Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Операционные системы

Студент: Трубач Д.С.

ФИТ 3 курс 5 группа

Преподаватель: Савельева М.Г.

Минск 2023

**Лабораторная работа №3**

Задание 1. Windows-приложение OS03\_01 на языке С++, выполняющее длинный цикл с временной задержкой и с выводом на консоль идентификатора процесса.

|  |  |
| --- | --- |
| Код программы | #include <iostream>  #include <Windows.h>  using namespace std;  int main()  {  for (short i = 1; i <= 10000; ++i)  {  cout << i << ". PID = " << GetCurrentProcessId() << "\n";  Sleep(1000);  }  return 0;  } |
| Скриншот запуска | E:\3course\5sem\OS\Lab3\Image\run 03_01.png |
| Просмотр через Task Manager | E:\3course\5sem\OS\Lab3\Image\task manager.png |
| Просмотр через tasklist | E:\3course\5sem\OS\Lab3\Image\tasklist.png |
| Просмотр через Powershell | E:\3course\5sem\OS\Lab3\Image\powershell.png |
| Просмотр через Perfomance Monitor | E:\3course\5sem\OS\Lab3\Image\monitor.png |

Задание 2. Windows-приложение OS03\_02 на языке С++, выполняющее цикл 100 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.

|  |  |
| --- | --- |
| Код программы | #include <iostream>  #include <Windows.h>  using namespace std;  // CMD: tasklist /FI "IMAGENAME eq OS03\_02\*"  int main()  {  cout << "OS03\_02\n\n";  LPCWSTR OS03\_02\_1 = L"E:\\3course\\5sem\\OS\\Lab3\\Debug\\OS03\_02\_1.exe";  LPCWSTR OS03\_02\_2 = L"E:\\3course\\5sem\\OS\\Lab3\\Debug\\OS03\_02\_2.exe";  STARTUPINFO si1; // структура для определения внешнего вида окна для нового процесса (ширина и высота окна, его смещение, заголовок)  STARTUPINFO si2;  PROCESS\_INFORMATION pi1; // структура с инфой о созданном процессе и его первичном потоке  PROCESS\_INFORMATION pi2;  ZeroMemory(&si1, sizeof(STARTUPINFO));  ZeroMemory(&si2, sizeof(STARTUPINFO));  si1.cb = sizeof(STARTUPINFO); // явное указание размера структуры si в байтах  si1.dwFlags = STARTF\_USEFILLATTRIBUTE; // флаг для возможности работы dwFillAttribute  si1.dwFillAttribute = (DWORD)FOREGROUND\_GREEN; // параметр цвета текста  si1.lpReserved = NULL; // lpReserved рекомендуется установить в null перед передачей si в CreateProcess  si2.cb = sizeof(STARTUPINFO);  si2.dwFlags = STARTF\_USEFILLATTRIBUTE;  si2.dwFillAttribute = (DWORD)FOREGROUND\_RED;  si1.lpReserved = NULL;  if (CreateProcess(  OS03\_02\_1, // lpApplicationName: имя исполняемого модуля - exe иди другой тип (MS-DOS или OS/2)  NULL, // lpCommandLine: параметры коммандной строки  NULL, // lpProcessAttributes: может ли возвращаемый дескриптор быть унаследован дочерними процессами (null - нельзя наследовать)  NULL, // lpThreadAttributes: может ли возвращаемый дескриптор быть унаследован дочерними потоками (null - нельзя наследовать)  FALSE, // bInheritHandles: true - каждый возвращаемый дескриптор наследуется дочерним процессом; false - не наследуется  CREATE\_NEW\_CONSOLE, // dwCreationFlags: флаги, управляющие приоритетом и параметрами процесса; конкретно этот создает новый инстанс консоли  NULL, // lpEnvironment: блок конфигурации нового процесса (пары ключ-значение); если null, то конфигурация наследуется от родителя  NULL, // lpCurrentDirectory: полный путь дочернего процесса; если null, то процесс создается в каталоге родительского процесса  &si1, // lpStartupInfo: структура STARTUPINFO (внешний вид окна)  &pi1)) // lpProcessInfo: структура PROCESS\_INFORMATION (дескрипторы процесса и первичного потока)  cout << "[OK] Process OS03\_02\_1 was created.\n";  else cout << "[ERROR] Process OS03\_02\_1 was not created.\n";  if (CreateProcess(OS03\_02\_2, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si2, &pi2))  cout << "[OK] Process OS03\_02\_2 was created.\n\n";  else cout << "[ERROR] Process OS03\_02\_2 was not created.\n\n";  for (short i = 1; i <= 100; ++i)  {  cout << i << ". PID = " << GetCurrentProcessId() << "\n";  Sleep(1000);  }  WaitForSingleObject(pi1.hProcess, INFINITE); // бесконечно будет ожидаться завершение программы до момента,  WaitForSingleObject(pi2.hProcess, INFINITE); // пока оба дочерних процесса не пошлют сигнал о завершении работы  CloseHandle(pi1.hThread); // CloseHandle закрывает дескрипторы в структуре PROCESS\_INFORMATION  CloseHandle(pi2.hThread);  CloseHandle(pi1.hProcess); // Явно закрываются первичный поток и процесс  CloseHandle(pi2.hProcess);  return 0;  } |
| Скриншот запуска |  |
| Просмотр через tasklist |  |

Задание 3. Windows-приложение OS03\_03 на языке С++, выводящее на консоль перечень выполняющихся процессов в данный момент в OS.

|  |  |
| --- | --- |
| Код программы | #include <iostream>  #include <Windows.h>  #include "TlHelp32.h"  #include <iomanip>  using namespace std;  int main()  {  DWORD pid = GetCurrentProcessId();  HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS\_SNAPALL, 0); // делает снапшот текущих процессов системы  // 1-ый параметр: флаг TH32CS\_SNAPALL включает в снапшот все процессы и потоки системы, а также кучу и модули процесса th32ProcessID (второй параметр)  // 2-ой параметр: PID процесса, включаемого в снапшот. 0 - текущий процесс  PROCESSENTRY32 peProcessEntry; // структура со списком процессов из снапшота  peProcessEntry.dwSize = sizeof(PROCESSENTRY32); // явное указание размера  wcout << L"Current PID: " << pid << endl;  try  {  // извлекает инфу о первом процессе снапшота. возвращает true, если первая  // запись снапшота списка процессов была скопирована в буфер, иначе false  if (!Process32First(snapshot, &peProcessEntry))  throw L"Process32First";  do  {  if (peProcessEntry.th32ProcessID == pid) // если один ин PID в снапшоте равен записанному PID текущего процесса  wcout << " ! CURRENTLY RUNNING PROCESS !\n";  wcout << L"Name\t\t" << peProcessEntry.szExeFile << "\n";  wcout << L"PID\t\t" << peProcessEntry.th32ProcessID << "\n";  wcout << L"Parent PID\t" << peProcessEntry.th32ParentProcessID << "\n";  wcout << L"--------------------------------------\n";  } while (Process32Next(snapshot, &peProcessEntry)); // итератор по снапшоту  }  catch (char\* errMessage)  {  wcout << L"[ERROR] " << errMessage;  }  system("pause");  return 0;  } |
| Скриншот запуска |  |

Задание 4. Linux-приложение OS03\_04 на языке С, выполняющее длинный цикл с временной задержкой и с выводом на консоль идентификатора процесса.

|  |  |
| --- | --- |
| Код программы | #include <stdio.h>  #include <stdlib.h>  #include <sys/types.h>  #include <unistd.h>  int main()  {  pid\_t pid = getpid();  for (short i = 1; i <= 10000; ++i)  {  printf("%d. PID = %d\n", i, pid);  sleep(1);  }  exit(0);  } |
| Скриншот запуска и применение команды ps | E:\3course\5sem\OS\Lab3\Image\OS03_04 (ps).png |
| Скриншот запуска и применение команды proc | E:\3course\5sem\OS\Lab3\Image\OS03_04 (proc).png |

Задание 5. Linux-приложение OS03\_05 на языке С, выполняющее цикл 100 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.

Приложение OS03\_05 должно создавать один дочерний процесс OS03\_05\_1 с помощью системного вызова fork. Процесс OS03\_05\_1 в этом случае не является отдельным модулем, а встроен (fork) в программный модуль OS03\_05.

Процесс OS03\_05\_1 - консольное Linux-приложение выполняющее цикл 50 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.

|  |  |
| --- | --- |
| Код программы | #include <stdio.h>  #include <stdlib.h>  #include <errno.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/wait.h>  void cycle(int counter, char\* message)  {  for (int i = 1; i <= counter; ++i)  {  printf("%d. PID = %d [%s]\n", i, getpid(), message);  sleep(2);  }  }  int main()  {  // fork() создает максимально подробную копию родительского процесса (копирует дескрипторы файлов итд).  // дочерний процесс получает новый pid (обычно ppid+1) и имеет pid старого процесса в качестве ppid.  // так как два процесса запускают один и тот же код, можно понять кто есть кто по возвращаемому значению.  // дочерний возвращает 0, а родительский возвращает pid дочернего  pid\_t pid;  switch (pid = fork())  {  case -1: perror("[ERROR] Fork() returned -1"); // fork возвращает -1 при ошибке (например нехватка свободных дескрипторов или памяти)  exit(-1);  case 0: cycle(50, "OS03\_05\_1"); // дочерний процесс возвращает 0  exit(0);  default: printf("[INFO] default = %d\n", pid); // родительский процесс возвращает pid дочернего  cycle(100, "OS03\_05");  wait(0); // ожидается завершение дочернего процесса  }  exit(0);  }  // Найти форкнутые процессы: ps -A | grep .out |
| Скриншот запуска и применение команды ps | E:\3course\5sem\OS\Lab3\Image\OS03_04 (ps).png |
| Скриншот запуска и применение команды proc | E:\3course\5sem\OS\Lab3\Image\OS03_04 (proc).png |