Лабораторная работа 03

OC, ПОИТ-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.
2. Продемонстрируйте информацию о процессе **OS03\_01** в с помощью утилит **Task Manager**, **tasklist,**  **PowerShell ISE** и **Performance Monitor**.

**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.
2. Приложение **OS03\_02** должно создавать два дочерних процесса **OS03\_02\_1** и **OS03\_02\_2.**
3. Процесс **OS03\_02\_1** - консольное Windows-приложение выполняющее цикл 50 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.
4. Процесс **OS03\_02\_2** - консольное Windows-приложение выполняющее цикл 125 итераций с временной задержкой в 1 сек. с выводом на консоль идентификатора процесса.
5. 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.
2. 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.
2. Продемонстрируйте информацию о процессе **OS03\_04** с помощью файловой системы **/proc**.
3. 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**.

**Задание 08.ответьте на следующие вопросы**

1. Что такое POSIX?

**POSIX**  — набор стандартов, описывающих интерфейсы между [операционной системой](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) и [прикладной программой](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) (системный [API](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9)), библиотеку языка C и набор приложений и их интерфейсов. Стандарт создан для обеспечения совместимости различных [UNIX](https://ru.wikipedia.org/wiki/UNIX)-подобных операционных систем и переносимости прикладных программ на уровне [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4), но может быть использован и для не-Unix систем.

1. Что такое системный вызов?

системный вызов - механизм вызова прикладной программой функции ядра OS. Системный вызов осуществляется с помощью программного прерывания (RISC/x86 – int) или новый механизм Intel x86\_64 - SYSENTER/SYSEXIT, AMD - SYSCALL/SYSRET).

1. Что такое аппаратное прерывание, программное прерывание?

Прерывание — сигнал к процессору, испускаемый аппаратными средствами или программным обеспечением, и указывающий на событие, которое требует немедленного внимания.

**Аппаратные прерывания** используются устройствами для передачи информации о том, что они требуют внимания со стороны операционной системы. Внутренние аппаратные прерывания реализуются с использованием электронных сигналов оповещения, которые отправляются процессору от внешнего устройства, которое является частью самого компьютера, например контроллер диска, или внешнее периферийное устройство.

**Программное прерывание**: прерывание, генерируемое в процессоре путем выполнения инструкции. Программные прерывания часто используются для реализации системных вызовов, поскольку они приводят к вызову подпрограммы с изменением уровня вызова ЦП.

1. Что такое процесс?

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

1. Что такое контекст процесса?

В ядре находится таблица процессов, каждая запись которой описывает состояние одного из активных процессов в системе. В пространстве процесса хранится дополнительная информация, используемая в управлении протеканием процесса. Запись в таблице процессов и пространство процесса составляют в совокупности контекст процесса.

содержит

адресное пространство, содержимое регистров (общего назначения, счетчик команд, состояния процессора, вершина стека, …), объекты ядра OS (объекты процессов, потоков, безопасности, файлов и пр.), стек ядра (для этого процесса).

1. Что такое родительский и дочерний процесс?

Процесс, который в ходе своей работы создает один или несколько дочерних процессов, называется родительским. Создаваемый процесс называется дочерним

1. Что такое процесс инициализации OS?

Процесс, который первым стартует при запуске OS и порождает другие процессы, необходимые для работы OS.

Является корневым узлом в дереве процессов.

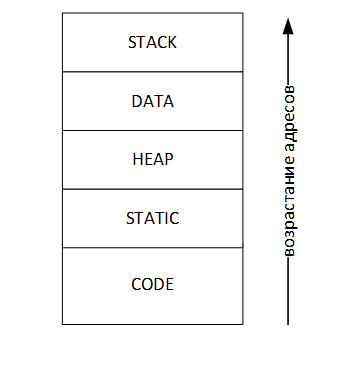
1. Перечислите области памяти процесса и поясните их назначение.

Каждый процесс в многозадачной ОС выполняется в собственной “песочнице”. Эта песочница представляет собой **виртуальное адресное пространство**, которое в 32-битном защищенном режиме **всегда имеет размер равный 4 гигабайтам**.

Kernel space

User mode space

(отношение обычно 1:3, 2:2, 3:1)



* **Стек** — это область памяти, где программа хранит информацию о вызываемых функциях, их аргументах и каждой локальной переменной в функции.
* **Данные:** данные машинных команд которые обрабатываются вместе с ними.
* **Куча** — это область памяти где программист волен делать всё что угодно.
* **Код** — область памяти, где будут храниться инструкции ЦП скомпилированной программы. Эти инструкции генерируются компилятором, но могут быть написаны вручную. Обратите внимание, что сегмент кода обычно делится на три части (текст, данные и BSS), но мы не будем так глубоко погружаться.

1. Чем отличаются системные процессы от пользовательских?

Системные процессы являются частью ядра ОС и всегда распо-

ложены в оперативной (основной) памяти.

Выполняемые инструкции

и данные этих процессов находятся в ядре системы, и поэтому они

могут вызывать функции и обращаться к данным, недоступным для

остальных процессов, например диспетчер страничного замещения,

диспетчер памяти ядра, диспетчер буферного кэша и другие.

(Пользовательские)Прикладные процессы, как правило, порождаются в рамках

пользовательского сеанса. Они могут выполняться как в интерактив-

ном, так и в фоновом режимах.

1. Что такое Windows-сервисы, Linux-демоны?

**Деймоны** - Это процессы, которые работают в фоновом режиме и не являются интерактивными. У них нет управляющего терминала.

Они выполняют определенные действия в заранее определенное время или в ответ на определенные события.

**Сервисы** - в Windows демоны называются **сервисами** .

1. С помощью каких системных вызовов можно создать дочерний процесс в Windows? Поясните разницу.

CreateProcess,

CreateProcessAsUser – заупскается в контексте системы безопасности пользователя

CreateProcessWithTokenW – новый процесс выполняется в контексте безопасности указанного токена

CreateProcessWithLogonW – разница в том, что вызывающей программе не нежуно вызывать функцию LogonUser, чтобы подтвердить подлинность пользователея и получить маркер доступа

1. С помощью каких системных вызовов можно создать дочерний процесс в Linux? Поясните разницу.

fork()

Начинается с Fork(), он создает точный клон вызывающего процесса, так называемый «дочерний» процесс

fork() создает новое адресное пространство, которое полностью идентично адресному пространству основного процесса. После выполнения этого системного вызова мы получаем два абсолютно одинаковых процесса - основной и порожденный. Функция fork() возвращает 0 в порожденном процессе и PID (Process ID - идентификатор порожденного процесса) - в основном. PID - это целое число.

В адресное пространство порожденного с помощью fork() процесса будет загружена новая программа и ее выполнение начнется с точки входа (адрес функции main).

1. Какие потоки данных доступны любому процессу автоматически?й

Поток ввода(0), поток вывода(1), поток вывода ошибок(2)

1. Поясните назначение системного вызова WaitForSingleObject в Windows-приложении.

Ожидание завершения работы дочернего процесса

1. Поясните назначение системного вызова wait в Linux-приложении.

Системный вызов wait переводит процесс в спящий режим и ожидает завершения дочернего процесса. Затем он заполняет аргумент кодом выхода дочернего процесса (если аргумент не NULL ).

1. Дайте развернутое определение процесса OS.

**Проце́сс** — это в выполняемая в данный момент программа. Выполнение процесса должно осуществляться последовательно. Процесс определяется как сущность, представляющая основную единицу работы, которая должна быть реализована в системе.