کرنل (Kernel) : کرنل هسته اصلی سیستم عامل است که به طور مستقیم با سخت افزار سیستم ارتباط دارد و مسئول مدیریت فرایندهای سیستم مثل پردازنده، حافظه و دستگاه‌های ورودی و خروجی (I/O) است. کرنل دستورات عملیات های اصلی سیستم را مدیریت می‌کند.

وظایف:

- مدیریت فرایندها و وظایف چندگانه (multi-tasking)
- مدیریت حافظه (Memory Management)
- مدیریت دستگاه‌های سخت افزاری (I/O mangement)
- مدیریت سیستم‌های فایل (File System Management)

سیستم کال (System Call) : سیستم کال یک واسط است که توسط برنامه‌های کاربری استفاده می‌شود تا به سرویس‌ها و منابع مختلف سیستم عامل دسترسی پیدا کنند. چون برنامه‌های کاربری نمی‌توانند مستقیم به سخت افزار دسترسی داشته باشند، از سیستم کال‌ها برای درخواست سرویس از کرنل استفاده می‌کنند.

مثال) سیستم کال open() ← این سیستم کال برای باز کردن یک فایل در سیستم عامل استفاده می‌شود. وقتی open() فراخوانی می‌شود، کنترل به کرنل منتقل می‌شود و کرنل عملیات لازم را برای باز کردن فایل انجام می‌دهد.

کرنل مود (Kernel Mode) : حالتی است که در آن کرنل سیستم عامل دسترسی‌های خاصی از کرنل‌های سطح پایین سیستم عامل اجرا می‌شوند. در این حالت، سیستم عامل دسترسی کامل به تمام منابع سخت افزاری و حافظه دارد.

کاربرد: در این حالت، کرنل می‌تواند عملیات‌های حساس مثل مدیریت حافظه و تخصیص آن، مدیریت ورودی و خروجی، کنترل فرایندها را انجام دهد. برنامه‌های کاربری نمی‌توانند مستقیماً در کرنل مود اجرا شوند تا امنیت و پایداری سیستم تضمین شود.

(2) یوزر مود (User Mode): حالتی است که در آن برنامه‌های کاربردی معمولی (مثل ویرایشگرها و مرورگرها) اجرا می‌شوند. در این حالت، برنامه‌ها دسترسی مستقیم به منابع سخت‌افزاری و حافظه ندارند و فقط می‌توانند عملیات‌های پیچیده را انجام دهند. در این حالت، برنامه‌ها فقط از طریق سیستم کال‌ها از کرنل، درخواست خدمات می‌کنند

پایه‌ی کار سیستم عامل براساس Interrupt می‌باشد.

دو نوع کار با Interrupt ها:

۱- ایجاد اینتراپت: هنگامی که یک دیوایس خاص اتفاق می‌افتد، یک Interrupt ایجاد می‌شود.

۲- متوقف کردن فرایند جاری: پردازنده، اجرای فرایند جاری را متوقف کرده و وضعیت آن را ذخیره می‌کند

۳- انتقال به سرویس اینتراپت (ISR): پردازنده به یک تابع خاص با نام "روتین سرویس اینتراپت" (ISR) منتقل می‌شود که وظیفه پردازش اینتراپت را برعهده دارد.

۴- پاسخ به اینتراپت: ISR وظایف لازم را انجام می‌دهد، مثل خواندن داده‌ها از یک دستگاه یا تنظیم رجیسترهای مورد نیاز

۵- پاسخ به فرایند قبلی: پس از اتمام ISR، وضعیت قبلی پردازنده بازیابی می‌شود و کنترل به فرایند قبلی بازگردانده می‌شود.

« System call ها برای پیاده سازی از مفهوم Interrupt استفاده می کنند »

برای اجرای سیستم کال، برنامه باید از حالت User Mode به حالت Kernel Mode منتقل شود.
این جابجایی توسط اینتراپت نرم افزاری انجام می شود. اینتراپت نرم افزاری برای درخواست سیستم کال ها اجرا می شود.

فرایند به این صورت است:

- برنامه کاربری کدی اجرا می کند که یک اینتراپت نرم افزاری خاص (مثل اینتراپت 0x80) را فعال می کند.

- وقتی این اینتراپت اجرا می شود، کنترل اجرای برنامه به هسته سیستم عامل منتقل می شود و حالت سیستم از کاربر به هسته تغییر می یابد.

=> پس از اجرای سیستم کال و انجام عملیات مورد نظر، نتیجه اجرای سیستم کال (مثل داده خوانده شده یا کد وضعیت) به برنامه کاربر بازگردانده می شود. پس سیستم عامل با استفاده از اینتراپت نرم افزاری دوباره به حالت کاربر برمی گردد و اجرای برنامه ادامه می یابد.

« تفاوت کلی برای راحتی در پیاده سازی یک سیستم عامل »
جداسازی Mechanism و Policy

Mechanism ↙ : چطوری انجامش بدیم؟
Policy ↘ : برای انجام هرکاری که داریم به چی نیاز داریم؟

مکانیزم دونه از Policy جدا شد، کاربرد Flexibility می شه و مثلا با تغییر مکانیزم می تونیم پارامترهای Policy رو عوض کنیم

« مقایسه بین زبان های برنامه نویسی سطح بالا و سطح پایین »

زبان های سطح بالا :

* با توجه به نزدیکی بودن به زبان انسان و مفهوم بهتر Syntax ها، سرعت بالا تری در کدزنی داریم .

* با توجه به حضور کامپایلر در تبدیل زبان سطح بالا به زبان ماشین، کدهای نهایی بهینه و بهتر از آن خواهد شد که بجز اهمیم خودمون حافظه رو مدیریت میکنیم و به مسائل و مشکلی فکر میکنیم .

* پرتابل بودن : زبان های سطح بالا مثل C++ ، کتابخانه های مشابهی با خود دارد درصورتی که در زبان های سطح پایین کتابخانه ها متفاوت تر خواهند شد .

* در زبان های سطح بالا، کتابخانه ها و Structure های زیاد و مفیدی برای برنامه نویسی داریم .

* در زبان های سطح بالا به دلیل بالا بودن تعداد کتابخانه ها و هماهنگی بین آن ها سرعت پایین تری میگیرند

* در زبان های سطح بالا، به دلیل استفاده از Data Structure های پیشرفته، حافظه زیادی استفاده میکنند .

(Emulator) : شبیه ساز یا emolator به نرم افزاری گفته میشود که به کاربر اجازه میدهد تا سیستم کامپیوتری را بر روی پلتفرم متفاوت شبیه سازی کند . در واقع شبیه سازها این امکان را میدهند که برنامه ها یا سیستم عامل هایی که برای یک سخت افزار خاص طراحی شده اند، روی سخت افزار یا پلتفرم دیگری اجرا شوند .

« Modular و Monolithic » در پیاده سازی

سیستم عامل

Monolithic : در این روش از طراحی، کرنل رو به صورت پیوسته به هم کد نویسی کردند و از برنامه های سیستمی یکپارچه جدا کردند و یه جور Compact است.

مزیت : سادگی در پیاده سازی. چون هر چیزی که به نظر مهم بوده رو درش ریختند و در کرنل، در کنار دیگر بخش ها گذاشتند

عیب : دیباگ کردن و عیب یابی سخت تر است

Modular : بخش های مختلف کرنل، به صورت جداگانه و ماژولار پیاده سازی شده اند

« Layer Approach »

در طراحی لایه ای، هر بخش از سیستم، جداگانه پیاده سازی شده و با لایه های بالا و پایین خودش در ارتباطه. در این پیاده سازی اگه لایه ی 2 بخواد با لایه صفر ارتباط برقرار کنه، باید حتما از طریق لایه 1 این کار رو انجام بده.

مزیت این روش flexebility آن است چون هر بخش جدا است از بقیه ولی به دلیل ارتباط لایه ای overhead در سیستم داریم یه عیب محسوب میشه

← Linux، هم monolithic است و هم modular

هر بخشی که نیاز بوده جداگانه ماژولار کردن و اون قسمت هایی که از flexability بیشتری برخوردار بودن رو یکپارچه کردن

(6)

معماری میکروکرنل: در این معماری، بخش‌های اساسی کرنل تو هسته قرار میگیرند و بخش‌های بیشتری از آن را در User Space پیاده‌سازی می‌کنیم.

مزیت: ۱. با توجه به حساس بودن بخش کرنل، هرچه دستورات کمتری در آن اجرا شود، بهتره!

۲. بالا رفتن portability

عیب: ۱. بالا رفتن overhead، چرا که بخش‌هایی که در User Space پیاده‌سازی شده‌اند، برای برقراری ارتباط با بخش‌های کرنلی از پیام استفاده می‌کنند ← macos با میکروکرنل پیاده‌سازی شده است.

LKM: Loadable Kernel Modules: کرنل‌هایی که به شکل LKM پیاده‌سازی می‌شوند این قابلیت را دارند که ماژول حذف یا اضافه کنند، بدون اینکه کرنل کامپایل دوباره شود.

← طراحی به سبک LKM، یه چیزی تو مایه‌های Layer هست با این تفاوت که به دلیل افزودن (load) و کم کردن ماژول‌ها، انعطاف‌پذیری بیشتری دارد.

windows: microkernel + monolithic

macos: Layer + microkernel

Linux: monolithic + modular

Building and Booting Linux

کانفیگ کردن کرنل لینوکس پس از دانلود نسخه جدید: make menuconfig

کامپایل کردن: make

پس یک image به نام "vmlinuz" ایجاد می‌شود.

با دستور "make modules" می‌تونیم ماژول‌های دیگه‌ای که توی بخش قبلی ایجاد نبودن رو کامپایل کنیم.

با دستور "make modules_installs" کرنل این ماژول‌ها را در vmlinuz نصب می‌کنند.

کرنل جدید با دستور "make install" نصب می‌شود.

(7) مراحل ساخت و اجرای یک سیستم عامل

۱- نوشتن Source code سیستم عامل

۲- کامپایل کردن سیستم عامل برای اجرا روی ساخت افزارهای مختلف.

۳- کامپایل کردن سیستم عامل

۴- نصب سیستم عامل

۵- بوت کردن کامپیوتر و سیستم عامل جدیدش.

System Boot

((می‌خواهیم به سیستم این رو بفهمیم که وقتی روشن شد، نرم‌افزار سیستم عامل بیاد بالا))

روی کامپیوترها، یک برنامه کوچک اولیه به اسم BIOS وجود داره که اصطلاحا بهش boot strap loader می‌گیم.

این برنامه روی ROM یا EEPROM ذخیره می‌شه

BIOS یکی از اجزای حیاتی برای اطمینان از راه‌اندازی صحیح سیستم و ارتباط بین سخت‌افزار و BIOS است.

این برنامه‌ای است که پس از روشن سیستم اجرا می‌شود و وظیفه بوت کردن سیستم را برعهده دارد.

پس از BIOS یه boot loader دیگه هم اجرا می‌شه که یا حال دیفورت یا یه boot loader دیگه مثل GRUB لینوکس است.

در کامپیوترهای جدید از UEFI که جدیدتر از BIOS است استفاده می‌شود

BIOS روی ROM ذخیره بود و اجرا می‌شد، درحالی که UEFI روی یک پارتیشن مشخص از هارد دیسک ذخیره می‌شه.

UEFI امکانات بیشتری دارد.

UEFI به دلیل تفاوت در نوع ساخت دیباگ‌سازی سخت‌تری نسبت به BIOS دارد.

UEFI می‌تواند دستورات ۳۲ یا ۶۴ بیتی را اجرا کند درحالی که BIOS فقط ۱۶ بیتی لود می‌شد. به همین دلیل هم، bootloader ای که BIOS لود می‌کنه ۱۶ بیتی است ولی UEFI می‌تونه ۳۲ یا ۶۴ بیتی لود کنه و حتا این از تنظیمات خیلی بیشتره.

UEFI امکانات بیشتری دارد، مثلا درایورهای بیشتری از جمله درایور ماوس رو داره.

* پس از اینکه BIOS کار خود را در انجام تست های اولیه سخت افزار و شناسایی دستگاه های بوت به اتمام رساند، کنترل را به GRUB یا هر برنامه دیگری که در پیکربندی سیستم تنظیم شده باشد، واگذار می کند.

GRUB سپس با توجه به تنظیمات خود، لیستی از سیستم عامل های موجود را نمایش می دهد یا سیستم عاملی که به عنوان پیش فرض تنظیم شده است را بوت می کند.

operating system
Debugging

دستگاه های ذخیره سازی، core dump و crash dump. هر دو فایل هایی هستند که اطلاعاتی از وضعیت حافظه و پردازنده هنگام بروز کرش را ذخیره می کنند. این اطلاعات به تحلیل خطاها و دستیابی علت بروز کرش در نرم افزار و سخت افزار کمک می کنند.

core dump & crash dump

core dump : یک فایل است که وضعیت حافظه یک برنامه خاص را هنگام متوقف شدن ناگهانی ذخیره می کند.
core dump اطلاعاتی را از قبیل وضعیت حافظه، رجیسترهای پردازنده، داده های هیپ و پشته و دستورالعمل های نقطه ای که برنامه در حال کرش کرده را ذخیره می کند.

crash dump : وضعیت کل سیستم را هنگام کرش یا بروز خطای سیستمی ثبت می کند. این نوع دامپ نه تنها اطلاعاتی از یک برنامه خاص، بلکه اطلاعاتی از کل سیستم از جمله کرنل، درایورهای سخت افزار و تنظیمات سیستم عامل را در خود دارد.

« Virtualization and Computing Environments »

به وسیله ی Virtualization به وجود می آید.

process : os abstraction of processor, main memory, an I/O devices for a running program ← چیزی که process به آن دسترسی دارد می تواند اجرا شود، از حافظه استفاده کند و با دستگاه های I/O کار کند.

Virtual memory : os abstraction of Memory ← هر process دید خودش از حافظه را دارد. انگار که حافظه مختص به خودش و در اختیار خودش است.

File : os abstraction of I/O devices ← همه ورودی و خروجی های سیستم به شکل فایل هستند که می توان در آن ها نوشت یا از آن خواند.

پس به عنوانی این abstraction یعنی بخش انتزاعی سیستم عامل از سخت افزار است که به ما ارائه می‌شد

=> با این تعریف، Process رفتار مشترک با OS برای پردازنده، main memory و دستگاه های ورودی خارجی یک برنامه در حال اجرا. Virtual Memory، رفتار مشترک با OS و memory و File یک تجرید از I/O device هاست.

"CPU Virtualizatio"

مثال قابل درک در واقع که به عنوان همیشه سروکار داریم، run کردن چندین برنامه و process است که در این حالت، CPU به جوری واسه ما virtualization می‌شه انگار همه process ها دارن باهم اجرا می‌شن در حالی که از یک سری مکانیزم واسه تخصیص CPU مثل Time sharing داره استفاده می‌شه.

"Memory Virtualization"

Memory رو به شکل آرایه ای از بایت های تو ذهنم بسپم.

واسه خواندن از Memory (Read) نیاز به آدرس خانه ای داریم که باید ازش بخونیم.

واسه نوشتن در Memory (write) " " " داده ای که بنویسیم علاوه بر Data ای که باید ذخیره شه.

در نتیجه دو عملیات اصلی در Memory Virtualization، load & Store هستند.

یعنی اینکه هر دستوری از برنامه هم در Memory است باید Fetch شه.

← دسترسی به حافظه ما خیلی زیاده و هر کاری بخواهیم بکنیم بهش احتیاج داریم.

Memory Virtualization، یعنی هر process طوری Memory رو می‌بینه انگار کل Memory واسه خودشه. و هیچ تداخلی بین تخصیص حافظه برای برنامه ها نیست که CPU آن را manage می‌کند

← به عنوان مثال هر سه process، 1، 2، 3 اگر همزمان اجرا شوند، و هر کدام دارای یک پوینتر باشد، این پوینتر در هر سه می‌تواند به یک خانه از حافظه اشاره کند و یکسان باشد. چرا که سیستم هر process انگار کل حافظه را در اختیار دارد

← نکته: در map شدن این سه آدرس در مثال بالا به Phisical Memory، آدرس ها متفاوت خواهند شد و این همان abstraction of Memory OS است.

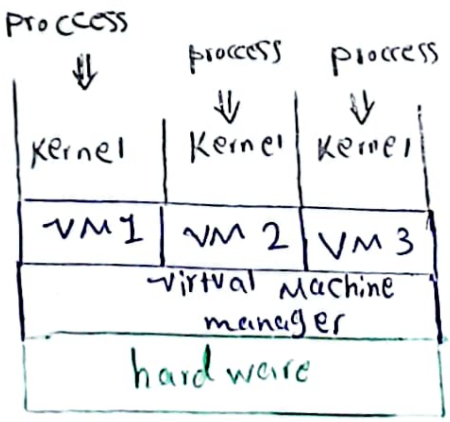
"Concurrency"

همزمانی

وقتی دوتا process روی یک چیز کار می‌کنند، ممکن است همزمانی رخ دهد. مثلا فرض کنید دوتا thread داریم که هر دو قرار با Counter ای رو ++ کنند. در لحظه‌ای که هر دوی این‌ها همزمان ++ می‌کنند، انگار که فقط یک بار به Counter افزوده شده و این یعنی همان Concurrency .

"Virtual Machine"

تقسیم که Virtual Memory و Virtual CPU به این شکل بود که انگار هر process به طور کامل CPU و Memory رو در اختیار داره. کار VM شباهت زیادی به همین موارد داره با این تفاوت که کار VM، اینه که OS های مختلف روی یک سیستم راه‌اندازی می‌شه به طوری که همه این‌ها انگار از یک سری منابع hardware استفاده می‌کنن در حالی که انگار کل منابع سیستم در اختیار هر یک از OS ها هست و VM، این‌ها رو برای هر کدام abstract کرده



Emulation : در بحث Virtual Machine، مفهوم Emulation هم مطرح می‌شه. به این شکل که هر کدام از این VM ها CPU متفاوتی نسبت به CPU source داشته باشند (مثلا)

« Container-Based Virtualizatio »

برای اجرای اپلیکیشن ها و سرویس ها به صورت جداگانه و ایزوله در یک سیستم عامل مشترک استفاده می شود. در این روش، هر کانتینر شامل کد، کتابخانه ها و وابستگی های لازم برای اجرای اپلیکیشن مورد نظر است.
برخلاف مجازی سازی VM، سیستم عامل مستقلی ندارد و برروی سیستم عامل میزبان اجرا می شود. به همین علت کانتینرها سریع تر و سبک تر از VM ها هستند و مصرف منابع کمتری دارند.
کانتینرهای Docker معروف ترین هستند

hypervisor چیست؟ که به آن ناظر ماشین مجازی گفته می شود، نرم افزاری است که یک لایه مجازی سازی ایجاد می کند که به سیستم عامل ها اجازه می دهد به منابع سخت افزاری مثل پردازنده، حافظه بدون دسترسی مستقیم به سخت افزار فیزیکی دسترسی داشته باشند

hypervisor type 1: نوع اول آن، مستقیماً روی سخت افزار نصب می شود و نیازی به سیستم عامل پایه ندارد و بیشتر برای محیط های سازمانی بزرگ مناسب است.

hypervisor type 2: نوع دوم آن است که روی سیستم عامل موجود نصب می شود و مثل یک نرم افزار معمولی رفتار می کند مثل VMware.

« Computing Environments »

distributed systems: چندین سیستم که با یک network باهم در ارتباط هستند و به شکل یک سیستم واحد عمل می کنند ← این سیستم، نیازمند سیستم عامل خاصی است با اسم Network OS.

دستگاه هایی که هدفشان از چنین پردازنده برای سیستم استفاده می‌شود، مزایای خوبی حاصل است:

۱- Increased throughput ← سریع‌تر اجرا کردن برنامه‌ها

۲- Economy of Scale ← اجرای چنین برنامه

۳- Increased reliability ← یعنی اگر یکی از پردازنده‌ها از کار افتاد، بقیه‌ای به اجرای برنامه خود و بقیه پردازنده‌ها تقسیم شده و روند اجرا ادامه یابد.

« دو نوع قرارگیری و استفاده از چنین پردازنده‌ای »

1- Asymetric Multiproccessing ⟶ نامتقارن

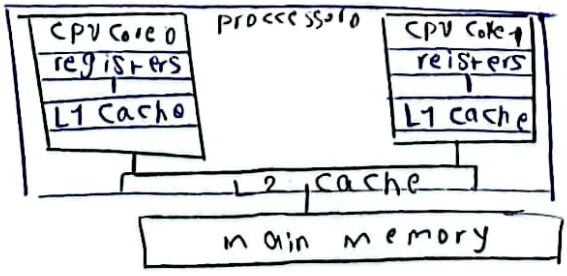
در این حالت چنین پردازنده، تقسیم کنترل یک پردازنده master در سیستم مقایسه می‌شود. که هر پردازنده‌ها را خاصی را باید انجام دهد.

2- Symmetric " ⟶ متقارن

در این حالت master نداریم و پردازنده‌ها با هم تراشه‌ها را حل می‌کنند.

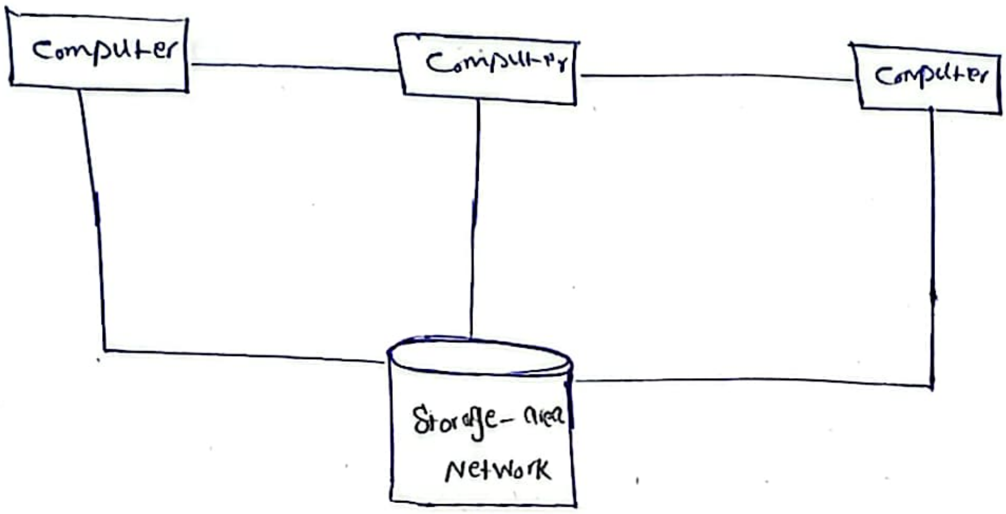
« Dual Core Design »

در این حالت یک پردازنده، به اصطلاح یک چیپ، حاوی دو (یا بیشتر که بیشتر هم باشد) CPU مجزا است که هر CPU، cache خود را دارد و سپس به یک cache مشترک وصل هستند



"Clustered Systems"

"تا حدی ممکن است چندین کامپیوتر که هر کدام می‌توانند Single/multi processor یا multi core باشند، با قابل شبکه با هم تشکیل یک سیستم بدهند که همه این ها از یک Storage-are network (SAN) استفاده می‌کنند.

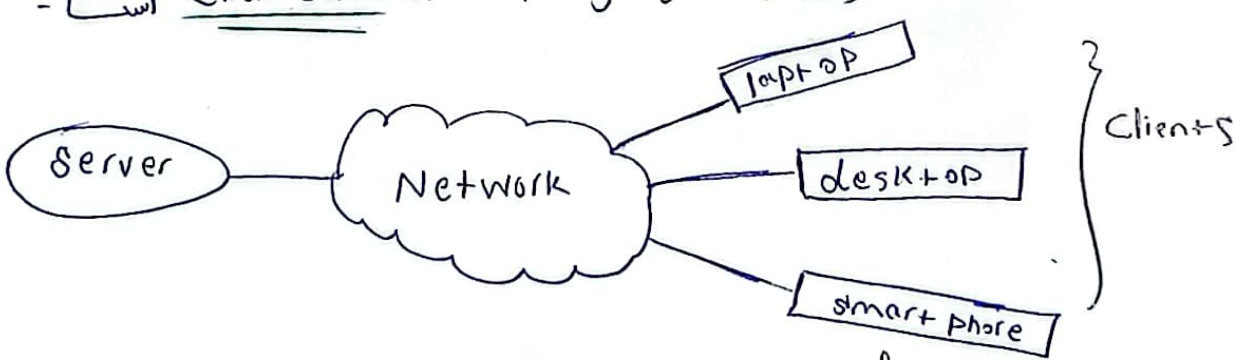


در دیزاین clustered هم می‌توان Asymmetric یا Symmetric داشت

به این شکل که در Asymmetric یک کامپیوتر داشته باشیم به عنوان master و بقیه زیر نظر کنترل این master فعالیت‌ها را task های خود را انجام دهند.

و در حالت Symmetric چندین کامپیوتر با هم تشکیل سیستم دهند و هر برنامه از آن ها می‌توان task های سیستم را دارد.

"حالت دیگری از Computing System ها Client Server است.



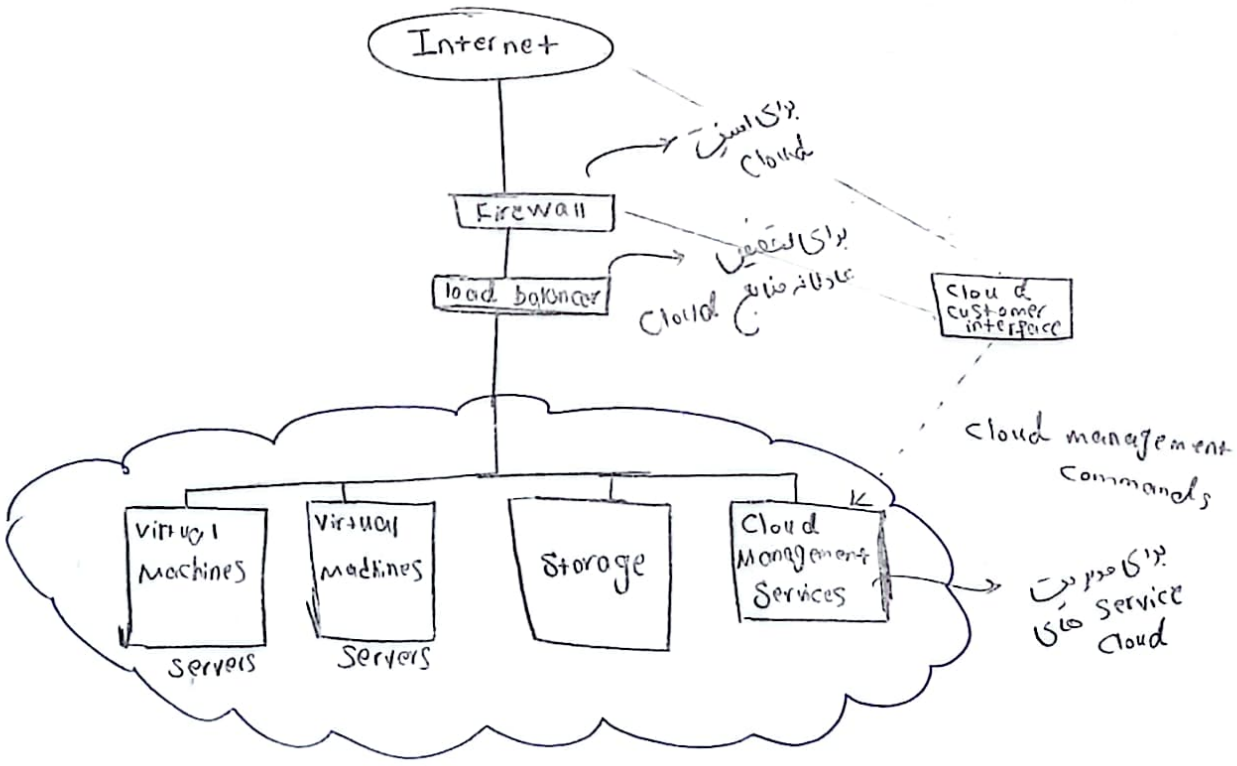
در این حالت، شکل clustered نیازی نیست ها client ها به طور فیزیکی در کنار یکدیگر باشند و Server از طریق Network که می‌تونه چیزی مثل اینترنت باشه با client ها ارتباط برقرار کنه

حالت دیگری از distributed system ها، peer-to-peer می‌باشد که در این حالت Server نداریم و Client ها باهم ارتباط دارند و اطلاعات را ذخیره می‌کنند.

"Cloud Computing"

دسترسی هایی مثل Computing و Storage را به Client ها میده و Cloud یک جورایی یک Service است

لایه‌های زیادی در درون هر Cloud هست که کارهای مختلفی (سرویس‌های مختلف) را ارائه می‌کنند.

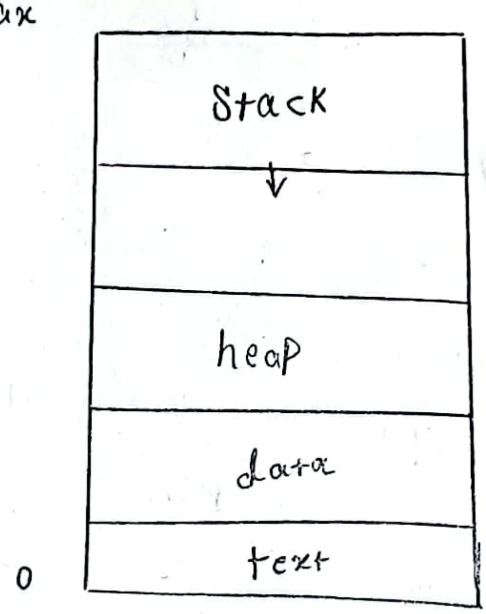


process به یک برنامه در حالت اجرا (execution) گفته می‌شود.

process بخش active و program بخش passive است. ← program روی دیسک و process روی مموری است. (لود شدن در memory)

دستور top، لیست process ها رو میده. هر پروسه یک key ID منحصر به فرد داره.

process in Memory



⇐ متغیرهای local، ورودی‌های توابع، خروجی‌هایی که از توابع می‌گیریم، همه در stack ذخیره می‌شه

⇐ اون فضاهایی که allocate می‌کنیم در heap ذخیره می‌شن

⇐ متغیرهای گلوبال در data هستند.

⇐ بخش text متن کدهاست.

* هر پروسه یه مقداری از صفر تا max، از Memory می‌گیره که max بستگی به سیستم داره.

* فضای stack از بالای این بخش و heap از بالای data، شروع شده و این فضای خالی بین اون‌ها یعنی سایزشون نسبت به پروسه، می‌تونه تغییر کنه.

« process state »

۱- New : حالتی که برنامه در memory قرار گرفته (تازه) است و ایجاد شده است.

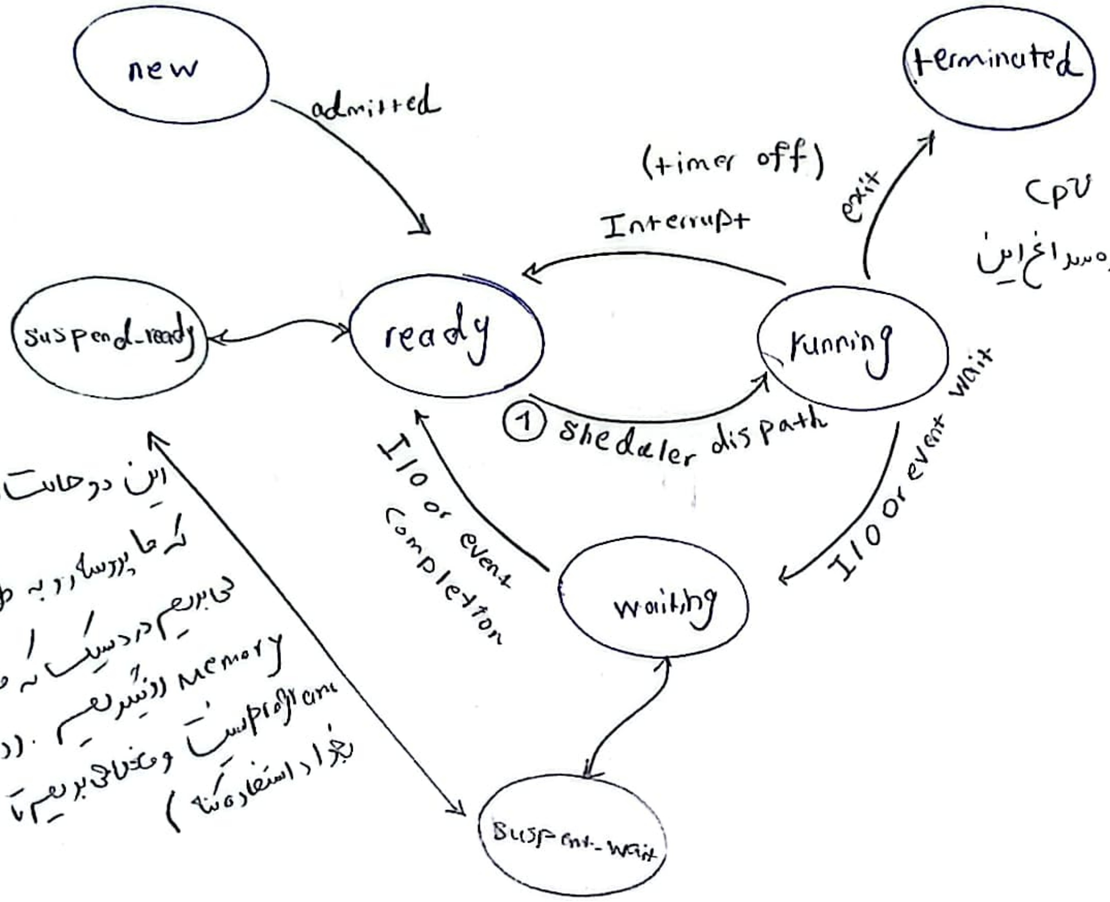
۲- Running : حالتی که پروسه در حال اجراست.

۳- Waiting : حالتی که پروسه به جایی رسیده که منتظر یک چیزی باشه، مثل ورودی ای که طرف بده.

۴- Ready : حالتی که برنامه در Memory است ولی منتظر آن است که cpu اجراش کنه.

۵- Terminated : حالتی که پروسه تموم شده باشه.

« Diagram of process state »



① در این حالت CPU تصمیم می‌گیره که بره سراغ این فرایند

این دو حالت وقتی است که ما پروسه رو به طور موقت می‌بریم در دیسک که فضای memory رو نشر بدیم. (وقتی سیستم که program است و وقتی می‌بریم تا CPU دوباره بخواد استفاده کنه)

«process control block»

PCB

اطلاعات مربوط به هر پروسه رو در PCB نگهداری می‌کنیم.

* Process State: این که پروسه در کدام state قرار داره
* Program Counter: مکان instruction بعدی
* CPU registers: مجموعه رجیسترهایی که پردازنده برای این پروسه داره رو نگهداری می‌کنیم
* CPU Sheduling: زمان‌بندی سیستم، اولویت پروسه رو نگه می‌داریم
* Memory_management: نگهداری موقعیت فعلی پروسه درون حافظه فیزیکی
* Accounting Information: اطلاعاتی از قبیل مقدار استفاده پروسه از CPU، این که چند تا کلاک دیگه عوض شده یا چه قدر اجرا شده.
* I/O Status Information: اطلاعات I/O device های allocate شده برای پروسه.

اطلاعات PCB در لینوکس در task_struct قرار گرفته است

« proccess sheduling »

scheduler سیستم با بررسی کیوهای انتظار، میاد مشخص می‌کنه کدوم پروسه، بره توی core پردازنده و اجرا شه

⇐ هدف، بیشترین استفاده از CPU و به سرعت سوئیچ کردن بین پروسه‌هاست

scheduling از دو حالت استفاده می‌شه:

1- Ready Queue: همه پروسه‌های قرار گرفته در Memory، آماده برای اجرا

2- Waiting Queue: همه پروسه‌هایی که منتظر یک event هستند

« Context Switch »

Context Switch، عاملی است که پس از scheduler، اجرا شده و مراحل تغییر به پروسه دیگر رو انجام میده.

این عامل در ابتدا، اطلاعات فعلی پروسه رو با اطلاعات جدید آپدیت می‌کنه و در PCB، save می‌کنه. سپس اطلاعات پروسه بعدی را از PCB آن لود کرده و در دست اجرا برای CPU قرار می‌دهد

hyper threading های cpu core: هسته‌هایی از پردازنده که میتونن چندین رشته (دسته) دستور داشته باشند.
در این حالت اطلاعات چند PCB را همزمان در CPU می‌توانیم load کنیم

« Shedulers »

انواع مختلفی از scheduler ها داریم:

1- short-term: همون زمانبندی که همش رو اسمش صدا می‌کنیم و همون که مشخص می‌کنه کدوم پروسه، وارد CPU بره تا اجرا شه. این نوع scheduler، خیلی سریع انجام میشه در حد milliseconds

2- long-term: زمانبندی که مشخص می‌کنه کدوم پروسه بره توی صف ready. نه این نوع، سریع انجام می‌شه و در حد چند ثانیه و دقیقه است.

⇐ همان‌هایی که دارای PCB در task_struct سیستم عامل هستند. long-term-scheduler، درجه multiprogramming رو کنترل می‌کند یعنی تعداد process های موجود در system.

I/O bound process : پردازنده هایی که بیشتر وقت خود را صرف خواندن از I/O دارند

CPU bound process : پردازنده هایی که بیشتر وقت خود را صرف محاسبات اطلاعات CPU صرف می کنند.

یکی از کارهایی که long-term scheduler ها انجام میدن، برقراری یک تعادل بین پروسه های I/O bound, CPU bound است.

۳- Medium-term : این کار به پروسه برده توی حالت Suspend، توسط این زمانبند ارغام میشه. به طور مثال اگر یک پروسه جز اهداف new به Memory وجا نداشته باشه، این Scheduler میاد اون پروسه رو برمیداره توی صف waiting قرار داده، این رو میبره توی دیسک و یک دیگه رو میاره توی Memory.

به این فرایند Swapping گفته می شه.