مریم سعیدمهر
۹۶۲۹۳۷۳

تکلیف تئوری اول درس سیستم عامل
ترم اول ۹۸ – ۹۹

۱- تعریف کرده و مقایسه کنید

الف) Device Controller و Device driver

* **دیوایس درایور :** یک نرم افزار است که رابط بین کنترلر و سیستم عامل میباشد به زبان عامیانه میداند چگونه با دیوایس کار کند یعنی دستورات سیستم عامل را به زبان قابل فهم(صفر و یک) برای دیوایس کنترلر ترجمه میکند. همچنین دیوایس درایور قابل تغییر است و کاربر میتواند یک درایور جدید بنویسد (میتواند از قبل در سیستم عامل نباشد!)

* **دیوایس کنترلر :** بین سخت افزار (دیوایس) و درایور قرار دارد و جزئی از سیستم عامل نیست و سخت افزار است ، دارای بافر برای انتقال اطلاعات بین دیوایس و درایور است.

ب) kernel mode و user mode

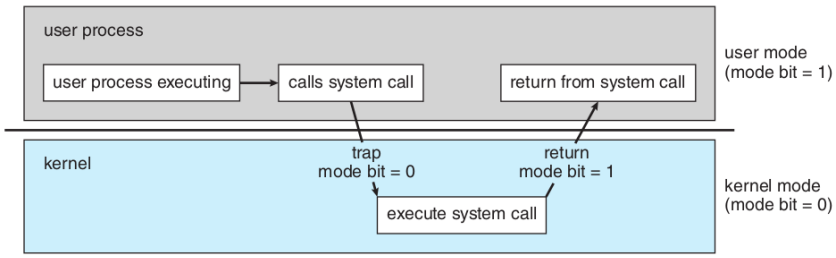

Figure 1.13 Transition from user to kernel mode.

از آنجایی که سیستم عامل و کاربرهایش باید منابع سخت افزار و نرم افزاری را بین خود تقسیم کنند، سیستم عامل موظف است به طریقی اطمینان حاصل کند که یک برنامه ی نادرست یا مخرّب نتواند در اجرای سایر برنامه ها یا خود سیستم عامل مشکل ایجاد کند پس باید بتواند اجرای کدهای سیستم عامل و کاربر تمایز قائل شود در نتیجه به حداقل دو مد متفاوت نیاز داریم :

وقتی کامپیوتر در حال اجرای برنامه های سمت کاربر است، سیستم در مد کاربر میباشد اما اگر برنامه ی کاربر به سرویسی از سیستم عامل نیاز داشته باشد(از طریق system call) سیستم باید از مد کاربر به مد کرنل برود (تصویر فوق این تغییر مد را نشان میدهد)

به علاوه یک بیت که mode bit خوانده میشود به سخت افزار کامپیوتر اضافه شده تا مد فعلی سیستم را مشخص کند به این صورت که در مد کرنل مقدار صفر دارد و در مد کاربر ، یک است .

ج) clustered systems و multiprocessor system

یک سیستم خوشه ای از چندین کامپیوتر برای انجام یک وظیفه محاسباتی بصورت توزیع شده استفاده می کند. سیستم های خوشه ای از دو یا چند سیستم جداگانه (از لحاظ فیزیکی) تشکیل شده که از طریق یک شبکه LAN به یکدیگر متصل شده اند. از طرف دیگر، سیستم های چندپردازنده ای یک سیستم متشکل از چندین پردازنده است که در آن دو یا چند پردازنده باهم کار میکنند تا بیشتر از یک برنامه را همزمان پردازش کنند. این به سیستم عامل کمک می کند تا کارهای مختلف را درمدت زمان کوتاهتری به پایان برساند. پردازنده ها در سیستم های چندپردازنده ای نسبت به سیستم های خوشه ای ارتباط تنگاتنگ تری دارند و به حافظه مشترک، باس مشترک و... متصل هستند

۱- سیستم های خوشه ای غالبا از تبادل پیام برای ارتباط با یکدیگر استفاده می کنند، در حالیکه پردازنده ها درسیستم چندپردازنده ای با استفاده از حافظه مشترک با هم ارتباط دارند.

۲- در سیستم های خوشه ای از شبکه بندی LAN استفاده شده است، ولی در سیستم های چندپردازنده ای مورد نیاز نمی باشد.

۳- در سیستم های خوشه ای، هدف اصلی فراهم کردن دسترسی بالاست. در حالیکه در سیستم های چندپردازنده ای هدف صرفه جویی در زمان و بالا بردن گذردهی است

د) system call و system call API

سیستم کال یک رابط برای سرویس هایی است که سیستم عامل ارائه میدهد. این کال ها عمدتا به صورت توابعی هستند که به زبان C و C++ یا اسمبلی نوشته شده اند. هر برنامه ساده ممکن است استفاده ی زیادی از سیستم عامل کند یعنی سیستم در هر ثانیه در حال اجرای میلیون ها سیستم کال است در حالی که برنامه نویس هیچ وقت این سطح از جزیئات را نمی بیند! معمولا برنامه نویسان برنامه هایشان را بر اساس یک API انجام میدهند. API لیستی از توابع موجود برای استفاده ی برنامه نویس ، پارامترهای ورودی این توابع و نوع خروجی های توابع را مشخص میکند. سه مدل API داریم: ویندوز ، POSIX و جاوا

در پشت صحنه ، بعد از اجرای اینترفیس سیستم کال ، سیستم کال حقیقی در کرنل فراخوانی میشود مثلا برای ویندوز تابع CreatProcess() در حقیقت سیستم کال NTCreatProcess() درکرنل ویندوز را فراخوانی میکند.

دلیل استفاده از API ، قابل اجرا بودن برنامه ها روی سیستم های مختلفی که از API یکسان پشتیبانی میکنند ، است. به علاوه API برنامه نویسی را راحت تر میکند.
(هرچند که در حقیقت API ویندوز و POSIX خیلی شبیه به سیستم کال های اصلی مورد استفاده در سیستم عامل های ویندوز و لینوکس است)

ه) UEFI و BIOS

بعضی سیستم ها از بوت چند مرحله ای استفاده میکنند : موقعی که کامپیوتر روشن میشود ابتدا BIOS اجرا میشود و یک بوت لودر دیگر را لود میکندکه در بوت بلاک قرار دارد و این بوت لودر ثانویه ، آدرس سیستم عامل در storage را میداند و آن را به RAM می آورد تا شروع به اجرا شدن کند.
کامپیوترهای امروزی UEFI را جایگزین BIOS کرده اند. مزیت UEFI نسبت به BIOS ، پشتیبانی بهتر از سیستم های ۶۴ بیتی و دیسک های بزرگتر است ولی مهم ترین مزیت آن این است که UEFI یک بوت منیجر کامل است(چند مرحله ای نیست) در نتیجه سریع تر از بوت منیجرهای چندمرحله ای است(مث BIOS)

۲- نقش مجازی سازی در محاسبات ابری ؟

مجازی سازی یک تکنولوژی است که به ما اجازه میدهد که سخت افزار یک سیستم واحد را بین چندین محیط اجرایی به گونه ای تقسیم کنیم که گویا هر کدام از این محیط های اجرایی روی یک سیستم مجزا در حال اجرا هستند(تجرید) (همچنین به کمک VM manager که بین سخت افزار و VMها قرار میگیرد و نحوه ی تقسیم منابع سخت افزاری بین VMها را مدیریت میکند (مدیریت منابع))
محاسبات ابری گونه ای از محاسبات است که storage ، computation و حتی applications را به عنوان سرویسی از اینترنت ارائه میدهد. در حقیقت محاسبات ابری تعمیمی از مجازی سازی است به این صورت که منابع سخت افزاری و نرم افزاری متعددی به صورت فیزیکی در نقطه ای از جهان وجود دارد و کاربران زیادی که به اینترنت دسترسی داشته باشند میتوانند به این منابع متصل شوند و سرویس های مورد نیازشان را دریافت کنند حال اینکه از دید سرویس دهنده ، تمام کاربران متصل شده مثل یک VM هستند یعنی هر دو ویژگی تجرید (از دید کاربر که قصد استفاده از این ابر را دارد ، گویا در حال استفاده از یک سیستم مجزا به تنهایی است) و مدیریت منابع (به این صورت که این منابع فیزیکی

حقیقی موجود به گونه ای عادلانه بین کلاینت ها تقسیم میشود) که در مجازی سازی بیان شد ، در محاسبات ابری نیز قابل تعریف هستند.

۳- درایور از طریق کدام مفهوم پیاده میشود ؟ کرنل ، ماژول یا سیستم کال ؟

دیوایس درایور از طریق مفهوم ماژول پیاده سازی میشود ، به این صورت که ما برای هر سخت افزار یک درایور خاص تهیه میکنیم و کرنل ممکن است از ابتدا درایور نداشته باشد و در حقیقت فقط شامل سرویس های سیستم عامل باشد و ما باید به وسیله ی ماژول های مربوطه، درایورهای مورد نیاز را در کرنل نصب کنیم.

۴- آیا ممکن است بین انتقال داده توسط DMA و اجرای همزمان برنامه توسط CPU تداخلی رخ دهد؟در چه شرایطی ؟ اولویت دسترسی به حافظه با کدام است ؟

بله امکان تداخل هست اگر CPU و DMA بخواهند همزمان به حافظه دسترسی پیدا کنند ، مشکل ایجاد می شود. CPU و DMA برای memory bus cycle رقابت می کنند. کنترلر حافظه سعی خواهد کرد bus cycle را بین CPU و DMA به صورت عادلانه اختصاص دهد. بنابراین ، هر برنامه CPU که قادر به استفاده از تمام memory bus cycle باشد می تواند در حالی که DMA فعال است کندتر عمل کند.(در زمانی که کنترلر حافظه bus cycle را در اختیار DMA میگذارد ، بدیهی است که برنامه در حال اجرا در CPU در وضعیت انتظار قرار میگیرد تا interrupt مربوطه برای CPU ارسال شود.)

۵- ساختار ریزهسته ، لایه ای و یکپارچه را درمورد معیارهای زیر مقایسه کنید :

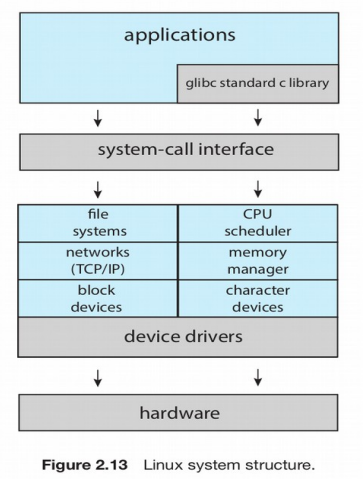

Figure 2.13 Linux system structure.

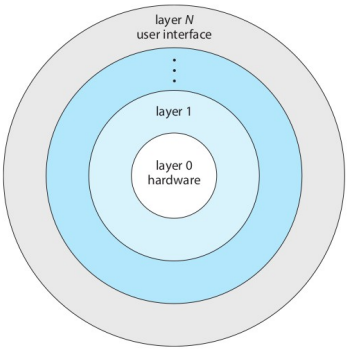

Figure 2.14 A layered operating system.

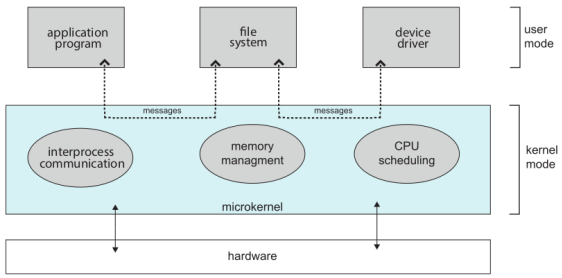

Figure 2.15 Architecture of a typical microkernel.

performance -

* در ساختار یکپارچه به دلیل سادگی آشکارش و سربار (overhead) بسیار کمی که در ارتباط با سیستم کال ها دارد و همچنین ارتباط درون کرنلی سریعش ، مزایای اجرایی متفاوت و زیادی دارد!
* در ساختار لایه ای به طور کلی از لحاظ performance ضعیف است به دلیل سربار (overhead) زیادی که در ارتباط لایه های متعدد نیاز است.
* در ساختار ریزهسته متاسفانه به دلیل سربار (overhead) زیاد توابع سیستمی (system-function) ، که شامل کپی کردن پیام های ارتباطی سرویس های سطح کاربر و سویئچ کردن بین پروسس هاست، از لحاظ performance قوی نیست

- نحوه ارتباط برنامه های سطح کاربر با خدمات سیستم عامل

* در ساختار یکپارچه ، اپلیکیشن ها معمولا از کتابخانه ی glibc که یک کتابخانه ی استاندارد زبان C است برای ارتباط با اینترفیس سیستم کال ها و کرنل استفاده میکنند. (مطابق تصویر ۲.۱۳)
* در ساختار لایه ای همانطور که از نامش پیداست ، ارتباط اپلیکیشن ها با خدمات سیستم عامل از طریق لایه ها انجام میشود به این صورت که رابط کاربری در بالاترین لایه (لایه ی N) قرار دارد و میتواند از توابع و سرویس های لایه های زیرینش استفاده کند و همین طور تا به پایین ترین لایه یعنی سخت افزار(لایه ی صفرم) برسیم!
* در ساختار ریزهسته ، کرنل فقط شامل CPU scheduling و memory management و ارتباط های بین پروسسی است ، یعنی یک سری از کامپوننت هایی که قبلا در کرنل بوده اند، در این ساختار به عنوان برنامه های سطح کاربر تعریف شده اند(تصویر ۲.۱۵) پس برای استفاده از خدمات سیستم عامل ، درواقع باید با یک سری از همین کامپوننت هایی که گفتم در ارتباط باشیم و این ارتباط از طریق رد و بدل کردن پیغام هایی بین این برنامه هاست(ارتباط مستقیم ندارند همچنین این پیام ها از طریق کرنل جابه جا میشوند)

Debugging and verification -

* در ساختار یکپارچه ، دقیقا به دلیل یکپارچه بودنش ، هرگونه تغییر در هر قسمتی از کرنل ، قسمت های دیگر را هم تحت تاثیر میگذارد پس عملیات خطایابی و بررسی صحت عملکرد بسیار دشوار میشود

به این دلیل که در صورت یافتن هرگونه باگ و اصلاح آن ممکن است سایر قسمت ها دچار تغییر یا اشکال شوند.

* در ساختار لایه ای و ریزهسته به دلیل ماژولار بودنشان ، عملیات خطایابی و بررسی صحت عملکرد بسیار راحت میباشد. مثلا در مورد ساختار لایه ای ، دیباگ کردن را از لایه صفرم که سخت افزار است آغاز میکنیم و بعد از بررسی سالم بودن قطعات به دیباگ کردن لایه ی اول میپردازیم. از آنجایی که لایه ی صفرم کاملا بررسی شده و عملکرد درستی دارد پس هرگونه ایراد در این لایه دیده شود فقط مربوط به همان لایه است و همین طور تا بالاترین لایه به راحتی دیباگ میشوند. در مورد ریزهسته هم به نحو مشابه (بررسی هر ماژول و دیباگ کردن آنها به صورت مجزا) است.

۶- دو رویکرد متفاوت در برنامه نویسی مفسر فرمان(CLI) وجود دارد. درباره ی bash در این زمینه تحقیق کنید.

در برنامه نویسی CLI دو رویکرد داریم :

۱-Internal or built-in commands : وقتی که مفسرفرمان در خود شامل کدهایی باشد تا کامند مربوطه را اجرا کند

۲-External or script commands : وقتی که کامند ها از طریق برنامه های سیستمی پیاده سازی شوند

درباره bash ، از هر دو رویکرد استفاده شده است. مثلا در مورد دستور rm ، مفسر فرمان اصلا متوجه این دستور نمیشود و باید فایلی با عنوان rm را در RAM لود کند تا کدهای مربوطه اجرا شود و در نهایت نتیجه ی مربوطه حاصل شود. عمدتاً از اسکریپت برای راحتی کار استفاده میکنند به خصوص وقتی که قرار است چند دستور به صورت متوالی و مکرراً استفاده شود .

به عنوان مثال دستورات cd , ls و ... به صورت built-in و دستورات ps , pwd , rm و ... به صورت script هستند. (میتوان فایل کدهای این دستورات اسکریپت را در پوشه ی /bin پیدا کرد)

۷- ۴ مورد از توابع system call API برای فایل و دیوایس نام ببرید. در برخورد یکسان سیستم عامل با فایل و دیوایس چه معایب و مزایایی هست ؟

در تصویر زیر چند مورد از سیستم کال هایی برای مدیریت فایل و دیوایس در دو سیستم عامل به ترتیب از چپ به راست ، ویندوز و Unix آورده شده است.

File management	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()
Device management	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()

مزیت برخورد یکسان سیستم عامل با دیوایس و فایل در این است که تعداد سیستم کال ها به حداقل میرسد و برنامه نویسی و درایور نویسی را راحت تر میکند

عیب این طرز برخورد هم در این است که ، از آنجایی که دیوایس ها متفاوت هستند تعریف سیستم کال ها هم باید متفاوت باشد (مثلا عمل نوشتن یا write در پرینتر با صفحه نمایش متفاوت است) پس در حقیقت باید برای هر دیوایس مجددا این توابع را تعریف کنیم(مشابه مفهوم ارث بری از یک کلاس abstract که توابع تعریف شده در کلاس والد باید در هر یک از کلاس های فرزند تعریف شوند و تعریف این توابع در فرزندان میتواند متفاوت ازیکدیگر باشد)

تحویل در تاریخ ۱۳۹۸/۷/۲۶

مریم سعیدمهر