آریا خلیق
شماره دانشجویی: ۹۵۲۴۰۱۴
تکلیف اول سیستم‌های عامل

۱)
زمانی که یک دیوایس نیاز به یک ورودی/خروجی دارد یک وقفه ایجاد می‌کند. CPU از این وقفه مطلع شده و وارد interrupt service routine می‌شود.
برای **ورودی** دیوایس ما زمانی CPU را مطلع می‌کندکه یک ورودی وجود دارد و دیوایس ما می‌خواهد خواندن از آن را شروع کند.
برای **خروجی** دیوایس ما زمانی وقفه ایجاد می‌کند که آماده قبول دیتای جدید است.

روند
- درایور دیوایس ورودی/خروجی یک درخواست I/O ایجاد می‌کند (CPU در این موقع مشغول کار خودش است).
- درایور دیوایس به کنترلر I/O دستگاهی که اینتراپت را شروع کرد را مشخص می‌کند.
- دیوایس ورودی خروجی به کنترلر I/O اطلاع می‌دهد که آماده دریافت ورودی، خروجی است یا اروری رخ داده است.
- CPU یک وقفه دریافت می‌کند و سپس interrupt handler routine را صدا و اجرا می‌کند.
- Interrupt handler علت و منشاء این وقفه را مشخص کرده و عملیات مربوط را انجام می‌دهد و سپس عملیات return from را انجام می‌دهد.
- CPU به برنامه‌ای که در حال اجرای آن بود باز می‌گردد.

روند پردازش وقفه در حالت کلی ز:
- دیوایس یک interrupt به پردازنده می‌دهد.
- پردازنده اجرای instruction را قبل از سریس دادن به دیوایسی که وقفه ایجاد کرده تکمیل می‌کند.
- پردازنده یک سیگنال acknowledged به دیوایس مربوطه می‌دهد.
- پردازنده حالت برنامه‌ای که در حال اجرای آن بوده را ذخیره می‌کند و کنترل به interrupt routine می‌دهد.

- پردازنده مکان شروع interrupt-handling routine را داخل PC لود می‌کند.
- پردازنـده بـه instruction cycle بعـدی می‌رود، چـون مکـان شـروع interrupt-handling routine داخل PC قرار دارد شروع به اجرای آن می‌کند.
- چـون بایـد اطلاعـات اجـرای برنامـه قبـل ذخـیره شـود معمـولاً interrupt-handler شروع به ذخیره آن داخل stack می‌کند.
- interrupt handler شروع به اجرای interrupt می‌کند.
- زمانی که به وقفه پاسخ داده شد و تکمیل شد، اطلاعـات رجیستر برنامـه قبلی کـه ذخیره شده بود بازیابی می‌شود.
- PSW و PC هم بازیابی می‌شوند و در cycle بعدی برنامـه قبلی بـه حـالت اجـرا بـاز می‌گردد.

۲)
۳ روش برای انتقال ورودی خروجی وجود دارد:
۱) programmed I/O
زمانی که پردازنده به یک دستور مرتبط با I/O می‌رسد به وسیله یک دستور عملیات مربوط به آن I/O را اجرا می‌کند(عملیات خواندن/نوشتن توسط پردازنده انجام می‌شود). در این حالت ماژول I/O بیت‌های مرتبط را در status register تغییر می‌دهد ولی دیگر کار دیگری برای اطلاع داده به پردازنده انجام نمی‌دهد. به همین دلیل پردازنده باید چک کند که چه موقع دستور I/O تمام می‌شود به همین منظور پردازنده دایم بیت status ماژول ورودی/خروجی را ک کرده تا زمانی که از طریق این بیت بفهمد عملیات تمام شده.
از معایب این روش کاهش بازدهی به دلیل چک کردن مداوم status بیت می‌باشد.

۲) interrupt-driven I/O
مانند حالت قبلی یک دستور به ماژول ورودی خروجی برای اجرای دستور می‌دهد ولی ماژول را رها کرده و به دنبال کار دیگری می‌رود (عملیات خواندن/نوشتن توسط پردازنده انجام می‌شود). ماژول I/O پس از انجام کار پردازنده را با خبر می‌کند سپس پردازنده اطلاعات و ... را که در ماژول ورودی خروجی وجود دارد منتقل می‌کند و سر کار قبلی خود باز می‌گردد.
از معایب این روش می‌توان گفت که مانند programmed I/O نیاز به درگیری مستقیم CPU با ماژول ورودی خروجی برای انتقال اطلاعات دارد.

۳)direct memory access
زمانی که پردازنده نیاز به خواندن یا نوشتن اطلاعات دارد یک دستور به دیوایس ورودی خروجی می‌دهد و اطلاعات زیر را می‌فرستد:

- خواندن یا نوشتن
- آدرس دیوایس ورودی/خروجی
- آدرس شروع حافظه برای خواندن یا نوشتن
- تعداد کلمات برای خواندن یا نوشتن

سپس دنبال کار دیگری می‌رود. از این به بعد کارها را ماژوی ورودی/خروجی انجام می‌دهد و بدون دخالت پردازنده اطلاعات را می‌خواند یا می‌نویسد. در زمان پایان کار یک interrupt به پردازنده برای اتمام کار می‌دهد و طی این روش فقط در شروع و پایان با پردازنده ارتباط داشت.

۳)

SISD:

به سیستمی گفته می‌شود که یک پردازنده تنها یک دستور را در یک زمان اجرا می‌کند.

SIMD:

در این سیستم فقط یک control unit داریم ولی بیش از یک processing unit وجود دارد. در این حالت دستورات توسط PU های مختلف اجرا و توسط یک CU کنترل می‌شود. CU سیگنال‌های کنترلی را به PUها می‌دهد و یک دستور روی دیتاهای مختلف اجرا می‌شود.

MISD:

این نوع سیستم‌ها برای محاسبه موازی زمانی که PUها باید عملیات مختلفی را روی یک دیتا انجام دهند استفاده می‌شود.

MIMD:

در این نوع سیستم‌ها هر PU می‌تواند دستور مختلفی را به صورت غیر هم‌زمان ب بقیه روی دیتاهای جدا انجام دهد.

این نوع کامپیوترها می‌توانند در memory poolها یا distributed systemها استفاده شوند.

۴)

user-accessible registers:

این نوع رجیسترها می‌توانند توسط دستورات ماشین خوانده یا نوشته شوند. این نوع رجیستر ها خود به دو نوع تقسیم می‌شوند:

- data registers: می‌توانند مقادیر عددی را داخل خود نگه دارند(int،float، کاراکتر و ...)
- address registers: آدرس‌ها را نگه می‌دارد.
- general purpose registers: این نوع رجیسترها هم می‌توانند دیتا و هم آدرس را در خود نگه دارند.
- status registers: دیتاهایی را نگه می‌دارند که مربوط به status های مختلف است مانند انواع دسترسی و ...
- floating point registers: برای نگه داشتن اعداد ممیزدار هستند و فقط برای این کار استفاده می‌شوند.
- Vector registers: برای عملیات برداری استفاده می‌شوند و در معماری SIMD کاربر دارند.

Internal registers:

این نوع رجیسترها به وسیله دستورات قابل دسترسی نیستند و برای پردازش‌ها و محاسبه‌های درونی استفاده می‌شوند.

- Instruction register: این رجیستر instructionای را که در حال اجرا است را در خود نگه می‌دارد.
- رجیسترهایی برای عملیات fetch از RAM: مجموعه‌ای از رجیسترها هیتند که روی چیپ‌های مختلف پیدازنده قرار دارند.

Architectural register:

رجیسترهایی که به وسیله نرم‌افزار قابل مشاهده هستند ولی لزوماً معادل حالت سخت‌افزاری خود نیستند (امکان تغییر نام وجود دارد).

۵)

ساختار Layered:

در این حالت سیستم‌عامل از لایه‌های مختلف تشکیل می‌شود و هر لایه یک ماشین مجازی بهبودیافته را به لایه بالایی ارایه می‌دهد. این کار بعث ماژولار شدن سیستم‌عامل می‌شود. تغییر در لایه‌های این نوع سیستم‌عامل ها با تغییر درونی لایه و حفظ اینترفیس قبلی ممکن است.

تقسیم‌بندی در ۶ لایه:

Layer 5: Job Managers
Layer 4: Device Managers
Layer 3: Console Manager
Layer 2: Page Manager
Layer 1: Kernel
Layer 0: Hardware

ساختار Monolithic:

در این معماری عملیات سیستم عامل به وسیله function call ها که توسط کرنل انجام می‌شود. کرنل یک برنامه بزرگ است که در صورت نیاز درایورها هم روی آن اجرا می‌شوند و جزیی از آن به حساب می‌آیند.

مثال: Linux, Unix systems

معایب: برای نگه‌داری و درک سخت هستند.

ساختار Client-Server یا Microkernel:

ایده این است که سیستم‌عامل به چند process تقسیم شود که هر بخش یک کار واحد را انجام می‌دهد مثل I/O server, memory server. هر server در حالت user mode اجرا می‌شود و برای Client یک سرویسی را ارایه می‌دهد. هر Client می‌تواند یک process از سیستم‌عامل یا یک برنامه باشد که سرویس را از هر Server درخواست می‌کند. هر Server هم به درخواست‌های مختلف پاسخ می‌دهد.

ساختار Virtual Machine:

ایده Virtual Machine این است که یک interface از سخت‌افزار ایجاد می‌کند که سیستم‌عامل‌های مختلف به صورت هم‌زمان روی سخت‌افزار اجرا شوند در حالتی که هر سیستم‌عامل فرض کند که سخت‌افزار
مختص خود را دارد.

۶)
در این حالت یک بعضی از Instruction ها را حالت Privileged می‌کنیم. این دستورها فقط می‌توانند در حالت kernel mode اجرا شوند. برای اجرای درست سیستم‌عامل باید بتوانیم بین اجرای عملیات سیستم و کاربر تفاوت قایل شویم.
در حالت پایه عملیات را به دو مود کاربر و کرنل تقسیم می‌کنیم. یک بیت مود را مشخص می‌کند که به سخت‌افزار اضافه می‌شود.
زمانی که سیستم در حال اجرای برنامه‌های کاربر است در حالت مود کاربر اجرا می‌شود و زمانی که برنامه کاربر از سیستم‌عامل سرویس می‌خواهد در حالت مود کرنل اجرا می‌شود.
در زمان بوت سخت‌افزار در حالت مود کرنل است سپس سیستم‌عامل لود می‌شود و به حالت مود کاربر می‌رود و برنامه‌های کاربر را اجرا می‌کند. زمانی که یک وقفه رخ می‌دهد از حالت مود کاربر به حالت مود کرنل می‌رود. سیستم همواره قبل از دادن کنترل به برنامه کاربر به حالت مود کاربر می‌رود.
عملیات مود دوگانه سیستم‌عامل را از کاربران و کاربران را در مقابل هم به وسیله دسترسی محافظت می‌کند. سخت‌افزار فقط اجازه دسترسی به دستوراتی که privileged هستند را به مود کرنل می‌دهد و اجازه اجرا را برای مود کاربر نمی‌دهد.
مثال‌هایی از privileged instructions:
کنترل I/O، مدیریت تایمر و مدیریت وقفه

۷)
یک سیستم متشکل از چند فرایند است که در جاهای مختلف مثل سیستم‌عامل و فرایندهای کاربر در حال اجرا می‌باشند. یک برنامه یک موجودی passive است ولی یک process یک موجودی active است که CPU دستورات یک برنامه را یکی پس از دیگری اجرا می‌کند.
وظایف کلی سیستم عامل:
برنامه‌ریزی فرایندها و threadها در اجرا توسط CPU
سیستم‌عامل مشخص می‌کند که هر فرایند در چه نوبتی، کی و چگونه اجرا شود. آیا در صف منتظر دریافت خدمات باشد یا خیر و ... . همچنین رشته‌های مختلف را مدیریت می‌کند.
ایجاد و حذف فرایندهای کاربر و سیستم
سیستم عامل باید بتواند برنامه را اجرا کند و به آن CPU اختصاص بدهد و همچنین در صورت نیاز آن را متوقف و حذف کند.
توقف و ادامه فرایندها
سیستم‌عامل می‌تواند هر فرایندی را به حالت Suspended(با توجه به شرایط روی RAM یا دیسک) در آورد و در صورت نیاز آن را دوباره روی RAM لود کرده و اجرای آن را ادامه دهد.
مهیا کردن مکانیزم‌هایی برای فرایندهای هم‌زمان
مهیا کردن مکانیزم‌هایی برای ایجاد ارتباط
هر فرایند باید بتواند با فرایندهای دیگر ارتباط برقرار کند که این کار باید توسط سیستم‌عامل انجام شود.

۸)
برای اینکه مطمین شویم که برنامه داخل یک لوپ بی‌نهایت نمی‌افتد یا موارد مشابه از وقفه تایمر استفاده می‌کنیم. وقفه تایمر می‌تواند زمان ثابت یا متغیر داشته باشد.
در هر بار کلاک counter یک بار کمتر می‌شود و زمانی که مقدار آن به صفر می‌رسد یک وقفه تایمر رخ می‌دهد. قبل از دادن کنترل به کاربر سیستم عامل باید چک کند که وقفه فعال است یا خیر. هر وقفه تایمر کنترل را به سیستم‌عامل می‌دهد که سیستم عامل وقفه را به عنوان یک ارور می‌بیند یا دوباره به برنامه فرصت می‌دهد. هر دستوری که می‌تواند تایمر را تغییر دهد باید privilege داشته باشد.
می‌توان از تایمر برای جلوگیری از اجرای طولانی برنامه کاربر استفاده کرد.
یک مثال:
شمارنده را طبق زمان مجاز اجرای برنامه ست می‌کنیم. برای مثال برای برنامه ۲ دقیقه‌ای مقدار ۱۲۰ را ست می‌کنیم. هر ثانیه یک بار یک وقفه اینتراپت رخ می‌دهد و مقدار شمارنده یکی کم می‌شود. تا زمانی که شمارنده مثبت است کنترل به آن برمی‌گردد و زمانی که منفی شد سیستم عامل برنامه را تمام می‌کند.

۹)
بافر مکانی روی حافظه است که زمانی که اطلاعاتی از جایی به جای دیگر منتقل می‌شوند،برای ذخیره سازی موقت اطلاعات استفاده می‌شود.
بافرها در ارتباط I/O با سخت‌افزارهایی مانند دیسک، ارسال یا دریافت اطلاعات از شبکه یا پخش صدا روی یک اسپیکر استفاده می‌شوند.
بافرها می‌توانند بازدهی برنامه را توسط عملیات هم‌زمان افزایش می‌دهند.
مثلا خواندن یا نوشتن روی فایل می‌تواند به سرعت انجام شود همزمان با اینکه منتظر وقفه سخت‌افزار برای اجازه دسترسی به دیسک هستیم و به برنامه اجازه می‌دهد به عملیات خود ادامه دهد در حالی که کرنل در حال تکمیل کردن عملیات دیسک در پس زمینه است.
یا مثلاً از بافرینگ در مشاهده ویدیوهای آنلاین برای جلوگیری از لگ و تأخیر رخ می‌دهد، کامپیوتر ما داده‌های دریافتی را به میزان مشخصی بافر می‌کند و پس از آنکه به حد قابل قبولی رسید دیتاهای بافر شده نمایش داده می‌شوند.

۱۰)
System Call برای یک برنامه راهی می‌باشد که در آن یک سرویس سیستم‌عاملی که در آن اجرا می‌شود را می‌خواهد. برنامه‌ها توسط System Call با سیستم‌عامل تعامل می‌کنند. یک برنامه یک زمانی که یک درخواست از سیستم عامل دارد یک System Call می‌کند و آن سپس آن درخواست در سیستم‌عامل اجرا می‌شود.
System Call یک سرویس سیستم‌عامل را برای برنامه به وسیله API فراهم می‌کند. API یک اینترفیس بین برنامه و سیستم‌عامل در سطح کاربر ایجاد می‌کنند که در می‌توانند به این روش با هم تعامل کنند. به وسیله استفاده از API به جای فراخوانی مستقیم می‌توان انتظار این را داشت که برنامه روی هر سیستمی که API مشخص را ساپورت کند قابل اجرا باشد.
برای بیشتر زبان‌های برنامه نویسی run-time support system یک system call interface را مهیا می‌سازد که یک لینک به system call هایی که به وسیله سیستم‌عامل قابل اجرا است می‌باشد.

۱۱)
به صورت دقیق‌تر در سؤال یک گفته شده و خلاصه:
CPU در حال پردازش یک برنامه بوده است. یک وقفه رخ می‌دهد. سیستم عامل interrupt handler routine را اجرا می‌کند. در این مرحله تشخیص داده می‌شود که نوع وقفه چیست و چه کارهایی باید انجام شود. در طی interrupt handler routine نوع دیوایس تشخیص داده شده و با عملیات branch روی interrupt service routine که توسط درایور دستگاه مشخص شده می‌رود و آن را انجام می‌دهد سپس پردازنده کار قبلی خود را ادامه می‌دهد.
روند پردازش وقفه:

- دیوایس یک interrupt به پردازنده می‌دهد.
- پردازنده اجرای instruction را قبل از سریس دادن به دیوایسی که وقفه ایجاد کرده تکمیل می‌کند.
- پردازنده یک سیگنال acknowledged به دیوایس مربوطه می‌دهد.
- پردازنده حالت برنامه‌ای که در حال اجرای آن بوده را ذخیره می‌کند و کنترل به interrupt routine می‌دهد.
- پردازنده مکان شروع interrupt-handling routine را داخل PC لود می‌کند.
- پردازنده به instruction cycle بعدی می‌رود، چون مکان شروع interrupt-handling routine داخل PC قرار دارد شروع به اجرای آن می‌کند.
- چون باید اطلاعات اجرای برنامه قبل ذخیره شود معمولاً interrupt-handler شروع به ذخیره آن داخل stack می‌کند.
- interrupt handler شروع به اجرای interrupt می‌کند.
- زمانی که به وقفه پاسخ داده شد و تکمیل شد، اطلاعات رجیستر برنامه قبلی که ذخیره شده بود بازیابی می‌شود.
- PSW و PC هم بازیابی می‌شوند و در cycle بعدی برنامه قبلی به حالت اجرا باز می‌گردد.

۱۲)
PSW یک رجیستر است که حاوی بیت‌های condition است که به وسیله عملیات مقایسه‌ای، CPU و موارد دیگر ست می‌شود. برنامه‌های کاربر تمام PSW را می‌خوانند ولی فقط تعدادی از بیت‌های آن را در صورت لزوم تغییر می‌دهد.
در سیستم‌هایی که چند مود داریم یک بیت داخل PSW این مودها را کنترل می‌کند.

۱۲)
Hardware:
ترکیبی از اجزای سخت‌افزاری است که سخت‌افزار سیستم را تشکیل می‌دهند. مانند: Hard Disk، CPU، RAM و ...
Kernel:
بخش اصلی سیستم‌عامل است که CPU، memory، و دیوایس‌های جانبی را مدیریت می‌کند. کرنل پایین‌ترین رده سیستم‌عامل است و مسیول اجرای بخش‌های مختلف و اصلی سیتم‌عامل روی سخت‌افزار می‌باشد.
کرنل ارتباط بین دیوایس‌ها و نرم‌افزار را مدیریت می‌کند و منابع سیستم را مدیریت می‌کند همچنین با قرار گرفتن بین برنامه‌نویس و سخت‌افزار امکان استفاده راحت‌تر از سخت‌افزار را به وسیله تعریف کردن اینترفیس برای برنامه نویس ایجاد می‌کند.
Shell:
شل به عنوان یک اینترفیس برای کاربر عمل می‌کند و کاربر می‌تواند دستورات خود را در آن اجرا کند و برای مثال یک برنامه باز کند. در بعضی از سیستم‌ها command interpreter نامیده می‌شود.
Command:
دستوراتی هستند که کاربران برای تعامل با Shell و با هدف تعامل با سیستم‌عامل وارد می‌کنند.
Users:
کاربران سیستم‌عامل هستند که با استفاده از دستورات و شل با سیستم‌عامل تعامل می‌کنند.

برنامه‌های کاربردی: به وسیله شل با کرنل و سخت‌افزار تعامل می‌کند.
برنامه‌های سیستمی: این برنامه‌ها داخل کرنل هستند.
فراخوانی‌های سیستم: به صورت نرم‌افزاری داخل کرنل هستند.

با این معماری موافق هستم زیرا در سیستم‌عامل‌های موجود مانند لینوکس استفاده می‌شود.

۱۳)
زمانی که از API استفاده نکنیم باید به تمامی ماژول‌های سخت مسلط باشیم و ازشان آگاه باشیم و توانایی استفاده از آن‌ها را داشته باشیم و از طرفی اگر برای یک سخت‌افزار خاص نوشتیم برای سخت‌افزار دیگر کار نخواهد کرد به همین دلیل از API استفاده می‌کنیم.
API یک سری از دستورات را که برنامه‌نویس می‌تواند از آن‌ها استفاده کند را مشخص می‌کند. برای مثال پارامترهای توابع و مقادیر بازگشتیشان و... .
سه نوع مرسوم API ها POSIX ،Windows و Java هستند.
برنامه‌نویس به وسیله کتاب‌خانه‌ای که توسط سیستم‌عامل مشخص شده به API دسترسی پیدا کرده و از آن استفاده می‌کند. برای UNIX و LINUX این کتاب‌خانه linc نام دارد. در وشت سر و آن‌ور API تعدادی system call فراخوانی می‌کند.

۱۴)

کامپیوترها برای شروع اجرا به یک برنامه اولیه براس اجرا نیاز دارند. این برنامه اولیه bootstrap program نام دارد و روی سخت‌افزار کامپیوتر و ROM ذخیره می‌شود. این برنامه تمامی اجزای سیستم را مانند رجیسترهای CPU و کنترل دیوایس‌ها و حافظه را مقداردهی اولیه می‌کند. این برنامه باید بداند که چگونه سیتم‌عامل را لود و اجرا کند و برای این کار برنامه bootstrap باید کرنل سیستم‌عامل را پیدا کرده و داخل حافظه لود کند.

زمانی که کرنل سیستم‌عامل لود شد و شروع به اجرا کرد می‌تواند سرویس‌های مورد نظر را بدهد.

۱۵)

در حالت کلاسترینگ نامتقارن یک ماشین در حالت hot-standby قرار می‌گیرد در صورتی که بقیه ماشین‌ها در حال اجرای برنامه هستند. ماشینی که در حالت hot-standby است سرورهای فعال را رصد می‌کند.

در حالت کلاسترینگ متقارن تعداد ۲ یا بیشتر ماشین در حال اجرای برنامه هستند و همدیگر را رصد می‌کنند.

حالت کلاسترینگ متقارن به صرفه تر است چون از تمام سخت‌افزار استفاده بهینه می‌کنیم ولی نیاز دارد که بیشتر از یک برنامه برای اجرا در دسترس باشد.

۱۶)

اصطلاح peer-to-peer چیست و چه مزیتی نسبت به معماری client-server میتواند داشته باشد؟

در این مدل بر خلاف client-server، سرور و client از هم جدا نمی‌شوند و هر node بر حسب نیاز client یا سرور نامیده می‌شود.

در این معماری بر خلاف client-server دیگر server به عنوان گلوگاه شناخته نمی‌شود.

چرا در سیستم‌ها (مثلاً در system call) استفاده از API به جای استفاده مستقیم از سخت افزار، پورت‌ها و ... مهم است؟

ما به وسیله استفاده از API دیگر درگیر پیاده‌سازی داخلی و سختی‌ها و تفاوت‌های آن‌ها نخواهیم شد. فرض کنید یک برنامه‌نویس به جای استفاده اتصال به wifi مستقیم از پورت‌های کارت شبکه استفاده کند. این برنامه باید روی تعداد بسیار زیادی کامپیوتر اجرا شود و عملاً ممکن نیست برنامه‌نویس به صورت دستی همه این موارد را دانسته و پیاده‌سازی کند به همین دلیل از API به عنوان یک رابط استفاده می‌کنیم و دیگر درگیر پیاده‌سازی داخلی نمی‌شویم.

۱۷)

روند boot شدن کامپیوتر چگونه است و آیا برنامه یا سیستمی برای این کار لازم است؟

در کامپیوترها یک parent board داریم که یک برنامه برای BIOS روی آن قرار دارد. BIOS حاوی برنامه‌های سطح پایین I/O و چک کردن سیستم است.

زمانی که سیستم boot می‌شود، BIOS شروع به کار می‌کند و در ابتدا چک می‌کند چقدر RAM نصب شده و آیا پیوایس‌های ساده I/O مثل ماوس و کیبورد وصل شده‌اند و به درستی پاسخ می‌دهند.

چرا virtualization و virtual machine در web hosting مهم است؟

فرض کنید ایده‌ای به نام virtual machine وجود نداشت. هر مشتری باید یا یک کامپیوتر واقعی و کامل با سیستم‌عامل اجاره می‌کرد که اصلاً به صرفه نبود یا باید از هاست‌های اشتراکی که اصلاً کنترل زیادی روی آن ندارد استفاده می‌کرد.

ولی با ایده virtual machine می‌توان یک سیستم و کامپیوتر داشت و آن را به چندین ماشین تقسیم‌کرد که دقیقاً مانند ماشین‌های جدا از لحاظ سخت افزاری و فیزیکی رفتار می‌کنند.

۱۸)

چرا LINUX با اینکه monolithic است، بعد از اضافه شدن هر قابلیت نیاز با اجرای دوباره ندارد؟

لینوکس از معماری microkernel استفاده نمی‌کند ولی با استفاده از معماری ماژولار و تشکیل شدن از ماژول‌های مختلف که در زمان نیاز به صورت اتوماتیک لود می‌شوند استفاده می‌کند. به این ماژول‌ها loadable block گفته می‌شود. در حقیقت ماژول‌ها می‌توانند در صورت نیاز به کرنل لینک شده یا لینک آن‌ها پس گرفته شود.

چرا CACHE در کامپیوترها مهم است؟

در کامپیوترها بحث تمرکز روی یک قسمت از RAM را داریم یعنی CPU عموما در یک بازه زمانی تنها با قسمتی از RAM سروکار دارد و بخش‌های دیگر استفاده زیادی نمی‌شوند از طرفی سرعت کم RAM هم گلوگاه ما است و سرعت ما را کاهش می‌دهد. ممکن است به یک بخش RAM هم چندین بار دسترسی پیدا کنیم.

در این میان هم یک بده بستان بین حافظه و سرعت وجود دارد.

برای رفع مشکل از حافظه‌های فوق سریعی به نام CACHE استفاده می‌کنند که زمانی که خانه‌ای از RAM در CPU استفاده شد در خود CACHE می‌ماند و CPU به جای استفاده از RAM داده را از خود CACHE می‌خواند. در این حالت و کامپیوترهای امروزی CACHE بین CPU و Main Memory قرار میگیرد و خود CACHE هم چند لول مختلفی دارد و با عبور از لایه‌ها hit rate ما بیشتر می‌شود.