Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Операционные системы

Студент: Дмитрук И.И.

ФИТ 3 курс 5 группа

Преподаватель: Савельева М.Г.

Минск 2023

**Лабораторная работа №5**

Задание 1. 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.

|  |  |
| --- | --- |
| Код программы OS05\_01 | #include <iostream>  #include "Windows.h"  using namespace std;  // приоритет (приоритетный класс) текущего процесса;  void PrintProcessPriority(HANDLE hp)  {  int priority = GetPriorityClass(hp);  switch (priority)  {  case IDLE\_PRIORITY\_CLASS:  cout << "Priority: IDLE\_PRIORITY\_CLASS" << endl;  break;  case BELOW\_NORMAL\_PRIORITY\_CLASS:  cout << "Priority: BELOW\_NORMAL\_PRIORITY\_CLASS" << endl;  break;  case NORMAL\_PRIORITY\_CLASS:  cout << "Priority: NORMAL\_PRIORITY\_CLASS" << endl;  break;  case ABOVE\_NORMAL\_PRIORITY\_CLASS:  cout << "Priority: ABOVE\_NORMAL\_PRIORITY\_CLASS" << endl;  break;  case HIGH\_PRIORITY\_CLASS:  cout << "Priority: HIGH\_PRIORITY\_CLASS" << endl;  break;  case REALTIME\_PRIORITY\_CLASS:  cout << "Priority: REALTIME\_PRIORITY\_CLASS" << endl;  break;  }  }  // приоритет текущего потока;  void PrintThreadPriority(HANDLE ht)  {  int priority = GetThreadPriority(ht);  switch (priority)  {  case THREAD\_PRIORITY\_IDLE:  cout << "Thread priority: THREAD\_PRIORITY\_IDLE" << endl;  break;  case THREAD\_PRIORITY\_LOWEST:  cout << "Thread priority: THREAD\_PRIORITY\_LOWEST" << endl;  break;  case THREAD\_PRIORITY\_BELOW\_NORMAL:  cout << "Thread priority: THREAD\_PRIORITY\_BELOW\_NORMAL" << endl;  break;  case THREAD\_PRIORITY\_NORMAL:  cout << "Thread priority: THREAD\_PRIORITY\_NORMAL" << endl;  break;  case THREAD\_PRIORITY\_ABOVE\_NORMAL:  cout << "Thread priority: THREAD\_PRIORITY\_ABOVE\_NORMAL" << endl;  break;  case THREAD\_PRIORITY\_HIGHEST:  cout << "Thread priority: THREAD\_PRIORITY\_HIGHEST" << endl;  break;  case THREAD\_PRIORITY\_TIME\_CRITICAL:  cout << "Thread priority: THREAD\_PRIORITY\_TIME\_CRITICAL" << endl;  break;  }  }  //маска (affinity mask) доступных процессу процессоров в двоичном виде;  void PrintAffinityMask(HANDLE hp, HANDLE ht) {  DWORD\_PTR pa = NULL, sa = NULL, icpu = -1;  char buf[10];  if (!GetProcessAffinityMask(hp, &pa, &sa))  throw "[FATAL] GetProcessAffinityMask threw an exception.";  \_itoa\_s(pa, buf, 2);  cout << "Process affinity mask: " << buf;  cout << " (" << showbase << hex << pa << ")\n";  \_itoa\_s(sa, buf, 2);  cout << "System affinity mask: " << buf;  cout << " (" << showbase << hex << sa << ")\n";  SYSTEM\_INFO sys\_info;  GetSystemInfo(&sys\_info);  cout << "Max processors count: " << dec << sys\_info.dwNumberOfProcessors << "\n";  icpu = SetThreadIdealProcessor(ht, MAXIMUM\_PROCESSORS);  cout << "Thread IdealProcessor: " << dec << icpu << "\n";  }  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  HANDLE hp = GetCurrentProcess();  HANDLE ht = GetCurrentThread();  // идентификатор текущего процесса;  cout << "Process ID: " << GetCurrentProcessId() << endl;  // идентификатор текущего main потока;  cout << "Thread ID: " << GetCurrentThreadId() << endl;  PrintProcessPriority(hp);  PrintThreadPriority(ht);  PrintAffinityMask(hp, ht);  system("pause");  return 0;  } |
| Скриншот запуска |  |

Задание 2. Консольное Windows-приложение OS05\_02x, выполняющее цикл в 1млн итераций.

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.

|  |  |
| --- | --- |
| Код программы OS05\_02x | #include <Windows.h>  #include <iostream>  using namespace std;  void PrintProcessPriority(HANDLE hp)  {  int priority = GetPriorityClass(hp);  switch (priority)  {  case IDLE\_PRIORITY\_CLASS:  cout << "Priority: IDLE\_PRIORITY\_CLASS" << endl;  break;  case BELOW\_NORMAL\_PRIORITY\_CLASS:  cout << "Priority: BELOW\_NORMAL\_PRIORITY\_CLASS" << endl;  break;  case NORMAL\_PRIORITY\_CLASS:  cout << "Priority: NORMAL\_PRIORITY\_CLASS" << endl;  break;  case ABOVE\_NORMAL\_PRIORITY\_CLASS:  cout << "Priority: ABOVE\_NORMAL\_PRIORITY\_CLASS" << endl;  break;  case HIGH\_PRIORITY\_CLASS:  cout << "Priority: HIGH\_PRIORITY\_CLASS" << endl;  break;  case REALTIME\_PRIORITY\_CLASS:  cout << "Priority: REALTIME\_PRIORITY\_CLASS" << endl;  break;  }  }  // приоритет текущего потока;  void PrintThreadPriority(HANDLE ht)  {  int priority = GetThreadPriority(ht);  switch (priority)  {  case THREAD\_PRIORITY\_IDLE:  cout << "Thread priority: THREAD\_PRIORITY\_IDLE" << endl;  break;  case THREAD\_PRIORITY\_LOWEST:  cout << "Thread priority: THREAD\_PRIORITY\_LOWEST" << endl;  break;  case THREAD\_PRIORITY\_BELOW\_NORMAL:  cout << "Thread priority: THREAD\_PRIORITY\_BELOW\_NORMAL" << endl;  break;  case THREAD\_PRIORITY\_NORMAL:  cout << "Thread priority: THREAD\_PRIORITY\_NORMAL" << endl;  break;  case THREAD\_PRIORITY\_ABOVE\_NORMAL:  cout << "Thread priority: THREAD\_PRIORITY\_ABOVE\_NORMAL" << endl;  break;  case THREAD\_PRIORITY\_HIGHEST:  cout << "Thread priority: THREAD\_PRIORITY\_HIGHEST" << endl;  break;  case THREAD\_PRIORITY\_TIME\_CRITICAL:  cout << "Thread priority: THREAD\_PRIORITY\_TIME\_CRITICAL" << endl;  break;  }  }  //маска (affinity mask) доступных процессу процессоров в двоичном виде;  void PrintAffinityMask(HANDLE hp, HANDLE ht) {  DWORD\_PTR pa = NULL, sa = NULL, icpu = -1;  char buf[10];  if (!GetProcessAffinityMask(hp, &pa, &sa))  throw "[FATAL] GetProcessAffinityMask threw an exception.";  \_itoa\_s(pa, buf, 2);  cout << "Process affinity mask: " << buf;  cout << " (" << showbase << hex << pa << ")\n";  \_itoa\_s(sa, buf, 2);  cout << "System affinity mask: " << buf;  cout << " (" << showbase << hex << sa << ")\n";  SYSTEM\_INFO sys\_info;  GetSystemInfo(&sys\_info);  cout << "Max processors count: " << dec << sys\_info.dwNumberOfProcessors << "\n";  icpu = SetThreadIdealProcessor(ht, MAXIMUM\_PROCESSORS);  cout << "Thread IdealProcessor: " << dec << icpu << "\n";  }  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  HANDLE processHandle = GetCurrentProcess();  HANDLE threadHandle = GetCurrentThread();  int i = 0;  clock\_t start = clock();  while (i < 1000000)  {  if (i % 1000 == 0)  {  cout << "=================================" << endl;  cout << "Номер итерации: " << i << endl;  cout << "Идентификатор процесса: " << GetCurrentProcessId() << endl;  cout << "Идентификатор потока: " << GetCurrentThreadId() << endl;  cout << "Класс приоритета процесса: ";  PrintProcessPriority(GetCurrentProcess());  cout << "Приоритет потока: ";  PrintThreadPriority(GetCurrentThread());  cout << "Номер назначенного процессора: " << GetCurrentProcessorNumber() << endl;  cout << "\n";  cout << "=================================" << endl;  Sleep(200);  }  i++;  }  clock\_t end = clock();  std::cout << "Time: " << (end - start) / 1000.0 << std::endl;  system("pause");  } |
| Скриншот запуска |  |

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.

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.

|  |  |
| --- | --- |
| Код программы OS05\_02 | // 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.  // 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.  #include <windows.h>  #include <iostream>  using namespace std;  DWORD intToProcessPriority(int i) {  switch (i)  {  case 1: return IDLE\_PRIORITY\_CLASS;  case 2: return BELOW\_NORMAL\_PRIORITY\_CLASS;  case 3: return NORMAL\_PRIORITY\_CLASS;  case 4: return ABOVE\_NORMAL\_PRIORITY\_CLASS;  case 5: return HIGH\_PRIORITY\_CLASS;  case 6: return REALTIME\_PRIORITY\_CLASS;  default: throw "Unknown priority class";  }  }  int main(int argc, char\* argv[])  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  //case 1:  // mask = 0xff (255);  // priority1 = NORMAL\_PRIORITY\_CLASS (3);  // priority2 = NORMAL\_PRIORITY\_CLASS (3);  // break;  //case 2:  // mask = 0xff (255);  // priority1 = BELOW\_NORMAL\_PRIORITY\_CLASS (2);  // priority2 = HIGH\_PRIORITY\_CLASS (5);  // break;  //case 3:  // mask = 0x01 (1);  // priority1 = BELOW\_NORMAL\_PRIORITY\_CLASS (2);  // priority2 = HIGH\_PRIORITY\_CLASS (5);  // break;  try  {  if (argc == 4)  {  HANDLE processHandle = GetCurrentProcess();  DWORD\_PTR pa = NULL, sa = NULL, icpu = -1;  char buf[13];  int parm1 = atoi(argv[1]);  int parm2 = atoi(argv[2]);  int parm3 = atoi(argv[3]);  if (!GetProcessAffinityMask(processHandle, &pa, &sa))  throw "Error in GetProcessAffinityMask";  cout << "\t\tBefore applying parameters:\n";  \_itoa\_s(pa, buf, 2);  cout << "Process affinity Mask: " << buf << endl;  \_itoa\_s(sa, buf, 2);  cout << "System affinity Mask: " << buf << endl;  if (!SetProcessAffinityMask(processHandle, parm1))  throw "ERROR in SetProcessAffinityMask";  if (!GetProcessAffinityMask(processHandle, &pa, &sa))  throw "Error in GetProcessAffinityMask";  cout << "\t\tAfter applying parameters:\n";  \_itoa\_s(pa, buf, 2);  cout << "Process affinity Mask: " << buf << endl;  \_itoa\_s(sa, buf, 2);  cout << "System affinity Mask: " << buf << endl;  \_itoa\_s(parm1, buf, 2);  cout << "Child 1 PriorityClass: " << parm2 << endl;  cout << "Child 2 PriorityClass: " << parm3 << endl;  LPCWSTR path1 = L"D:\\Labs\\ОС\\Lab5\\Lab5\\x64\\Debug\\OS05\_02x.exe";  LPCWSTR path2 = L" D:\\Labs\\ОС\\Lab5\\Lab5\\x64\\Debug\\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(path1, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE | intToProcessPriority(parm2), NULL, NULL, &si1, &pi1))  cout << "-- Process OS05\_02 1 was created\n";  else cout << "-- Process OS05\_02 1 wasn't created\n";  if (CreateProcess(path2, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE | intToProcessPriority(parm3), NULL, NULL, &si2, &pi2))  cout << "-- Process OS05\_02 2 was created\n";  else cout << "-- Process OS05\_02 2 wasn't created\n";  WaitForSingleObject(pi1.hProcess, INFINITE);  WaitForSingleObject(pi2.hProcess, INFINITE);  CloseHandle(pi1.hProcess);  CloseHandle(pi2.hProcess);  }  else  cout << "No parameters provided" << endl;  }  catch (string err)  {  cout << err << endl;  }  system("pause");  } |

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.

|  |  |
| --- | --- |
| Параметры | P1 = 0xff (255);  P2 = NORMAL\_PRIORITY\_CLASS (3);  P3 = NORMAL\_PRIORITY\_CLASS (3); |
| Скриншот запуска |  |

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.

|  |  |
| --- | --- |
| Параметры | P1 = 0xff (255);  P2 = BELOW\_NORMAL\_PRIORITY\_CLASS (2);  P3 = HIGH\_PRIORITY\_CLASS (5); |
| Скриншот запуска |  |

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.

|  |  |
| --- | --- |
| Параметры | P1 = 0x01 (1);  P2 = BELOW\_NORMAL\_PRIORITY\_CLASS (2);  P3 = HIGH\_PRIORITY\_CLASS (5); |
| Скриншот запуска |  |

Задание 3. 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.

|  |  |
| --- | --- |
| Код программы OS05\_03 | /\* ﻿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.  Приложение OS05\_03 включает в себя потоковую функцию TA, выполняющую цикл в 1млн итераций, аналогичный циклу в задании 02.  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. \*/  #include <iostream>  #include <cstdlib>  #include "Windows.h"  using namespace std;  DWORD intToProcessPriority(int i)  {  switch (i)  {  case 1: return IDLE\_PRIORITY\_CLASS;  case 2: return BELOW\_NORMAL\_PRIORITY\_CLASS;  case 3: return NORMAL\_PRIORITY\_CLASS;  case 4: return ABOVE\_NORMAL\_PRIORITY\_CLASS;  case 5: return HIGH\_PRIORITY\_CLASS;  case 6: return REALTIME\_PRIORITY\_CLASS;  default: throw "Unknown priority class";  }  }  DWORD intToThreadPriority(int i)  {  switch (i)  {  case 1: return THREAD\_PRIORITY\_LOWEST;  case 2: return THREAD\_PRIORITY\_BELOW\_NORMAL;  case 3: return THREAD\_PRIORITY\_NORMAL;  case 4: return THREAD\_PRIORITY\_ABOVE\_NORMAL;  case 5: return THREAD\_PRIORITY\_HIGHEST;  case 6: return THREAD\_PRIORITY\_IDLE;  default: throw "No such priority class";  }  }  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 << "ERROR: PROCESS PRIORITY NOT RECOGNIZED\n\n"; break;  }  }  void getThreadPriority(HANDLE ht)  {  DWORD prty = GetThreadPriority(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 << "ERROR: THREAD PRIORITY NOT RECOGNIZED\n\n"; break;  }  }  void childThread()  {  DWORD pid = GetCurrentProcessId();  DWORD tid = GetCurrentThreadId();  HANDLE hp = GetCurrentProcess();  HANDLE ht = GetCurrentThread();  DWORD icpu = SetThreadIdealProcessor(ht, MAXIMUM\_PROCESSORS);  for (int i = 0; i < 1000000; i++)  {  if (i % 1000 == 0)  {  cout << "------------------------------------------\n";  cout << "Номер итерации: " << i << endl;  cout << "Идентификатор процесса: " << pid << "\nИдентификатор потока: " << tid << endl;  cout << "Класс приоритета процесса: \n";  getProcessPriority(hp);  cout << "Приоритет потока: \n";  getThreadPriority(ht);  cout << "Thread IdealProcessor: " << dec << icpu << endl << endl;  Sleep(200);  }  }  return;  }  int main(int argc, char\* argv[])  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  /\*case 1:  P1 = 0xff (255);  P2 = NORMAL\_PRIORITY\_CLASS (3);  P3 = THREAD\_PRIORITY\_NORMAL (3);  P4 = THREAD\_PRIORITY\_NORMAL (3);  case 2:  P1 = 0xff (1);  P2 = NORMAL\_PRIORITY\_CLASS (3);  P3 = THREAD\_PRIORITY\_LOWEST (1);  P4 = THREAD\_PRIORITY\_HIGHEST (5);  case 3:  P1 = 0x01 (1);  P2 = NORMAL\_PRIORITY\_CLASS (3);  P3 = THREAD\_PRIORITY\_LOWEST (1);  P4 = THREAD\_PRIORITY\_HIGHEST (5);  \*/  try  {  if (argc == 5)  {  HANDLE processHandle = GetCurrentProcess();  DWORD\_PTR pa = NULL, sa = NULL, icpu = -1;  char buf[13];  int parm1 = atoi(argv[1]);  int parm2 = atoi(argv[2]);  int parm3 = atoi(argv[3]);  int parm4 = atoi(argv[4]);  if (!GetProcessAffinityMask(processHandle, &pa, &sa)) throw "Error in GetProcessAffinityMask";  cout << "\t\tBefore applying parameters:\n";  \_itoa\_s(pa, buf, 2);  cout << "Process affinity Mask: " << buf << endl;  \_itoa\_s(sa, buf, 2);  cout << "System affinity Mask: " << buf << endl;  if (!SetProcessAffinityMask(processHandle, parm1)) throw "ERROR in SetProcessAffinityMask";  if (!GetProcessAffinityMask(processHandle, &pa, &sa)) throw "Error in GetProcessAffinityMask";  cout << "\t\tAfter applying parameters:\n";  \_itoa\_s(pa, buf, 2);  cout << "Process affinity Mask: " << buf << endl;  \_itoa\_s(sa, buf, 2);  cout << "System affinity Mask: " << buf << endl;  SetPriorityClass(processHandle, intToProcessPriority(parm2));  cout << "Main PriorityClass: " << parm2 << endl;  DWORD childId\_T1, childId\_T2 = NULL;  HANDLE hChild1 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)childThread, NULL, CREATE\_SUSPENDED, &childId\_T1);  HANDLE hChild2 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)childThread, NULL, CREATE\_SUSPENDED, &childId\_T2);  SetThreadPriority(hChild1, intToThreadPriority(parm3));  SetThreadPriority(hChild2, intToThreadPriority(parm4));  getThreadPriority(hChild1);  getThreadPriority(hChild2);  ResumeThread(hChild1);  ResumeThread(hChild2);  WaitForSingleObject(hChild1, INFINITE);  WaitForSingleObject(hChild2, INFINITE);  CloseHandle(hChild1);  CloseHandle(hChild2);  }  else cout << "No parameters provided" << endl;  }  catch (string err) {  cout << err << endl;  }  system("pause");  } |

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.

|  |  |
| --- | --- |
| Параметры | P1 = 0xff (255);  P2 = NORMAL\_PRIORITY\_CLASS (3);  P3 = THREAD\_PRIORITY\_NORMAL (3);  P4 = THREAD\_PRIORITY\_NORMAL (3) |
| Скриншот запуска |  |

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.

|  |  |
| --- | --- |
| Параметры | P1 = 0xff (255);  P2 = NORMAL\_PRIORITY\_CLASS (3);  P3 = THREAD\_PRIORITY\_LOWEST (1);  P4 = THREAD\_PRIORITY\_HIGHEST (5) |
| Скриншот запуска |  |

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.

|  |  |
| --- | --- |
| Параметры | P1 = 0x01 (1);  P2 = NORMAL\_PRIORITY\_CLASS (3);  P3 = THREAD\_PRIORITY\_LOWEST (1);  P4 = THREAD\_PRIORITY\_HIGHEST (5) |
| Скриншот запуска |  |

Задание 4. 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.

|  |  |
| --- | --- |
| Код программы OS05\_04 | #define \_GNU\_SOURCE  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/types.h>  #include <sched.h>  int main()  {  pid\_t pid = getpid();  pid\_t tid = gettid();  int nicity = nice(0);  cpu\_set\_t set;  CPU\_ZERO(&set);  printf("PID = %d\nTID = %d\nNice = %d\n", pid, tid, nicity);  if (sched\_getaffinity(0, sizeof(cpu\_set\_t), &set) == 0)  {  int is = 0;  for (int i = 0; i < CPU\_SETSIZE; ++i)  {  if (CPU\_ISSET(i, &set))  printf("CPU = %i is set\n", i);  }  }  else  printf("[ERROR]: shed\_getaffinity returned -1.");  exit(0);  } |
| Скриншот запуска |  |

Задание 5. Консольное Linux-приложение OS05\_05 на языке С, выполняющее длинный цикл.

Запустите приложение OS05\_05.

Зафиксируйте (скриншот) текущее значение nicе, полученное с помощью команды top.

Увеличьте приоритет для OS05\_05 до максимального значения (самого привилегированного). Зафиксируйте (скриншот) текущее значение nicе, полученное с помощью команды top.

Уменьшите приоритет для OS05\_05 до минимального значения (самого ничтожного). Зафиксируйте (скриншот) текущее значение nicе, полученное с помощью команды top

|  |  |
| --- | --- |
| Код программы OS05\_05 | #define \_GNU\_SOURCE  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/types.h>  #include <sched.h>  /\*  sudo nice -n <nice> <command> Run with some nice  sudo renice -n <nice> -p <pid> Change nice of running cmd  \*/  int main()  {  pid\_t pid = getpid();  pid\_t tid = gettid();  int nicity = nice(0);  for (int i = 0; i < 10000; ++i)  {  printf("PID: %d\nTID: %d\nNice: %d\n\n", pid, tid, nicity);  sleep(1);  }  } |
| Скриншот запуска |  |

Задание 6. Консольное Linux-приложение OS05\_06 на языке С, выполняющее длинный цикл с задержкой в 1сек в каждой итерации.

Продемонстрируйте запуск нескольких приложения OS05\_06 в фоновом режиме, и команды bg, fg, jobs, Ctrl+Z, kill -9

|  |  |
| --- | --- |
| Код программы OS05\_06 | #define \_GNU\_SOURCE  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/types.h>  #include <sched.h>  /\*  Ctrl+Z: Stop process  bg: Move process to background  fg [%n]: Move process [%n] to foreground  jobs: All processes in this bash console  kill -9: Terminate process  <cmd> &: Start command in bg  vmstart 5 - command for easier demonstration  \*/  int main()  {  pid\_t pid = getpid();  pid\_t tid = gettid();  for (int i = 0; i < 1000000; ++i)  {  printf("PID: %d\nTID: %d\n\n", pid, tid);  sleep(1);  }  } |
| Скриншот запуска |  |

Задание 7. 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.

С помощью команды watch ps продемонстрируйте работу этих потоков и их значение nice.

|  |  |
| --- | --- |
| Код программы OS05\_07 | #define \_GNU\_SOURCE  #include <stdio.h>  #include <stdlib.h>  #include <unistd.h>  #include <sys/types.h>  #include <sched.h>  #include <errno.h>  #include <sys/wait.h>  #include <sys/syscall.h>  #include <pthread.h>  #include <sched.h>  /\*  Get nice via ps command: ps -eo "%p %P %c %n"  \*/  void thread1()  {  pid\_t pid = getpid();  int cnice = nice(0);  printf("Nice: %d\n", cnice);  for (int i = 0; i < 1000000; ++i)  {  if (i == 10)  {  cnice = nice(10);  printf("Nice: %d\n", cnice);  }  printf("[CHILD] PID: %d\n", pid);  sleep(2);  }  }  void thread2()  {  pid\_t pid = getpid();  for (int i = 0; i < 1000000; i++)  {  printf("[MAIN] PID: %d\n", pid);  sleep(2);  }  }  int main()  {  pid\_t pid;  switch (pid = fork())  {  case -1:  perror("[ERROR] Fork() returned -1.");  exit(-1);  case 0:  thread1();  exit(0);  default:  thread2();  wait(NULL);  }  exit(0);  } |
| Скриншот запуска |  |