Модульный контроль №1

«Операционные системы»

Поперешняк С.В.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49

**1.** **ТЕОРИЯ ОС**

1.1.Место ОС в структуре компьютера

**Расположение ОС в общей структуре компьютера.**

· Оборудование, аппаратура: физический устройства, микроархитектура, машинный язык.

· **Системные программы: ОС**, компиляторы, редакторы, интерпретаторы.

· Программы-приложения: Банковская система, заказ билетов, веб-браузер, ворд и т.д.

1.2.История ОС

Спочатку були комп'ютери, які взагалі обходились без ОС. На них запускалась одна програма яка завантажувалась з перфокарти.

Потім з'явилось допоміжне ПЗ, таке як асемблери, чи стрічки з допоміжними бібліотеками (наприклад для вводу і виводу), які й стали зародком ОС.

Із ростом швидкодії час роботи програм став на порядок меншим за час передачі обладнання в руки іншому користувачу. Черга на виконання перетворилась з буквальної черги програмістів перед дверима операційного залу в ряди котушок з магнітними стрічками, та стовпчики перфокарт. З'явилась необхідність в бібліотеках для керування послідовністю завантаження перфокарт, вибору магнітного пристрою для читання та запису, логування часу та помилок.

Ці бібліотеки стали невидимим ПЗ, яке запускалось перед першою задачею користувача, і керувало її завантаженням та виконанням, виділяло ресурси, записувало результати роботи, слідкувало за правильним завершенням та звільненням ресурсів та негайно після цього переходило до наступної задачі. Такі фонові програми, ще до впровадження терміну ОС називались «моніторами».

При переході в еру персональних комп'ютерів відбувся зсув в розумінні поняття операційної системи. Першим автомобілям бракувало спідометрів, радіо та склоочисників, які згодом стали стандартними компонентами, так само першим ОС бракувало багато необов'язкових функцій (наприклад текстові редактори, файлові менеджери і т.д), які на ПК стали обов'язковими компонентами ОС. Сучасний користувач навіть не уявляє ОС без графічного інтерфейсу. Проте деякі програми, такі як СУБД, чи електронні таблиці, досі вважаються додатками, і постачаються окремо. А справжнім нащадком перших ОС є те, що сьогодні називається ядром. В технічних колах все ще використовують старе поняття ОС, як лише ядра, через розробку вбудованих систем для різних видів пристроїв з можливостями обробки даних — від наручних годинників до промислових роботів. Вбудовані ОС сьогодні не надто відрізняються від своїх предків з 1950-тих.

Вважається, що першою ОС була GM-NAA I/O, створена в General Motors для IBM 704. Багато інших ранніх ОС для комп'ютерів IBM були створені її клієнтами.

Перші ОС були дуже різноманітними та чисельними, бо кожен виробник створював одну чи кілька ОС для кожної моделі своїх комп'ютерів.

Стан справ змінився в 1960-тих, коли IBM, на той час вже лідер поставки апаратного забезпечення, припинив роботу над існуючими системами, і зоседерив всі зусилля над розробкою серії машин System/360, які використовували однаковий набір інструкцій, та архітектуру вводу-виводу. Для серії під керуванням Фреда Брукса була створена єдина операційна система — OS/360. Проблеми з якими зіткнулись розробники стали легендарними, і пізніше були описані в книзі Фреда Брукса «Міфічний людино-місяць». Через проблеми з відмінністями поміж апаратного забезпечення серії, та затримки в розробці, замість однієї OS/360 було створене ціле сімейство ОС.

1.3.Операционная система как расширенная машина

Архитектура большинства компьютеров на уровне машинного языка примитивна и не удобна для работы с программами. Программа, скрывающая истину об аппаратном обеспечении и предоставляющая простой поименованный список файлов, с которыми можно непосредственно работать является ОС.

Операционная система скрывает множество не приятной работы ,которая связана с прерываниями, счетчиками времени, организацией памяти и другими элементами низкого уровня. С точки зрения пользователя ОС выполняет функцию виртуальной машины, в которой проще программировать и работать.

**Вывод**: ОС предоставляет пользователю ряд возможностей, которые могут использовать программы с помощью специальных команд, которые называют системными вызовами.

**Пометка**: данная концепция предоставляет схему сверху вниз.

1.4.Операционная система как менеджер ресурсов

Концепция снизу вверх дает представление об ОС как о механизме присутствующем в компьютере для управления всеми частями этой машины. Работа ОС заключается в обеспечении организованного и контролируемого распределения процессоров, памяти, устройство ввода/вывода, между различными программами, состязающимися за их использование.

Основная задача ОС заключается в отслеживании того кто какой ресурс использует, в обработке запроса на ресурс, в подсчете коэффициента загрузки и решении конфликтующих ситуаций.

Управление ресурсами включает в себя мультиплексирование двумя способами:

1. Во времени. Пользователь и программы используют его по очереди.

2. В пространстве. Пользователи и программы получают часть ресурсов.

1.5.Лидеры ОС в настоящее время

Різні категорії комп'ютерів використовують широкий спектр операційних систем, і частка використання відрізняється в одній категорії від інших.

У деяких категоріях операційні системи одного сімейства є домінуючим. Наприклад, більшість настільних комп'ютерів використовують [Microsoft Windows](http://uk.wikipedia.org/wiki/Microsoft_Windows) а більшість суперкомп'ютерів використовують [Linux](http://uk.wikipedia.org/wiki/Linux). В інших категоріях, таких як [смартфони](http://uk.wikipedia.org/wiki/%D0%A1%D0%BC%D0%B0%D1%80%D1%82%D1%84%D0%BE%D0%BD) і [сервери](http://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80), існує велика різноманітність і конкуренція.

Інформацію про частку операційної системи важко отримати. У більшості категорій нижче, не існує надійного первинного джерела або методології відбору.

Існує небагато відкрито опублікованої інформації про частку використання операційних систем на настільних комп'ютерах і [ноутбуках](http://uk.wikipedia.org/wiki/%D0%9D%D0%BE%D1%83%D1%82%D0%B1%D1%83%D0%BA).

У промові до інвесторів у лютому 2009 року [Стів Балмер](http://uk.wikipedia.org/wiki/%D0%A1%D1%82%D1%96%D0%B2_%D0%91%D0%B0%D0%BB%D0%BC%D0%B5%D1%80) з [Microsoft](http://uk.wikipedia.org/wiki/Microsoft) представ слайд на основі досліджень Microsoft: поки він не показав цифри, кругова діаграма для [Linux](http://uk.wikipedia.org/wiki/Linux) і [Apple](http://uk.wikipedia.org/wiki/Apple) Макінтош, виглядала як приблизно 5-6% від домашніх і офісних ПК кожна.

Частка використання веб-клієнтів часто слугує основою для визначення частки настільних, але багато комп'ютерів не використовуються для веб-серфінгу. Статистика веб-клієнта дозволяє припустити, що [Microsoft Windows](http://uk.wikipedia.org/wiki/Microsoft_Windows) має близько 87%, [Apple](http://uk.wikipedia.org/wiki/Apple) [Mac OS](http://uk.wikipedia.org/wiki/Mac_OS) 7% і [Linux](http://uk.wikipedia.org/wiki/Linux) 1%. Співвідношенню між частками настільних і веб-клієнтів також кидає виклик мобільний доступ в Інтернет, який виріс до 1% в 2009 році і 4% в 2010 році.

Фінансовий директор Microsoft Пітер Кляйн заявив, у липні 2010 року, що [Windows 7](http://uk.wikipedia.org/wiki/Windows_7) тепер працює на більш ніж 15% всіх комп'ютерів у всьому світі. Дослідження [Forrester Research](http://uk.wikipedia.org/wiki/Forrester_Research) настільних операційних систем, що використовуються в Північній Америці і європейських країнах у 2010 році виявили Windows 7 на 10% всіх комерційних настільних комп'ютерів, [Windows XP](http://uk.wikipedia.org/wiki/Windows_XP) на 75% і [Vista](http://uk.wikipedia.org/wiki/Vista) на 7%.

1.6.ОС построенные на принципах UNIX

У [1983](http://uk.wikipedia.org/wiki/1983) році [Річард Столлмен](http://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%BE%D0%BB%D0%BC%D0%B5%D0%BD_%D0%A0%D1%96%D1%87%D0%B0%D1%80%D0%B4) оголосив про створення проекту [GNU](http://uk.wikipedia.org/wiki/GNU) — спроби створити [вільної](http://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F) UNIX-подібної операційної системи з нуля, без використання оригінального [сирцевого коду](http://uk.wikipedia.org/wiki/%D0%A1%D0%B8%D1%80%D1%86%D0%B5%D0%B2%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4). Більша частина програмного забезпечення, розробленого в рамках цього проекту — такого, як [GNU toolchain](http://uk.wikipedia.org/wiki/GNU_toolchain), [Glibc](http://uk.wikipedia.org/wiki/Glibc) (стандартна бібліотека мови [Сі](http://uk.wikipedia.org/wiki/%D0%A1%D1%96_%28%D0%BC%D0%BE%D0%B2%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%29)) та [Coreutils](http://uk.wikipedia.org/w/index.php?title=Coreutils&action=edit&redlink=1) — відіграють ключову роль у інших вільних операційних системах. Однак, роботи з створення заміни для [ядра](http://uk.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_%28%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%BE%D1%97_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B8%29) UNIX, необхідного для повного виконання задач GNU, відбувались дуже повільно. На теперішній час [GNU Hurd](http://uk.wikipedia.org/wiki/GNU_Hurd) — спроба створити сучасне ядро на основі [мікроядерної](http://uk.wikipedia.org/wiki/%D0%9C%D1%96%D0%BA%D1%80%D0%BE%D1%8F%D0%B4%D1%80%D0%BE) архітектури [Mach](http://uk.wikipedia.org/wiki/Mach) — все ще далека від завершення.

У [1991](http://uk.wikipedia.org/wiki/1991) році, коли [Лінус Торвальдс](http://uk.wikipedia.org/wiki/%D0%A2%D0%BE%D1%80%D0%B2%D0%B0%D0%BB%D1%8C%D0%B4%D1%81_%D0%9B%D1%96%D0%BD%D1%83%D1%81) опублікував ядро [Linux](http://uk.wikipedia.org/wiki/Linux_%28%D1%8F%D0%B4%D1%80%D0%BE%29) та залучив помічників, використання інструментів, розроблених у рамках проекту [GNU](http://uk.wikipedia.org/wiki/GNU), було очевидним вибором. Об'єднавшись з ядром Linux, програмне забезпечення GNU стало основою для UNIX-подібної операційної системи, відомою як [Linux](http://uk.wikipedia.org/wiki/Linux). [Дистрибутиви](http://uk.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D0%B8%D0%B2_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%BE%D1%97_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B8) цієї системи (такі як [Red Hat](http://uk.wikipedia.org/wiki/Red_Hat) та [Debian](http://uk.wikipedia.org/wiki/Debian)), які включають ядро, утиліти GNU та додаткове програмне забезпечення стали популярними як серед аматорів, так і серед фахівців.

У результаті регулювання юридичної справи, відкритою [UNIX Systems Laboratories](http://uk.wikipedia.org/w/index.php?title=UNIX_Systems_Laboratories&action=edit&redlink=1) проти [університету Берклі](http://uk.wikipedia.org/wiki/%D0%A3%D0%BD%D1%96%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%82%D0%B5%D1%82_%D0%91%D0%B5%D1%80%D0%BA%D0%BB%D1%96) та **Berkeley Software Design Inc.**, було встановлено, що університет може розповсюджувати BSD UNIX, в тому числі і безкоштовно. Після цього були відновлені експерименти, пов'язані з [BSD](http://uk.wikipedia.org/wiki/BSD)-версією UNIX. Незабаром розробка BSD UNIX була продовжена у декількох напрямах одночасно, що призвело до появи проектів, відомих як [FreeBSD](http://uk.wikipedia.org/wiki/FreeBSD), [NetBSD](http://uk.wikipedia.org/wiki/NetBSD), [OpenBSD](http://uk.wikipedia.org/wiki/OpenBSD) та [DragonFlyBSD](http://uk.wikipedia.org/wiki/DragonFlyBSD).

На теперішній час Linux та представники сімейства BSD швидко відвойовують ринок у комерційних UNIX-систем та одночасно проникають як у персональні комп'ютери користувачів, так і на мобільні вмонтовані системи. Одним із свідчень даного успіху служить той факт, що коли фірма [Apple](http://uk.wikipedia.org/wiki/Apple) шукала основу для своєї операційної системи, вона вибрала [NEXTSTEP](http://uk.wikipedia.org/wiki/NEXTSTEP) — операційну систему з вільно розповсюджуваним ядром, розроблену фірмою [NeXT](http://uk.wikipedia.org/wiki/NeXT) та перейменованою у [Darwin](http://uk.wikipedia.org/wiki/Apple_Darwin) після придбання фірмою Apple. Ця система відноситься до сімейства BSD та базується на ядрі [Mach](http://uk.wikipedia.org/wiki/Mach). Застосування **Darwin BSD UNIX** у [Mac OS X](http://uk.wikipedia.org/wiki/Mac_OS_X) робить його однією з найбільш розповсюджених версій UNIX.

Більшість виробників [відкритих](http://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D0%BA%D1%80%D0%B8%D1%82%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F) *UNIX*-систем не домагаються сертифікації *UNIX* для свого продукту навіть в якості компромату: вартість сертифікації вважається неприпустимою. Для таких систем зазвичай використовують термін «Freenix». Прикладом є [*GNU*](http://uk.wikipedia.org/wiki/GNU), [*Linux*](http://uk.wikipedia.org/wiki/Linux), [*Minix*](http://uk.wikipedia.org/wiki/Minix), [*OpenSolaris*](http://uk.wikipedia.org/wiki/OpenSolaris), [*Plan 9*](http://uk.wikipedia.org/wiki/Plan_9) і [*BSD*](http://uk.wikipedia.org/wiki/BSD) зі своїми нащадками, такими як [*FreeBSD*](http://uk.wikipedia.org/wiki/FreeBSD), [*NetBSD*](http://uk.wikipedia.org/wiki/NetBSD) і [*OpenBSD*](http://uk.wikipedia.org/wiki/OpenBSD).

1.7.Свободное ПО

**Вільне програмне забезпечення** ([англ.](http://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *free software, software libre, чи libre software*) — [програмне забезпечення](http://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F), яке надає користувачу ряд свобод:

* запускати програму («свобода 0»);
* вивчати й змінювати її [початковий код](http://uk.wikipedia.org/wiki/%D0%94%D0%B6%D0%B5%D1%80%D0%B5%D0%BB%D1%8C%D0%BD%D0%B8%D0%B9_%D0%BA%D0%BE%D0%B4) відповідно до власних потреб («свобода 1»);
* вільно розповсюджувати копії програми («свобода 2»);
* розповсюджувати модифіковані версії програми («свобода 3»)[[1]](http://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F#cite_note-gnu-fs-1).

Якщо хоча б однієї із цих свобод немає, програма не належить до вільного програмного забезпечення. Таким чином, якщо програма надається безкоштовно, це ще не означає, що програма є вільним софтом: існує ряд безкоштовних програмних продуктів, джерельний код яких не публікується, або на які існують обмеження використання чи розповсюдження. Такі програми не є вільним програмним забезпеченням.

При тому вільне програмне забезпечення не обов'язково мусить бути безкоштовним: копії можна розповсюджувати і за гроші, але не можна при тому заборонити безкоштовно копіювати програму далі[[2]](http://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%B5_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F#cite_note-2) чи обмежувати користувача в будь-якій зі свобод, зазначених вище.

Оскільки кожен, хто має копію вільної програми, має право передавати її будь-кому безкоштовно, то найчастіше ВПЗ є безкоштовним. Бізнесові моделі ВПЗ базуються на додаткових послугах на кшталт технічної підтримки, навчанні, сертифікації чи інтеграції. Проте ВПЗ забороняє бізнесові моделі, засновані на абсолютній відсутності будь-яких прав у користувача і вимозі оплачувати право використання ПЗ.

Термін **вільне програмне забезпечення** ввів [Річард Столмен](http://uk.wikipedia.org/wiki/%D0%A1%D1%82%D0%BE%D0%BB%D0%BC%D0%B5%D0%BD_%D0%A0%D1%96%D1%87%D0%B0%D1%80%D0%B4), засновник проекту [GNU](http://uk.wikipedia.org/wiki/GNU)

1.8.Многообразие операционных систем

**Множество ОС:**

· Операционные системы мэйнфреймов. В основном ориентированы на обработку множества заданий, большинство из которых требуют огромного количества операций ввода/вывода. Они предлагают три вида обслуживания: пакетную обработку, обработку транзакции (групповые операции), разделение времени.

· Серверные ОС. Работают на серверах, которые представляют собой или очень большие персональные компьютеры или рабочие станции или возможно даже мэйнфреймы. Они одновременно обслуживают множество пользователей и позволяют им делить между собой аппаратное и программное обеспечение.

· Многопроцессорные ОС. Более частое использование увеличение мощности компьютера состоит в соединении нескольких процессоров в одной системе. В зависимости от вида соединений процессоров и разделения работы, такие системы называются параллельными компьютерами, мультикомпьютерами и многопроцессорными системами. Для них требуется специальный вид ОС, но чаще всего это серверные ОС со специальными возможностями связи.

· ОС для ПК. Работа данных ОС заключается в предоставлении удобного интерфейса для одного пользователя.

· ОС реального времени. Главным параметром таких систем является время. Системы делятся на жесткие и гибкие системы реального времени.

· Встроенные ОС. Встроенные системы управляют действиями устройств, работают на машинах, которые обычно не считаются компьютерами (холодильники, стиральные машины и т.д.).

· Операционные системы для смарт-карт. Самые маленькие операционные системы работают на смарт-картах и содержат центральный процессор. В данных ОС жесткие требования на размер памяти и ограничения по мощности процессора. Часто они являются патентованными системами, некоторые смарт-карты джава-ориентированные.

1.9.Роль и функции операционной системы

**Операционная система управляет следующими аппаратными ресурсами:**

· процессорами

· памятью

· устройствами ввода/вывода

**К базовым функциям ОС относят следующие:**

· управление процессорами и распределение процессорного времени

· управление и распределение памяти

· управление устройствами ввода/вывода.

**ОС обеспечивает следующие стандартные функции для прикладного ПО и его пользователей:**

· обеспечение разделения аппаратных ресурсов между задачами и пользователями

· обеспечение эффективного выполнения операций ввода/вывода

· обеспечение возможности работать с общими данными в режиме коллективного пользования

· планирование доступа пользователей к общим ресурсам

· обеспечение интерфейса пользователя

· обеспечение безопасности и защиты информации

1.10. Цели работы ОС

Основные **цели** работы операционной системы следующие.

1. **Обеспечение удобства, эффективности, надежности, безопасности выполнения пользовательских программ**. Для пользователя самое главное – чтобы его программа работала, вела себя предсказуемо, выдавала необходимые ему правильные результаты, не давала сбоев, не подвергалась внешним атакам. Вычислительную среду для такого выполнения программ и обеспечивает операционная система.
2. **Обеспечение удобства, эффективности, надежности, безопасности использования компьютера**. Операционная система обеспечивает максимальную полезность и эффективность использования компьютера и его ресурсов, обрабатывает прерывания, защищает компьютер от сбоев, отказов и хакерских атак. Эта деятельность ОС может быть не столь заметной для пользователя, но она осуществляется постоянно.
3. **Обеспечение удобства, эффективности, надежности, безопасности использования сетевых, дисковых и других внешних устройств, подключенных к компьютеру**. Особая функция операционной системы, без которой невозможно использовать компьютер, - это работа с внешними устройствами. Например, ОС обрабатывает любое обращение к жесткому диску, обеспечивая работу соответствующего **драйвера** (низкоуровневой программы для обмена информацией с диском) и **контроллера** (специализированного процессора, выполняющего команды ввода-вывода с диском). Любая "флэшка", вставленная в USB-слот компьютера, распознается операционной системой, получает свое логическое имя (в системе Windows – в виде буквы, например, G) и становится частью файловой системы компьютера на все время, пока она не будет извлечена (демонтирована).
4. Подчеркнем особую важность среди функций современных ОС обеспечения **безопасности, надежности и защиты данных**. Следует учитывать, что компьютер и операционная система работают в сетевом окружении, в котором постоянно возможны и фактически происходят атаки хакеров и их программ, ставящие своей целью нарушение работы компьютера, "взлом" конфиденциальных данных пользователя, хранящихся на нем, похищение логинов, паролей, использование компьютера как "робота" для рассылки реклам или вирусов и др. В связи с этим в 2002 г. фирма Microsoft объявила **инициативу по надежным и безопасным вычислениям (trustworthy computing initiative**), целью которой является повышение надежности и безопасности всего программного обеспечения, прежде всего – операционных систем. В данном курсе мы будем подробно останавливаться на том, какие действия по обеспечению надежности, безопасности и защите данных предпринимают современные ОС.

1.11. Определение ОС. Классификация ОС

**ОС** – системное программное обеспечение, обеспечивающее управление ресурсами компьютера и предоставляющее прикладному ПО стандартные функции, доступные через определенные интерфейс.

**Классификация ОС.**

**1)** По количеству одновременно выполняемых задач:

a. Однозадачные

b. Многозадачные

**2)** По количеству поддерживаемых процессоров:

a. Многопроцессорные – поддерживают многопроцессорные вычислительные системы.

b. Однопроцессорные

**3)** По количеству пользователей

a. Однопользовательские

b. Многопользовательские

**4)** По наличию встроенных средств сетевого обмена

a. Несетевые – не имеют встроенных средств поддержки вычислительных сетей

b. Сетевые

**5)** По поддержке построение систем реального времени

a. Реального времени (реактивные) – поддерживают решение задач управления объектами реального времени.

b. Слабореактивные – не предназначены для управления объектом реального времени.

**6)** По типу пользовательского интерфейса

a. С командным интерфейсом пользователя

b. С графическим интерфейсом пользователя

1.12. Требования к современным операционным системам

Главным требованием, предъявляемым ОС, является выполнение основных функций эффективного управления ресурсами и обеспечение удобного интерфейса для пользователя и прикладных программ. Также к ОС предъявляются эксплуатационные требования:

· **Расширяемость** – если код ОС написан таким образом, что дополнения и изменения могут вноситься без нарушения целостности системы, то такую ОС называют расширяемой. Расширяемость достигается за счет модульной структуры ОС, при которой программы могут строиться из отдельных модулей, взаимодействующих через функциональный интерфейс.

· **Переносимость** – в то время как аппаратная часть устаревает за несколько лет, код ОС должен, легко переносится с аппаратной платформы одного типа на аппаратную платформу другого типа, с процессора одного типа на процессор другого типа. Переносимая ОС имеет несколько вариантов реализации для разных платформ и такое свойств ОС называют наноплатформностью.

· **Совместимость** – для пользователей, переходящих по тем или иным причинам, с одной ОС на другую, необходима возможность запуска в новой ОС, привычного приложения. Если ОС имеет средства для выполнения прикладных программ, написанных для других ОС, то говорят, что она совместима, с данными ОС. Существует совместимость на уровне исходных текстов и двоичных кодов.

· **Надежность и отказоустойчивость** – **система** должна быть защищена от внутренних и от внешних сбоев. Ее действия должны быть всегда предсказуемыми, а предложения не должны навредить ОС. Надежность и отказоустойчивость ОС определяется архитектурным решением, а также отлаженностью и качеством реализаций кода.

· **Безопасность** – современная ОС должна защищать данные и другие ресурсы от несанкционированного доступа. Для этого, система должна обладать, как минимум, средствами аутентификации (определение легальности пользователя), авторизация (полномочия, для данного аккаунта), аудит (фиксация всех, подозрительных для безопасности системы, событий).

· **Производительность** – ОС должна обладать на столько, хорошим быстродействием и временем реакции, насколько позволяет аппаратная платформа. На быстродействие влияет много факторов, но один из главных – архитектурный фактор.

1.13. Классификации компьютерных систем

**Классификация ОС.**

**1)** По количеству одновременно выполняемых задач:

a. Однозадачные

b. Многозадачные

**2)** По количеству поддерживаемых процессоров:

a. Многопроцессорные – поддерживают многопроцессорные вычислительные системы.

b. Однопроцессорные

**3)** По количеству пользователей

a. Однопользовательские

b. Многопользовательские

**4)** По наличию встроенных средств сетевого обмена

a. Несетевые – не имеют встроенных средств поддержки вычислительных сетей

b. Сетевые

**5)** По поддержке построение систем реального времени

a. Реального времени (реактивные) – поддерживают решение задач управления объектами реального времени.

b. Слабореактивные – не предназначены для управления объектом реального времени.

**6)** По типу пользовательского интерфейса

a. С командным интерфейсом пользователя

b. С графическим интерфейсом пользователя

1.14. Функциональные компоненты ОС автономного компьютера

**Функциональные компоненты операционных систем автономного компьютера**

-программы управления вводом/выводом;

-программы, управляющие файловой системой и планирующие задания для компьютера;

-процессор командного языка, который принимает, анализирует и выполняет команды адресованные операционной системе.

Каждая операционная система имеет свой командный язык, который позволяет пользователю выполнять те или иные действия:

-обращаться к каталогу;

-выполнять разметку внешних носителей;

-запускать программы;

-... другие действия.

Анализ и исполнение команд пользователя, включая загрузку готовых программ из файлов в оперативную память и их запуск, осуществляет командный процессор операционной системы. Для управления внешними устройствами компьютера используются специальные системные программы - драйверы. Драйверы стандартных устройств образуют в совокупности базовую систему ввода - вывода (BIOS), которая обычно заносится в постоянное ЗУ компьютера

1.15. Архитектура ОС. Ядро и вспомогательные модули ОС

**Наиболее подходящим подходом структуризации ОС является разделение всех ее модулей на две группы:**

1) Ядро – модули, выполняющие основные функции ОС. Модули ядра выполняют такие базовые функции ОС, как управление памятью, процессорами и устройствами ввода/вывода. В состав ядра входят функции, решающие внутрисистемные задачи организации вычислительного процесса (ВП), а именно, переключение контекста, загрузка/выгрузка страниц, обработка прерываний и т.д. То есть это функции, которые не доступны для приложений. Функции, выполняющиеся ядром, наиболее часто используемые, поэтому скорость их выполнения влияет на производительность всей системы. Таким образом, они постоянно располагаются в ОЗУ и являются резидентными.

2) Модули, выполняющие вспомогательные функции ОС. Эти модули выполняют весьма полезные, но менее обязательные функции, то есть, это может быть архивация данных, дефрагментация данных и т.д. Вспомогательные модули обычно выполняются в виде приложений либо в виде библиотек процедур. Поскольку, вспомогательные модули ОС оформлены в виде приложений, бывает сложно провести границу между ОС и приложениями (пользовательскими).

**Вспомогательные модули ОС обычно подразделяются на следующие группы:**

· Утилиты – программы. Решающие отдельные задачи управления и сопровождения компьютерной системы, такие, например, как программы сжатия дисков, архивирования данных на магнитную ленту;

· Системные обрабатывающие программы – текстовые или графические редакторы, компиляторы, компоновщики, отладчики;

· Программы предоставления пользователю дополнительных услуг – специальный вариант пользовательского интерфейса, калькулятор и даже игры;

· Библиотеки процедур различного назначения, упрощающие разработку приложений, например, библиотека математических функций, функций ввода/вывода и т.д.

1.16. Архитектура операционной системы с ядром в привилегированном режиме.

Для надежного управления кодом выполнения приложения, ОС должна иметь по отношению к приложениям определенные привилегии. Не одно приложение не должно иметь возможности: получать без ведома ОС дополнительную область памяти, занимать процессор дольше разрешенного времени, непосредственно управлять совместно используемыми внешними устройствами. Обеспечить привилегии ОС невозможно без специальных средств аппаратной поддержки.

Аппаратура компьютера должна поддерживать как минимум два режима работы:

1) Пользовательский

2) Привилегированный (режим ядра или режим супервизора)

Так как ядро ОС работает в привилегированном режиме, то данное свойство иногда служит определением ядра.

Приложения ставятся в подчинение за счет запрета выполнения в пользовательском режиме некоторых критических команд: команды, которые связаны с переключение процессора с одной программы на другую, управление вводом/выводом, команды, которые занимаются распределением памяти.

Повышенная устойчивость ОС обеспечиваемая перехода ядра в привилегированный режим достигается за счет некоторого замедления выполнения системных вызовов.

1.17. Многослойная структура ОС. Структура ядра.

**Многослойная структура ОС**

Вычислительную систему, работающую под управление ОС, на основе ядра, можно рассматривать как систему, состоящую из трех иерархически расположенных слоев. Нижний слой составляет аппаратура, промежуточный ядро, а утилиты, системные программы и приложения составляют верхний слой. Слои в структуре ОС принято исполнять в виде концентрических колец, показывая то, что каждый слой может взаимодействовать только со смежными слоями. Каждый вышележащий слой, выполняя набор функций, которые образуют межслойный интерфейс, обслуживает слой, который находится над ним и используют возможности слоя, которые находятся под ним.

Данная концепция имеет много достоинств, она существенно упрощает разработку системы, так как позволяет при анализе «сверху вниз» определить функции слоев и межслойные интерфейсы, а затем постепенно наращивать функции слоев, двигаясь «снизу вверх». При модернизации системы можно изменить модули внутри слоя, без необходимости изменения модулей в других слоях и межслойного интерфейса.

**Ядро может состоять из следующих слоев:**

· Средства аппаратной поддержки ОС. ОС – это совокупность программных и аппаратных средств. К операционной системе относят не все аппаратные средства, а только средства аппаратной поддержки ОС, то есть те, которые прямо участвуют в организации вычислительного процесса (средства поддержки привилегированного режима, система прерывания, средства переключения контекста процессах, средства защиты областей памяти и т.д.).

· Машинно-зависимые компоненты ОС. Этот слой образуют программные модули, в которых отражается специфика аппаратной платформы компьютера. В идеале данный слой полностью экранирует вышележащие слои от особенностей аппаратуры. Это позволяет разрабатывать вышележащие слои, на основе машинно-независимых модулей, существующих в единственном экземпляре для всех типов платформ поддерживаемых данной ОС.

· Базовые механизмы ядра. Этот слой выполняет наиболее примитивные операции ядра, такие как, программные переключение процессов, диспетчеризация прерываний, перемещении страницы с памяти и обратно и т.д. Модули данного уровня не принимают решения, а только выполняют решения принятые выше. Иногда данный слой называют исполнительным механизмом модулей.

· Менеджеры ресурсов. Данный слой состоит из мощных функциональных модулей, которые реализуют стратегические задачи по управлению основными ресурсами ВС (вычислительной системы). Обычно в данном слое работают менеджеры ресурсов процессора, ввода/вывода и оперативной памяти. Внутри слоя менеджера существуют тесные взаимосвязи, отображающие тот факт, что для выполнения процесса необходим доступ одновременно к нескольким ресурсам.

· Интерфейс системных вызовов. Этот слой взаимодействует непосредственно с приложениями и системными утилитами, обрабатывающими программами и т.д. образуя прикладной программный интерфейс ОС.

Выбор количества слоев диаграмм является ответственным и сложным делом: увеличение числа слоев ведет к некоторому замедлению ядра, за счет дополнительных прикладных расходов на межслойное взаимодействие, а уменьшение числа слоев ведет к ухудшению расширяемости и логичности системы.

Обычно ОС прошедшие долгий путь развития (UNIX, напримерJ) имеют неупорядоченное ядро с небольшим числом четко выделенных слоев. У относительно молодых ОС ядро разделено на большое кол-во слоев и взаимодействие между ними форматизировано в гораздо высшей степени.

1.18. Средства аппаратной поддержки ОС

**Типовые средства аппаратной поддержки ОС**

· Средства поддержки привилегированного режима . Решения, какие функции будут выполнятся аппаратно, а какие программно принимает разработчик.

Тем не менее практически все современные ОС имеют типичный набор аппаратных средств поддержки ОС. Средства поддержки привилегированного режима обычно основан на системном регистре процессора. Данные регистр содержит некоторые признаки, определяющие режим работы процессора. Смена режима привилегий выполняется в результате прерывания или выполнения привилегированной команды.

· Средства трансляции адресов выполняет преобразование виртуальных адресов, которые содержатся в кодах процесса, в физические адреса. Таблицы, предназначены для трансляции адресов, имеют большой объем, поэтому для их хранения используются области оперативной памяти, а аппаратура процессора содержит указатели на эти области. Средства трансляции адресов используют данные указатели для доступа к элементам таблицы.

· Средства переключения процессов предназначены для быстрого сохранения контекста приостанавливаемого процесса и соответственно восстановление контекста процесса, которой становиться… содержимое контекста обычно включает содержимое всех регистров общего назначения, регистры флагов операция, системные регистры и указатели, указатели в таблицу адресов памяти и т.д.

· Система прерываний. Позволяет реагировать на внешние события, синхронизировать выполнение процессов и работу устройств ввода/вывода, быстро переходить с одной программы на другую. Механизм прерывания нужен, для того чтобы оповестить систему о возникновении вычислительной системы непредсказуемого события или события которое не синхронизировано циклом работы процессора.

· Системный таймер. Часто реализуется в виде быстродействующего регистра счетчика, необходим ОС для выдержки интервала времени.

· Средства защиты областей памяти. Обеспечивают на аппаратном уровне проверку возможности программного кода осуществлять данными определенной области такие операции как чтение, запись или выполнение. Если аппаратура компьютера поддерживает механизм трансляции адресов, то данное средство встраивается в этот механизм. Функции аппаратуры по защите памяти обычно состоят в сравнении привилегий текущего кода процессора и сегмента памяти, к которому обращается.

1.19. Переносимость операционной системы. Требования

**Переносимость ОС**

Если код ОС может быть сравнительно легко перенесен с аппаратной платформы одного типа на другую, такая ОС переносимая.

Понятие переносимости не бинарное состояние, а состояние степени. Т.е. на сколько легко она переносится.

Для того чтобы ОС обеспечить свойство мобильности разработчики должны следовать следующим правилам:

1. Большая часть кода должна быть написана на языке, трансляторы которого имеются на всех машинах, куда предполагается переносить системы (такими языками являются языки высокого уровня)

2. Объем машинно-зависимых частей кода, который непосредственно взаимодействует с аппаратными средствами, должен быть по возможности минимизирован. Например, необходимо избегать манипулирования регистрами, исключить возможность использовать по умолчанию стандартных конфигураций аппаратуры или их характеристик.

3. Аппаратно-зависимый код должен быть надежно изолирован в нескольких модулях, а не должен быть распределен по всей системе.

1.20. Микроядерная архитектура. Преимущества и недостатки микроядерной архитектуры

Таким образом, главная функция микроядра, является наличие в ОС удобного и эффективного способа вызова процедур одного процесса и другого.

Микроядерная архитектура является альтернативой классическому способу построения ОС. Суть микроядерной архитектуры состоит в следующем: в привилегированном режиме остается работать только небольшая часть ОС, которая называется, микроядро.

Микроядро защищено от остальных частей ОС и приложений. В состав микроядра обычно входят машинно-зависимые модули, а также модули, выполняющие базовые (но не все) функции ядра, по управление процессоров, обработки прерываний, виртуальной памятью, пересылке сообщений, управления устройствами ввода/вывода.

**Преимущества и недостатки микроядерной архитектуры**

ОС основаны на концепции микроядра высокой степени удовлетворяют большинству требований, предъявляемых современным ОС. Обладают переносимостью, расширяемостью, надежностью, создают хорошие условия для распределенных приложений. За эти преимущества приходится платить снижением производительности, что является главным и весомым недостатком микроядерной архитектуры.

1. Высокая степень переносимости обусловлена тем, что весь машинно-зависимый код изолирован в микроядре. Поэтому для переноса системы на новый процессор требуется меньше изменений, и все они логически группированы.

2. Расширяемость присуща микроядерной архитектуре в очень высокой степени. Традиционной системе даже при наличии строго иерархии слоев достаточно сложно удалить один слой и поменять его на другой. Микроядерная архитектура позволяет не только добавлять, но и сокращать число компонентов.

3. Использование микроядерной архитектуры повышает надежность ОС. Каждый сервер выполняется в виде отдельного процесса, в своей собственной области памяти и тем самым защищен от других серверов или приложений. Размеры микроядра очень уменьшены в объеме, что снижает вероятность появления ошибок при программировании.

4. Модель с микроядром хорошо подходит для поддержки распределенных вычислений, так как использует механизмы аналогичные сетевым: взаимодействие путем обмена сообщениями.

1.21. Совместимость ОС.

Если пользователь, имеет возможность выполнять в приложения, написанные для других ОС, в своей ОС, то такое свойство ОС называется совместимостью. Необходимо различать совместимость на уровне двоичных кодов и на уровне исходных текстов.

Приложения обычно хранятся в ОС в виде файлов содержащих двоичные образы кода. Двоичная совместимость достигается в том случае, когда можно взять исполняемую программу и запустить ее в среде другой ОС.

Совместимость на уровне исходных текстов требует наличия соответствующего компилятора в составе программного обеспечения компьютера, на котором предполагается выполнять данное приложение. Совместимость на уровне библиотек и системных выводов. (двоичная для пользователей, исходные тексты для разработчиков принципиальны)

Самый главный фактор, влияющий на совместимость ОС – архитектура процессора на котором работает ОС. Если процессор использует тот же набор команд, тот же диапазон адресов, то двоичная совместимость может быть достигнута довольно просто. Для этого необходимо:

1. Вызов функции интерфейса прикладного программирования, которые содержат приложения, должны поддерживаться данной ОС.

2. Внутренняя структура, исполняемая файлом должна соответствовать структуре исполняемых файлов данной ОС.

1.22. Классификация компьютерных архитектур.

### Методы классификации компьютеров

Номенклатура видов компьютеров сегодня огромная: машины различаются по назначению, мощности, размерам, элементной базе и т.д. Поэтому классифицируют ЭВМ по разным признакам. Следует заметить, что любая классификация является в некоторой мере условной, поскольку развитие компьютерной науки и техники настолько бурное, что, например, сегодняшняя микроЭВМ не уступает по мощности миниЭВМ пятилетней давности и даже суперкомпьютерам недавнего прошлого. Кроме того, зачисление компьютеров к определенному классу довольно условно через нечеткость разделения групп, так и вследствии внедрения в практику заказной сборки компьютеров, где номенклатуру узлов и конкретные модели адаптируют к требованиям заказчика. Рассмотрим распространенные критерии классификации компьютеров.

### Классификация по назначению

* большие электронно-вычислительные машины (ЭВМ);

Применяют для обслуживания крупных областей народного хозяйства. Они характеризуются 64-разрядными параллельно работающими процессорами (количество которых достигает до 100), интегральным быстродействием до десятков миллиардов операций в секунду, многопользовательским режимом работы. Доминирующее положение в выпуске компьютеров такого класса занимает фирма IBM (США). Наиболее известными моделями суперЭВМ являются: IBM 360, IBM 370, IBM ES/9000, Cray 3, Cray 4, VAX-100, Hitachi, Fujitsu VP2000.

* миниЭВМ;

Похожа на большие ЭВМ, но меньших размеров. Используют на крупных предприятиях, научных учреждениях и организациях. Часто используют для управления производственными процессами. Характеризуются мультипроцессорной архитектурой, подключением до 200 терминалов, дисковыми запоминающими устройствами, которые наращиваются до сотен гигабайт, разветвленной периферией. Для организации работы с миниЭВМ, нужен вычислительный центр, но меньший чем для больших ЭВМ.

* микроЭВМ;

Доступны многим учреждениям. Для обслуживания достаточно вычислительной лаборатории в составе нескольких человек, с наличием прикладных программистов. Необходимые системные программы покупаются вместе с микроЭВМ, разработку прикладных программ заказывают в больших вычислительных центрах или специализированных организациях.

Программисты вычислительной лаборатории занимаются внедрением приобретенного или заказанного программного обеспечения, выполняют его настройку и согласовывают его работу с другими программами и устройствами компьютера. Могут вносить изменения в отдельные фрагменты программного и системного обеспечения.

* персональные компьютеры.

Бурное развитие приобрели в последние 20 лет. Персональный компьютер (ПК) предназначен для обслуживания одного рабочего места и способен удовлетворить потребности малых предприятий и отдельных лиц. С появлением Интернета популярность ПК значительно возросла, поскольку с помощью персонального компьютера можно пользоваться научной, справочной, учебной и развлекательной информацией.

Персональные компьютеры условно можно поделить на профессиональные и бытовые, но в связи с удешевлением аппаратного обеспечения, грань между ними размывается.

### Классификация по уровню специализации

* универсальные;
* специализированные.

На базе универсальных ПК можно создать любую конфигурацию для работы с графикой, текстом, музыкой, видео и т.п.. Специализированные ПК созданы для решения конкретных задач, в частности, бортовые компьютеры в самолетах и автомобилях. Специализированные миниЭВМ для работы с графикой (кино- видеофильмы, реклама) называются графическими станциями. Специализированные компьютеры, объединяющие компьютеры в единую сеть, называются файловыми серверами. Компьютеры, обеспечивающие передачу информации через Интернет, называются сетевыми серверами.

### Классификация по размеру

* настольные (desktop);
* портативные (notebook);
* карманные (palmtop).

Наиболее распространенными являются настольные ПК, которые позволяют легко изменять конфигурацию. Портативные удобны для пользования, имеют средства компьютерной связи. Карманные модели можно назвать "интеллектуальными" записными книжками, разрешают хранить оперативные данные и получать к ним быстрый доступ.

### Классификация по совместимости

Существует великое множество типов компьютеров, которые собираются из деталей, изготовленных разными производителями. Важным является совместимость обеспечения компьютера:

* аппаратная совместимость (платформа IBM PC и Apple Macintosh)
* совместимость на уровне операционной системы;
* программная совместимость;
* совместимость на уровне данных.

1.23. Мультипрограммирование. Критерия эффективности ОС

Это способ организации вычислительного процесса, при котором на одном процессоре попеременно выполняется сразу несколько программ. Эти программы совместно используют не только процессор, но и другие ресурсы компьютера. Цель мультипрограммирования – повысить эффективность использования ОС.

Критерии эффективности ВС

Пропускная способность – кол-во задач выполняемых ВС в единицу времени

Удобство работы пользователя – заключается в том, что пользователи имеют возможность интерактивно работать с несколькими приложениями.

Реактивность системы – способность выдерживать заранее заданные интервалы времени между запуском программы и получением результата.

В зависимости от выбранного критерия ОС делятся на

-Системы пакетной обработки(многозадачность, большее количества решенных задач) – пропускная способность.

-Системы разделения времени

-Системы реального времени

1.24. Мультипрограммирование в системах пакетной обработки

Мультипрограммирование в системах пакетной обработки – при использовании мультипрограммирования главной целью являются минимизация простоя всех устройств компьютера, прежде всего ЦП. Системы пакетной обработки предназначались для решения задач в основном вычислительного характера, главное цель – решение максимального числа задач в единицу времени.

Для достижения этой цели в системах пакетной обработки используется следующая схема функционирования:

1) Вначале работы формируется пакет заданий каждое задание содержит требования к системным ресурсам

2) Из этого пакета формируется мультипрограммная смесь, то есть множество одновременно выполняемых заданий. Для одновременного выполнения выбираются задания предъявляющие разные требования к ресурсам что бы обеспечивалась сбалансированная загрузка всех устройств.

В системах пакетной обработки невозможно гарантировать выполнение определенной задачи в течении конкретного периода времени.

1.25. Мультипрограммирование в системах разделения времени

Повышение эффективности работы пользователя является целью система разделения времени. В системах разделения времени предоставляется возможность работы пользователя сразу с несколькими приложениями, для этого каждое приложение должно иметь возможность регулярного общения с пользователем.

В системах разделения времени данная проблема решается за счет того что ОС периодически принудительно останавливает приложения не дожидаясь пока они добровольно не завершат процесс. Всем приложениям попеременно выделяется квант процессорного времени, таким образом реализуется возможность диалога с программами.

Производительность системы снижается за счет возросших накладных расходов связанных с переключением с одной задачи на другую.

Вместе с тем мультипрограммное выполнение интерактивных приложений повышает пропускную способность компьютера. Аппаратура загружается больше, поскольку пока одно приложение ждет пользователя, другое приложение может обрабатывается процессором.

1.26. Мультипрограммирование в системах реального времени

Мультипрограммирование в системах реально времени предназначено для управления от компьютера различными техническими объектами (станки, спутники, научные установки) или технологическими процессами. (гальваническая линия, доменный процесс). Во всех этих случаях существует предельно допустимое время в течении которого должна быть выполнена та или иная программа. Таким образом в критерии эффективности является способность выдерживать заранее заданный интервал времени между запуском и получением результат. Данное время является временем реакции системы, данное свойство называется реактивностью.

В системах реального времени мультипрограммная смесь представляет собой фиксированный набор заранее разработанных программ, а вывод на программное выполнения осуществляется по прерываниям.

Способность аппаратуры компьютера и ОС для быстрого ответа зависит в основном от скорости переключения между задачами. Если во время прерывания процессор должен опросить сотни потенциальных источников прерывания, то реакция будет очень плачевна. Плак-плак

Время обработки прерывания в системах реального времени является главным требованием к процессору даже при небольшой его загрузке.

В системах реально времени не стремятся максимально загружать все устройства, а наоборот при проектировании данной системы обычно закладывают некоторый запас на случай пиковой нагрузки.

1.27. Мультипроцессорная обработка

Мультипроцессирование или мультипроцессорная обработка это способ организации вычислительного процесса в системах с некоторыми процессами при которых несколько задач могут выполнятся одновременно(АДНАВРЕМЕННА) на разных процессорах.

Мультипроцессорные системы бывают симметричные и несимметричные. При этом следует только определять к какому аспекту ВС(вычислительная система)относится эта характеристика. Либо к типу архитектуры либо к способу организации ВП(вычислительного процесса)

Симметричная архитектура мультипроцессорной системы предполагает единство или однородность всех процессоров и единообразие включения процессоров в общую схему мультипроцессорной системы. Традиционно симметричная мультипроцессорная конфигурация разделяет одну память между всеми процессами. Масштабируемость – возможность наращивания числа процессоров. Масштабируемость в симметричных системах ограничена вследствие того что они ограничены памятью. Такая конструкция называется масштабируемой по вертикали. В данной схеме используется одна память.

В следствии этого могут очень быстро обмениваться данными так что обеспечивается высокая производительность для тех приложений которые взаимодействуют между собой.

В асимметрической архитектуре разные процессоры могут отличатся своими характеристиками(производительность, надежность, система команд модель микропроцессора и т.д.) так и функциональной ролью. Функциональная неоднородность в ассиметричной архитектуре влечет за собой структурные отличия в фрагментах системы содержащих разные процессоры системы.

Масштабируемость реализуется иначе, так как нету жесткого требования к единому корпусу. Система может состоять из нескольких устройств, каждое из которых состоит из некоторых процессоров. Наз. масштабируемостью по горизонтали.

Ассиметричное мультипроцессирование является наиболее простым способ организации вычислительного процесса. Данный способ часто называют «ведущий-ведомый». Один процессор выделяет в качестве ведущего, на котором работает ОС, который управляет всеми ведомыми процессорами. Ведущий процессор берет на себя функцию распределения задач и ресурсов, а ведомый работает только как обрабатывающее устройство.

ОС полностью децентрализирована, все процессоры работают с одной и той же динамически выравниваемой нагрузкой.

1.28. Основные функции подсистемы управления процессами

Запущенная на выполнение программа порождает в системе один или более процессов (или задач). Подсистема управления процессами контролирует:

I Создание и удаление процессов

П Распределение системных ресурсов (памяти, вычислительных ресурсов) между процессами

III Синхронизацию процессов

IV Межпроцессное взаимодействие

Очевидно, что в общем случае число активных процессов превышает число процессоров компьютера, но в каждый конкретный момент времени на каждом процессоре может выполняться только один процесс. Операционная система управляет доступом процессов к вычислительным ресурсам, создавая ощущение одновременного выполнения нескольких задач.

Специальная задача ядра, называемая *распорядителем* или *планировщиком процессов* (scheduler), разрешает конфликты между процессами в конкуренции за системные ресурсы (процессор, память, устройства ввода/вывода). Планировщик запускает процесс на выполнение, следя за тем, чтобы процесс монопольно не захватил разделяемые системные ресурсы. Процесс освобождает процессор, ожидая длительной операции ввода/вывода, или по прошествии кванта времени. В этом случае планировщик выбирает следующий процесс с наивысшим приоритетом и запускает его на выполнение.

Модуль *межпроцессного взаимодействия* отвечает за уведомление процессов о событиях с помощью сигналов и обеспечивает возможность передачи данных между различными процессами

1.29. Понятия «процесс» и «поток».

В настоящие время используются две единицы работ, более крупная единица называется процесс или задача в некоторых ОС, которая требует для своего выполнения нескольких более мелких работ, для обозначения которых используют обозначение поток или нить.

Любая вычислительная система заключается в работе некоторой программы, поэтому из процессов и из потоков связывают программный код, который оформляется в виде исполняемого модуля. Чтобы данный программный код можно было выполнить, его необходимо загрузить в оперативную память, возможно, выделить некоторое место на диске, для хранения данных, предоставить доступ к устройствам ввода/вывода. Невозможно выполнить программу без предоставления процессорного времени.

В ОС, где существуют процессы и потоки, процесс рассматривается, как заявка на потребление всех видов ресурсов, кроме процессорного времени. Процессорное время распределяется между потоками.

В ОС, на ряду, с процессами используется механизм распараллеливания вычислений, который учитывает более тесные связи, между отдельными ветвями одного процесса. Для того, чтобы процессы не могли вмешиваться в распределение ресурсов, а также не могли повредить коды и данные друг друга, важнейшей функцией является изоляция процессов друг от друга. Для этого ОС обеспечивает каждый процесс отдельным адресным пространством, так что не один процесс не может получить доступ к командам другого процесса.

В отличие от процессов, создание потоков требует меньше накладных расходов, так как они используют адресное пространство и набор ресурсов, которые выделены их процессу. Мультипрограммирование на уровне потоков более эффективно, чем на уровне процессов.

Использование потока связано с повышением производительности, за счет параллельных вычислений, которые принадлежат одному процессу и могут выполняться на различных процессорах, а также с целью создание более читабельных и логичных программ.

1.30. Создание и завершение процессов и потоков.

**Создать процесс** – это, прежде всего, создать описателя процесса, в качестве которого выступает одна или несколько информационных структур, содержащих все сведения о процессе, необходимые ОС, для управления им (идентификатор процесса, данные о расположении в памяти исполняемого модуля, степень привилегированности процесса и т.д.). Примерами описателей процесса является управляющий блок процесса, дескриптор процесса, объект или другое (в разных системах по-разному).

Создание описателя процесса, говорит о появления в системе еще одного претендента на вычислительные ресурсы и с данного момента ОС должна принимать во внимание потребности нового процесса.

Создание процесса включает в себя загрузку кода и данных исполняемых программой с диска в оперативную память. Для этого, ОС должна обнаружить место данной программы на диске, затем перераспределить оперативную память, выделить в ней место, затем считать программу в выделенные участки оперативной памяти.

В многопоточной системе при создании процессов сразу создается как минимум 1 поток. При создании потока, ОС генерирует специальную информационную структуру – описатель потока, который содержит идентификатор потока, данные прав доступа, приоритетов и другую информацию о потоке. Во многих ОС поток может обратиться с запросом на создание потоков потомков. В зависимости от ОС в некоторых работа родителей и потомков синхронизируется, а в некоторых нет. Во многих ОС порождение потомков является основным механизмом

Работа по определению того, в какой момент необходимо прервать работу текущего активного потока и какому потоку предоставить возможность выполняться, называется **планированием**.

При планировании потока принимается во внимание приоритеты потока, время ожидания их в очереди, накопленной время выполнения, интенсивность обращения к вводу/выводу.

1.31. Процессы в ОС UNIX

Процесс в ОС UNIX - это программа, выполняемая в собственном виртуальном адресном пространстве. Когда пользователь входит в систему, автоматически создается процесс, в котором выполняется программа командного интерпретатора. Если командному интерпретатору встречается команда, соответствующая выполняемому файлу, то он создает новый процесс и запускает в нем соответствующую программу, начиная с функции main. Эта запущенная программа, в свою очередь, может создать процесс и запустить в нем другую программу (она тоже должна содержать функцию main) и т.д.

Схемы взаимодействия между процессами соответствуют механизму сопрограмм. Использование сопрограмм упрощает логику ядра системы и требует одного выделенного процесса, который создается нестандартным образом. С него начинается работа системы после запуска. В ОС UNIX этот процесс называется "диспетчерским" ("swapper"), он не имеет пользовательской фазы.

Все процессы в ОС UNIX, кроме диспетчерского, создаются операцией "Порождение". В этой операции участвуют два процесса: порождающий и порожденный. Порождающий выполняет системный вызов (fork), в результате появляется порожденный процесс.

Адресное пространство процесса состоит из 3 сегментов:

* текстового сегмента (инструкции);
* cегмент данных;
* сегмент стека.

1.32. Состояния потока

**Состояния потока:**

· Выполнение – активное состояние потока, во время которого поток обладает всеми необходимыми ресурсами и непосредственно выполняется процессором.

· Ожидание – пассивное состояние потока, находясь в котором, поток заблокирован по своим внутренним причинам (ждет осуществления некоторого события, например, завершения операции ввода/вывода, получения сообщения от другого потока или освобождения какого-либо необходимого ему ресурса).

· Готовность – также пассивное состояние потока, но в этом случае поток заблокирован в связи с внешним по отношению к нему обстоятельством (имеет все требуемые для него ресурсы, готов выполняться, однако процессор занят выполнением другого потока).

1.33. Планирование и диспетчеризация потоков

**Планирование потоков включает в себя решение двух задач:**

1. Определение момента времени для смены текущего активного потока

2. Выбор для выполнения потока из очереди готовых потоков.

Диспетчеризация заключается в реализации, найденного в результате планирования решения, то есть, переключение процессора с одного потока на другой.

**Диспетчеризация потока включает в себя:**

1. Сохранение контекста текущего потока, который требуется сменить

2. Загрузка контекста нового потока, выбранного в результате планирования

3. Запуск нового потока на выполнение.

1.34. Вытесняющие и невытесняющие алгоритмы планирования

Не вытесняющие алгоритмы основаны на том, что активному потоку, позволяется выполняться, пока он сам, по собственной инициативе, не отдаст управление ОС, для того чтобы та выбрала из очереди готовых потоков другой поток на выполнение.

Вытесняющие алгоритмы – это такие способы планирования потоков, в которых решения, о переключении процессора с выполнения одного потока, на выполнение другого принимается ОС.

Основным различием между вытесняющими и не вытесняющими алгоритмами является степень централизации механизма планирования потоков. При вытесняющем мультипрограммировании, функции планирования потоком, целиком сосредоточены в ОС. При этом программиста, когда пишет приложения, не заботится о том, что оно будет выполняться одновременно с другими приложениями. При этом, ОС выполняет следующие функции:

1. Определяет момент снятия активной задачи с выполнения

2. Запоминает его контекст

3. Выбирает из очереди готовых потоков, следующий на выполнение

4. Загружает новый поток на выполнение, загружая его контекст

При не вытесняющем мультипрограммировании механизм планирования распределен между ОС и прикладными приложениями. Прикладная программа, которая получила управление от ОС, сама определяет момент завершения очередного цикла своего выполнения, только после этого передает управление ОС. ОС формирует очереди потока и выбирает, согласно правилам, следующий поток на выполнение.

1.35. Алгоритмы планирования, основанные на квантовании

В соответствии с данной концепцией ,каждому потоку, поочередно, для выполнения предоставляется ограниченный, непрерывный период процессорного времени – **квант**.

Смена активного процесса, при данной схеме, происходит если:

1. Поток завершился и покинул систему

2. Произошла ошибка

3. Поток перешел в состояние ожидания

4. Исчерпан квант процессорного времени, отведенного данному потоку (поток переходит в состояние готовности).

Кванты, выделяемые потоком, могут быть одинаковыми для всех потоков и различными. Рассмотрим, например, случай, когда всем потокам предоставляется квант одинаковой длинны q. Если в системе содержится n потоков, то время, которое поток проводит в ожидании следующего кванта, будет равняться q(n-1). Чем больше потоков в системе, тем больше времени проводят потоки в ожидании, однако если величину кванта q выбрать достаточно малой, то у пользователя будет создаваться ощущение интерактивной работы со всеми приложениями.

Если квант короткий, то суммарное время, которое проводит поток в ожидании процессора, прямо пропорционально времени, требуемого для его выполнения.

Чем больше квант, тем больше вероятность, что потоки завершаться в результате первого цикла выполнения и тем менее явным становиться зависимость времени ожидания, от времени их выполнения. При достаточно большом кванте, данный алгоритм выражается в алгоритм последовательной обработки.

Кванты, выделяемые одному потоку, могут быть как фиксированной величины, а могут изменяться в течении жизни потока.

1.36. Алгоритмы планирования, основанные на приоритетах

Приоритетное планирование предполагает наличие у потоков некоторых изначально известной характеристики – приоритета, на основании которого определяется порядок их выполнения. Приоритет это – число, характеризующее степень привилегированности потока, при использовании ресурсов вычислительной машины, а именно: чем выше приоритет, тем меньше времени в очередях будет проводить поток. Приоритет может выражаться целым или дробным, положительным или отрицательным числом. В различных ОС разные схемы приоритета, в некоторых, чем больше число, тем больше приоритет, в некоторых наоборот.

В большинстве ОС, поддерживающих приоритеты, приоритет потока непосредственно связан с приоритетом процесса. Приоритет процесса назначается ОС при его создании.

Во многих ОС, предусматривается возможность изменения приоритета в течении жизни потока. Изменение приоритета могут происходить по инициативе самого потока (когда он обращается к соответствующим вызовам ОС) или по инициативе пользователя (когда он выполняет соответствующую команду). Кроме того, ОС сама может изменять приоритеты потоков, зависимо от ситуации. Данные типы приоритетов называются динамическими в отличии от фиксированных, которые не меняются в течении жизни.

Существуют две разновидности приоритетного клонирования:

1. Обслуживание с относительными приоритетами

2. Обслуживание с абсолютными приоритетами.

В обоих случаях, процесс выбора потока из очереди готовых, осуществляется одинаково: выбирается поток или процесс с наибольшим приоритетом. Определение момента смены активного потока определяется по-разному.

В системах с относительными приоритетами, активный поток выполняется до тех пор, пока сам не покинет систему.

В системе с абсолютными приоритетами, выполнение активного потока прерывается, кроме указанных причин, еще при одном условии, а именно, когда в очереди готовых потоков появляться поток, приоритет которого выше приоритета активного, тогда текущий поток переходит в состояние готовности.

1.37. Планирование в системах реального времени

В системах реального времени главным критерием эффективности является обеспечение временных характеристик вычислительного процесса. При разработке алгоритмов планирования для систем реального времени необходимо условие, какие последствия возникают в этих системах при несоответствии времени. Если последствия катастрофичны, то данная система реального времени называется **жесткой**, в противном случае системы реального времени называется **мягкой** или **гибкой**.

В жестких системах реального времени, время завершения выполнения каждой из критических задач должно быть гарантировано, для любого набора заданий. Такие гарантии могут быть даны либо результатом исчерпывающего тестирования, либо в результате построения статического расписания, либо в результате построения математически обоснованного математического расписания.

В качестве простейшего критерия, при котором заданные временные характеристики удовлетворяют расписание, может служить следующее условие: разность между предельным сроком выполнения задачи и временем на ее вычисления должно быть положительным.

В зависимости от характера выполнения запросов задачи делятся на два типа: **периодические** и **спорадические**.

Практическое планирования алгоритмы зависимых задач требует больших вычислительных ресурсов, и решения этой проблемы возможно за счет выполнения следующих мер: разведение проблемы планирования **на две части** таким образом, чтобы одна более сложная часть выполнялась перед запуском системы, вторая более простая во время работы системы; введения ограничивающих предположений о поведении набора задач.

Суть алгоритма состоит в том, что всем задачам назначают статические приоритеты, в соответствии с величиной их периода выполнения. Затем самым хорошим задачам наивысший приоритет, а задачи с наибольшим кол-вом периодом – наименьший.

Существуют алгоритмы с динамическим изменением приоритета, которые назначаются в соответствии с текущими параметрами задачи. При необходимости назначения некоторой задачи, на выполнение, выбирается та, у которой разница между конечным сроком выполнения и временем необходимым для выполнения является наименьшим.

1.38. Моменты перепланировки

Моменты перепланировки

Для реализации алгоритма планирования ОС должна получать управление всякий раз, когда в системе происходит событие, требующее перераспределения процессорного времени. К таким событиям могут быть отнесены следующие:

- Прерывание от таймера, сигнализирующее, что время, отведенное активной задаче на выполнение, закончилось. Планировщик переводит задачу в состояние готовности и выполняет перепланирование.

Активная задача выполнила системный вызов, связанный с запросом на ввод-вывод или на доступ к ресурсу, который в настоящий момент занят (например, файл данных). Планировщик переводит задачу в состояние ожидания и выполняет перепланирование.

- Активная задача выполнила системный вызов, связанный с освобождением ресурса. Планировщик проверяет, не ожидает ли этот ресурс какая-либо задача. Если да, то эта задача переводится из состояния ожидания в состояние готовности. При этом, возможно, что задача, которая получила ресурс, имеет более высокий приоритет, чем текущая активная задача. После перепланирования более приоритетная задача получает доступ к процессору, вытесняя текущую задачу.

- Внешнее (аппаратное) прерывание1, которое сигнализирует о завершении периферийным устройством операции ввода-вывода, переводит соответствующую задачу в очередь готовых, и выполняется планирование.

- Внутреннее прерывание сигнализирует об ошибке, которая произошла в результате выполнения активной задачи. Планировщик снимает задачу и выполняет перепланирование.

1.39. Назначение и типы прерываний

Периодические [**прерывания**](http://make-smart-room.com/electronics/2013/11/19/struktura-preryvaniy-mk51.html) от таймера вызывают **смену** [**процессов**](http://make-smart-room.com/unix/2013/03/02/processy-v-linux-chast-2.html) в мультипрограммной ОС, а прерывания от устройств ввода-вывода управляют потоками данных, которыми вычислительная система обменивается с внешним миром.

Система прерываний переводит процессор на выполнние потока команд, отличного от того, который выполнялся до сих пор, с последующим возвратом к исходному коду.

Механизм прерываний очень похож на механизм выполнения процедур. Переключение по прерыванию отличается от переключения, которое происходит по команде безусловного или условного перехода, предусмотренной программистом в потоке команд приложения. Переход по команде происходит в заранее определенных программистом точках программы в зависимости от исходных данных, обрабатываемых программой. **Прерывание же происходит в произвольной точке потока команд программы**, которую программист не может прогнозировать. Прерывание возникает либо в зависимости от внешних по отношению к процессу выполнения событий, либо при появлении непридвиденных аварийных ситуаций в процессе выполнения данной программы. **Сходство же прерываний с процедурами** состоит в том, что в обоих случаях **выполняется некоторая подпрограмма, обрабатывающая специальную ситуацию, а затем продолжается выполнение основной ветви программы**.

Прерывания:

— **внутренние**

называемые также исключениями, происходят синхронно выполнению программы при появлении аварийной ситуации в ходе исполнения некоторой инструкции программы, например, **ошибка деления на нуль**, **ошибки защиты памяти**, **обращение по несуществующему адресу** и т.п.

— **внешние**

могут возникать в результате действий пользователя или оператора за терминалом, или же в результате поступления сигналов от аппаратных устройств — [сигналов](http://make-smart-room.com/unix/2013/05/16/sistemnyy-vyzov-signal-ustanovka-sobstvennogo-obrabotchika-signala.html) завершения операций ввода-вывода, вырабатываемых контроллерами внешних устройств компьютера, такими как принтер или накопитель на жестких дисках, или же сигналов от датчиков управляемых компьютером технических объектов.

— **программные**

отличаются от предыдущих двух классов тем, что они по своей сути не являются истинными прерываниями, Программное прерывание возникает при выполнении особой команды процессора, выполнение которой **имитирует прерывание**, то есть переход на новую последовательность инструкций.

В зависимости от источника возникновения сигнала прерывания делятся на:

* асинхронные, или внешние (аппаратные) — события, которые исходят от внешних источников (например, периферийных устройств) и могут произойти в любой произвольный момент: сигнал от таймера, сетевой карты или дискового накопителя, нажатие клавиш клавиатуры, движение мыши. Факт возникновения в системе такого прерывания трактуется как *запрос на прерывание* ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Interrupt request, IRQ*);
* синхронные, или внутренние — события в самом процессоре как результат нарушения каких-то условий при исполнении [машинного кода](http://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4): деление на ноль или [переполнение стека](http://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D0%B5_%D1%81%D1%82%D0%B5%D0%BA%D0%B0), обращение к недопустимым адресам памяти или недопустимый код операции;
* программные (частный случай внутреннего прерывания) — инициируются исполнением специальной [инструкции](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%B4_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%B8) в коде [программы](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0). Программные прерывания как правило используются для обращения к функциям встроенного программного обеспечения ([firmware](http://ru.wikipedia.org/wiki/Firmware)), [драйверов](http://ru.wikipedia.org/wiki/%D0%94%D1%80%D0%B0%D0%B9%D0%B2%D0%B5%D1%80) и [операционной системы](http://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).

1.40. Механизм прерываний

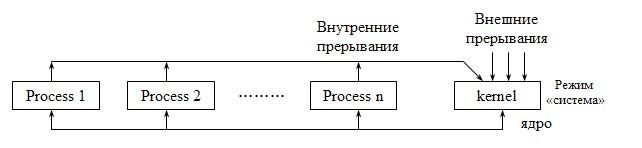
**Прерывание** - это событие, при котором меняется последовательность выполнения команд.

*Обработка прерывания идет следующим образом:*

* управление передается ОС;
* ОС запоминает состояние прерванного процесса;
* ОС анализирует прерывание и передает управление соответствующей процедуре обработки прерывания.

**СХЕМЫ ОБРАБОТКИ ПРЕРЫВАНИЙ**

На рисунке 1 представлен первый способ обработки прерывания



**Рис.1. Схема обработки прерываний**

Существуют несколько источников прерываний:

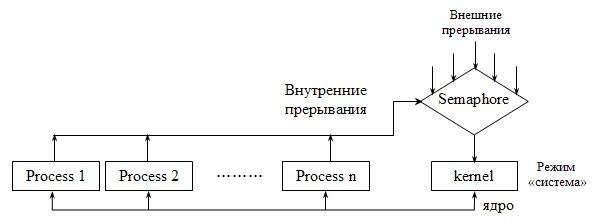
внешние могут возникать, например, при сбое напряжения питания, при выполнении несуществующей команды и т.д., внутренним прерыванием являются все вызовы SV.

Может случиться, что ядро, не успев обработать одно прерывание, должно реагировать на следующее. Поэтому в системе ОС UNIX существует несколько уровней обработки прерываний. При обработке прерываний необходимо исключить вмешательство тех процессов, которые требуют тот же ресурс, на котором находится прерываемый процесс.

Можно сказать, что допускается нахождение только одного прерывания внутри ядра системы. Для достижения этого существуют способы:

* если одно прерывание находится на ядре системы, все остальные должны быть запрещены, независимо от причины;
* в ОС UNIX ядро является ресурсом, рассматривается как CS, вход и выход из него контролируется семафором, связанным с очередью процессов.
* Очередь организована на уровне списков.

На рисунке 19.2 представлен второй способ обработки прерывания



**Рис.2. Схема обработки прерываний с использованием семафора.**

Семафор связан с изъятием из списка оответствующего требования. Такой способ гарантирует, что ядро на каждое прерывание реагирует в течении времени d t.

Каждое прерывание, которое не может обрабатываться в течении определенного времени, или ему необходим доступ к глобальной переменной, должно превратится в так называемый fork-process. Для управления такими процессами введены две дополнительные переменные:

* целочисленный семафор - State, указывающий состояние системы;
* очередь - fork, которая формируется из форк-процессов.

**Порядок обработки прерывания:**

* выполняется первичная (предварительная) обработка прерывания с закрытыми масками прерывания;
* выполняется вызов подпрограммы fork;
* вторичная обработка прерывания с использованием системных переменных и услуг.

**Особенности**: при первичной обработке выполняются самые необходимые действия (распознавание и вызов), после этого прерывание для вторичной обработки помещается в fork-очередь.

1.41. Системные вызовы

**Систе́мный вы́зов** ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *system call*) в [программировании](http://ru.wikipedia.org/wiki/%D0%9F%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%D0%B5) и [вычислительной технике](http://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0) — обращение [прикладной программы](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) к [ядру операционной системы](http://ru.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%BE_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B) для выполнения какой-либо операции.

Современные операционные системы (ОС) предусматривают [разделение времени](http://ru.wikipedia.org/w/index.php?title=%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8&action=edit&redlink=1) между выполняющимися [вычислительными процессами](http://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81) (многозадачность) и [разделение полномочий](http://ru.wikipedia.org/w/index.php?title=%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BF%D0%BE%D0%BB%D0%BD%D0%BE%D0%BC%D0%BE%D1%87%D0%B8%D0%B9&action=edit&redlink=1), препятствующее исполняемым программам обращаться к данным других программ и оборудованию. Ядро ОС исполняется в привилегированном режиме работы [процессора](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80). Для выполнения межпроцессной операции или операции, требующей доступа к оборудованию, программа обращается к ядру, которое, в зависимости от полномочий вызывающего процесса, исполняет либо отказывает в исполнении такого вызова.

*Системный вызов* возникает, когда пользовательский процесс (наподобие *emacs*) требует некоторой службы, реализуемой ядром (такой как открытие файла), и вызывает специальную функцию (например, open). В этот момент пользовательский процесс переводится в режим ожидания. Ядро анализирует запрос, пытается его выполнить и передает результаты пользовательскому процессу, который затем возобновляет свою работу. Системный вызов должен возвращать значение типа int и только int. В соответствие с принятыми соглашениями, возвращаемое значение равно 0 или любому положительному числу в случае успеха и любому отрицательному числу в случае неудачи.

1.42. Цели и средства синхронизации. Необходимость синхронизации и гонки

**Цели и средства синхронизации**

Существует достаточно обширный класс средств операционной системы, с помощью которых обеспечивается взаимная синхронизация процессов и потоков1. Потребность в синхронизации потоков возникает только в мультипрограммной операционной системе и связана с совместным использованием аппаратных и информационных ресурс об вычислительной системы. Синхронизация необходима для исключения гонок и тупиков при обмене данными между потоками, разделении данных, при доступе к процессору и устройствам ввода-вывода.

Во многих операционных системах эти средства называются средствами межпроцессного взаимодействия — Inter Process Communications (IPC), что отражает историческую первичность понятия «процесс» по отношению к понятию «поток». Обычно к средствам IPC относят не только средства межпроцессной синхронизации, но и средства межпроцессного обмена данными.

Выполнение потока в мультипрограммной среде всегда имеет асинхронный характер. Очень сложно с полной определенностью сказать, на каком этапе выполнения будет находиться процесс в определенный момент времени. Даже в однопрограммном режиме не всегда можно точно оценить время выполнения задачи. Это время во многих случаях существенно зависит от значения исходных данных, которые влияют на количество циклов, направления разветвления программы, время выполнения операций ввода-вывода и т. п. Так как исходные данные в разные моменты запуска задачи могут быть разными, то и время выполнения отдельных этапов и задачи в целом является весьма неопределенной величиной.

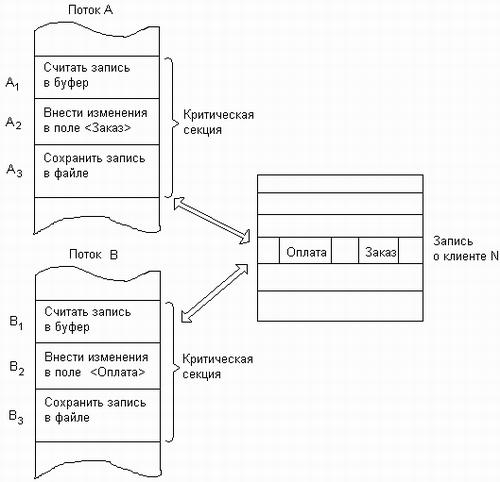
**Необходимость синхронизации и гонки**

Пренебрежение вопросами синхронизации в многопоточной системе может привести к неправильному решению задачи или даже к краху системы. Рассмотрим, например (рис. 4.16), задачу ведения базы данных клиентов некоторого предприятия. Каждому клиенту отводится отдельная запись в базе данных, в которой среди прочих полей имеются поля Заказ и Оплата. Программа, ведущая базу данных, оформлена как единый процесс, имеющий несколько потоков, в том числе поток А, который заносит в базу данных информацию о заказах, поступивших от клиентов, и поток В, который фиксирует в базе данных сведения об оплате клиентами выставленных счетов. Оба эти потока совместно работают над общим файлом базы данных, используя однотипные алгоритмы, включающие три шага.

1. Считать из файла базы данных в буфер запись о клиенте с заданным идентификатором.

2. Внести новое значение в поле Заказ (для потока А) или Оплата (для потока В).

3. Вернуть модифицированную запись в файл базы данных.



**Рис. 4.16.** Возникновение гонок при доступе к разделяемым данным

Обозначим соответствующие шаги для потока А как Al, A2 и A3, а для потока В как Bl, B2 и ВЗ. Предположим, что в некоторый момент поток А обновляет поле Заказ записи о клиенте N. Для этого он считывает эту запись в свой буфер (шаг А1), модифицирует значение поля Заказ (шаг А2), но внести запись в базу данных (шаг A3) не успевает, так как его выполнение прерывается, например, вследствие завершения кванта времени.

Предположим также, что потоку В также потребовалось внести сведения об оплате относительно того же клиента N. Когда подходит очередь потока В, он успевает считать запись в свой буфер (шаг В1) и выполнить обновление поля Оплата (шаг В2), а затем прерывается. Заметим, что в буфере у потока В находится запись о клиенте N, в которой поле Заказ имеет прежнее, не измененное значение.

1.43. Критическая секция

**Критическая секция** — участок исполняемого кода программы, в котором производится доступ к общему ресурсу (данным или устройству), который не должен быть одновременно использован более чем одним потоком исполнения. При нахождении в критической секции двух (или более) процессов возникает состояние «гонки» («состязания»). Для избежания данной ситуации необходимо выполнение четырех условий:

1. Два процесса не должны одновременно находиться в критических областях.
2. В программе не должно быть предположений о скорости или количестве процессоров.
3. Процесс, находящийся вне критической области, не может блокировать другие процессы.
4. Невозможна ситуация, в которой процесс вечно ждет попадания в критическую область.

**Критическая секция** ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *critical section*) — объект синхронизации потоков, позволяющий предотвратить одновременное выполнение некоторого набора операций (обычно связанных с доступом к данным) несколькими потоками. Критическая секция выполняет те же задачи, что и [мьютекс](http://ru.wikipedia.org/wiki/%D0%9C%D1%8C%D1%8E%D1%82%D0%B5%D0%BA%D1%81).

Между мьютексом и критической секцией есть терминологические различия, так процедура, аналогичная захвату мьютекса, называется входом в критическую секцию ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *enter*), снятию блокировки мьютекса — выходом из критической секции ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *leave*).

Процедура входа и выхода из критических секций обычно занимает меньшее время, нежели аналогичные операции мьютекса, что связано с отсутствием необходимости обращаться к ядру ОС.

В операционных системах семейства [Microsoft Windows](http://ru.wikipedia.org/wiki/Microsoft_Windows) разница между мьютексом и критической секцией в том, что мьютекс является объектом ядра и может быть использован несколькими процессами одновременно, критическая секция же принадлежит процессу и служит для синхронизации только его потоков.

1.44. Блокирующие переменные

Блокирующие переменные - некоторые глобальные переменные, к которым имеют доступ все потоки данного процесса. Каждому набору критических данных ставится в соответствие некоторая двоичная переменная. Поток входит в эти данные переменная=0; выходит-1. Перед входом в критическую секцию поток проверяет значение блокирующей переменной(не входит ли другой поток.): если 1, то можно работать и установится 0. При завершении работы устанавливается 1. Если 0, то критические данные заняты другим потоком. Структурная схема работы блокирующих переменных: см. в тетрадь, стр.47. Особенность: нельзя прерывать поток между операцией проверки и установки значения блокирующей переменной. Существует неделимая команда(см. схему), если ее нет, то между этими операциями запрещается прерывание. Преимущества: гарантирован доступ к критической секции только одного потока. Недостаток: если ресурс занят?, то поток будет все время обращаться к нему - тратить свой квант времени впустую, следовательно, используются вызовы, которые не переводят поток в состояние ожидания. Если ресурс занят?, то системный вызов не выполняет циклический поток, а переводит его в процесс ожидания.

Для синхронизации потоков одного процесса прикладной программист может использовать **глобальные блокирующие переменные**.

С этими переменными, к которым все потоки процесса имеют прямой доступ, программист работает, не обращаясь к системным вызовам ОС.

Блокирующие переменные могут использоваться и при доступе к разделяемым ресурсам любого вида.

Если все потоки написаны с учетом вышеописанных соглашений, то взаимное исключение гарантируется. При этом потоки могут быть прерваны операционной системой в любой момент и в любом месте, в том числе в критической секции.

Одно ограничение на прерывания - нельзя прерывать поток между выполнением операций проверки и установки блокирующей переменной.

1.45. Семафоры

**Семафо́р** — объект, ограничивающий количество [потоков](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F), которые могут войти в заданный участок кода. Определение введено [Эдсгером Дейкстрой](http://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0,_%D0%AD%D0%B4%D1%81%D0%B3%D0%B5%D1%80_%D0%92%D0%B8%D0%B1%D0%B5).

Семафоры используются при передаче данных через разделяемую память.

Вот некоторые из проблем, которые могут решать семафоры.

· запрет одновременного выполнения заданных участков кода;

· поочерёдный доступ к критическому ресурсу (важному [ресурсу](http://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5_%D1%80%D0%B5%D1%81%D1%83%D1%80%D1%81%D1%8B), для которого невозможен одновременный доступ).

Следующий пример показывает, как наладить поочерёдный доступ к [консоли](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D0%BE%D0%BB%D1%8C).

Проблемы семафоров

Во-первых, можно написать программу с «утечкой семафора», вызвав *enter()* и забыв вызвать *leave()*. Реже встречаются ошибки, когда дважды вызывается leave().

Во-вторых, семафоры чреваты [взаимной блокировкой](http://ru.wikipedia.org/wiki/%D0%92%D0%B7%D0%B0%D0%B8%D0%BC%D0%BD%D0%B0%D1%8F_%D0%B1%D0%BB%D0%BE%D0%BA%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0) потоков.

1.46. Тупики

При некотором стечение обстоятельств потоки могут блокировать друг друга, такую ситуацию называют тупиком (или дедлокам-deadlock). Пусть двум потокам, принадлежащим разным процессам нужны принтер и последовательный порт. Поток А запрашивает сначала принтер, затем порт. Предположим ОС назначила потоку А принтер, установила связанную с принтером переменную. Затем поток А был прерван. Поток Б выполнил запрос получения ком порта. При выполнение следующей команды был заблокирован, так как принтер занят другим потоком. Управления вновь получил поток А, сделал попытку занять порт и был заблокирован. В такой тупиковой ситуации потоки могут находиться бесконечно.

Система должна предоставить администратору средства с помощью которой он мог бы распознать тупик. Тупики могут быть предотвращены на стадии выполнения программ.

Кроме того существуют методы основанные на ведение таблиц распределения ресурсов и таблиц запросов к занятым ресурсам. Анализ таблиц позволяет обнаружить взаимные блокировки.

1.47. Сигналы

**Сигналы** в [UNIX](http://ru.wikipedia.org/wiki/UNIX), [Unix-подобных](http://ru.wikipedia.org/wiki/Unix-%D0%BF%D0%BE%D0%B4%D0%BE%D0%B1%D0%BD%D0%B0%D1%8F_%D0%BE%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 других [POSIX](http://ru.wikipedia.org/wiki/POSIX)-совместимых [операционных системах](http://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) являются одним из способов [взаимодействия между процессами](http://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%B6%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BD%D0%BE%D0%B5_%D0%B2%D0%B7%D0%B0%D0%B8%D0%BC%D0%BE%D0%B4%D0%B5%D0%B9%D1%81%D1%82%D0%B2%D0%B8%D0%B5) ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *IPC, inter-process communication*). Фактически, сигнал — это асинхронное уведомление процесса о каком-либо событии. Когда сигнал послан процессу, операционная система прерывает выполнение процесса. Если процесс установил собственный *обработчик сигнала*, операционная система запускает этот обработчик, передав ему информацию о сигнале. Если процесс не установил обработчик, то выполняется обработчик по умолчанию.

Названия сигналов «SIG…» являются числовыми константами (макроопределениями [Си](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%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))) со значениями, определяемыми в заголовочном файле signal.h. Числовые значения сигналов могут меняться от системы к системе, хотя основная их часть имеет в разных системах одни и те же значения. Утилита [kill](http://ru.wikipedia.org/wiki/Kill) позволяет задавать сигнал как числом, так и символьным обозначением.

1.48. Мьютексы

**Мью́текс** ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *mutex*, от *mutual exclusion* — «взаимное исключение») — одноместный [семафор](http://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), служащий в [программировании](http://ru.wikipedia.org/wiki/%D0%9F%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%D0%B5) для [синхронизации](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) одновременно выполняющихся [потоков](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F).

Мьютексы — это один из вариантов семафорных механизмов для организации взаимного исключения. Они реализованы во многих [ОС](http://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), их основное назначение — организация взаимного исключения для потоков из одного и того же или из разных процессов.

Мьютексы — это простейшие двоичные семафоры, которые могут находиться в одном из двух состояний — отмеченном или неотмеченном (открыт и закрыт соответственно). Когда какой-либо поток, принадлежащий любому процессу, становится владельцем объекта mutex, последний переводится в неотмеченное состояние. Если задача освобождает мьютекс, его состояние становится отмеченным.

Задача мьютекса — защита объекта от доступа к нему других потоков, отличных от того, который завладел мьютексом. В каждый конкретный момент только один поток может владеть объектом, защищённым мьютексом. Если другому потоку будет нужен доступ к переменной, защищённой мьютексом, то этот поток засыпает до тех пор, пока мьютекс не будет освобождён.

Цель использования мьютексов — защита данных от повреждения в результате асинхронных изменений ([состояние гонки](http://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5_%D0%B3%D0%BE%D0%BD%D0%BA%D0%B8)), однако могут порождаться другие проблемы — такие, как [взаимная блокировка](http://ru.wikipedia.org/wiki/%D0%92%D0%B7%D0%B0%D0%B8%D0%BC%D0%BD%D0%B0%D1%8F_%D0%B1%D0%BB%D0%BE%D0%BA%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0) (клинч).

Мьютекс это одна из реализаций [спинлок](http://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D0%BD%D0%BB%D0%BE%D0%BA).

Мьютекс отличается от [семафора](http://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D1%84%D0%BE%D1%80_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) общего вида тем, что только владеющий им [поток](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B2%D1%8B%D0%BF%D0%BE%D0%BB%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F) может его освободить, т.е. перевести в отмеченное состояние.

1.49. Мониторы

**Монитор** — в языках программирования, высокоуровневый механизм взаимодействия и синхронизации процессов, обеспечивающий доступ к неразделяемым ресурсам.[[1]](http://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BD%D0%B8%D1%82%D0%BE%D1%80_(%D1%81%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F)#cite_note-1) Подход к синхронизации двух или более компьютерных [задач](http://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C), использующих общий ресурс, обычно [аппаратуру](http://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D0%B0%D1%80%D0%B0%D1%82%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) или набор [переменных](http://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_(%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%D0%B5)).

При многозадачности, основанной на мониторах, [компилятор](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80) или [интерпретатор](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B0%D1%82%D0%BE%D1%80) прозрачно для программиста вставляет код блокировки-разблокировки в оформленные соответствующим образом процедуры, избавляя программиста от явного обращения к примитивам синхронизации.

Монитор состоит из:

· набора [процедур](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0), взаимодействующих с общим ресурсом

·  [мьютекса](http://ru.wikipedia.org/wiki/%D0%9C%D1%8C%D1%8E%D1%82%D0%B5%D0%BA%D1%81)

· переменных, связанных с этим ресурсом

·  [инварианта](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D0%BD%D1%82_(%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%D0%B5)), который определяет условия, позволяющие избежать [состояние гонки](http://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5_%D0%B3%D0%BE%D0%BD%D0%BA%D0%B8)

Процедура монитора захватывает мьютекс перед началом работы и держит его или до выхода из процедуры, или до момента ожидания условия (см. ниже). Если каждая процедура гарантирует, что перед освобождением мьютекса инвариант истиннен, то никакая задача не может получить ресурс в состоянии, ведущем к гонке.

**2.** **ОС UNIX.**

1 2 3 4 5 6 7 8 9 10 11

2.1.Основные характеристики ОС UNIX.

ОС UNIX имеет следующие основные характеристики:

* переносимость;
* *вытесняющая многозадачность* на основе процессов, работающих в изолированных адресных пространствах в виртуальной памяти;
* поддержка одновременной работы многих пользователей;
* поддержка асинхронных процессов;
* иерархическая файловая система;
* поддержка независимых от устройств операций ввода-вывода (через специальные файлы устройств);
* стандартный интерфейс для программ (программные каналы, IPC) и пользователей (командный интерпретатор, не входящий в ядро ОС);
* встроенные средства учета использования системы.

2.2.Что такое командный интерпретатор?

Командный интерпретатор в среде UNIX выполняет две основные функции:

* представляет интерактивный интерфейс с пользователем, т.е. выдает приглашение, и обрабатывает вводимые пользователем команды;
* обрабатывает и исполняет текстовые файлы, содержащие команды интерпретатора (командные файлы);

В последнем случае, операционная система позволяет рассматривать командные файлы как разновидность исполняемых файлов. Соответственно различают два режима работы интерпретатора: интерактивный и командный.

**Кома́ндная оболо́чка UNIX** ([англ.](http://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Unix shell*, часто просто «шелл» или «sh») — [командный интерпретатор](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%BE%D0%B9_%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B8), используемый в [операционных системах](http://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) семейства [UNIX](http://ru.wikipedia.org/wiki/UNIX), в котором пользователь может либо давать команды операционной системе по отдельности, либо запускать скрипты, состоящие из списка команд. В первую очередь, под shell понимаются [POSIX](http://ru.wikipedia.org/wiki/POSIX)-совместимые оболочки, восходящие к [Bourne shell](http://ru.wikipedia.org/wiki/Bourne_shell) (шелл Борна), появившемуся в [Unix Version](http://ru.wikipedia.org/wiki/UNIX_System_V) 7.

## Разновидности

* [sh](http://ru.wikipedia.org/wiki/Bourne_shell) — оригинальный шелл Борна; [dash](http://ru.wikipedia.org/wiki/Debian_Almquist_shell), [bash](http://ru.wikipedia.org/wiki/Bash), [zsh](http://ru.wikipedia.org/wiki/Zsh) — другие современные клоны Bourne shell (расположены по функциональности).
  + [ksh](http://ru.wikipedia.org/wiki/Korn_shell) (**K**orn**Sh**ell) — [клон](http://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%BE%D0%BD_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29) шелла Борна, разработанный [Дэвидом Корном](http://ru.wikipedia.org/w/index.php?title=%D0%9A%D0%BE%D1%80%D0%BD,_%D0%94%D1%8D%D0%B2%D0%B8%D0%B4&action=edit&redlink=1) из [AT&T](http://ru.wikipedia.org/wiki/AT%26T) Labs. Синтаксис совместим, функциональность интерактивности увеличена[[1]](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D0%BE%D0%BB%D0%BE%D1%87%D0%BA%D0%B0_UNIX#cite_note-zsh-1).
  + [pdksh](http://ru.wikipedia.org/w/index.php?title=Pdksh&action=edit&redlink=1) (**p**ublic **d**omain **ksh**) — открытая (почти целиком является [общественным достоянием](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%89%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B4%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5)[[2]](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D0%BE%D0%BB%D0%BE%D1%87%D0%BA%D0%B0_UNIX#cite_note-2)) реализация ksh.
  + [bash](http://ru.wikipedia.org/wiki/Bash) (**b**ourne **a**gain **sh**ell) (эмуляция совместимости POSIX[[1]](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D0%BE%D0%BB%D0%BE%D1%87%D0%BA%D0%B0_UNIX#cite_note-zsh-1)) расширенная Борном свободная (разработанная в рамках [проекта GNU](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82_GNU)) оболочка ash, сходная с pdksh. Стандартная оболочка в [Linux](http://ru.wikipedia.org/wiki/Linux).
* C shell — (несовместима с POSIX shell) оболочка, с синтаксисом на основе [Си](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8_%28%D1%8F%D0%B7%D1%8B%D0%BA_%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%29), созданная [Университетом Беркли](http://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BB%D0%B8%D1%84%D0%BE%D1%80%D0%BD%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%83%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%82%D0%B5%D1%82_%D0%B2_%D0%91%D0%B5%D1%80%D0%BA%D0%BB%D0%B8) в рамках проекта по реализации [BSD Unix](http://ru.wikipedia.org/wiki/BSD).
  + [csh](http://ru.wikipedia.org/wiki/Csh) (**C**-**Sh**ell) — оболочка из состава дистрибутива [BSD](http://ru.wikipedia.org/wiki/BSD), имеет [Си](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8_%28%D1%8F%D0%B7%D1%8B%D0%BA_%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%29)-образный синтаксис и не является [POSIX](http://ru.wikipedia.org/wiki/POSIX)-совместимой. Впервые введены возможности [управления заданиями](http://ru.wikipedia.org/w/index.php?title=%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D1%8F%D0%BC%D0%B8&action=edit&redlink=1) и произведены другие улучшения.
  + [tcsh](http://ru.wikipedia.org/wiki/Tcsh) (**csh**) — реализация csh с интерактивными возможностями, не уступающими bash[[1]](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D0%BE%D0%BB%D0%BE%D1%87%D0%BA%D0%B0_UNIX#cite_note-zsh-1). Удобна для интерактивной работы. Совместима с csh.
* [ash](http://ru.wikipedia.org/wiki/Almquist_shell) (Almquist shell, оболочка Альмквиста), [BusyBox](http://ru.wikipedia.org/wiki/BusyBox) — современные микро-версии, предназначенные для [встраиваемых систем](http://ru.wikipedia.org/wiki/%D0%92%D1%81%D1%82%D1%80%D0%B0%D0%B8%D0%B2%D0%B0%D0%B5%D0%BC%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), а также используемые в мини-[дистрибутивах](http://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D0%B8%D0%B2_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B).

2.3.Типы доступных командных интерпретаторов в ОС UNIX. Как можно узнать какой командный интерпретатор используется в данный момент?

## Разновидности

* [sh](http://ru.wikipedia.org/wiki/Bourne_shell) — оригинальный шелл Борна; [dash](http://ru.wikipedia.org/wiki/Debian_Almquist_shell), [bash](http://ru.wikipedia.org/wiki/Bash), [zsh](http://ru.wikipedia.org/wiki/Zsh) — другие современные клоны Bourne shell (расположены по функциональности).
  + [ksh](http://ru.wikipedia.org/wiki/Korn_shell) (**K**orn**Sh**ell) — [клон](http://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%BE%D0%BD_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29) шелла Борна, разработанный [Дэвидом Корном](http://ru.wikipedia.org/w/index.php?title=%D0%9A%D0%BE%D1%80%D0%BD,_%D0%94%D1%8D%D0%B2%D0%B8%D0%B4&action=edit&redlink=1) из [AT&T](http://ru.wikipedia.org/wiki/AT%26T) Labs. Синтаксис совместим, функциональность интерактивности увеличена[[1]](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D0%BE%D0%BB%D0%BE%D1%87%D0%BA%D0%B0_UNIX#cite_note-zsh-1).
  + [pdksh](http://ru.wikipedia.org/w/index.php?title=Pdksh&action=edit&redlink=1) (**p**ublic **d**omain **ksh**) — открытая (почти целиком является [общественным достоянием](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%89%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B4%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0%B5)[[2]](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D0%BE%D0%BB%D0%BE%D1%87%D0%BA%D0%B0_UNIX#cite_note-2)) реализация ksh.
  + [bash](http://ru.wikipedia.org/wiki/Bash) (**b**ourne **a**gain **sh**ell) (эмуляция совместимости POSIX[[1]](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D0%BE%D0%BB%D0%BE%D1%87%D0%BA%D0%B0_UNIX#cite_note-zsh-1)) расширенная Борном свободная (разработанная в рамках [проекта GNU](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82_GNU)) оболочка ash, сходная с pdksh. Стандартная оболочка в [Linux](http://ru.wikipedia.org/wiki/Linux).
* C shell — (несовместима с POSIX shell) оболочка, с синтаксисом на основе [Си](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8_%28%D1%8F%D0%B7%D1%8B%D0%BA_%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%29), созданная [Университетом Беркли](http://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BB%D0%B8%D1%84%D0%BE%D1%80%D0%BD%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%83%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%82%D0%B5%D1%82_%D0%B2_%D0%91%D0%B5%D1%80%D0%BA%D0%BB%D0%B8) в рамках проекта по реализации [BSD Unix](http://ru.wikipedia.org/wiki/BSD).
  + [csh](http://ru.wikipedia.org/wiki/Csh) (**C**-**Sh**ell) — оболочка из состава дистрибутива [BSD](http://ru.wikipedia.org/wiki/BSD), имеет [Си](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8_%28%D1%8F%D0%B7%D1%8B%D0%BA_%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%29)-образный синтаксис и не является [POSIX](http://ru.wikipedia.org/wiki/POSIX)-совместимой. Впервые введены возможности [управления заданиями](http://ru.wikipedia.org/w/index.php?title=%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B7%D0%B0%D0%B4%D0%B0%D0%BD%D0%B8%D1%8F%D0%BC%D0%B8&action=edit&redlink=1) и произведены другие улучшения.
  + [tcsh](http://ru.wikipedia.org/wiki/Tcsh) (**csh**) — реализация csh с интерактивными возможностями, не уступающими bash[[1]](http://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BE%D0%B1%D0%BE%D0%BB%D0%BE%D1%87%D0%BA%D0%B0_UNIX#cite_note-zsh-1). Удобна для интерактивной работы. Совместима с csh.
* [ash](http://ru.wikipedia.org/wiki/Almquist_shell) (Almquist shell, оболочка Альмквиста), [BusyBox](http://ru.wikipedia.org/wiki/BusyBox) — современные микро-версии, предназначенные для [встраиваемых систем](http://ru.wikipedia.org/wiki/%D0%92%D1%81%D1%82%D1%80%D0%B0%D0%B8%D0%B2%D0%B0%D0%B5%D0%BC%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), а также используемые в мини-[дистрибутивах](http://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D0%B8%D0%B2_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B).

Чтобы узнать о том, какой командный интерпретатор используется в данный момент, введите следующую команду):

**$ grep foo /etc/passwd**

где **foo —** имя, с которым вы регистрируетесь в системе, а **$** — приглашение командной строки

2.4.Дайте пояснение понятию «терминал».

Текстовый принцип работы с системой позволяет отвлечься от конкретных частей компьютера, вроде клавиатуры и видеокарты с монитором, рассматривая единое оконечное устройство, посредством которого пользователь вводит текст и передает его системе, а система выводит необходимые пользователю данные и сообщения. Такое устройство называется *терминалом*. В общем случае терминал — это точка входа пользователя в систему, обладающая способностью передавать текстовую информацию.

2.5.Дайте пояснение понятию «консоль».

У мейнфреймов имелся особый терминал, который предназначался для системного администратора и назывался консолью. Консоль обычно подсоединялась к компьютеру не по последовательному интерфейсу, а через отдельные разъемы (иногда в качестве устройства вывода в ее состав вместо монитора входило печатающее устройство).

Поскольку в UNIX-системах обычно соблюдаются традиции, клавиатура и монитор персонального компьютера ведут себя так же, как ранее консоль. Преимущество такого решения состоит в том, что все старые программы, создававшиеся для администраторов UNIX, без проблем работают и на новом типе системной консоли.

Но, кроме консоли, Linux позволяет подключать к компьютеру и удаленные терминалы и, более того, обеспечивает возможность работы с несколькими виртуальными терминалами с одной консоли.

2.6.Дайте пояснение понятию «оболочка».

Теперь надо сказать несколько слов об оболочке. Оболочка, или просто shell (это слово часто не переводят, а оставляют в английском написании), — это программа, которая осуществляет все общение с пользователем. Именно оболочка воспринимает все команды, вводимые пользователем с клавиатуры, и организует исполнение этих команд. Поэтому оболочку можно назвать еще командным процессором (более привычный термин для пользователя DOS , не правда ли?). Строго говоря, когда выше говорилось, например, "система выводит приглашение", это неправильно, поскольку приглашение выводит именно оболочка, ожидая ввода пользователем очередной команды. Каждый раз, когда очередной пользователь входит в систему, команда login запускает для него командный процессор — оболочку. Если вы логировались со второго терминала под именем пользователя jim (или под другим выбранным вами именем), то обратите теперь внимание на различие в приглашениях у пользователей root и jim. У пользователя rootприглашение оканчивается символом #, а у всех остальных пользователей — символом $.

2.7.Получение справки. Разделы справочных руководств.

Для доступа к странице справочного руководства используется команда

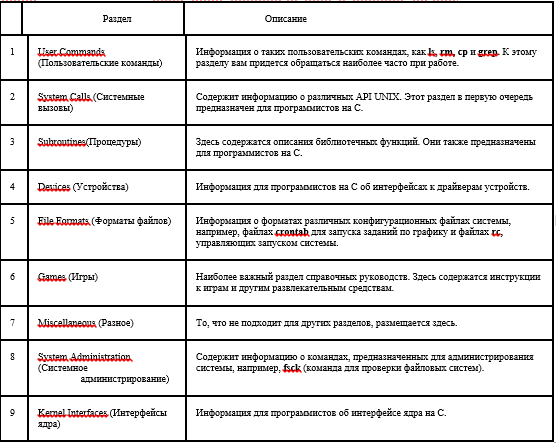
***man имя\_команды***

Например, чтобы прочесть справку о самой команде man, введите команду

***man man***

Иногда известно, что делает команда, но ее название почему-то не вспоминается. В этом случае можно воспользоваться опцией -k команды man.

Предположим, что вы хотите узнать о командах поиска в текстовых файлах определенных слов или фраз и вывода тех участков файла, где они содержатся. Вначале нужно запустить команду

***man -k search***

2.8.Имена файлов в ОС UNIX. Работа с с нестандартными именами файлов.

В ОС UNIX файл - это любая информация, такая как письмо, отчет доклад, программа и т.д., которую вы храните. Файл имеет несколько частей: имя , содержание и индексный дескриптор**inode** (information node). В индексном дескрипторе хранится служебная и системная информация такая как длина файла, место хранения содержания файла на диске и т.д.

Существует четыре основных типа файлов: **обычный** файл(regular), **каталог** (directory), **специальный** (special)и **символическая связь** (symbolic link).

**обычные** файлы: хранение текста

письма, заметки, данные, программы

**каталоги**: хранение имен других файлов

действует как оглавление

группирует вместе родственные файлы

**специальные** файлы: используются для устройств

принтеры, терминалы и т.д.

**символические связи**: содержат инструкции, которые указывают на другой файл

Когда вы создаете файл, вы должны выбрать для него имя. Имя файла должно отражать его содержание. При именовании файла вы МОЖЕТЕ использовать:

* до 14 бит в System V и до 256 бит в UFS файловой системе. Система обрезает более длинные имена.
* прописные и строчные буквы (английского алфавита). ОС UNIX чувствительна к типу буквы (файл с именем **mydoc** не тоже самое, что файл с именем **MYDOC**).
* цифры
* точку(.)
* знак подчеркивания(\_)

Так как некоторые символы имеют специальное значение в командной строке, избегайте использования:

* \* ! / [ ] ( ) ; & ^ | < > ' " ` { } + - \
* пробелов

**ИМЕНА ФАЙЛОВ -- ПРАВИЛА**

* Должны описывать содержание
* до 14 или 256 бит
* Рекомендуются прописные и строчные буквы, цифры, подчеркивание(\_) или точка(.)
* Избегать использования специальных символов или пробела

2.9.Знать команды и уметь использовать.

2.9.1. Перемещение по файловой системе.

**cd** <путь> - перейти к заданому пути, пример

**cd** //home/student

**cd** .. - назад на один каталог  
**cd** // - задает самый начальный путь  
**cd** ~/Музыка - задает адрес домашнего каталога

2.9.2. Копирование файлов и каталогов

Команда ***cp*** – копіює файли; має такі ключі:

*-d* – копіювати символьні зв’язки;

*-f* – видаляти наявні файли;

*-i* – запитувати про видалення наявних файлів;

*-l* - створювати твердий зв’язок.

2.9.3. Перемещение и переименование файлов и каталогов.

mv dir1 new\_dir переименовать или переместить файл или директорию

2.9.4. Удаление файлов и каталогов.

Команда ***rm*** видаляє вказаний файл. За замовчуванням не видаляє каталогів. Якщо файл помічений як такий, що призначений тільки для читання, то запитує підтвердження. Має ключі:

*-d* – видаляти каталоги (тільки для користувача *root*);

*-i* – запитувати щодо кожного файла;

*-r* – видаляти зміст вкладених каталогів;

*-v* – друкувати імена файлів, які видаляються.

***rmdir*** *–* видаляє порожні каталоги*.*

2.9.5. Создание файлов и папок

***mkdir*** *– створює каталоги.*

***touch*** *<назва файлу> - створює текстові файли*

2.9.6. Просмотр списка активных процессов и пользователей.

***ps*** – видає список активних процесів (*-l* - повна інформація);

***who*** – видає список користувачів, що працюють у системі;

2.9.7. Просмотр содержимого каталогов (разные варианты) и объяснить результат.

**ls, dir, vdir**

**LS** - Команда ls сначала выводит список всех файлов (не каталогов), перечисленных в командной строке, а затем выводит список всех файлов, находящихся в каталогах, перечисленных в командной строке. Если не указано ни одногофайла, то по умолчанию аргументом назначается "." (текущий каталог). Опция -d заставляет ls не считать аргументы-каталоги каталогами. Будут отображаться только файлы, которые не начинаются с `.' или все файлы, если задана опция -a.

**DIR** - dir видае те ж саме що і ls -C -b

**VDIR -** vdir видае те ж саме що і ls -l -b

2.9.8. Метасимволы и символы-заместители

У командах оболонки можна вказувати множини файлів, використовуючи спеціальні метасимволи:

- „*\** ” – будь-яка кількість будь-яких символів в імені;

- „*?”* – один будь-який символ;

- *[abc]* – один символ з указаної множини („*a*”, або „*b*”, або „*c*”).

2.9.9. Конвееры и перенаправления ввода-ввывода

Переключення вихідного потоку команди здійснюється за допомогою символу "*>*". Наприклад, можна перенаправити вихід команди *ls* у файл *filelist*:

*ls > filelist*

При цьому вихідні дані команди будуть виводитися не в стандартний потік консолі, а у файл з указаним іменем.

Використання символів *">>"* дозволяє додавати вихід команди до змісту наявного файла.

*ps >> filelist*

Переключення стандартного вводу здійснюється символом *"<"*, при цьому, ввід здійснюється з указаного файлу, наприклад:

*sort < filelist.*

Програмні канали дозволяють здійснювати безпосередню взаємодію процесів, не використовуючи допоміжні файли. У командній оболонці можна організовувати конвеєри команд, які будуть виконуватися одночасно, при цьому кожна попередня команда передаватиме результати роботи на вхід наступної. Для визначення конвеєра необхідно записати назви команд через символ " | ", наприклад:

*cat file, file2, file3 | sort > file4* (з’єднання трьох файлів з передачею результату у команду сортування рядків з подальшим занесенням у файл *file4*).

2.9.10. Работа с текстовыми файлами средствами ОС UNIX.

2.9.10.1. режимы работы в текстовом редакторе

**Нормальный режим** обусловливает немедленную реакцию на нажатие клавиш. Для выхода в нормальный режим необходимо нажать клавишу ESC.

**Командный режим** обусловливает набор команды в строке в нижней части экрана. Переход в этот режим осуществляется с нормального режима путем набора символа ":".

Переход в другие режимы (визуальный, вставки и т.д.) осуществляется с нормального с помощью некоторых клавиш, с последующим уточнением действий.

**Сразу после запуска редактор vi будет находиться в командном режиме.**

В этом режиме нажатия клавиш интерпретируются как команды редактору, а не как текст, вводимый в документ. Чтобы переключиться в режим ввода текста, необходимо нажать одну из клавиш: a, i или о.

2.9.10.2. команды для установки ввода текста

***Клавиша Действие***

* a додати текст після курсору;
* A додати текст у кінець рядку;
* i додає текст перед курсором;
* o відкрити новий рядок під поточним;
* O відкрити новий рядок над поточним;

2.9.10.3. перемещение по тексту в редакторе

* h Перемещает курсор влево на один символ.
* j Перемещает курсор вниз на один символ.
* k Перемещает курсор вверх на один символ.
* l Перемещает курсор вправо на один символ.
* w Перемещает курсор вперед на одно слово.
* b Перемещает курсор назад на одно слово.
* е Перемещает курсор в конец следующего слова.
* 0 Перемещает курсор в начало строки.
* $ Перемещает курсор в конец строки.
* Н Перемещает курсор на первую строку на экране.
* L Перемещает курсор на последнюю строку на экране.

2.9.10.4. команды редактирование текста

* D Удаляет текст от позиции курсора до конца строки,
* dd Удаляет всю текущую строку целиком.
* ndd Здесь n — число удаляемых строк.
* rc Здесь с — символ. Эта команда заменяет символ в позиции курсора символом, который указан за r.
* R Текст, вводимый после этой команды, замещает текущий текст, начиная с позиции курсора. Выход из режима - Escape.
* S Удаляет текущую строку и начинает ввод текста на новой пустой строке.
* х Удаляет символ в позиции курсора и сдвигает следующие за ним символы влево.
* N x видалити N символів після курсору;
* X Удаляет символ перед курсором и сдвигает следующие за ним символы влево.
* ~ Заменяет букву на позиции курсора той же буквой другого регистра.
* J Объединяет текущую строку с предыдущей.
* :r {файл} вставити зміст файлу під курсор;
* :r! {команда} вставити стандартний вихідний потік команди під курсор;
* v початковий символ, який виділений;
* N u відмінити N змін;
* CTRL-R повторити зміни;
* ga показати кодіровку символу над курсором;

2.9.10.5. Операции над файлами и выход из редактора.

* :е *filename* Загружает заданный файл в редактор vi для редактирования. Если заданный файл не существует, то создается новый файл.
* :е! Отбрасывает все изменения и перезагружает с диска старый вариант файла.
* :w\_f {файл} записати в файл;
* :wq {файл} записати файл і вийти із редактору;
* vim {файл} запустити редактор і завантажити файл;
* :f {файл} перейменувати файл.

2.9.10.6. Поиск в тексте и замена текста.

* */шаблон* Здесь *шаблон* — это фрагмент текста (слово, словосочетание, фраза и т. п.), который требуется найти. Редактор vi осуществляет поиск в файле в прямом направлении до первого совпадения с заданным шаблоном.
* / Повторяет поиск заданного образца в прямом направлении до обнаружения следующего совпадения.
* *?шаблон* Здесь *шаблон* — это фрагмент текста, который требуется найти. Редактор vi осуществляет поиск в файле в обратном направлении до первого встреченного совпадения с заданным шаблоном.
* ? Повторяет поиск шаблона в обратном направлении до обнаружения очередного совпадения.
* % Перемещает курсор на соответствующую парную скобку. Эта команда полезна для программистов.
* *:s/шаблон1 /шаблон2* Заменяет в текущей строке каждое совпадение *шаблона1* на *шаблон2.*
* *:%s/шаблон1 /шаблон2* Заменяет в файле каждое совпадение *шаблона1* на *шаблон2.*

2.9.10.7. копирование, вырезание и вставка текста в редакторе.

* yw Помещает в буфер слово, на котором в данный момент находится курсор.
* у$ Помещает в буфер текст от текущей позиции курсора до конца данной строки.
* уу Помещает в буфер всю текущую строку.
* nуу Здесь *п —* число строк, заносимых в буфер. Например, команда **5уу** помещает в буфер текущую строку, а также четыре строки, следующие за ней.
* Текст из буфера можно вставить на любое место документа, перемещая на требуемое место курсор и применяя команду р или Р. Команда р вставляет текст в документ после курсора. Команда Р вставляет текст перед курсором.

2.10. Изменение прав доступа к файлам и каталогам

Для кожного файла існують права доступу, що визначені для трьох категорій суб’єктів: володаря файла, членів групи, до якої належить володар, і інших користувачів.

Існує три типи прав: читання (*r*); запис (*w*); запуск на виконання (*x*).

Користувачеві надається право на здійснення операції, якщо воно дозволено для будь-якої категорії, до якої належить користувач. Група файлу та його володар зберігаються у дескрипторі файлу та можуть бути змінені адміністратором або поточним володарем файлу за допомогою команд *chgrp* та *chown*.

Для встановлення прав використовується команда

*chmod <режим> <файл>*

*u* – володар; *+* – додати; *r* – читання;

*g* – група; *-* – прибрати; *w*  – запис;

*o* – інші; *=*  – призначити; *x* – запуск.

*a* – все;

Приклад:

*chmod a=r, u+w file1 file2*

2.11. Добавление и удаление учетных записей пользователей

У системі можуть виконувати дії тільки ті користувачі, які мають обліковий запис та здійснили вхід до неї. Всі облікові записи користувачів системи зберігаються в файлі */etc/passwd* у такому форматі:

*ім’я\_користувача: пароль: ідентифікатор\_користувача: ідентифікатор\_групи: коментар: каталог\_користувача: оболонка\_користувача*

Користувач може бути включеним до однієї або кількох груп безпеки. Облікові записи груп безпеки та їх склад зберігаються у файлі */etc/group* у такому форматі:

*ім’я\_групи: пароль\_групи: ідентифікатор\_групи: список\_членів\_групи*

Для кожного різновиду користувачів – **користувач (user),** **група (group)** і **інші (others)** – існує набір бітів повноважень: **читання (read), записування (write)** і **виконання (execute).**

**Користувач Вид зміни Біт доступу**

u - володар; **+** - додати; r - читання;

g - група; **-** - убрати; w - записування;

o - інші; **=** - назначить; x - запуск.

a - всі;