Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Операционные системы

Студент: Лопатнюк П.В.

ФИТ 3 курс 1 группа

Преподаватель: Бернацкий П.В.

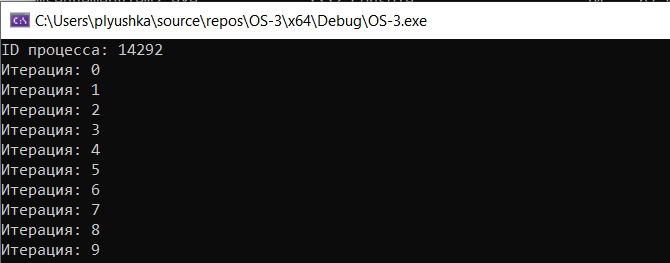
Минск 2024

**Лабораторная работа 3**

**Задание 01**

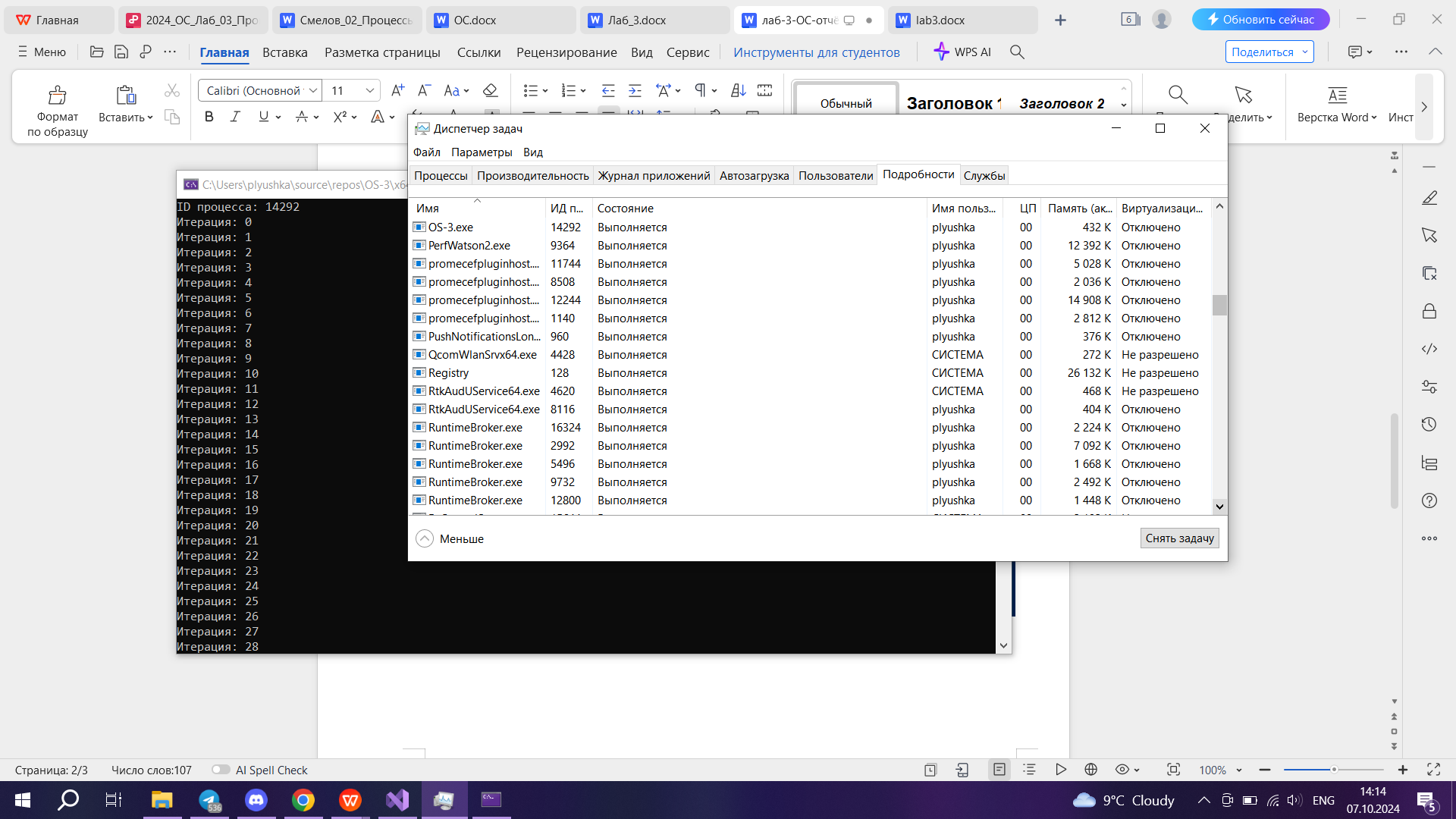
1. Разработайте консольное Windows-приложение OS03\_01 на языке С++, выполняющее длинный цикл с временной задержкой и с выводом на консоль идентификатора процесса.

|  |
| --- |
| #include <Windows.h>  #include <iostream>  int main()  {  setlocale(LC\_ALL, "rus");  DWORD processId = GetCurrentProcessId();  std::cout << "ID процесса: " << processId << std::endl;  for (int i = 0; i < 100; i++)  {  Sleep(1000);  std::cout << "Итерация: " << i << std::endl;  }  return 0;  } |

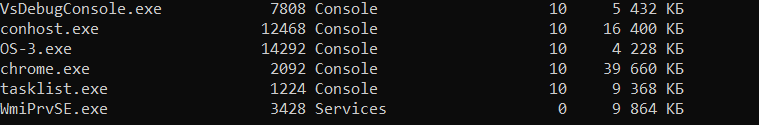


1. Продемонстрируйте информацию о процессе OS03\_01 с помощью утилит Task Manager, tasklist, PowerShell ISE и Performance Monitor.

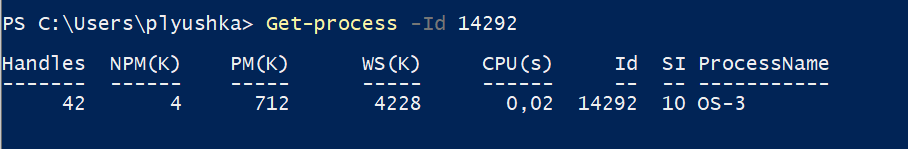
**Task Manager:**



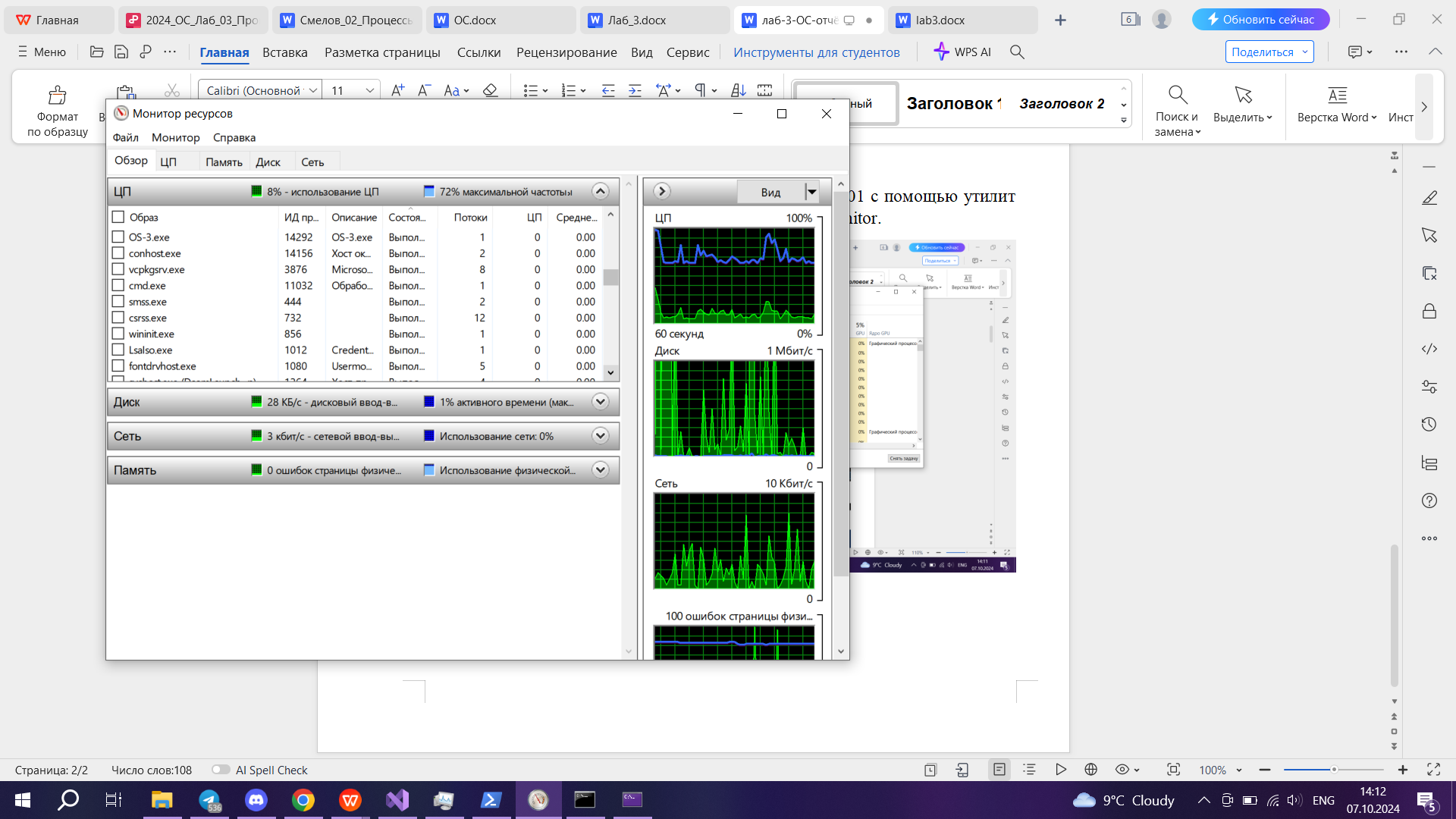
**Tasklist:**



**PowerShell ISE:**



**Performance Monitor:**



**Задание 02**

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

4. Приложение **OS03\_02** должно создавать два дочерних процесса **OS03\_02\_1** и **OS03\_02\_2.**

|  |
| --- |
| #include <Windows.h>  #include <iostream>  int main()  {  setlocale(LC\_ALL, "rus");  DWORD processId = GetCurrentProcessId();  std::cout << "ID процесса: " << processId << std::endl;  // создание двух дочерних процессов  STARTUPINFO si1, si2;  PROCESS\_INFORMATION pi1, pi2;  ZeroMemory(&si1, sizeof(si1));  si1.cb = sizeof(si1);  ZeroMemory(&pi1, sizeof(pi1));  ZeroMemory(&si2, sizeof(si2));  si2.cb = sizeof(si2);  ZeroMemory(&pi2, sizeof(pi2));  // запуск первого дочернего процесса  if (!CreateProcess(L"C:/Users/plyushka/source/repos/OS03\_02\_1/x64/Debug/OS03\_02\_1.exe",  NULL, // командная строка для нового процесса  NULL, // указатель на атрибуты безопасности для процесса  NULL, // указатель на атрибуты безопасности для потока  FALSE, // флаг наследования дескрипторов. FALSE = дескрипторы не наследуются дочерним процессом  0, // флаги создания процесса  NULL, // указатель на блок среды для нового процесса  NULL, // указатель на рабочую директорию для нового процесса  &si1, // указатель на структуру STARTUPINFO для конфигурации окна нового процесса  &pi1)) // указатель на структуру PROCESS\_INFORMATION для получения информации о новом процессе  {  std::cout << "Не удалось создать процесс OS03\_02\_1." << std::endl;  return 1;  }  std::cout << "Запущен процесс OS03\_02\_1. ID: " << pi1.dwProcessId << std::endl;  // запуск второго дочернего процесса  if (!CreateProcess(L"C:/Users/plyushka/source/repos/OS03\_02\_2/x64/Debug/OS03\_02\_2.exe",  NULL, NULL, NULL, FALSE, 0, NULL, NULL, &si2, &pi2))  {  std::cout << "Не удалось создать процесс OS03\_02\_2." << std::endl;  return 1;  }  std::cout << "Запущен процесс OS03\_02\_2. ID: " << pi2.dwProcessId << std::endl;  for (int i = 0; i < 100; i++)  {  Sleep(1000);  std::cout << "Основной процесс. Итерация: " << i << std::endl;  }  // ожидание завершения дочерних процессов  WaitForSingleObject(pi1.hProcess, INFINITE);  WaitForSingleObject(pi2.hProcess, INFINITE);  // закрытие дескрипторов  CloseHandle(pi1.hProcess);  CloseHandle(pi1.hThread);  CloseHandle(pi2.hProcess);  CloseHandle(pi2.hThread);  return 0;  } |

5. Процесс **OS03\_02\_1** - консольное Windows-приложение, выполняющее цикл 50 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.

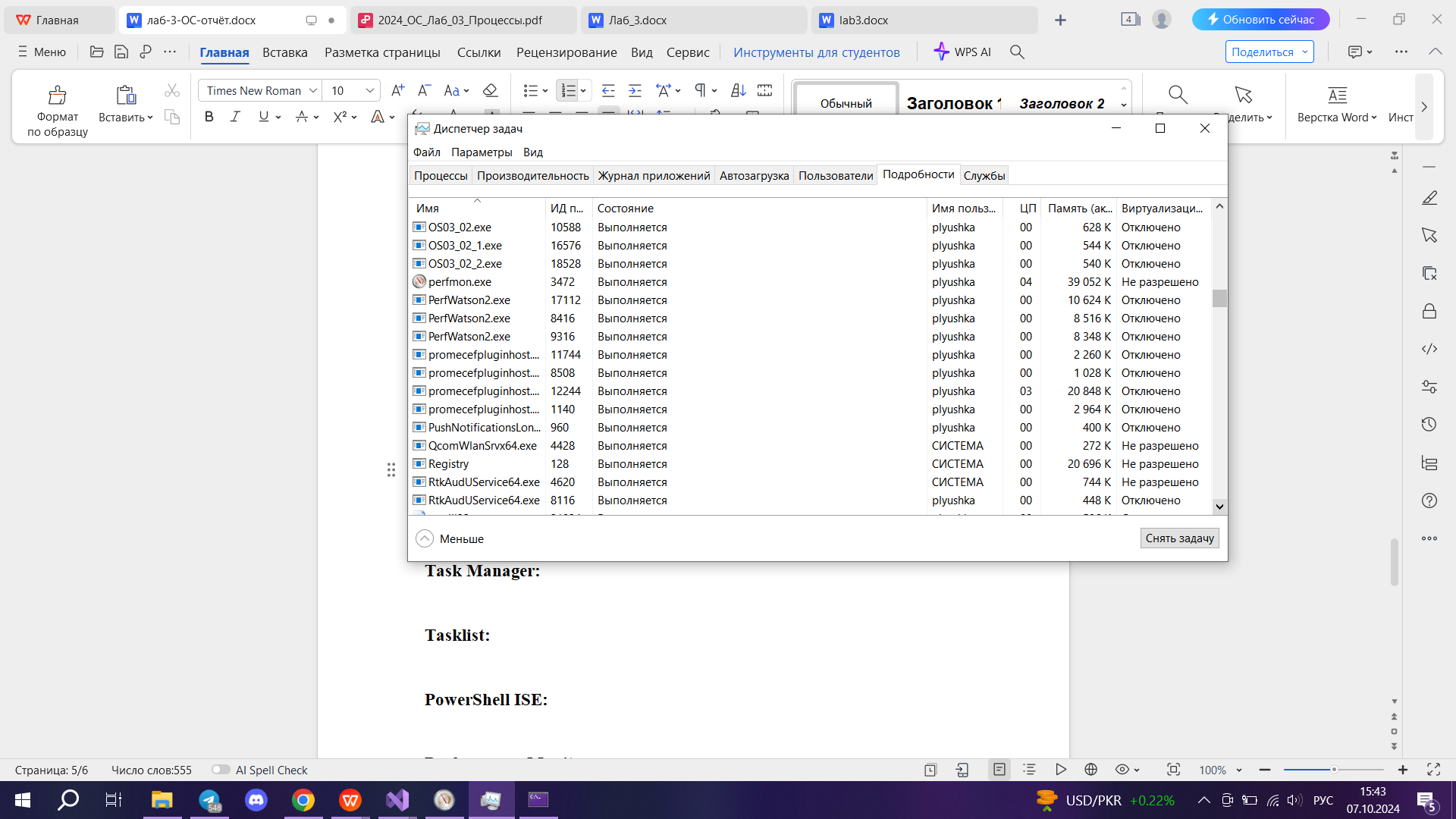
|  |
| --- |
| #include <Windows.h>  #include <iostream>  int main()  {  setlocale(LC\_ALL, "rus");  DWORD processId = GetCurrentProcessId();  std::cout << "ID процесса OS03\_02\_1: " << processId << std::endl;  for (int i = 0; i < 50; i++)  {  Sleep(1000);  std::cout << "OS03\_02\_1. Итерация: " << i << std::endl;  }  return 0;  } |

6. Процесс **OS03\_02\_2** - консольное Windows-приложение выполняющее цикл 125 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.

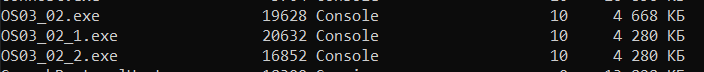
|  |
| --- |
| #include <Windows.h>  #include <iostream>  int main()  {  setlocale(LC\_ALL, "rus");  DWORD processId = GetCurrentProcessId();  std::cout << "ID процесса OS03\_02\_2: " << processId << std::endl;  for (int i = 0; i < 125; i++)  {  Sleep(1000);  std::cout << "OS03\_02\_2. Итерация: " << i << std::endl;  }  return 0;  } |

1. Продемонстрируйте информацию о процессах **OS03\_02, OS03\_02\_1** и **OS03\_02\_2** с помощью утилит **Task Manager**, **tasklist, PowerShell ISE** и **Performance Monitor**.

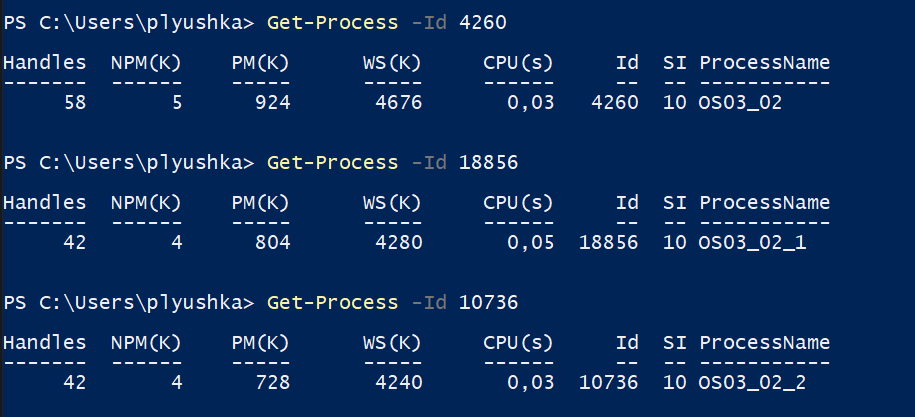
**Task Manager:**



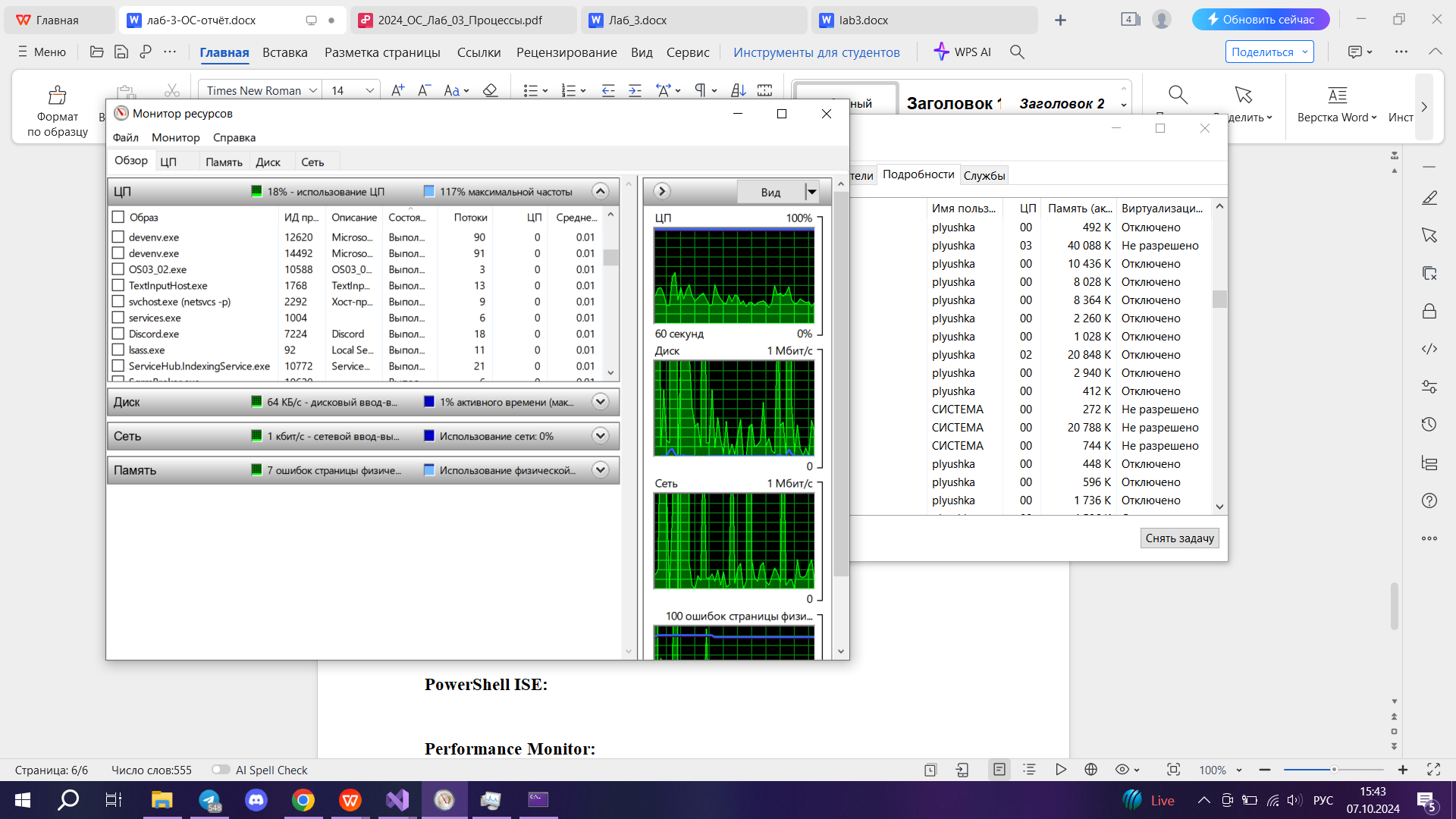
**Tasklist:**

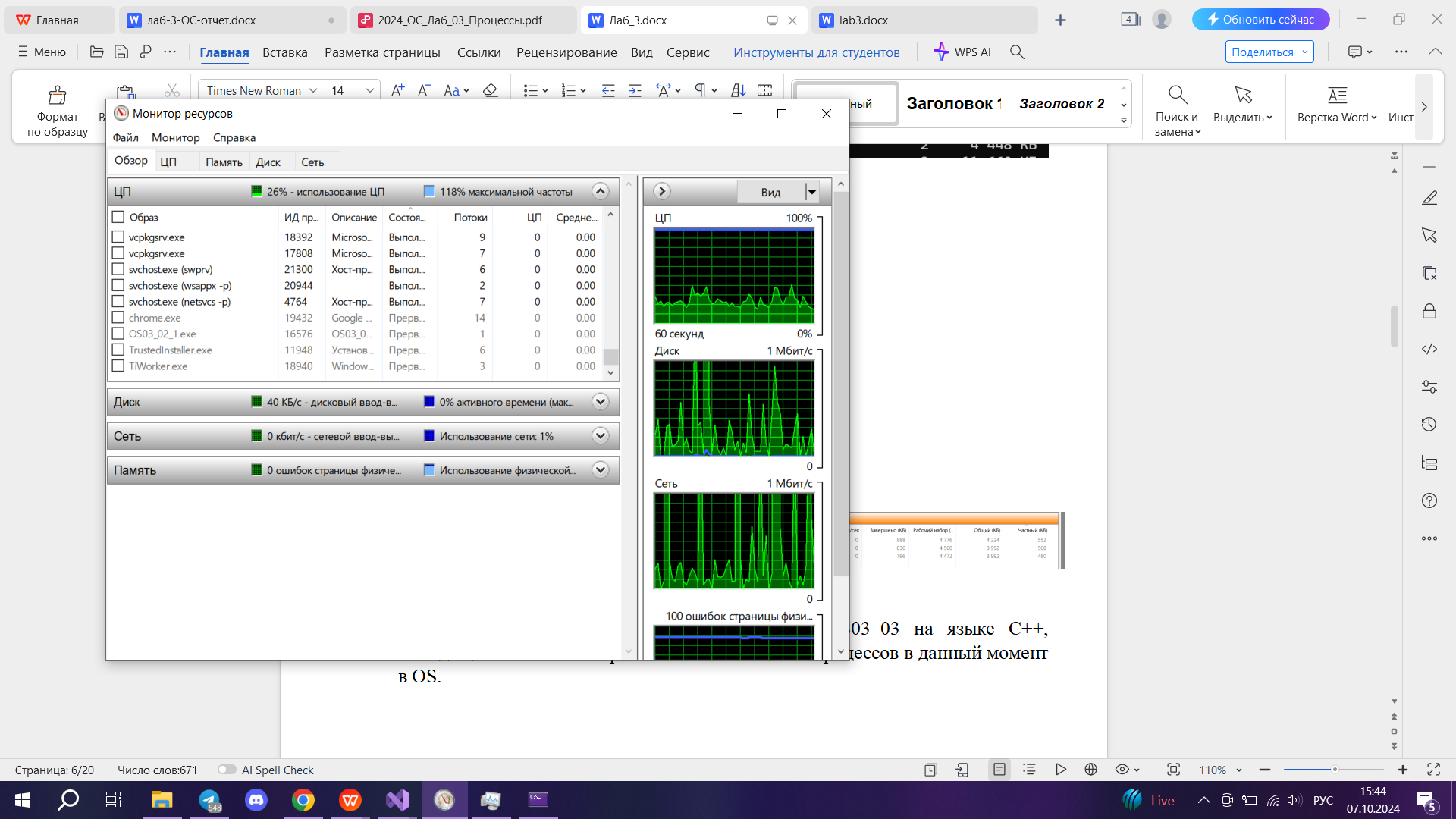


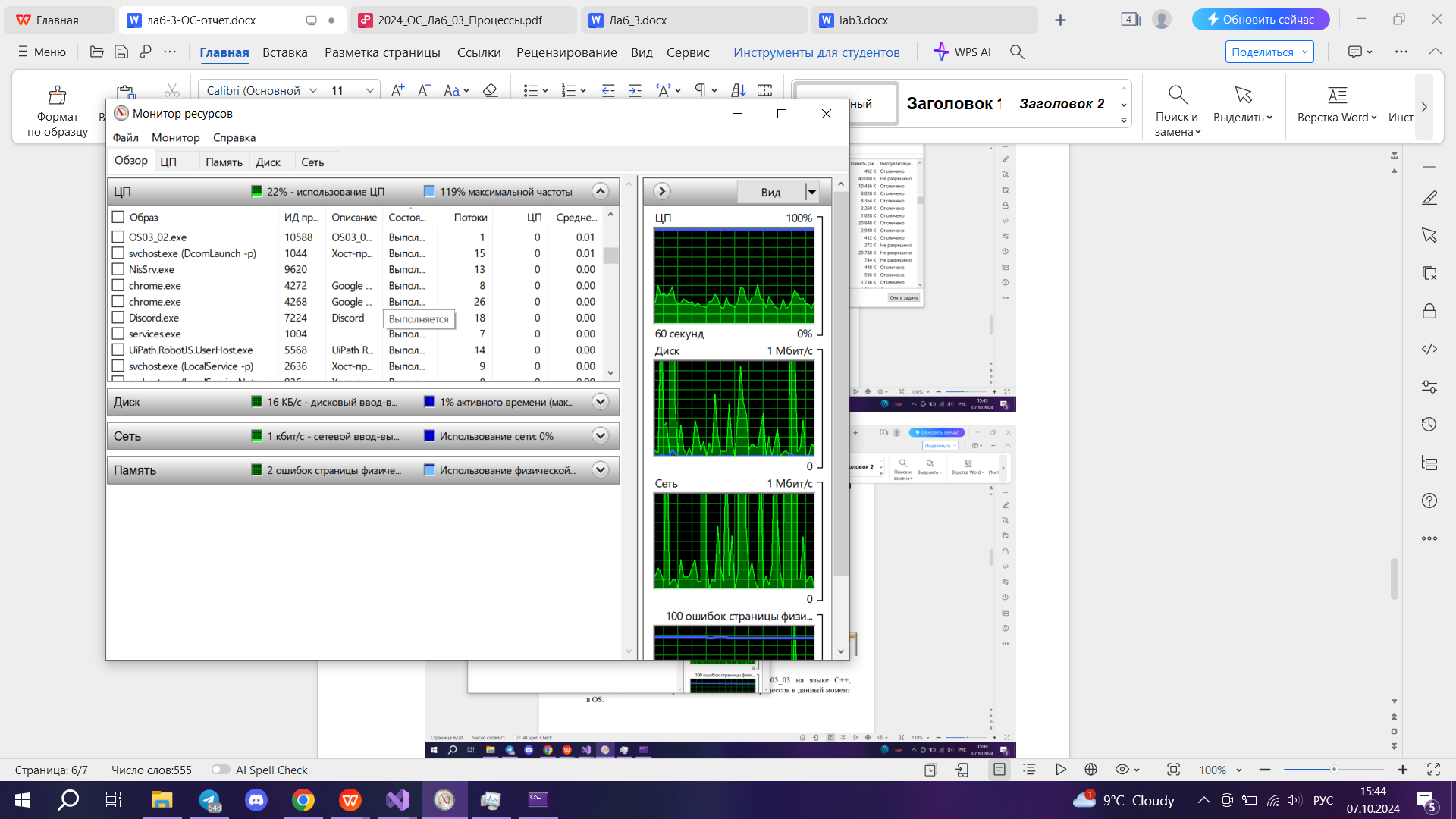
**PowerShell ISE:**



**Performance Monitor:**





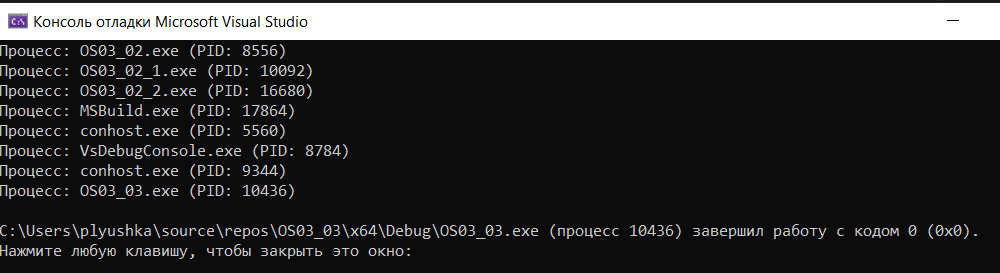


**Задание 03.**

8. Разработайте консольное Windows-приложение **OS03\_03** на языке С++, выводящее на консоль перечень выполняющихся процессов в данный момент в OS.

|  |
| --- |
| #include <Windows.h>  #include <tlhelp32.h>  #include <iostream>  void ListProcesses()  {  // снимок всех процессов в системе  HANDLE hProcessSnap = CreateToolhelp32Snapshot(TH32CS\_SNAPPROCESS, 0);  if (hProcessSnap == INVALID\_HANDLE\_VALUE)  {  std::cerr << "Ошибка создания снимка процессов." << std::endl;  return;  }  PROCESSENTRY32 pe32;  pe32.dwSize = sizeof(PROCESSENTRY32);  // первый процесс  if (!Process32First(hProcessSnap, &pe32))  {  std::cerr << "Ошибка при получении первого процесса." << std::endl;  CloseHandle(hProcessSnap);  return;  }  // все процессы в снимке  do  {  std::wcout << L"Процесс: " << pe32.szExeFile << L" (PID: " << pe32.th32ProcessID << L")" << std::endl;  } while (Process32Next(hProcessSnap, &pe32)); // переход к следующему процессу  CloseHandle(hProcessSnap); // закрываем снимок  }  int main()  {  setlocale(LC\_ALL, "rus");  std::cout << "Перечень процессов, выполняющихся в данный момент:" << std::endl;  ListProcesses();  return 0;  } |

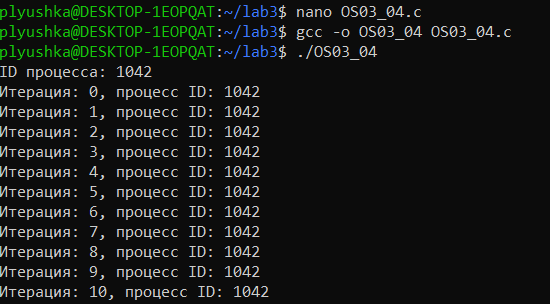
9. Запустите приложение **OS03\_02** и продемонстрируйте с помощью приложения **OS03\_03** в перечне процессов **OS03\_02, OS03\_02\_1, OS03\_02\_2** и **OS03\_03**.



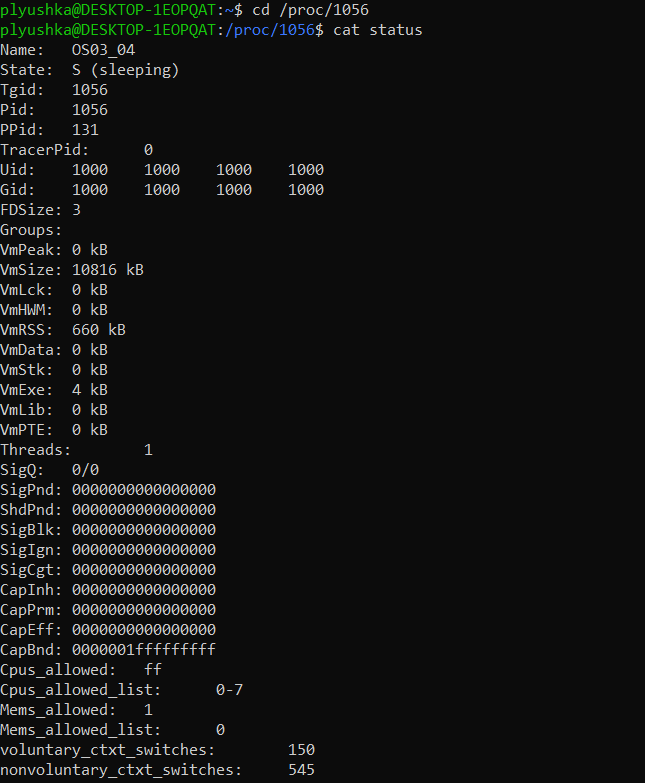
**Задание 04**

10. Разработайте консольное Linux-приложение **OS03\_04** на языке С, выполняющее длинный цикл с временной задержкой и с выводом на консоль идентификатора процесса.

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <sys/types.h>  int main()  {  pid\_t process\_id = getpid();  printf("ID процесса: %d\n", process\_id);  for (int i = 0; i < 100; i++) {  printf("Итерация: %d, процесс ID: %d\n", i, process\_id);  sleep(1);  }  return 0;  } |



11. Продемонстрируйте информацию о процессе **OS03\_04** с помощью файловой системы **/proc**.



**State:** Состояние процесса. Буква S означает "спящий, то есть процесс ожидает какое-то событие.

**Tgid:** Идентификатор группы потоков (Thread Group ID). Он совпадает с PID для однопоточных процессов.

**Pid:** Идентификатор процесса (Process ID).

**PPid:** Идентификатор родительского процесса (Parent Process ID).

**TracerPid:** Если процесс трассируется отладчиком (например, gdb), то это PID процесса, который его трассирует.

**Uid, Gid**: Это идентификаторы пользователя (User ID) и группы (Group ID), от имени которых выполняется процесс.

**FDSize:** Размер таблицы дескрипторов файлов (File Descriptor Size). Указывает количество файлов, которые процесс может открыть одновременно.

**Groups:** Группы, к которым принадлежит процесс.

**VmPeak:** Пиковый объем виртуальной памяти, который был использован процессом.

**VmSize**: Текущий объем виртуальной памяти, выделенной процессу.

**VmLck**: Количество заблокированной виртуальной памяти. Это память, которая не может быть выгружена на диск.

**VmHWM**: Пиковый объем используемой физической памяти (Resident Set Size, RSS).

**VmRSS:** Объем физической памяти, который процесс использует в данный момент.

**VmData, VmStk, VmExe, VmLib, VmPTE:** Различные сегменты виртуальной памяти:

**Threads:** Количество потоков, созданных процессом.

**SigQ:** Состояние очереди сигналов (сигналы, отправленные процессу и ожидающие обработки).

**CapInh, CapPrm, CapEff, CapBnd:** Это различные наборы прав доступа, используемые ядром Linux для управления возможностями процесса (Capabilities).

**Cpus\_allowed, Cpus\_allowed\_list:** Процесс может выполняться на процессорах, указанных в этих полях.

**Mems\_allowed, Mems\_allowed\_list:** Разрешенные узлы памяти для NUMA (неоднородная архитектура доступа к памяти).

**voluntary\_ctxt\_switches:** Количество добровольных переключений контекста. Процесс приостановился и уступил процессор.

**nonvoluntary\_ctxt\_switches:** Количество недобровольных переключений контекста, вызванных, например, вытеснением процессора другими процессами.

12. Продемонстрируйте информацию о процессе **OS03\_04** с помощью утилиты **ps**.



пользователь **plyushka** запустил процесс с идентификатором (PID) 1058.

**0.0 0.0** — процесс использует 0% процессорного времени и 0% оперативной памяти.

**10816** — объем виртуальной памяти (VmSize), который занимает процесс (в КБ).

**660** — объем физической памяти (VmRSS), используемой процессом (в КБ).

**tty1** — терминал, к которому привязан процесс (в данном случае tty1).

**S** — состояние процесса (Sleeping — спит/ожидает).

**19:34** — время запуска процесса.

**0:00** — количество времени, которое процесс активно выполнялся на CPU.

**./OS03\_04** — имя и путь исполняемого файла, который запущен в данном процессе.

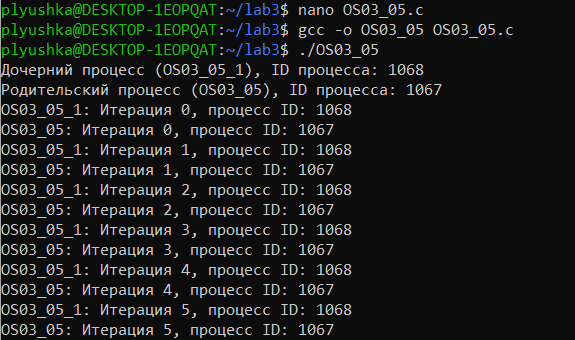
**Задание 05**

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

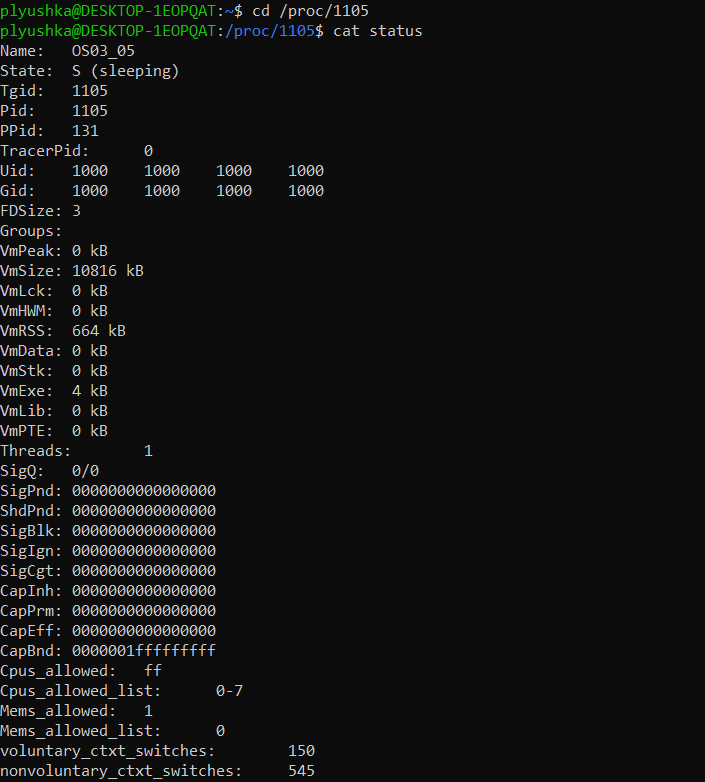
14. Приложение **OS03\_05** должно создавать один дочерний процесс **OS03\_05\_1** с помощью системного вызова **fork.** Процесс **OS03\_05\_1** в этом случае не является отдельным модулем, а встроен (fork) в программный модуль **OS03\_05**.

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

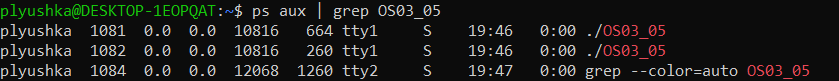
|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <sys/types.h>  #include <stdlib.h>  int main() {  pid\_t pid = fork(); // создание дочернего процесса  if (pid < 0) {  perror("Ошибка создания дочернего процесса");  return 1;  } else if (pid == 0) {  // дочерний процесс  printf("Дочерний процесс (OS03\_05\_1), ID процесса: %d\n", getpid());  for (int i = 0; i < 50; i++) {  printf("OS03\_05\_1: Итерация %d, процесс ID: %d\n", i, getpid());  sleep(1);  }  printf("OS03\_05\_1 завершен.\n");  } else {  // родительский процесс  printf("Родительский процесс (OS03\_05), ID процесса: %d\n", getpid());  for (int i = 0; i < 100; i++) {  printf("OS03\_05: Итерация %d, процесс ID: %d\n", i, getpid());  sleep(1);  }  printf("OS03\_05 завершен.\n");  }  return 0;  } |



1. Продемонстрируйте информацию о процессах **OS03\_05** и **OS03\_05\_1** с помощью файловой системы **/proc**.



17. Продемонстрируйте информацию о процессах **OS03\_05** и **OS03\_05\_1** с помощью утилиты **ps**.



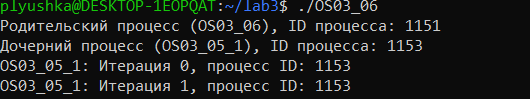
**Задание 06**

18. Разработайте консольное Linux-приложение **OS03\_06** на языке С, выполняющее цикл 100 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.

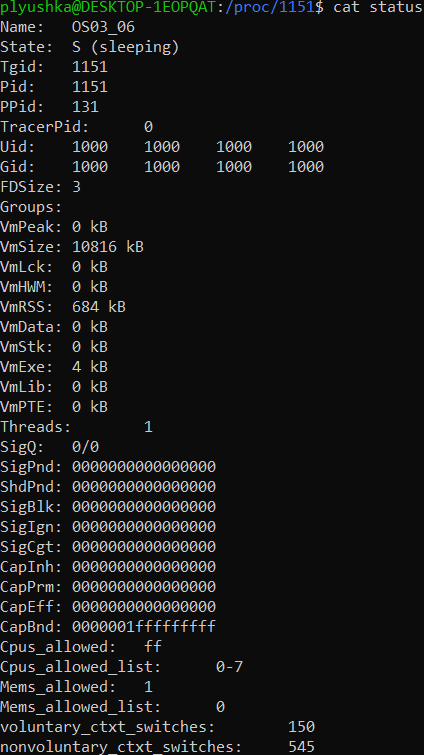
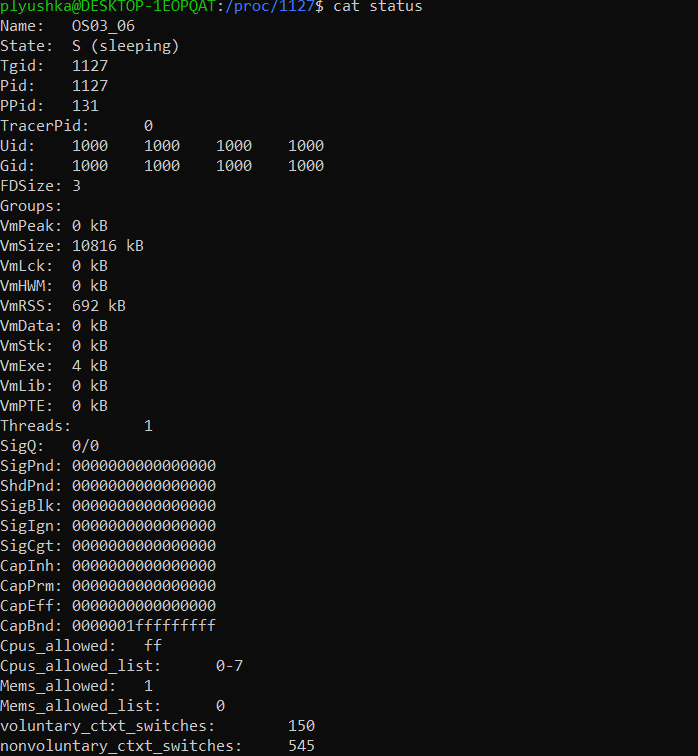
|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <stdlib.h>  int main() {  printf("Родительский процесс (OS03\_06), ID процесса: %d\n", getpid());  // запуск дочернего процесса  if (system("./OS03\_05\_1") == -1) {  perror("Ошибка запуска дочернего процесса OS03\_05\_1");  return 1;  }  // родительского процесс  for (int i = 0; i < 100; i++) {  printf("OS03\_06: Итерация %d, процесс ID: %d\n", i, getpid());  sleep(1);  }  printf("OS03\_06 завершен.\n");  return 0;  } |

19. Приложение **OS03\_06** должно создавать один дочерний процесс **OS03\_05\_1** (отдельный модуль) с помощью системного вызова **system.**

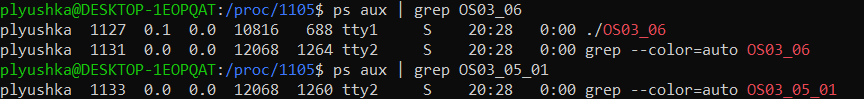
|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  int main() {  printf("Дочерний процесс (OS03\_05\_1), ID процесса: %d\n", getpid());  for (int i = 0; i < 50; i++) {  printf("OS03\_05\_1: Итерация %d, процесс ID: %d\n", i, getpid());  sleep(1);  }  printf("OS03\_05\_1 завершен.\n");  return 0;  } |



20. Продемонстрируйте информацию о процессах **OS03\_06** и **OS03\_05\_1** с помощью файловой системы **/proc**.



21. Продемонстрируйте информацию о процессах **OS03\_06** и **OS03\_05-1** с помощью утилиты **ps**.

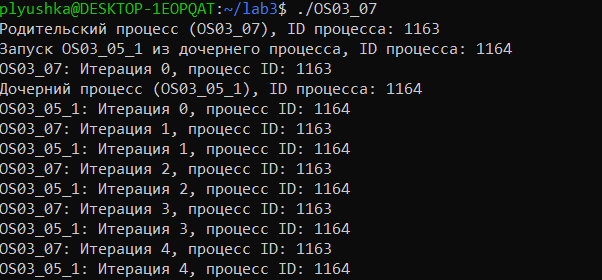


**Задание 07**

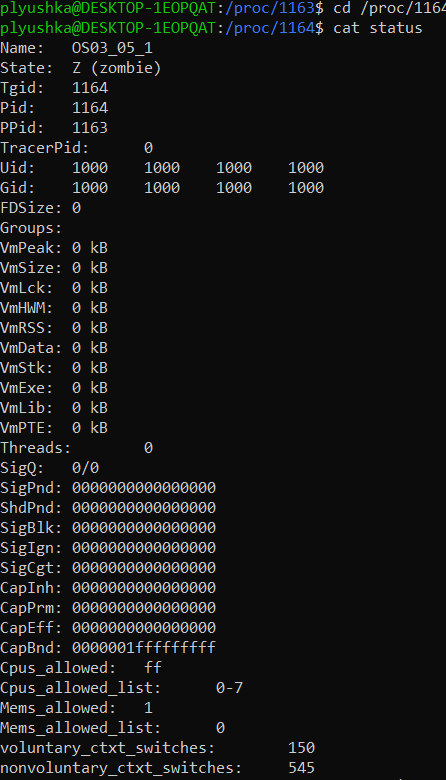
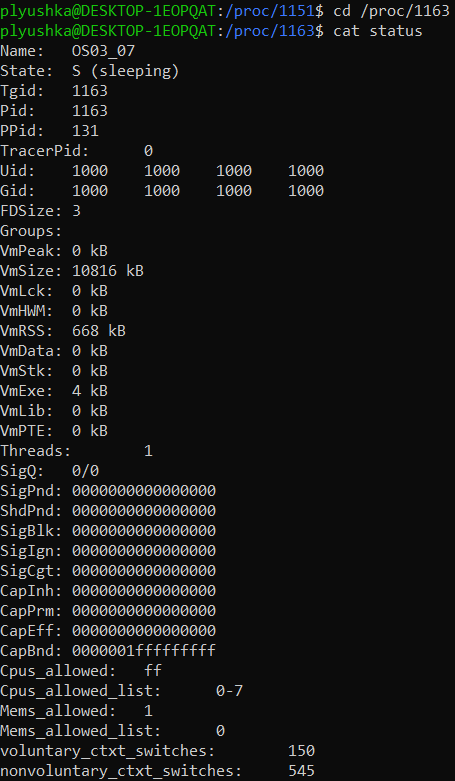
22. Разработайте консольное Linux-приложение **OS03\_07** на языке С, выполняющее цикл 100 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  #include <sys/types.h>  #include <sys/wait.h>  int main() {  pid\_t pid = fork();  if (pid < 0) {  perror("Ошибка создания дочернего процесса");  return 1;  } else if (pid == 0) {  printf("Запуск OS03\_05\_1 из дочернего процесса, ID процесса: %d\n", getpid());  execl("./OS03\_05\_1", "./OS03\_05\_1", NULL);  perror("Ошибка запуска OS03\_05\_1 через exec");  return 1;  } else {  printf("Родительский процесс (OS03\_07), ID процесса: %d\n", getpid());  for (int i = 0; i < 100; i++) {  printf("OS03\_07: Итерация %d, процесс ID: %d\n", i, getpid());  sleep(1);  }  wait(NULL);  printf("OS03\_07 завершен.\n");  }  return 0;  } |

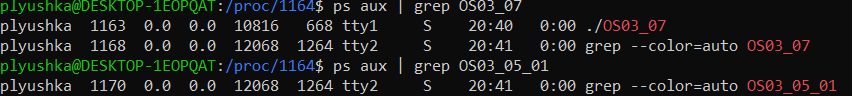
23. Приложение **OS03\_07** должно создавать один дочерний процесс **OS03\_05\_1** (отдельный модуль) с помощью системного вызова **exec.**



24. Продемонстрируйте информацию о процессах **OS03\_07** и **OS03\_05\_1** с помощью файловой системы **/proc**.



25. Продемонстрируйте информацию о процессах **OS03\_07** и **OS03\_05-1** с помощью утилиты **ps**.



26. Продемонстрируйте разницу системных вызовов **system** и **exec**.

1. **system()** вызывает командный интерпретатор (обычно /bin/sh), который интерпретирует и выполняет команду, как если бы она была введена вручную в командной строке.

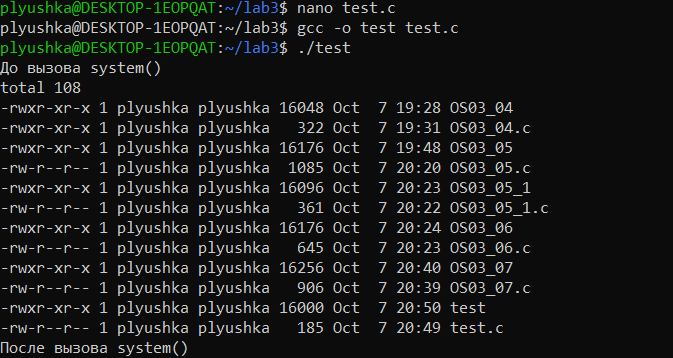
Поведение:

Родительский процесс создает дочерний процесс.

Дочерний процесс выполняет команду через оболочку.

Родительский процесс приостанавливается и ждет завершения команды.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main() {  printf("До вызова system()\n");  system("ls -l");  printf("После вызова system()\n");  return 0;  } |



2. **exec()** заменяет текущий процесс на новый процесс, указанный в аргументе функции. В отличие от system(), exec() не возвращает управление вызвавшему процессу, если не произошло ошибки.

Поведение:

При успешном выполнении exec() текущий процесс завершается, и начинается выполнение новой программы.

Использование fork() вместе с exec() позволяет создать новый дочерний процесс, который будет выполнять другую программу, оставляя родительский процесс нетронутым.

|  |
| --- |
| #include <stdio.h>  #include <unistd.h>  int main() {  printf("До вызо ва exec()\n");  execl("/bin/ls", "ls", "-l", NULL);  printf("После вызова exec()\n");  return 0;  } |

