Лабораторная работа 03

OC

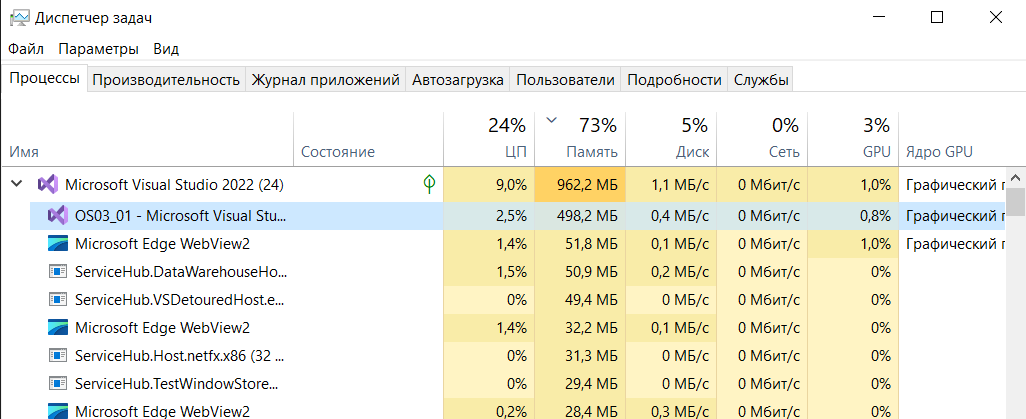
**Задание 01**

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

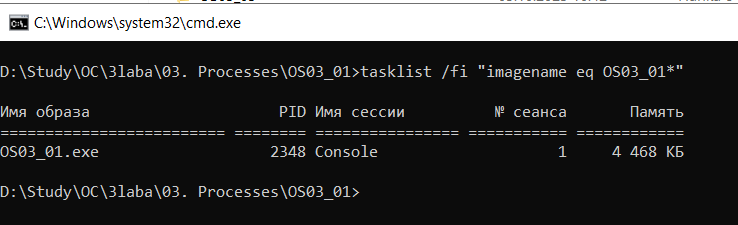
|  |
| --- |
| 1. #include <iostream> 2. #include <windows.h> 3. #include <process.h> 4. int main() 5. { 6. for (int i = 0; i < 10000; i++) { 7. Sleep(1000); 8. std::cout << "\t" << \_getpid(); 9. } 11. } |

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

**Task Manager.**

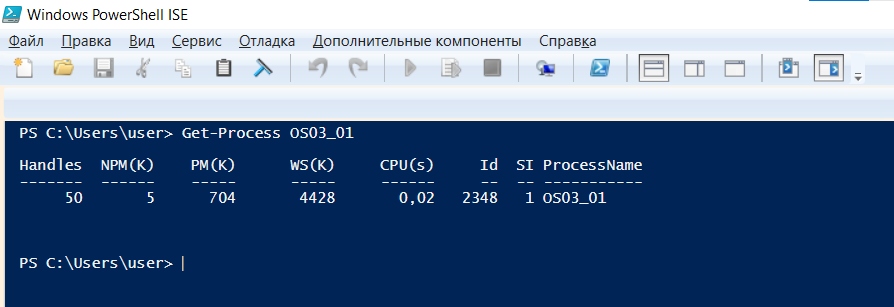
****

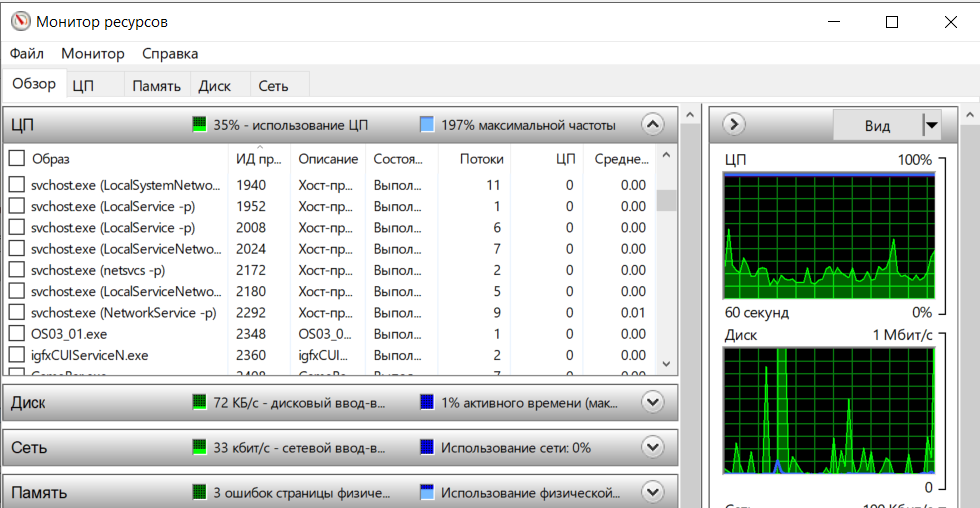
**TaskList**

****

**Без параметров**

****

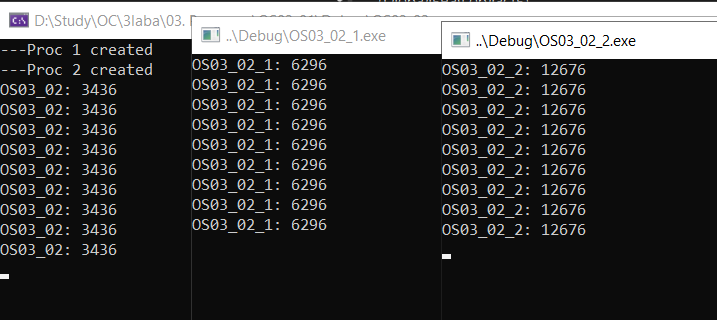
****

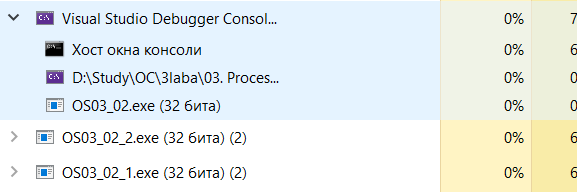
****

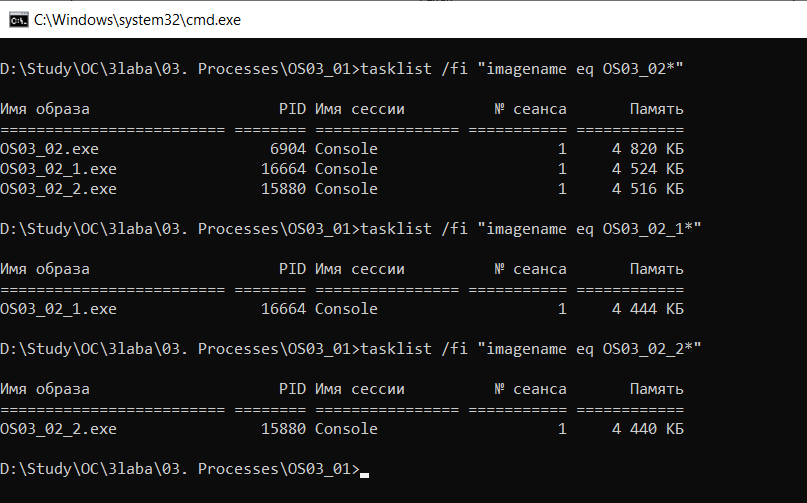
**Задание 02**

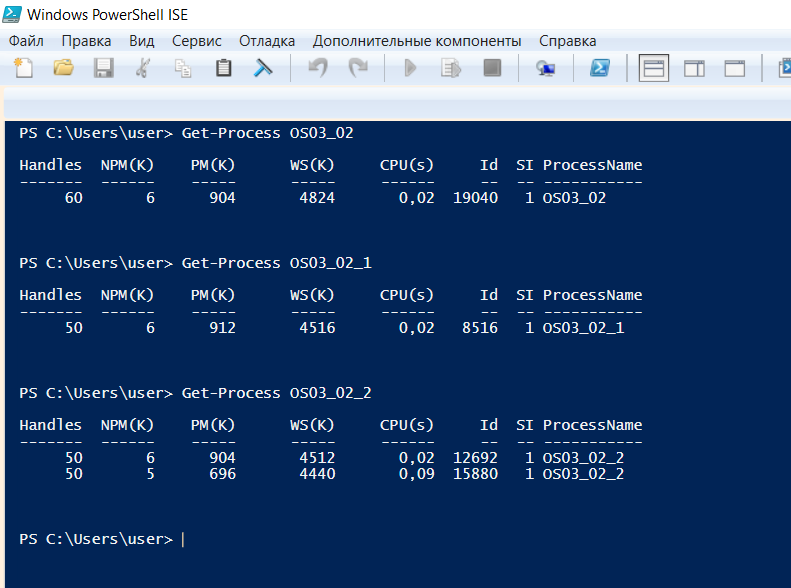
1. Разработайте консольное Windows-приложение **OS03\_02** на языке С++, выполняющее цикл 100 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.

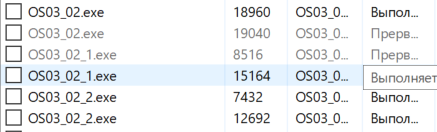
|  |
| --- |
| 1. #include <iostream> 2. #include <Windows.h> 3. #include <process.h> 4. int main() 5. { 6. LPCWSTR an = L"..\\Debug\\OS03\_02\_1.exe"; 7. LPCWSTR an1 = L"..\\Debug\\OS03\_02\_2.exe"; 8. STARTUPINFO si; 9. PROCESS\_INFORMATION pi, pi1; 10. ZeroMemory(&si, sizeof(STARTUPINFO)); 11. si.cb = sizeof(STARTUPINFO); 12. if (CreateProcess(an, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi) ) std::cout << "---Proc 1 created\n"; 13. else std::cout << "---Proc 1 not created\n"; 14. if (CreateProcess(an1, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE, NULL, NULL, &si, &pi1)) std::cout << "---Proc 2 created\n"; 15. else std::cout << "---Proc 2 not created\n"; 16. for (int i = 0; i < 100; i++) { 17. Sleep(1000); 18. std::cout << "OS03\_02: " << \_getpid() << "\n"; 19. } 20. WaitForSingleObject(pi.hProcess, INFINITE); 21. WaitForSingleObject(pi1.hProcess, INFINITE); 22. CloseHandle(pi.hProcess); 23. CloseHandle(pi1.hProcess); 24. return 0; 25. } |









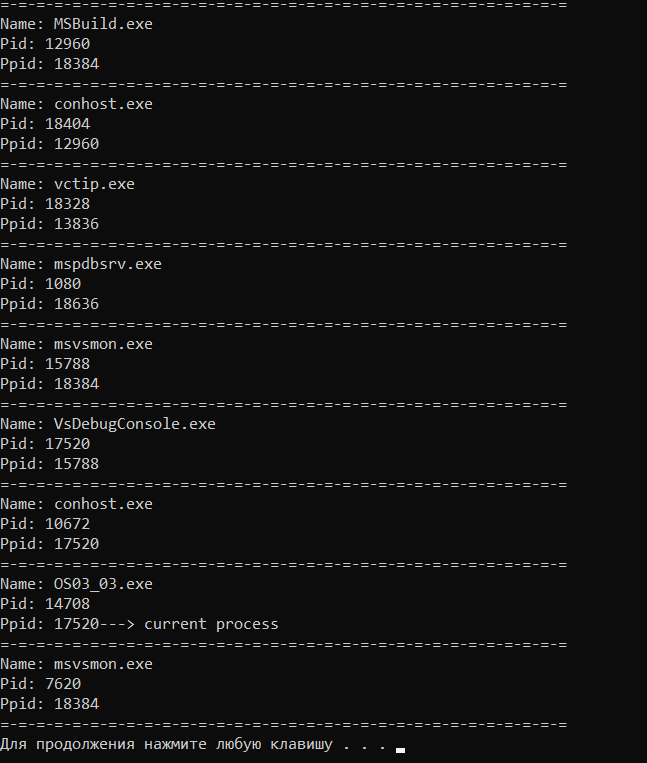


1. Приложение **OS03\_02** должно создавать два дочерних процесса **OS03\_02\_1** и **OS03\_02\_2.**
2. Процесс **OS03\_02\_1** - консольное Windows-приложение выполняющее цикл 50 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.
3. Процесс **OS03\_02\_2** - консольное Windows-приложение выполняющее цикл 125 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.
4. Продемонстрируйте информацию о процессах **OS03\_02, OS03\_02\_1** и **OS03\_02\_2** в с помощью утилит **Task Manager**, **tasklist,**  **PowerShell ISE** и **Performance Monitor**.

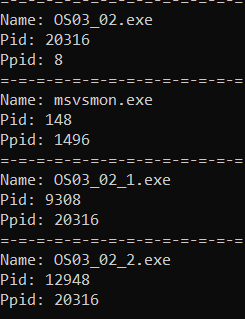
**Задание 03.**

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

|  |
| --- |
| #include <iostream>  #include <Windows.h>  #include <TlHelp32.h>  #include <iomanip>  int main()  {  DWORD pid = GetCurrentProcessId();  HANDLE snap = CreateToolhelp32Snapshot(TH32CS\_SNAPALL, 0);  PROCESSENTRY32 peProcessEntry;  peProcessEntry.dwSize = sizeof(PROCESSENTRY32);  std::wcout << L"Current pid: " << pid << "\n";  std::wcout << L"=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=\n";  try  {  if (!Process32First(snap, &peProcessEntry)) throw L"Process32First";  do  {  std::wcout << L"Name: " << peProcessEntry.szExeFile << "\n"  << L"Pid: " << peProcessEntry.th32ProcessID << "\n"  << L"Ppid: " << peProcessEntry.th32ParentProcessID;  if (peProcessEntry.th32ProcessID == pid) std::wcout << "---> current process";  std::wcout << L"\n=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=\n";  } while (Process32Next(snap, &peProcessEntry));  }  catch (char\* msg) {  std::wcout << L"ERROR: " << msg << "\n";  }  system("pause");  return 0;  }  // Run program: Ctrl + F5 or Debug > Start Without Debugging menu  // Debug program: F5 or Debug > Start Debugging menu  // Tips for Getting Started:  // 1. Use the Solution Explorer window to add/manage files  // 2. Use the Team Explorer window to connect to source control  // 3. Use the Output window to see build output and other messages  // 4. Use the Error List window to view errors  // 5. Go to Project > Add New Item to create new code files, or Project > Add Existing Item to add existing code files to the project  // 6. In the future, to open this project again, go to File > Open > Project and select the .sln file |



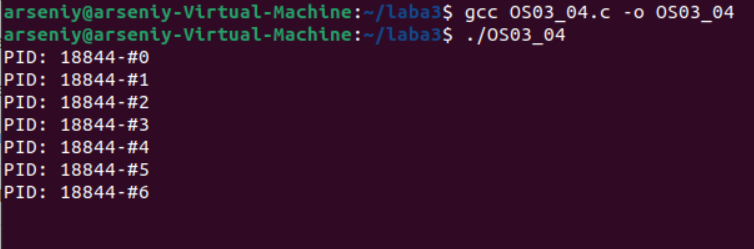
1. Запустите приложение **OS03\_02** и продемонстрируйте с помощью приложения **OS03\_03** в перечне процессов **OS03\_02, OS03\_02\_1, OS03\_02\_2** и **OS03\_03**.



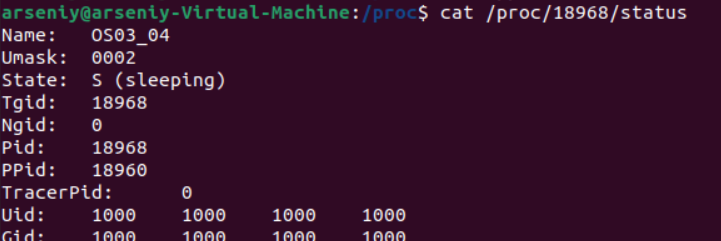
**Задание 04**

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

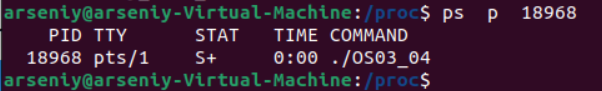
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <stdio.h>  #include <unistd.h>  #include <errno.h>  #include <sys/types.h>  #include <sys/wait.h>  int main(){  for (int i = 0; i < 1000; i++) {    printf("PID: %d-#%d\n", getpid(), i);  sleep(1);  }  return 0;  } |



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

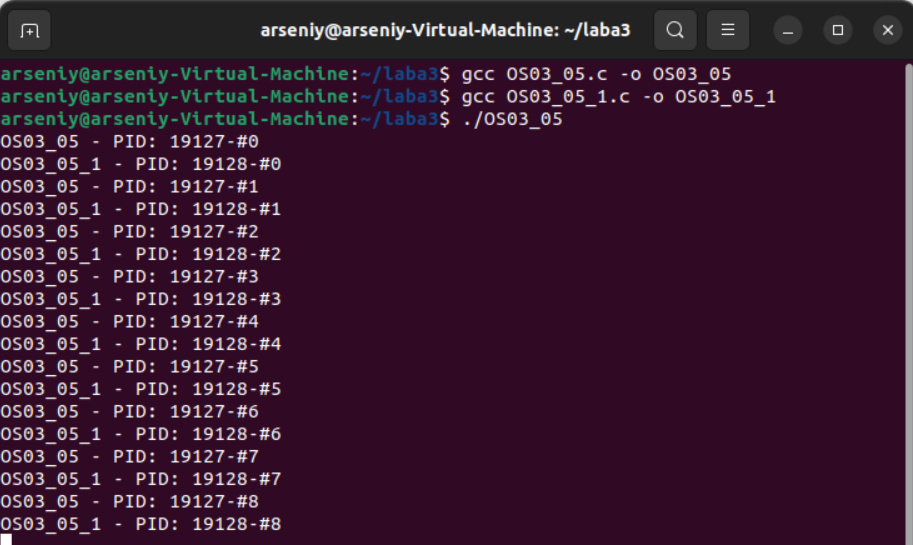


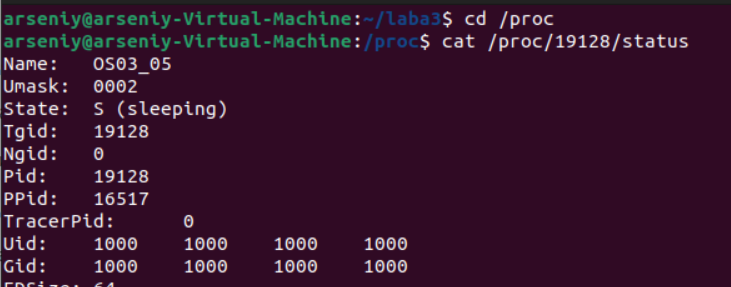
1. Продемонстрируйте информацию о процессе **OS03\_04** с помощью утилиты **ps**.

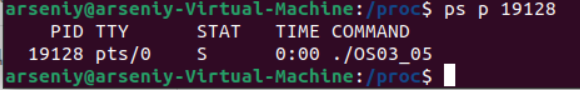


**Задание 05**

1. 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.
2. Приложение **OS03\_05** должно создавать один дочерний процесс **OS03\_05\_1** с помощью системного вызова **fork.** Процесс **OS03\_05\_1** в этом случае неявляется отдельным модулем, а встроен (fork) в программный модуль **OS03\_05**.
3. Процесс **OS03\_05\_1** - консольное Linux-приложение выполняющее цикл 50 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.
4. Продемонстрируйте информацию о процессах **OS03\_05** и **OS03\_05\_1** с помощью файловой системы **/proc**.
5. Продемонстрируйте информацию о процессах **OS03\_05** и **OS03\_05\_1** с помощью утилиты **ps**.

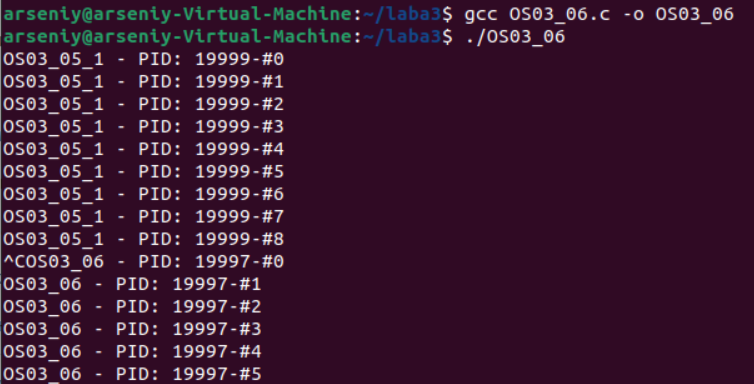


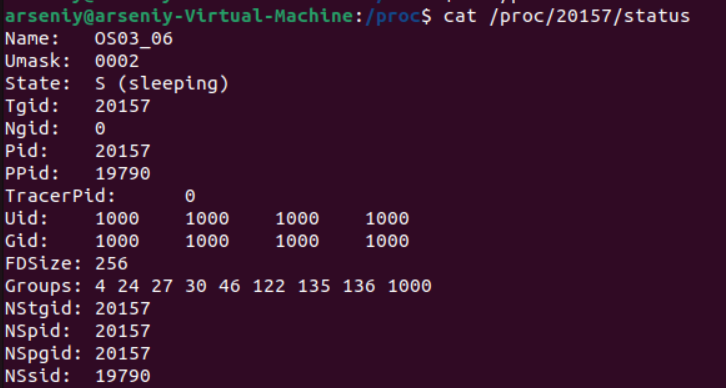


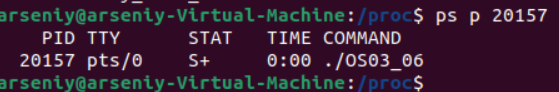


**Задание 06**

1. 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.
2. Приложение **OS03\_06** должно создавать один дочерний процесс **OS03\_05\_1** (отдельный модуль)с помощью системного вызова **system.**
3. Продемонстрируйте информацию о процессах **OS03\_06** и **OS03\_05\_1** с помощью файловой системы **/proc**.
4. Продемонстрируйте информацию о процессах **OS03\_06** и **OS03\_05-1** с помощью утилиты **ps**.

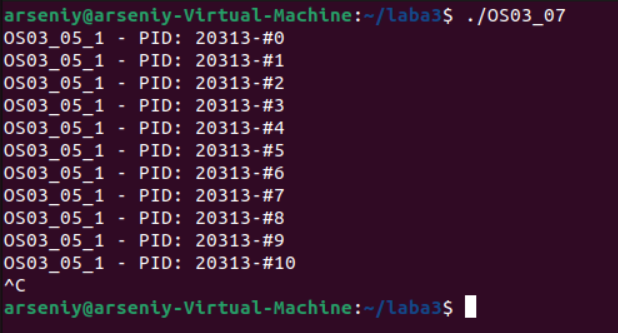


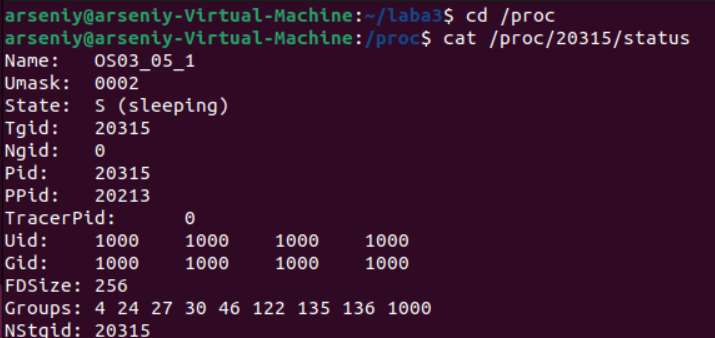


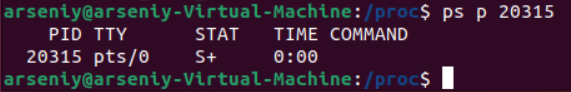


**Задание 07**

1. 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
2. Приложение **OS03\_07** должно создавать один дочерний процесс **OS03\_05\_1** (отдельный модуль)с помощью системного вызова **exec.**
3. Продемонстрируйте информацию о процессах **OS03\_07** и **OS03\_05\_1** с помощью файловой системы **/proc**.
4. Продемонстрируйте информацию о процессах **OS03\_07** и **OS03\_05-1** с помощью утилиты **ps**.
5. Продемонстрируйте разницу системных вызовов **system** и **exec**.







Различия system exec в том что, system создает новый поток для внешней команды. В то время, как exec заменяет текущий поток новым.

**Задание 08.ответьте на следующие вопросы**

1. 16 Дайте развернутое определение процесса OS.

**+1. Что такое POSIX?**

Набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой, библиотеку языка С и набор приложений и их интерфейсов

**+2. Что такое системный вызов?**

Механизм вызова прикладной программой функции ядра OS

**+3. Что такое аппаратное прерывание, программное прерывание?**

**Аппаратное** - возникает как реакция микропроцессоров на физический сигнал от некоторого устройства (клавиатура, системные часы, жесткий диск и т.д.), по времени возникновения эти прерывания асинхронны, т.е. происходят в случайные моменты времени

**Программное** - вызываются искусственно с помощью соответствующей команды из программы, предназначены для выполнения некоторых действий операционной системы, являются синхронными;

**+4. Что такое процесс?**

Выполняемая в данный момент программа. Единица работы OS - объект ядра

Процесс OS – единица работы OS - объект ядра OS+адресное пространство.

**+5. Что такое контекст процесса?**

данные, которые сохраняются при переключении процессов и предназначенные для продолжения работы

**6. Что такое родительский и дочерний процесс?**

Системный вызов fork() создает точную копию исходного процесса, называемого родительским процессом(parent process) Новый процесс называется дочерним процессом(child process)

**7. Что такое процесс инициализации OS?**

Процесс инициализации запускает все другие процессы, которые должны быть запущены, это родительский процесс для всего, что выполняется в системе. Другие процессы могут тоже создавать дочерние процессы, но если родительский процесс завершается, для его дочерних процессов родительским становится процесс инициализации.

**+8. Перечислите области памяти процесса и поясните их назначение.**

code, static, heap, data, stack

-----Возрастание адресов----->

**Code:** отображение программного кода в памяти  
**Static:** для константных объектов  
**Data:** отображение инициализированных переменных  
**Stack:** стек работает по схеме LIFO (последним вошел, первым вышел). Всякий раз, когда вызывается новый метод, содержащий примитивные значения или ссылки на объекты, то на вершине стека под них выделяется блок памяти.  
Когда метод завершает выполнение, блок памяти, отведенный для его нужд, очищается, и пространство становится доступным для следующего метода. Переменные в стеке существуют до тех пор, пока выполняется метод в котором они были созданы.

**Heap:** используется для выделения памяти во время выполнения программы. Но в отличие от стека, память, выделенная в куче, сохраняется и после того, как функция, вызвавшая выделение этой памяти, завершит работу. Язык С предоставляет программисту целый ряд средств управления памятью в куче.

**9. Чем отличаются системные процессы от пользовательских?**

Системные процессы являются частью ядра и всегда находятся в оперативной памяти. Такие процессы не имеют соответствующих им программ в виде исполняемых файлов и запускаются особым образом при инициализации ядра системы

**10. Что такое Windows-сервисы, Linux-демоны?**

**Windows-сервисы** - приложения, автоматически выполняемое системой при запуске операционной системы Windows и выполняющиеся вне зависимости от статуса пользователя

**Linux-демоны** - это программа, у которой есть определенная уникальная цель. Обычно, это служебные программы, которые незаметно работают в фоновом режиме для того чтобы отслеживать состояние и обслуживать определенные подсистемы и гарантировать правильную работу всей операционной системы в целом

**+11. С помощью каких системных вызовов можно создать дочерний процесс в Windows? Поясните разницу.**

CreateProcess - данный вызов имеет множество параметров, многие из которых «по умолчанию» Функция CreateProcess создает новый процесс и его первичный (главный) поток

CreateThread

Разница между CreateProcess и CreateThread заключается в том, что CreateProcess создает новый процесс со своим собственным адресным пространством, ресурсами и первичным потоком, в то время как CreateThread создает новый поток внутри существующего процесса, используя общие ресурсы и адресное пространство процесса.

**+12. С помощью каких системных вызовов можно создать дочерний процесс в Linux? Поясните разницу.**

fork, system

**fork()** создает точную копию вызывающего процесса. После выполнения системного вызова fork два процесса, родительский и дочерний, имеют единый образ памяти, единые строки описания конфигурации и одни и те же открытые файлы

**exec()** не создает новый процесс, а заменяет данные текущего процесса новыми данными. Вызов exec происходит таким образом, что переданная в качестве аргумента программа загружается в память вместо старой, которая вызвала exec.

**system()** неявно запускает новый процесс для обслуживания запроса и возвращает значение, полученное через дочерний процесс, который он изначально разветвил

**+13. Какие потоки данных доступны любому процессу автоматически?**

ввода, вывода, вывод ошибок

1. Поток ввода (stdin): Это поток данных, который предназначен для ввода данных в процесс. Обычно он связан с клавиатурой или другими устройствами ввода. Процесс может читать данные из потока ввода для получения ввода от пользователя или других источников.
2. Поток вывода (stdout): Это поток данных, который предназначен для вывода данных из процесса. Обычно он связан с экраном или другими устройствами вывода. Процесс может записывать данные в поток вывода для отображения результатов или информации для пользователя или других процессов.
3. Поток ошибок (stderr): Это поток данных, который предназначен для вывода сообщений об ошибках или предупреждений из процесса. Обычно он также связан с экраном или другими устройствами вывода. Процесс может записывать сообщения об ошибках в поток ошибок для информирования пользователя или других процессов о возникших проблемах.

**14. Поясните назначение системного вызова WaitForSingleObject в Windows-приложении.**

Приостанавливает выполнение потока до тех пор, пока не произойдет одно из двух событий:

* истечет таймаут ожидания;
* ожидаемый объект перейдет в сигнальное (signaled) состояние.

**15. Поясните назначение системного вызова wait в Linux-приложении.**

Блокирует вызывающий процесс до тех пор, пока один из его дочерних процессов не завершится (или не произойдет ошибка).

**16. Дайте развернутое определение процесса OS.**

процесс OS – единица работы OS - объект ядра OS+адресное пространство:

* создается ядром OS по системному вызову;
* адресное пространство (данные, программа, стек, куча);
* ресурсы: регистры, открытые файлы, родительский процесс, перечень связанных (дочерних) процессов, реальные страницы памяти, виртуальное адресное пространство, маркеры доступа (безопасность);
* процесс может создавать (с помощью системного вызова) дочерние процессы, в общем случае может образовываться дерево процессов;чя
* обычно соответствует работающей программе (например, exe-файлу в Windows);
* OS хранит список (или таблицу) объектов работающих процессов;
* при приостановке процесса в объекте процесса сохраняется вся информация (регистры, уведомления OS,… - контекст процесса), позволяющая возобновить работу процесса;
* процессы изолированы друг от друга;
* для обмена данными между процессами, применяется специальный механизм – механизм межпроцессного взаимодействия (IPC);

--------