**Принципи побудови інтерфейсів операційних систем.**

Нагадаємо що ОС завжди виступає як інтерфейс між апаратурою комп’ютера і користувачем з його задачами. Під інтерфейсами операційних систем тут і надалі слід розуміти спеціальні інтерфейси системного і прикладного програмування призначених для виконання слідуючи задач:

1. Управління процесами, котрі включають в себе слідуючий набір основних функцій:

Запуск, зупинка і зняття задачі з виконання;

Задання і змінення пріоритету задачі;

Взаємодія задач між собою;

REC (remote procedure call) – віддалений визов програм.

1. Управління пам’яттю:

Запит на виділення блока пам’яті:

Відображення файлів і виводом:

3.Управління вводом і виводом:

Запит на управління віртуальними приладами.

Файлові операції (запити до системи управління файлами на створення, змінення і знищення даних, організованих у файли).

Тут ми перерахували основні набори функції, котрі виповнюються ОС по відповідним запитам від задач. Що стосується користувацького інтерфейсу операційної системи, то він реалізується з допомогою спеціальних програмних модулів, котрі приймають його команди на відповідному язиці і транслюють їх в звичайні визови t відповідно основному інтерфейсу системи.

Зазвичай ці люди називають інтерпретатором команд. Так наприклад функції такого інтерпретатора в МS-POS виповнює модуль GOMMAND.COM.

Получив від користувача і синтаксичного аналізу або сам виповнює дії, або звертається до інших модулів ОС, використовує механізм АРІ.

Треба примітити, що в послідовні роки велику популярність отримали графічні інтерфейси. (GUI), в котрих задіяні відповідні маніпулятори типу “Миш” або “трек бал”.

**Інтерфейс прикладного програмування.**

Передусім необхідно однозначно розділити загальний термін ФРІ (applicatiio program intertace – інтерфейс прикладного програму3вання) на слідуючи напрямки:

АРІ як інтерфейс високого рівня, належить до бібліотек RTL;

АРІ прикладних і системних програм, що входять в поставку операційної системи; та інші АРІ.

АРІ представляє собою набір функцій, представлених системою програмування розробнику прикладної програми і орієнтованих на організацію взаємодії результуючої прикладної програми з цільовою враховуючою системою.

Цільова вираховуюча система представляє собою сокупність програмних і апаратних засобів в оточенні яких виконується результативна програма.

В практиці АРІ використовується не тільки прикладними, но і багатьма системними програмами як в складі ОС так і в складі системи програмування.

Функції АРІ позволяють розробнику будувати результуючу прикладну програму так, щоб використовувати засоби цільової вираховучою системи для виповнення типових операцій.

Існує декілька варіантів реалізації АРІ:

Реалізація на рівні ОС

Реалізація на рівні системи програмування.

Реалізація на рівні внутрішньої бібліотеки процедур і функцій.

Можливості АРІ можна оцінити з слідуючи позицій:

Ефективність виповнення функцій АРІ – враховують в себе швидкість виповнення функцій і об’єм вираховуючих ресурсі, потрібних для їх виконання;

Обсяг представлених можливостей;

Залежність прикладної програми від архітектури цільової враховуючої системи.

**Реалізація функції АРІ на рівні ОС**.

При реалізації функцій АРІ на рівні ОС за їх виконання відповідальність несе ос. Об’єктний код, виконуючий функції, або безпосередньо входить в склад ОС, або постачається в складі динамічно завантажуючи бібліотек, розроблених для даної ОС. Система програмування, відповідальна тільки за те, щоб організувати інтерфейс для виклику цього коду.

Недостатком організації АРІ по такій схемі являється практично Повна відсутність перенесення не тільки кода результуючої програми, но і кода вихідної програми. Програма, створена для одної архітектури враховуючої системи, не зможе виконувати на враховуючій систем іншої архітектури навіть після того, як її об’єктний код буде повністю перебудований. Скоріше всього систем програмування не зможе виконати перебудову вихідного коду для нової архітектури враховуючої системи, оскільки багато функцій АРІ орієнтовані на визначені ОС будуть у новій архітектурі просто відсутні.

**Реалізація функції АРІ на рівні системи програмування.**

Якщо функції АРІ реалізуються на рівні системи програмування то вони представляються користувачу у вигляді бібліотеки функцій відповідної мови програмування. RTL (rub time library). Система програмування представляє користування бібліотеку відповідної мови програмування в забезпечує підключення до результуючої програми об’єктного коду, відповідального за виконання усіх функцій. Очевидно, що ефективність функцій АРІ в такому варіанті буде трохи нижча ніж при безпосередньому звертанні до функцій ОС.

Існування всіх необхідних викликів і звертань до функції ОС у об’єктному коді RTL забезпечує система програмування. Однак перенесення вихідного коду програми у такому варіанті буде найвищим, оскільки син такс і семантика всіх функцій будуть суворо регламентовані у стандарті відповідного язика програмування. Вони залежать від мови і не залежать від архітектури цільової враховуючої системи. Тому для виконання прикладної програми на новій архітектурі враховуючої системи досить заново побудувати код результуючої програми за допомогою відповідної системи програмування.

Однообразне виконання функції мови, забезпечується системою програмування. При орієнтації на різні архітектури цільової враховуючої системи в системі програмування можуть бути потрібні різні комбінації викликів функції ОС для виконання одних і тих функцій вихідної мови. Це повинно бути враховано в коді RTL. У загальному випадку для кожної архітектури цільової враховуючої системи буде потрібен свій код RTL мови програмування. Вибір того чи іншого об’єктного коду RTL для підключення до результуючої програми система програмування забезпечує автоматично.

**Реалізація функції АРІ за допомогою зовнішніх бібліотек.**

При реалізації функції АРІ за допомогою зовнішніх бібліотек вони подаються користувачу у вигляді бібліотеки процедур і функцій, створеної стороннім розробником. При чому розробником такої бібліотеки може виступати той самий виробник. Система програмування відповідальна тільки за те, щоб підключити об’єктний код бібліотеки до результуючої програми. При чому зовнішня бібліотека може бути динамічно завантаженою. З точки зору ефективності виконання цей метод реалізації АРІ має найнижчі результати, оскільки зовнішня бібліотека звертається як до функцій ОС так і до функцій RTL мови програмування. Тільки при дуже високій якості зовнішньої бібліотеки її ефективність стає на одному рівні з бібліотекою RTL. Якщо говорити про перенесення вихідного коду, то тут потреба тільки одна – використована зовнішня бібліотека повинна бути доступна в любій з архітектур вираховуючих систем на котрі орієнтована прикладна програми. Тоді вдається досягнути перенесення. Це можливо, якщо використована бібліотека задовольняє який-небудь прийнятий стандарт, а система програмування підтримує це стандарт. Для більшості специфічних бібліотек окремих розробників це не так. Якщо користувач використовує якусь бібліотеку, то вона орієнтована на обмежений набір доступних архітектур цільової враховуючої системи.

В цілому розвиток функцій прикладного АЗІ йде в напрямку спроби створити бібліотеки АЗШ забезпечує широке перенесення вихідного коду. Розробка широко використовуваного стандарту ФЗШ поки що залишається справою майбутнього. Що стосується прикладних програм, то багато більшу перспективу для них представляють технології зв’язані з розробками в рамках архітектури “клієнт-сервер” або трьохрівневої архітектури створення прикладань. У цьому напрямку передові виробники ОС, СУБД і систем програмування скоріше дойдуть до згоди, ніж в напрямку стандартизації АРІ.

**ІІ. Функції файлової системи ОС і ієрархія даних.**

Нагадаємо, що під файлом звичайно розуміють набір даних організованих у вигляді сукупності записів однакової структури. Для управління цими даними створюються відповідні системи управління файлами. Можливість змінити діло з логічним рівнем структури даних і операцій, виконуючих над ними в процесі їх обробки, представляє файлова система.

Таким чином, файлова система цей набір специфікацій і відповідно програмного забезпечення, які відповідають за створення, знищення, організацію, зчитування, запис, модифікацію і переміщення файлової інформації. Іменно файлова система визначає спосіб організації даних на диску або на якому-небудь іншому носію даних. Система управління файлами являється основною підсистемою в більшості сучасних ОС, хоча в принципі можна обходитись і без неї.

По-перше, через систему управління файлами, зв’язуються по даних всі системи обробляючі програми. По-друге, за допомогою цієї системи вирішуються проблеми централізованого розподілення дискового простору і управлінням даними.

По-третє завдяки ви користуванню чи іншу систему управління файлами користувачам представляються слідуючи можливості:

Створення, зниження. Переназвання;

Робота з не дисковими периферичними устройствами так, як із файлами;

Обмін даними між файлами, між устройствами між файлом і устройством або навпаки.

Робота з файлами за допомогою звертань до програмних модулів від несанкціонованого доступу. В деяких ОС може бути декілька систем управління файлами, що забезпечує їм можливість працювати з декількома файловими системами.

Система керування файлами – організація зручного доступу до даних, організованих як файли.

Термін “файлова система” визначає перед усе принципи доступу до даних організованих у файли.

Система управління файлами – комплекс програмних модулів забезпечуючи роботу з файлами в конкретній операційній системі.

Будь-яка система управління файлами не існує сама по собі – вона розроблена для роботи в конкретній ОС. Для кожної ОС повинна бути розроблена відповідна система управління файлами.

**Структура магнітного диску.**

Для того щоби можна було загрузити з магнітного диску ОС, а потім з її допомогою організувати роботу тої чи іншої системи управління файлами були прийняті спеціальні системні відношення про структуру диска. Розташування структури даних, несучі інформацію про логічну організацію диска і найпростішу програму. За допомогою якої можна знаходити і завантажувати програми тої чи іншої ОС. Очевидно це самий перший сектор магнітного диску. Як відомого інформація на магнітних дисках розміщується і передається блоками. Кожний такий блок називається сектором, сектори розташовані на концентричних доріжках поверхності диска. Кожна доріжка створюється при обертанні магнітного диску під зафіксованою в деякому опреділеному положенні головкою читання запису. Накопичувач на жорстких магнітних дисках (НЖМД) має один або більше дисків.

Групи дорожок одного радіуса, розміщених на поверхностях магнітних дисків, утворюють так звані циліндри. Сечасні жорсткі диски можуть мати декілька десятків тисяч циліндрів, в той час як на поверхності дискети число доріжок тільки 80.

Кожний сектор складається з поля даних службової інформації обмежуючи і індитифікуючи його.

Розмір сектора установлюється контролером або драйвером.

Фізичний адрес сектора на диску визначається за допомогою трьох “координат” [c-h-s] де с – номер циліндра, h- номер робочої поверхні диску, S – номер сектора на доріжці.

Номер робочої поверхності диска “h” належить діапазону О.Н-1, де Н – число магнітних головок в накопичувачі.

Номер сектора на доріжці “S” указуються у діапазоні 1..S, де S – кількість секторів на доріжці.

Жорсткий диск може бути робочий на декілька розділів котрі в принципі потім можуть використовуватися або одною ДС або різними ДС. Причому самим головним являється те, що на кожному розділі може бути організована своя файлова система. Однаково для організації навіть єдиної файлової системи необхідної визначити хоч один розділ.

Розділи диска можуть бути двох типів – первинний і розширений (primary I extended).

Максимальне число primary – розмірів дорівнює чотирьом.

Якщо primary – розділів декілька. То тільки один із них може бути активним.

Згідно специфіці на одному жорсткому диску може бути тільки одних extended – розділ, котрий, в свою чергу, може бути розділений на велику кількість підрозділів – логічних дисків. Один із primary – розділів повинен бути активним і меню з нього повинна завантажуватись програма за грудки ОС.

# ІІІ. Дисципліни диспетчеризації

Коли говорять про диспетчеризацію, то завжди мають на увазі задачі на поняття процесу. Так як ці терміни часто використовуються іменно у такому смислі ми змушені будемо використовувати термін “процес” як синонім терміну “задача”.

Розрізняють дві дисципліни диспетчеризації – без пріоритетні і пріоритетні.

При без приорітетному обслуговуванні вибір задачі приводиться в деякому наперед установленому порядку без урахування їх відносної важливості і часу обслуговування.

При реалізації приоритетних дисциплін обслуговування окремим задачам надає переважаюче право попасти в стан виконання.

Перерахунок дисциплін обслуговування і їх класифікація зображені на мал. 3.1.

В порядку черги

Дисципліни диспетчеризації

Безпріоритетні

Пріоритетні

Випадковий вибір процесу

Багатопріоритетний циклічний

З абсолютним пріоритетом

З відносним пріоритетом

Адаптивне обслуговування

Пріоритет залежить від часу очікування

Пріоритет залежить від часу обслуговування

Циклічний алгоритм

Лінійні

Циклічні

З фіксованим пріоритетом

З динамічним пріоритетом

Запам’ятаємо про пріоритети слідуючи:

Пріоритет присвоєний задачі. Може являтися величиною настійною;

Пріоритет задачі може змінюватись в процесі її розв’язання.

Диспетчеризація з динамічним пріоритетом потребує додаткових витрат на вираховування значень пріоритетів виконуючих задач тому у багатьох ОС реального часу використовуються методи диспетчеризації на основі статистичних пріоритетів. Дисципліна обслуговування SLN потребує щоб для кожного завдання була відома оцінка в потребах машинного часу. Необхідність повідомляти ОС характеристики задач, в котрих описувались би потреби в ресурсах враховуючої системи., привела до тог, що були розроблені відповідні мовні засоби. Мова JCL була однією із найбільш відомих.

Користувачі змушені були вказувати передбачуваний час виконання, і для того, щоб вони не зловживали можливістю вказати менший час виконання ввели підрахунок реальних потрібностей. Диспетчер задач порівнював заказаний час і час виконання в у випадку перевищення вказаної оцінки у даному ресурсі ставив дане завдання не на початок а на кінець черги.

Дисциплінарне обслуговування SJN припускає, що є тільки одна черга завдань, готових до виконання. І завдання котрі в процесі свого виконання були тимчасово заблоковані (наприклад, чекали завершення операції вводу-виводу). Знову попадають в кінець черги готових до виконання на рівні з знову поступаючим. Дисципліна диспетчеризації RR –одна із найпоширеніших дисциплін. Однаково бувають ситуації, коли ОС не підтримує у явному вигляді дисципліну карусельної диспетчеризації. Наприклад в деяких ОС реального часу використовується диспетчер задач, працюючий за принципом абсолютних пріоритетів. Іншими словами, зняти задачу з виконання може тільки поява задачі з більш високим пріоритетом.

Тому, якщо потрібно організувати обслуговування задач таким чином, щоб всі вони почали процесорний час рівномірно і рівноправно, то системний оператор може сам організовувати цю дисципліну. Для цього одночасно всі користувацьким задачам присвоїти однакові пріоритети і створити одну високо пріоритетну задачу, котра не повинна нічого робити, але контра буде по таймеру плануватися на виконання. Ця задача зніме з виконання текуче прикладання, вона буде поставлена в кінець очереді і по-скільки цій високо пріоритетній задачі насправді нічого робити не треба, то вона зразу звільнить процесор і з черги готовності буде взята слідуюча задача. В своїй найпростішій реалізації дисципліна карусельної диспетчеризації передбачає, що всі задачі мають однаковий пріоритет.

Якщо необхідно ввести лаконізм пріоритетного обслуговування то це як правило робиться за рахунок організації декількох черг. Процесорний час буде представлений в першу чергу тим задачам, котрі знаходяться в найпривілейованій черзі.

Витісняючі і невітіснюючі алгоритми диспетчеризації. Диспетчеризація без пере розподілення процесорного часу це є не витісняюча багатозадачність – це такий спосіб диспетчеризації процесі при якому активний процес виконується до тих під поки від сам не віддасть управління диспетчеру задач для вибору із черги іншого готового до виконання процесу. Дисципліна SJN відноситься до невитісняючих.

Диспетчеризація з перерозподілом процесорного часу між задачами є витісняючою багатозадачнішою. Це такий спосіб при якому рішення про переключання процесора з виконання одного процесу на виконання іншого приймається диспетчером задач, а не активною задачою. При витісняючій багатозначності механізм диспетчеризації задач цілком зосереджений в операційній системі і програміст може писати своє прикладання, не турбуючись про те, як воно буде виконуватись правильно з іншими задачами. При цьому ос виконує слідуючі функції визначає момент зняття з виконання текучої задачі, зберігає її конте5кст у дескрипторі задач: вибирає у черги готових задач наступну і запускає її на виконання наперед завантажив її контекст.

Дисциплін RR і інші побудовані на її основі відносять до витісняючи.

При не витісняючій багатозначності механізм розподілу процесорно часу розподілений між системою і прикладними програмами.