << GetThreadPriority(ht) << endl;  }  void getAffinityMask(HANDLE hp, HANDLE ht) {  DWORD icpu = -1;  icpu = SetThreadIdealProcessor(ht, MAXIMUM\_PROCESSORS);  cout << "Thread IdealProcessor: " << dec << icpu << endl << endl;  } |

**os05\_03.exe 255 32 1 1**



**os05\_03.exe 255 32 -2 2**



**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?