Оперативни системи-теорија

1. Вовед

Компјутер-ЦПУ, главна меморија, диск, печатар, тастатура,глушец, екран, мрежна меѓуврска и други уреди за влез и излез. Компјутерите имаат соодветен софтвер – оперативен систем чијашто задача е да обезбеди подобар, поедноставен модел на компјутер и да се погрижи за управувањето со овие ресурси.

Хардверот се состои од чипови, плочи, дискови, тастатура, монитор и сл. Физички објекти. Над хардверот се наоѓа софтверот. Компјутерите имаат два режима на работење: јадрен и кориснички режим. Оперативниот систем енајосновниот софтвер и работи во јадрен(надзорен) режим т.е. има пристап до целиот хардвер и може да ги изврши сите функции на машината. Останатиот софтвер работи во кориснички режим. Најниското ниво на корсничкиот режим-Корисничката меѓуврска , школката(на основа на текст) или GUI (graphical user interface-на основа на икони) не е дел од оперативниот систем, туку го користи за извршување на задачите, т.е. му овозможува на корисникот да стартува др програми.

Се што работи во јадрен режим е дел од оперативниот систем, но и некои програми кои работат надвор од него може да се дел од него или да се тесно поврзани. Оперативните системи се состојат од пет милиони редови код и тоа само за делот што работи со јадрото, а корисничките програми како гуи, библиотеките и основниот апликативен софтвер имаат обем 10 до 20 пати поголем.

1.1. Оперативен систем

-да им обезбеди на програмерите на апликациите јасен сет на ресурси

-да управува со тие ресурси

Оперативните системи го претвараат лошиот хардвер во прекрасни апстракции.
Вистинските клиенти на оперативниот систем се апликативните програми.
Задачата на оперативниот систем е да обезбеди уредна и контролирана алокација на ЦПУ, мемориите и влезно-излезните уреди помеѓу различните програми што се натпреварuваат помеѓу нив. Неговат главна задача е да следи кои програми и кои ресурси ги користат, да одобрува барања за користење ресурси, да води сметка за користењето и да управува со спротивставени барања од различни програми и корисници.
Управувањето со ресурси вклучува мултиплексирање(споделување) на ресурсите на два начина: во простор и во време. Во време- различните програми или корисници го користат еден по друг (пр. ЦПУ, печатар). Во простор- секој добива дел од ресурсот (меморија, тврд диск).

1.2 Процеси

Клучен концепт во сите ос е процесот. Со секој процес е поврзан неговиот адресен простор, листа на мемориски локации од 0 до макс во кои процесот може да чита и пишува. Адресниот простор ја содржи извршната програма, податоците на програмата и нејзиниот стек. Со Секој процес се поврзани и сет ресурси, што обично вклучуваат регистри (вклучувајќи го програмскиот бројач и индикаторот на стекот), листа на отворени датотеки, предвидени аларми, листа на поврзани процеси и сите др инф за извршување на програмата.

1.3 Системски повици

Секој компјутер со една ЦПУ може да извршува само една инс во даден момент. Ако одреден процес извршува корисничка програма во корисничи режим и има потреба од системски сервис, како читање податоци од датотека, мора да изврши инструкција за преклопување за да ја пренесе контролата на ос. Потоа ос открива што бара процесот со проверка на параметрите и го извршува системскиот повик и ја враќа контролата на инструкцијата. Системски повик- аналогно како испраќање вид повик за постапка, со таа разлика што с.повици влегуваат во јадрото.

1.4 Зоолошка градина на ОС

-ОС за големи компјутери
-ОС за сервери
-Мултипроцесорски ОС
-ОС за персонални компј
-ОС за рачни компјутери
-Вградени ОС
-ОС за сензорски јазли
-ОС во реално време
-ОС за интелигентни картички

1.5. Структура на ОС

-Монолитни системи

ОС е програмиран како збир од постапки, поврзани во единствена извршна бинарна програма која работи во јадрен режим. Главна програма што ја повикува бараната сервисна постапка, збир од сервисни што ги извршуваат системските повици и збир од помошни што им помагаат на сервисните.

-Слоевити структури

Организирање на оперативниот систем во хиерархија од слоеви секој од нив надграден над оној под него. Првиот систем изграден на овој начин е THE системот. Кај МУЛТИКС внатрешните кругови се поприviлегирани од надворешните.

-Микројадра

Основната замисла е да се постигне висок степен на сигурност со делење на ОС во мали, добро дефинирани модули, од кои само еден – микројадрото работи во јадрен режим, а др. Работат како релативно немоќни кориснички процеси.
Sys-структура за јадрени повици. Драјверот за часовник исто така се наоѓа во јадрото затоа што распоредувачот има блиска интеракција со него.

-Виртуелни машини

Мултипрограмирање и проширена машина со покорисна меѓуврска отколку голиот хардвер. Идентични копии на голиот хардвер, не се продолжение на машините. Виртуелизација како начин сите услуги да работат на иста машина, без опасноста паѓањето на еден сервер да ги сруши останатите.
Хупервизор од тип 1 – монитор на виртуелна машина, можност за два или повеќе ОС за истовремено работење.
Хипервизор од тип 2 - работат како апликативни програми врз Windows, Linux или др ОС, познат како host OS – домаќин.

-Егзојадра

Наместо да се клонира цела машина, се дели така што секој корисник добива посет ресурси. Во долниот слој во јадрен режим се наоѓа програма наречена егзојадро. Негова задача е да ги распоредува ресурсите на виртуелните машини,а потоа да ги проверува обидите за користење, да не се обидува некоја ВМ да користи ресурси од друга ВМ. Предноста е што тука се заштедува еден слој за мапирање.

-Клиент сервер

Постоење процеси –сервиси и процеси-клиенти, комуникацијата се врши преку пренесување пораки. Се испраќаат барања и се враќаат одговори. Моделот клиент-сервер е апстракција што може да се користи на една машина или за мрежа од машини.

2. Процеси

2.1 Модел на секвенционални процеси

Целиот софтвер е организиран во неколку последователни процеси. Процесот е само инстанца на извршната програма, вклучувајќи ги и тековните вредности на програмскиот бројач, регистрите и променливите. Концепциск секој процес има сопствена ЦПУ (во реалноста, една ЦПУ се префрла од процес во процес –Мултипрограмирање).

Страна 92 – елементи во Process Control Block

2.2 Нишки

Во традиционалните ОС секој процес има свој адресен простор и една управувачка нишка. Тоа е и дефиниција за процесот. Често има ситуации со повеќе контролни нишки во ист адресен простор што работат во квазипаралелност, како да се различни процеси (освен заедничкиот адресен простор).

-Имплементација на нишки во кориснички простор

Пакетот нишки се ставе целосно во Кп и јадрото не знае за нив, тоа мисли дека работи со обични процеси со една нишка. Може да се примени на Ос што не подржува нишки. Нишките се извршуваат врз систем за програми во извршување којшто е збир процедури што управуваат со нишките. За секој процес е потребна оделна табела на нишки за да се следат сите нишки на тој процес. Системот за програми во извршување извршува нишки од својот процес додека јадрото не му ја земе ЦПУ.

-Имплементација на нишки во јадрото

Јадрото знае и управува со нишките. Јадрото има табела на нишки што ги следи сите во системот. Сите повици што можат да блокираат нишка се спроведуваат како системски повици, со многу поголема цена отколку повикот до постапка во системот за програми во извршување. Кога една нишка ќе блокира, јадрото по сопствено мислење распоредува др нишка. Се користи и рециклирарање на ншките за да се намали оптоварувањето на ресурсите. Предности-не се потребни ново неблокирачки системски повици.Недостатоци- цената на системскиот повик е значителна, ако има чести операции во врска со нишките ќе настане големо оптоварување.

-Хибридна примена

Да се користат нишки во јадрото, а потоа да се комбинира сигналот од повеќе нишки во корисничкиот простор во некои или во сите нишки во јадрото. Програмерот може да одреди колку нишки во јадрото ќе користи и од колку нишки од корисничко ниво ќе го користи сигналот.

3. IPC – Меѓупроцесна комуникација

Натпревари-два или повеќе процеси читаат или пишуваат заеднички податоци и конечниот резултат зависи од тоа кој процес кога ќе биде извршен.

-Инструкцијата ТСЛ

Ја вчитува содржината на меморискиот збор Lock(бравата) во регистарот RX, а потоа меморира вредност различна од нула во мемориската адреса. Ниту еден процесор не може да пристапи до меморискиот збор додека не заврши инструкцијата. ЦПУ што ја извршува ТСЛ ја заклучува мем. Магистрала за да го спречи пристапот на другите ЦПУ до меморијата додека не заврш. Кога бравата изнесува 0, секој процес може да ја смени во 1

со користење на ТСЛ, а потоа да прочита и запише во заедничката мемотија. Откако ќе заврши ја сетира бравата на 0, користејќи обична инструкција Move. Слика – првата инструкција ја копира старата вредност на бравата во регистарот и ја менува во 1. Потоа новата вредност се споредува со 0. Ако не изнесува 0, бравата е веќе наместена, програмата се враќа на почеток и се тестира. Кога ќе стане 0 (кога процесот што моментално се наоѓа во критичниот регион ќе излезе од него) и подрутината се враќа со наместена брава. Алтернативна инструкција на ТСЛ е XCHG што ги разменува содржините на две локации, регистар и збор од меморијата. Кодот е во суштина ист како кај решението со ТСЛ.

-Произведувач потрошувач

Може да настане натпревар затоа што пристапот до count не е ограничен. Може баферот да биде празен, променливата е 0 и потрошувачот само што ја прочитал, во тој момент распоредувачот одлучува да го извршува произведувачот. Тој става во баферот и променливата станува 1 и испраќа повик за будење кој се губи затоа што произведувачот е уште будене. Кога тој ќе почне да се извршува ќе види дека кај него промен е 0 и ќе заспие, кога баферот ќе се наполни ќе заспие и произведувачот. Двата процеса ќе спијат вечно. Брзо решение е да се постави бит за чекање за будењето, ако кога ќе се испрати повик за будење процесот е сеуште буден, се поставува овој бит, подоцна ако процесот проба да заспие а е поставен битот , процесот останува буден. Овој бит ја спасува ситуација, но проблемот сеуште постои.

-Семафори

Користење на цела варијабла што ќе го смета бројот на повици за будење за да се користи подоцна. Беше воведена нов вид променлива наречена семафор. Семафорот може да има вредност 0, што посочува дека нема меморирани повици за будење или некоја позитивна вредност доколку се меморирани еден или повеќе повици. Операцијата down на семафорот проверува дали вредноста е поголема од 0, ако е ја намалува(искористува мемориски повик) и продолжува. Ако е 0, процесот заспива без да ја изврши down во моментот. Проверувањето, менувањето на вредноста и по потреба заспвање се прават во една атомска-неделива постапка , што овозможува неден друг процес во тој момент да пристапи до семафорот – решение за синхронизација. Операцијата up ја зголемува вредноста на засегнатиот семафор, ако на тој семафор спијат повеќе процеси и не можат да ја извршат претходната down, системот избира еден од нив и му овозможува да ја изврши down. Така семафорот пак ќе изнесува 0, но на него ќе спие еден процес помалку. ОС накратко ги исклучува прекините додека го тестира семафорот, а воедно не се нанесува штета. Ако се користат повеќе ЦПУ се користи брава за да се обездеди само една ЦПУ во даден момент да го проверува семафорот.

-Семафори кај произведувач-потрошувач

Три семафори-еден-фулл, за број на зафатени места, емпту-за следење на бр на празни места и мутекс што обездедува потрошувачот и произведувачот да не пристапат до баферот истовремено. Фулл е – на почетокот, емпти е бр на места во баферот (се користат за да гарантираат дека нема да се случат низа настани), а мутексот е 1(се користи за заемно исклучување). Семафорите што се иницијализираат на 1 и се користат од два или повеќе процеси за да се обезбеди само еден од нив да може во даден момент да влезе во критичниот регион се нарекуваат **бинарни** семафори.

-Монитори

Wait , signal се слични на sleep, wakeup со таа разлика што автоматското заемно исклучување на постапките на мониторот гарантира дека ако, да речеме, произведувачот открие дека баферот е полн, ќе може да ја разврши wait без да се грижи распоредувачот да не се префрли на потрошувачот непосредно ред да заврши wait, тој воопшто нема да биде пуштен во мониторот.

Изгладнување-сите програми се извршуваат бесконечно, но не прават никаков напредок.

4.Распоредување

Покрај одбирањето на вистинскиот процес на извршување, распоредувачот треба да се покрижи и за ефикасно искористување на цПУ, бидејќи префрлањето меѓу процеси е скапо. В/И е кога процес ќе влезе во блокирана состојба чекајќи надворешен уред да заврши со работа. Слика – првите поголем дел од времето го минуваат во пресметки-зависни од пресметувањето, а другите- зависни од В/И. Клучен фактор е времетраењето на работата на ЦПУ. Како што ЦПУ стануваат побрзи, процесите се се позависни од В/И.

-Алгоритми за распоредување

Серверите спаѓаат во околина на интерактивни корисници, бидејќи обично опслужуваат повеќе корисници, а сите тие се брзаат. + Страна 150.
Не преемптивни алгоритми: FIFO, SJF … Преемптивен е SRTN.
Round Robin – Утврдување премногу краток квантум ја намалува ефикасноста на ЦПУ, но предолгиот може да предизвика бавна реакција на кратки интерактивни барања, затоа квантум со должина од 20-50 милисекунди е разумен компромис.
Повеќекратни редови на чекање- најприоритетната се извршува еден квантум, следната два итн. Секогаш кога еден процес ќе ги искористи сите квантуми се преместува една класа подолу.
Застарување-техника на проценување на следната вредност во серија со земање на просекот од тековните измерени вред и претходните претпоставки.

-Распоредување кај нишките во кориснички простор

Јадрото не знае за постоење на нишките и збира процес и му дава контрола врз неговиот квантум. Кај секој процес распоредувачот на нишки одлучува која нишка ќе се изврши и бидејќи нема преки на часовник може да се извршува колку долго сака. Ако го искористи целиот квантум јадрото ќе избере друг процес, а кога ќе се врати пак на тј процес оваа нишка ќе продолжи со изврпување. Во практика најчести се кружното и распоредување со процеси.

-Нишки на ниво на јадро

Тука јадрото ја одбира нишката која ќе се изврши и притоа не мора да земе предвид на кој процес припаѓа. Нишката добива квантум и ќе биде прекината ако го надмине квантумот.

Главната разлика кај нишките на корисничко ниво и во јадрото е во перформанците. За префрлање меѓу нишки во КН потребни се само неколку машински инструкции , а во јадрото целосно менување на контекстот. ОД друга страна, со нишки на ниво на јадро, една нишка не го блокира цел процес, како кај КН.

5. Linux, Windows – процеси

5.1 Linux

Линукс системот може да се гледа како еден вид пирамида, на дното е хардверот, составен од ЦПУ, меморија, дисковите , монитор, тастатура и др. Оперативниот систем работи на гол хардвер, неговата функција е да го контролира хардверот и да им обездеди интерфејс за системски повици(им дозволуваат на корисничките програми да создаваат и управуваат со процеси, датотеки и др ресурси) на сите програми – 3тиот слој Стандардна библиотека(read,close,open,write) со по една процедура за секој системски повик. Интерфејсот кон библиотеката го обездедува POSIX. Како дополнение на ОС и библиотеката, сите верзии на ЛИНУКС нудат голем број стандардни програми(школка, уредувачи, компилатори итн – 4ти слој). Токму овие програми ги повикува корисникот-5тиот слој-преку тастатурата-интерфејс создаден од некој пакет на услужни програми – кориснички интерфејс.

-Структура на јадрото

Јадрото лежи директно врз хардверот и овозможува директна комуникација на влезно-излезните уреди и единицата за управување со меморијата, но и го контролира пристапот што го ма централната процесорска единица во истите. На најниско ниво се наоѓаат обработувачите на прекините, а коишто се основнио начин за меѓусебна комуникација со уредите и со најниските механизми за отстранување. Разните потсистеми во јадрото ќе ги поделиме во три компоненти: Влезно-излезна, Меморика и процесна. Влезно-излезната ги содржи сите делови најадрото одговорни за меѓусебна комуникација

со уредите и спроведување операции за вмрежување и зачувување податоци. На највисоко ниво сите влезно-излезни операции се вклопени во сојот на виртуелниот систем на датотеки. На најниско ниво сите в-и операции поминуваат низ извесен драјвер на уредот. Сите драјвери се класифицираат или како драјвери за знаковен уред или како драјвери за блок-уредите, драјвери за мрежни уреди. Кога станува збор за нивото над драјверите, кодот на јадрото е различен за различни типови уреди. Знаковните уреди можат да се користат на два начини, да се внесуваат знаците како што се пишуваат или преку таканаречена линиска дисциплина. Над нив се наоѓаат терминалите. Слојот над мрежните драјвери е одговорен за насочување и обезбедува дека извесен пакет ќе биде упатен кон вистинскиот уред (протоколи). Над протоколите се наоѓа интерфејсот со приклучоците. Над драјверите за дискот е в-и закажувач одговорен за наредување и издавање барања за операции. Над колоната на блок уредот се наоѓаат системите на датотеките а тука е и генерирачкиот слој на блок-уредот што обезбедува апстракции за системите на датотеки. Задачите за управување со меморијат подразбираат одржување на мапирањеро на виртуелната во физичката меморија, одржување на кеш меморијата и добра политика за замена на страници. Одговорноста на компонентата за управување со процеси е создавање и прекинување процеси, тука е и закажувачот кој одредува кој процес,нишка следува и тука е и кодот за управување со сигналот. На крајот на самиот врв се наоѓа интерфејсот за системски повик на јадрото.

-Создавање процес

Системскиот повик за раздвојување- форк – создрава идентична копија на оригиналниот процес – родителски процес создрава процес-дете. Р и Д имаат свои засебни одрази во меморијат, но делат отворени датотеки. Процесите се нарекуваат според нивните ПИД. При создавање процес родителот го добива идентификаторот на процес на детето. А на детето повратна вредност е 0. За детето-процес се создаца нов дескриптор на датотека и корисничка објаст кои се полнат со информации од родителот. На детето му се доделува ПИД , се поставува мемотиска мапа и има заеднички пристап до датотеките на неговиот родител. Потоа се поставуваат неговите регстри и е спремен за работа. Кодот кој што работи кај детото процес прави СП exec и го поставува името на наредбата како параметар. Сега јадрото ја бара и верификува датотеката, ги копира аргументите и низите на околината на јадрото и го ослободува стариот адресен простор. Следно е создавање нов адресен простор и негово пополнување, сигналите се поставуваат повторно и регистрите се поставуваат на 0.

-Системски повици за управување со процеси во Линукс - Страна 745.

-Имплементација на процеси и нишки

Јадрото на Линукс ги претставува процесите и нишките како задачи и ги препознава преку ПИД. Јадрото ги организира сите процеси во двојно поврзана листа од структури на задачи.

-Нишки кај Линукс

Кога се создаваше нова нишка, првобитната нишка и новата делеа се освен регистрите. Конкртно, дескриптори на отворените датотеки, управувачи со сигнали, аларми и останати глобални својства се однесуваа на роцес, не по нишка. Clone овозможи секое од овие својства да се однесува и на процес и на нишка. Повикот создава нова нишка во тековниот процес или во некој нов процес во зависност од sharing flags-бит мапа што дозволува порафиниран начин на заедничко делење. Новата нишка почнува со извршување на Function, што се повикува со arg како единствен параметар и добива свој приватен магацин, а покажувачот кон него се иницијализира со stack ptr.

-Распоредување во Линукс

Клучната структура на податоци закажувачот е runqueue. Оваа структура се порзува со ЦПУ и одржува две низи активна и истечена, секое од овие полиња е покажувач кон низа од 140 глави од листата, секоја со соодветен приоритет. Главата од листата покажува кон двојно поврзана листа на процеси за даден приоритет. Закажувачот избира задача од активната низа со највисок приоритет.Доколку квантата истече се префрла во листа на истечени, со др ниво на приоритет.Доколку се блокира, пред да истече квантата, штом се случи очекуваниот настан и може да продолжи да се извршува се враќа во првобитната активна низа,а временската порција се намалува.Кога ниту една од активните низи не содржи задачи, тогаш истечената станува активна и обратно. => Нема прегладнување.

-Подигнување на Линукс

Кога ком започнува со работа , основниот влезно излезен систем БИОС ја извршува операцијата ПОСТ, како и првична проверка на уредите и нивната искористеност. Потоа, првиот сектор од дискот за подигање на системот, МБР се вчитува на фиксна локација во меморијата и се извршува. Овој сектор содржи мала 512 битна програма којашто вчитува програма боот, која се копира на фиксна адреса за да ослободи мем за ОС, па го вчитува осноиот директориум и ситемот за подигање и накрај го вчитува јадрото на ОС и влегува во него. Сега се поставува јадрениот магацин, се препознава ЦПУ, пресметка на РаМ меморија, онеспособување на прекините и повикување на маин за почеток на основниот дел на ОС. Потоа следува доделување на податочните структури и системот може да започне со автоконфигураиција. По конфигурацијата на хардверот, следува внимателна изработка на процесот 0, поставувае на магацин и актиивање, тој продолжува со иницијализација-програмирање на часовникот во реално време , создавање на инит и демонот на страницата. Инит ги проверува своите знамиња за да види дали има еден или повеќе корисници, за еден кориснил издвојува процес за извршување на школката, за повеќе ја извршува програмата гетту-ја поставува брзината на линијата и останатите својства и се обидува да го вчита корсничкото име од екранот , а потоа бара да се внесе и лозинката и го споредува со зачуваните лозинки во датотеката /етц/пассњд/. Доколку е + се заменува со школката, во спротивно пак се појавува логин. На сликата на терминал 2

има успешно најавување и школката има издвоено дете процес којшто треба да ја изврши цп.

5.2. Windows

-Структура на ОС

Централниот дел на ОС е НТОС јадрото и има 2 слоја, извршувач кој содржи најмногу услуги и потеник слој кој е наречен јадро и и го имплементира распоредот на нишките и апстракции на синхронизации и имплементира ракувачи со трап, нарушувања и други аспекти. Јадрениот слој на НТОС е над извршниот модул. Под НТОС извршните слоеви постои софтвер –хардверски слој за апстракција кој апстрахира хардверски делови на нискко ниво. Други главни компоненти на јадрениот слој се и двигателите на уредите, системи на датотеки и стекови на мреѓни протоколи итн. Највисокиот слој е системската библиотека која се извршува во кориснички модул.

-Процеси во Виндоус

Процесите се објекти за програми, го чуваат виртуелниот простор, рачките кои упатуваат на предмети од јадренот модул и нишките. Тие ги чуваат ресурсите кои се користат за извршување на нишките. Секој процес има процес блок на средина. Нишките се јадрена апстракција за распоред на компјутерот, нивните приоритети се задаваат врз однос на приоритетот на процесот. Секоја нишка има 2 оделн стекови на повици еден за кориснички и еден за јадрен модул. Исто така постои блок за нипка на средина кој ги чува податоците во корисничкиот модул кои се специфични за секоја нишка. Постои и др структура која јадрениот модул ја дели со секој процес –заемни податоци за корисници-содржи број на вредности кои се одржуваат во јадрото. Кога се создава процес – добива рачка за процесот кој му овозможува да го промени новиот процес со мапирање делови, распоредување ВМ, запишување параметри на средина и создавање нишки. Виндоус може да ги групира процесите во работи, за да се применат ограничувањата на нишките ко ги содржат. Еден процес може да биде најмногу во една работа. Нишките не содржат влакна. Влакната се создаваат со распоредување на стек и структура на податоци на влакна на кориснички модул за складирање регистри и податоци поврзани со влакно. Нишките се претвораат во влакна, но влакната можат да се создадат независно. Префрлањето меѓу влакна е прилично посниско од префр нишки. Во виндоус ОС избира нишка за извршување па затоа нишките имаат состојба и имаат ИД.

-Синхронизација

Сите видови на синхронизација работат на нишки. Семафорите се создаваат со КреатеСемафоре, постојат повици уп и доњн со чудни имиња. Мутексите се поедноставни од семафорите,не користат бројачи. Всушност тие се брави за заклучување и отклучување. Критичните делови се слични на мутексите , освен локалните на адресеиот простор на создавачката нишка, бидејќи не се предмети на јадрениот модул немаат експлицитни

покажувачи и не се пренесуваат меѓу процесите. Постојат два вида на настани: настани за известување и настани за синхронизаццција. Може да биде во сигнализирана и несигнализирана состојба.

-Приоритети

Системот има 32 приоритети, од 0 до 31. Комбинациите на приоритетните класи се мапираат на 32 апсолутни приоритетни нишки. За да се користат приоритетите , системот има низа од 32 листи на нишки кои соодветствуваат на приоритети од 0 до 31. Штом ќе се најде полна листа првата нишка се извршува за еден квантум. Има 4 категории приоритети – реален, кориснички, нула и мрзелив(-1). Приоритетите од 16-31 се реални.

-Приоритетна инверзија

Две нишки работат на проблем заедно на релација производувач-корисник. Производувачот има приоритет 12, корисникот-4. Поризведувачот наполнил заемен бафер и блокира на семафор. Пред да добие корисникот шанса, неповрза нишка со приоритет 8 станува подготвена и започнува со извршување. Производувачот нема да се извршува повторно се додека нишката со приоритет 8 не се откаже. Овој проблем се решава со хакирање. Системот следи колку долго поминало од последната подготвена нишка. Ако поминало некоја граница и дава приоритет 15 за два кванта. Ова е шанса за да се деблокира производувачот, по завршувањето на два кванта зголемувањето се прекинува. Подобро решение е да се казнуваат нишките што го искористуваат квантумот со намалување на приоритетот. Проблемот не бил предизвикан од гладна туку од лакома нишка.

6. Управување со меморија

Префрлање-подигнување на секој процес во целост, негово извршување зададено време и потоа негово враќање на дискот.Кога се префрлаат процесите кои динамички растат, се префрла само меморијата а не и дополнителната. Ако има два растечки сегменти им се доделува заедничка дополнителна меморија.

-Виртуелна меморија

Виртуелни адреси-адреси генерирани од програми,го формираат виртуелниот адресен простор. Кога се користи ВМ , ВА не одат директно на мемориската магистрала, туку одат на ММУ – мемориска едница за управување која ги мапира ВА на физичката М.

-Структура на влез на странична табела

Најважно поле е Број на странична рамка, т.е. целта на мапирањето е да се најде оваа вредност. До него се наоѓа битот Присутно/Отсутно и ако овој бит е 1 влезот е валиден и може да се користи, ако е 0-грешка на страницата. Полето Заштита ни кажува какви видови на простапи ни се дозволени. Пр. 0 за читање и запишување и 1само за читање, но може да се користат и 3 бита за читање, запишување и подигнување на стр. Битовите Променето и Повикано ја забележуваат употребата на страницата. Кога се

запишува страница хардверот го приспособува битот Променето. Ако стр во него е модифицирана мора повторно да се запише на дискот, во спротивно може да се напушти, затоа што копијата на дискот е валидна. Повикано се повикува секогаш кога се повкува страница за читање или запишување. Неговата улога е да му помогне на ОС при исфрлање на страница при појава на грешка. Последниот бит дозволува кеширањето да биде исклучено. Овој бит е важен за стр кои се мапираат на уредина регистрите. Машините кои имаат оделен В/И простор и не користат мапирана меморија со В/И немаат потреба од овој бит.

-ТЛБ (мал уред за мапирање на физ адреси без да се оди низ страничната табела)

Се состои од мал бр влезови, секој влез содржи инф за една страница, вклучувајќи го брјот на виртуелната стр, бит кој е приспособен кога се менува страницата, заштитен код и физичката стр рамка каде е сместена страницата, бит кој покажува дали е валиден или не.

-Двонивовски табели на страници

Целта е да се избегне чување на сите странични табели во меморијата цело време. Имаме табела со 1024 влезови што соодветствува на 10-битно ПТ1 поле. Најпрво се извлекува ова поле и ја користи оваа вредност како индекс во врвната странична табела. Секое од овие 1024 влеза претставува 4мб. Влезот 0 на врвната странична табела насочува кон табелата за програмски текст, 1 податоци и 1023 стек, другото не се користи. ПТ2 се користи како индекс во избраната странична табела од второ ниво за да се најде бројот на страничната рамка за самата страница. Значи за милион страници потребни се само 4 табели.

-Инвертирана странична табела

Се користи ТЛБ-ги содржи најчесто користените стр. Ако има промашување во ТЛБ, променетата стр е пребарувана од софтверот. Еден начин е да се има hash table hashed на виртуелната адреса. Сите виртуелни адреси во мем со иста hash вредност се поврзани заедно. Кога ќе се најде бр на стр рамка,новиот (вирт. , физички)пар се внесува во ТЛБ. слајд 60- маленко т-мом виртуелно време – време на последна употреба. Ако има повеќе со p=0 се отстранува таа со најголема возраст, најмала вредност на последно користено време, а во најлош случај сите се 1, се отстранува по случаен избор.

6.1 Управување со меморија 2

Едно решение е да се имаат 2 оделни простори, 1 за инструкции и податоци, наречени И-простор и Д-простор, за секој посебно. Секој оди од 0 до некој максимум 2^16-1 или 2^32-1. Секој си има своја странична табела, со свое мапирање на виртуелни во физ адреси.

-Заемни страници-Деливи

Се делат само они страници кои се читаат, меѓутоа страниците со податоци не се делат. Ако одделните И и Д простори се подржуват, делењето програми е едноставно ,

т.ш користат иста странична табела за И простор и различни за Д просторот. Секој процес има два посочувачи во неговата табела на процес.

-Заемни библиотеки

Кога се вчитува, се вчитува стр по стр, за да не биде целата во меморијата,т.е. функциите кои не се повикани да не бидат во рам. Заемните библиотеки се вчитваат кога програмата се вчитува или кога се повикуваат функциите прв пат. Ако друга програма ја користи, тогаш нема потреба да се вчитува повторно. Тие ги смалуваат датотеките кои се извршуваат и со тоа зачуваат простор во меморијата исто така ако се ажурира некоја функција да се поправи грешка, нема потреба да се компајлираат програмите кои се повикуваат . Има мал проблем, ако библиотеката се наоѓа на различна адреса за секој процес, тогаш решението еда се компилираат заемните библ со специјално компајлерско знаме кое му кажува да не произведува компајлерот кои било инс кои користат апсл адреси, туку се користат само релативни адреси.

-Трап инструкција

По донесувањето на потребната инс , ком мора да ја рестартира инс и пак предизвикува трап. За да се рестартира инс ос мора да определи каде е првиот бајт. Вредноста на програмскиот бројач зависи кој операнд згрешил и невозможно е за ос да знае каде почнала инс. Решението е во форма на скриен внатрешен регистар во кој се копира програмскиот бројач пред да се изврши секоја инструкција.

-Чување на стр на диск

а) Статичен дел- делот за префрлање на дискот е исто толку голем како и процесираниот виртуелен простор, кај секоја страница има статично место на кое се запишува кога се отстранува од мем, Стр постојано се складираат на него по редослед на нивниот бр на виртуелна страница. Стр која е во меморија секогаш има своја сенка копија на дискот, но оваа копија можеби е стара ако стр била променета од моментот кога била вчитана, тогаш се користи копијата на дискот.

б) Немаат статични адреси кога се отстранува стр се избира празна стр на дскот, а мапата на дискот се ажурира соодветно. Стр во мем нема копија на дискот. Влезовите на мапата во дискот содржат неточна адреса на дискот или бит кој означува дека не се во употреба.

6.3 Сегментација

Машина со целосно независни адресни простори, наречени сегменти. Секој сегмент се состои од линеарна низа од адреси од 0 до некој максимум. Различни сегменти имаат можат да имаат различни должина и можат да се менуваат за време на извршување, секој сегмент создава свој адресен простор. Сегментот е логичка целост и програмерот ја користи како таква. Сегментацијата овозможува делење процедури и податоци меѓу процесите. Распоред по шаховска табла-меморијата се дели на делови од кои некои се

сегменти а некои дупки, решение : збиеноста.
-Сегментација со страничење – Интел Пентиум

Пентиум има 16КБ независни сегменти, од кои секој има по 1 милјарда 32-битни зборови. Срцето на виртуелната меморија се состои од 2 табели – ЛТД и ГДТ глобална табела опишувач , секоја програма си има свој ЛТД но и еден ГТД кој се дели од сите програми на ком и опишува сегменти на системот. За да се пристапи сегмент Пентиум вчитува изборник за тој сегмент во еден од 6те регистри. Секој изборник е 16битен број, еден од битовите ни кажува дали е локален или глобален, а другите тринаесет ни специфицираат ЛТД или ГТД број на влез, па овие табели се ограничени на 8кб на опишувачи на сегмент. Другите 2 бита се за заштита. Ако сегментот е во мем растојанието му е во опсегот, па Пентиум додава Басе поле во опишувачот за да формира таканаречена линеарна адреса и се дели на 3 дела. Ако страничењето е оневозможено линеарната адреса се интерпретира како физичка и се испраќа до мем, а ако е возможно , лин се објаснува како виртуелна и се мапира во физ со користење странични табели. Секоја програма која се извршува има директориум на стр која се состои од 1024, 32-битни влезови на бит, тој е сместен на адреса која посочува на глобален регистер. Исто така посочува на странична табела, чии влезови посочуваат на странични рамки.

6.4 Линукс-управување со меморија

Системски повици – брк ја наведува големината на податочниот сегмент со задавање на адресата на првиот последователен бајтм ако вредноста е поголема од старата се зголемува.прот- за читање за запишување или комбинација. Флагс-дали е приватна или споделена. Фд да се отвори датотеката за да може да се мапира, оффсет со кој дел да се почне да се мапира.
Линукс користи шема на страничење на 4 нивоа. Виртуелната адреса има 5 полиња и тие се директориуми кои се користат како индекс во соодветниот директориум со страници во коишто постои посебен директориум за секој процес. Најдената вредност е покажувач кон директориум на повисоко ниво.
-Другарски алгоритам

Мем се состои од еднствен граничен дел. Потоа се дели на 2 дела итн.

6.5 Виндоус

Меморискиот управувач создава виртуелен адресен простор ВАД за процес, кој ги листа мапираните адреси. ВАД се организира во дрво на балансирање, за да може опишувачот ефикасно да се најде. Неискористените делови не користат ресурси па тие се слободни.
-Кога се отстранува стр таа оди на дното на листата на чекање или на листата за промена во зависност од чистината. Овие се уште валидни, па ако се јави грешка и нее потребна се враќа во работната група без В/И. Кога процес завршува, валидните стр во неговата

табела одат во слободната листа. Мапираниот и променетиот запишувач проверуваат дали има доволно чисти страни,ако нема ги земаат од врвот на променетата листа и ги запишуваат на дискот и ги преместуваат во листата на чекање. Ако некој процес одмапира страница таа оди на слободната листа, освен во случај кога е заемна. Кога стр грешка бара стр рамка да ја држи стр да се прочита зема од слободната листа. ЗероПаге се извршува на најнизок приоритет и ги брише стр од слободната листа и ги став а на нулта станица.

7. Датотеки

-Структура на датотека

а)Низа на бајти-неструктуирана, ОС не се грижи што има во датотеката, се што гледа тој е бајти. Секое значење мора да се присили на ниво на корисник.

б) За корисниците кои сакаат да прават невообичаени нешта, важна е структурата од записи. Датотеката е низа на податоци со статична дол, секоја со внатрешна структура.

в) Третиот вид структура е датотека која се состои од дрво со податоци, кои не мора да се со иста должина, но секоја содржи клучно поле во статична позиција во податоците. Дрвото е сортирано во клучно поле за да обезбеди брзо пребарување на одреден клуч.

-Начин на справување со имиња со променлива должина е да се направат сите податоци на директориумот со статична должина и чување на имињата заедно во куп.

-Првиот начин е секој податок од директориумот содржи статичен дел кој почнува со должина на податокот, а потоа следуваат податоци со статичен формат, по ова следува вистинското име на датотекат, секое име се прекинува со специјален карактер, вообичаено 0 и е дополнето до интегрален бр на зборови.

-Деливи датотеки- блоковите на дискот не се листаат во директориумите,туку во структура со малку податоци поврзана со самата датотека. Директориумите посочиуваат кон неа-индекс јазол. Или се користи таканаречено симболично поврзување.

-Дневник-операциите со лог мора да бидат импотентни т.е. да можат да се повторуваат колку што е потребно без да се направи штета, атомска трансакција-да се завршат сите операции во заграда или ниедна од нив.

-Виртуелни системи датотеки

Се користи концептот на ВФС-виртуелен систем на датотеки. Сите повици на системот кои се поврзуваат се пренасочуваат кон виртуелни системи на датотеки за почетно процесирање. Овие повици кои доаѓаат од процесите на корисникот се стандардно ПОСИХ повици. Оттука ВФС има горно поврзување до процесите на корисникот и тоа е познато ПОСИХ поврзување. ВФС исто така има долно поврзување до вистински системи на датотеки.

-Управување со простор – огромна големина на блок- секоја датотека зафаќа цел цилиндар – малите датотеки трошат дел од просторот на дискот. Мала големина на блок-повеќето дат ќе се рашират на повеќе блокови – трошиме време.

-Квоти на диск

Администраторот на системот доделува максимум простор на датотеки и блокови на еден корисник, а ОС спречува корисниците да ги надминат нивните квоти. Кога се отвара датотека, атрибутите се сместени во табела која кажува кој е сопственикот. Големината на датотеката ќе се додели на квотата на сопственикот и се внесува посочувач кон неа. Кога се додава блок, се зголемува бр на сопственикот. Втората табела содржи податоци за квотата – мека и цврста граница, бр на предупредувања (ако се надмине било која граница).

-Физичко складирање

Почнува со блок 0 на дискот, ги запишува сите блокови на дискот на излезната касета по ред и запира кога ќе го копира последниот. Контролорот се справува со лош блок без да знае Ос за тоа. Може да се избегне складирањето на неискористени блокови.

-Логичко складирање

Почнува во еден или повеќе специфични директориуми и складира постојано таму, сите дат и директориуми кои се промениле од некој датум. Касетата добива серии внимателно идентификувани директориуми и датотеки. Алгоритмот за складирање одржува битмап, индексиран од индекс-јазол со неколку битови на индекс-јазолот. Битовите ќе бидат уредени и исчистени во оваа мапа. Фаза 1 – од почетниот директориум, за секоја променета датотека , нејзиниот индекс јазол е означен во битмапата и секој директориум е означен без разлика дали е променет. Фаза 2 ги враќа директориумите кои немаат променети датотеки или директориуми под нив. Фаза 3 скенирање на индекс јазли по нумерички редослед и складирање на сите директориуми кои се означени за складирање. Секој директориум има префикс од атрибутите на директориумот за да можат да се вратат. Фаза 4 складирање од префикс на атрибутите.

-За да се справат со проблемот на непостојани системи на датотеки, повеќето компјутери имаат услужна програма која ја проверува постојаноста на системот датотеки, unix-fsck, windows-scandisk. Се извршува секогаш кога се стартува системот, особено по паѓањето. Две табели, секоја содржи бројач за секој блок наместен на 0. Почнувајќи од индекс јазолот , можно е да се направи листа на сите броеви на блокови кои се користат во соодветната листа.

-Кеширање

Највообичаена техника која се користи за намалување на времето на пристап. Означува збир на блокови кои логички припаѓаат на дискот, но се чуваат во мем за подобри перформанси. Се хешира редот и адресата на дискот и да се пребарува во хеш табелата. Слична е на страничење со таа разлика што повикувањата во кеш мем се ретки па може да се чуваат сите блокови в ЛРУ ред со поврзани листи.

-Намалување на движењето на магнетната игла

со ставање на блокови кои ќе бидат простапени во низа близу еден до друг во истиот

цилиндар – за зголемување на перформансите. Препрека на перформансите на дискот кои користат индекс јазли е 2та пристапи кон индекс јазлите и кон блокот. Вообичаеното сместување на индекс јазлите се близу до почетокот на дискот па просечната одалеченост од и-ноде до блокот ке биде половина од бр на цилиндарот, при што ќе бара долги побарување. Друга идеја е да се подели дискот на групи цилиндри секој со свои јазли, блокови и слободна листа. Може да биде избран било кој јазол и се проверува дали има слободен блок во истата група цилиндири ако не се бара во соседниот.

7.1 МФТ – систем Датотеки

-За големи директориуми, наместо линеарно листање се користи B+ дрвото за да се направи азбучно баање и да се олесни вметнување на нови имиња во директориумот на вистинското место.

8. В/И уреди

Компјутерот треба да ги адресира контролорите на уредот за да размени податоци со нив. Користи таканаречен ДМА – директен пистап до меморија. ОС може да користи ДМА само ако хардверот има ДМА контролор. Дма контролорот има пристап до системската магистрала независно од ком. Содржи неолку регистри кои може да се читаат и запишуваат, на мем адреса, за броење бајти и контролни регистри. Прво се програмира ДМА со приспособување на неговите регистри за да знае каде да се премести, се започнува трансерот со задавање барање за читање на магистралата. Чекор 3-запишување на меморијата. Чекор 4 праќа сигнал за завршетокот на дма контролорот.

-Прекини

Кога В/И ја завршил работата предизвикува нарушување и го прави тоа со праќање на сигнал до линијата на магистралата која била назначена. Сигналот се детектира од контролниот чип на родител табелата и потоа одлучува шти да прави. Ако нема други нарушувања контролорот веднаш го процесира.

-Програмиран В/И

Комп ја врши целата работа.Програмата, т.е. низата се склопува во бафер во просоторт на корисникот. Корисникот го добива уредот за користење со правење системски повик за отварањее, системски повик за извршување, се копира низата во јадрениот простор, се печати карактер по карактер.Уредот има статус регистар, дали е достапен или не. На крај контролата се враќа на корисникот. ПО покажувањето на карактерот се проверува дали е зафатен или не – зафатено чекање. Недостаток-ком е зафатен додека В/И не заврши работа.

-Прекински воден

Ги печати карактерите секој как шо престигнува. Кога се печати еден карактер, ком го повикува закажувачот и се извршува друг процес. Процесот е блокиран се додека не се

испечати низата. Кога печатарот ќе биде спремен испраќа нарушување кое го прекинува мом процес и ја зачувува мом состојба.
-Б/И со ДМА

Проблем е што нарушувањето се јавува на секој карактер, решение ДМА. Да се допушти на ДМА контролорот да му префрла карактери на печатарот еден по еден. ДМА е програмиран И/О само што дма контролорот ја завршува раб на ком. Подобра во повеќе случаи – наместо нар за секој карактер имаме нар за секој бафер.
-Баферирање

а) не бафериран влез – процесот прави повик на читање и блокирање на чекање за еден карактер. Секој дојдовен карактер-нарушување, го предава карактероти и се деблокира, чита друг карактер и пак се блокиран. Проблем-процесот треба да почне за секој карактер.

б)баферирање во кориснички простор – процесот на корисникот обезбедува н бафери на карактер во просторот на корисникот и прав и читање на н карактери. Процедурата ги става карак додека баферот не се пополни, потоа го буди процсот. Недостаток-баферот може да се заклучи во меморијата, намалување на слободните стр.

в)баферирање во јадрото – се создава бафер во јадрото и справувачот со нарушување таму ги става карактерите , кога баферот е полн , стр со баферот на корисникот се внесува внатре и баферот се копира во една операција. Проблем – додека се префрла баферот на корисникот во дискот, доаѓаат нови карактери кои нема каде да се сместат.

г) решение на претходниот проблем ти е 2 бафера во јадрото. Ако вториот се копира, првиот се користи за нови карактери – двојно баферирање.
=Проблеми

чекор1-јадрото го копира пакетото до јадрениот бафер и процесот продолжува
чекор2-кога се повикува двигателот, го копира пакетот до контролорот
чекор3-се копира и надвор од мрежата
чекор4-се копира и во јадренио бафер на примачот
чекор5-се копира во баферот на процесот кој го прима

9. Дискови

Магнетните дискови се организирани во цилиндри, од кои секој содржи толку ленти колку што се вертикално натрупани главите за читање и запишување. Лентите се поделени на сектори, при што нивниот број околу обемот на дискот кај дискетите изнесува од 8 до 32 и до неколку стотоци кај тврдите дискови. Бр на глави варира од 1 до 16.