پاسخ تکلیف اول سیستم عامل

پاییز ۹۸

جواب سوال ۱:

الف)

Device Controller یک مجموعه‌ی سخت افزاری (شامل بافرهای محلی و ریجسترها متناسب با Device) است که مسئولیت انتقال داده بین دستگاه های جانبی متصل به سیستم و بافرهای محلی خود را دارد.

Device Driver یک مجموعه‌ی نرم افزاری است و مسئولیت برقراری ارتباط بین سیستم عامل و یک Device Controller را برعهده دارد. هر Controller برای ارتباط با سیستم عامل تنها می‌تواند از طریق Driver مربوط به خود اقدام کند.

بنابر تعاریف فوق می توان نتیجه گرفت که Device Deriver یک نرم افزار است که بین کرنل و Device Controller قرار می گیرد اما Device Controller یک سخت افزار است که بین دستگاه جانبی و Device Driver قرار می گیرد.

ب)

User mode: یک حالت اجرایی محدود است که به برنامه‌های سطح کاربر داده می‌شود. این حالت تعیین می‌کند که برنامه‌ی موردنظر نمی‌تواند هر عملی انجام دهد و یا به هر منبعی دسترسی داشته باشد.

Kernle mode: یک حالت اجرایی نامحدود است که معمولا به برنامه‌های سطح پایین که نیازمند ارتباط با سخت افزار هستند داده می‌شود و اختیارات آن به مراتب بیشتر از User mode است.

تفاوت این دو حالت: برنامه‌هایی که در User mode اجرا می‌شوند از لحاظ دسترسی به منابع دارای محدودیت هستند(تنها مقدار معینی می‌توانند استفاده کنند) و برای دسترسی به منابع باید از فراخوانی های سیستمی استفاده کنند(یعنی به صورت غیر مستقیم دسترسی دارند). در واقع هنگامی که برنامه در این حالت اجرا می شوداگر خطایی رخ دهد به راحتی

قابل بازیابی است و در صورت آسیب دیدن منابع، تنها منابع مربوط به همان برنامه آسیب می‌بیند و بقیه ی برنامه ها به کار خود ادامه می‌دهند.

برنامه هایی که در حالت Kernel mode اجرا می‌شوند، به کل فضای سخت افزار و سیستم، به صورت مستقیم(نه از طریق فراخوانی های سیستمی) دسترسی دارند. یعنی برنامه هایی که از این حالت بهره می‌بند می توانند به فضای حافظه‌ی تمام برنامه‌های دیگر نیز دسترسی داشته باشند. معمولا این سطح دسترسی تنها به کرنل سیستم عامل داده می‌شود زیرا یکی از وظایف آن مدیریت منابع سیستم است و از این رو دسترسی آن باید نامحدود باشد. در این حالت اگر خطایی رخ دهد احتمال دارد تنها محدود به آن برنامه نشود و ضمن مختل کردن کل کار سیستم، ممکن است به سخت افزار نیز آسیب برساند.

یکی از فواید تعریف این دو حالت در سیستم عامل، جلوگیری از تخریب ناآگاهانه‌ی سیستم توسط کاربر است(زیرا برنامه‌هایی که توسط کاربر اجرا می‌شوند معمولا دارای خطا هستند).

ج)

Multiprocessor System: به سیستم‌هایی گفته می‌شود که دارای دو یا تعداد بیشتری پردازنده هستند و از طریق باس سیستم به هم متصل هستند همچنین از حافظه‌ی فیزیکی و دستگاه‌های جانبی مشترک استفاده می‌کنند

یعنی تمام پردازنده‌ها تنها از یک RAM استفاده می‌کنند و فرآیندهای مختلف را به صورت همزمان اجرا می‌کنند.

Clustered System: به مجموعه‌ای از سیستم ها گفته می‌شود که از طریق شبکه‌ی محلی به هم متصل هستند و با همکاری هم فرآیندها را انجام می‌دهند.

ممکن است هر کدام از این سیستم‌ها خود Multiprocessor System نیز باشد.

در این سیستم‌ها، یک نرم‌افزار وظیفه‌ی هماهنگی بین سیستم‌های مختلف را برعهده دارد.

تفاوت این دو، با توجه با توضیحات داده شده مشهود است.

د)

System Call: هر گاه یک نرم‌افزار از سطح کاربر نیاز به دسترسی به منابع سیستم و سخت افزار را داشته باشد، عملیات مربوطه درون سیستم عامل به صورت یک سرویس روتین interrupt باید اجرا شود که به این عمل **فراخوانی سیستمی** می‌گویند.

System Call API: مجموعه‌ای از توابع با الگوی مشخصی از پارامترهای ورودی و مقادیر خروجی که امکان استفاده از فراخوانی‌های سیستمی را فراهم می‌آورد و ارتباط برنامه‌نویس با کرنل را تسهیل می‌بخشد.

تفاوت: System Call API فقط یک API برای استفاده از System Call هایی است که در سطح کرنل اجرا می‌شوند. در واقع جزئیات و پیچیدگی‌های فرآیند فراخوانی System Call را برای برنامه نویس از بین می‌برد.

ه)

BIOS: یک نرم‌افزار است که به هنگام آغاز به کار سیستم به عنوان یک واسطه بین سیستم عامل و سخت افزار عمل می‌کند و در واقع در نقش یک مفسر ظاهر می شود. وظیفه‌ی اول BIOS مقداردهی اولیه‌ی اجزای سخت‌افزاری به صورت یک به یک و تست سالم بودن آنها است و سپس به بارگذاری سیستم عامل می‌پردازد(Boot Loader یا برنامه بوت مبتنی بر BIOS را بارگذاری می‌کند.). BIOS برای راه‌اندازی سیستم عامل سکتور اول هارد دیسک را بررسی می‌کند و اگر Boot Loader را بیابد آن را به حافظه ی سیستم منتقل می‌کند تا اجرا شود. BIOS فقط می‌تواند دستورات ۱۶ بیتی را اجرا نماید. بنابراین برنامه بوت مبتنی بر BIOS باید ۱۶ بیتی باشد.

UEFI: همانند BIOS یک نرم‌افزار سطح پایین است که همان کار BIOS را انجام می‌دهد اما به شیوه‌ای متفاوت. UEFI می‌تواند چند سخت افزار را به صورت همزمان مقداردهی اولیه بدهد و علاوه بر آن توانایی اجرای دستورات ۳۲ و ۶۴ بیتی را نیز دارا است.

تفاوت: برنامه بوت مبتنی برBIOS تنها بر روی هارد دیسکی قابل نصب است که از نوع MBR باشد و نهایتا می‌تواند از ۴ عدد پارتیشن فیزیکی پشتیبانی کند اما UEFI بر روی هارد دیسکی نصب می‌شود که از نوع GPT باشد و می‌تواند تعداد بیشتری پارتیشن فیزیکی را پشتیبانی نماید. همچنین هر پارتیشن هاردی که MBR دارد حداکثر ۲ ترابایت است در صورتی که برای UEFI پارتیشن‌های بزرگ‌تر نیز می‌توانند ساپورت شوند.

سرعت UEFI به دلایل گفته شده در بالا از جمله اجرای دستورات ۳۲ و ۶۴ بیتی به مراتب بیشتر از BIOS است که البته برای کاربر چندان قابل مشاهده نیست.

UEFI برخلاف Bios که روی firmware باید ذخیره شود، روی دایرکتوری /EFI/ روی یک حافظه غیرفرار ذخیره می‌شود بنابراین UEFI می‌تواند روی حافظه فلشی روی مادربرد ذخیره شود و یا روی هارد باشد و یا از شبکه لود شود. بنابراین در UEFI هم به هرحال نیاز به کد کوچکی است که از روی firmware اجرا شده و خود UEFI را بوت کند.

در UEFI تمام اطلاعات لازم جهت بوت سیستم در فایل efi به جای firmware ذخیره می‌شود. این فایل در پارتیشن خاصی به اسم ESP قرار می‌گیرد. همچنین پارتیشن esp شامل بوت‌لودرهای سیستم‌عامل‌های مختلف نصب‌شده روی سیستم است.

از لحاظ امنیت نیز UEFI بهتر از BIOS عمل می کند زیرا با ارائه‌ی آپشن Secure boot می‌تواند از بارگذاری بدافزارها هنگام شروع به کار سیستم جلوگیری کند.

جواب سوال ۲:

تمام منابع یک Cloud از طریق مجازی‌سازی برای یک درخواست مشخص تخصیص داده می‌شوند.

برای مثال وقتی کاربر یک سیستم با مشخصات معلوم CPU، حافظه و نرم‌افزارهای آن از جمله سیستم عامل را از Cloud درخواست می‌کند، Cloud منابع درخواستی را به گونه‌ای

جدا می‌کند که کاربر حس می‌کند تنها با یک سیستم یکپارچه کار می‌کند حال آنکه ممکن است هر منبع از سیستم‌های فیزیکی متفاوتی فراهم شده باشد.

توضیحات بیشتر در مورد رایانش ابری و مجازی سازی: رایانش ابری یعنی استفاده از سرورهای مجازی که به وسیله ی تکنیک مجازی نرم افزارها و برنامه های گوناگون ، سازی بوجود آمده اند و دارای سیستم عامل می باشد. در این سرویس کاربران متوجه نمی شوند به کدام سرور از نظر محل فیزیکی و شبکه ای متصل هستند .در واقع در ساختار ابر کاربر متوجه نمی شود که سیستم داده و قدرت پردازش چه قدر است و کاربر ارتباط ، عامل، فضای ذخیره سازی ورودی خروجی دارد. در رایانش ابری از روش ها و تکنیک های مجازی سازی استفاده می شود چرا که فقط از این طریق می تواند به این خواسته ها دسترسی پیدا کند.

مخصوصاً در IaaS(Infrastructure as a Service)، سیستم از طریق مفهوم مجازی سازی ساخته و در اختیار کاربر قرار می‌گیرد.

در واقع مجازی سازی یکی از عناصر حیاتی در فناوری Cloud Computing است که به فراهم آمدن قابلیت های امروزی این تکنولوژی کمک شایانی کرده است.

مجازی سازی ممکن است در سطوح مختلفی انجام شود:

- Network Virtualization
- Storage Virtualization
- Data Virtualization
- Application Virtualization

بررسی اهمیت تجرید در استفاده از سخت‌افزار و مقایسه‌ی آن با سیستم عامل:

تجرید در سیستم‌عامل جهت استفاده از سخت افزار فراهم می‌شود که این سخت‌افزار روی یک سیستم قرار دارد.

اما در Cloud تجرید از سخت‌افزارها و نرم‌افزارهای متنوع و مجزایی فراهم می‌شود تا همان گونه که در قسمت قبل بیان شد، کاربر درگیر رابطه‌ی مستقیم با سیستم توزیع شده در نودهای Cloud نباشد.

توضیحات بیشتر:

به بیان ساده در مدیریت نرم‌افزار یک Cloud، با استفاده از ماشین‌های مجازی قرار گرفته بر روی سرورها، سیستم برای کاربران یک ماشین مجازی مهیا کرده و در نتیجه کاربر استفاده کننده از رایانش ابری بدون اطلاع از جزئیات سخت افزاری سمت سرورها، از این فناوری استفاده میکند.

از طرفی در سیستم عامل نیز برای خود کرنل مجرد سازی سخت افزار صورت می گیرد. به طور کلی تجرید در سخت افزار برای کاربر یا برنامه نویس موجب آن می شود که این اشخاص بدون نیاز به درگیری با جزئیات و پیچیدگی های سخت افزاری، به سخت‌افزار به عنوان یک Black Box نگاه کرده و با استفاده از API های تهیه شده از طرف شرکت‌های تولید کننده سخت‌افزارها، بتوانند به راحتی با این ابزار کار کنند.

حال به صورت مشابه در فناوری Cloud Computing نیز کاربر بدون سر و کله زدن با سخت‌افزار از امکانات این فناوری بهره می‌برد.

بدین صورت گفته میشود که مجرد سازی سخت افزار صورت گرفته است.

بررسی اهمیت مدیریت منابع در فناوری Cloud Computing و مقایسه آن با سیستم عامل:

مدیریت منابع در سیستم عامل، همه منابع موجود روی یک سیستم را به نرم‌افزارها فرآیندهای آن سیستم اختصاص می‌دهد ولی در Cloud، همه منابع موجود روی هزاران نود به فرآیندهای کاربران تخصیص داده می‌شود که این مساله خود موجب می‌شود مدیریت منابع روی Cloud به نسبت پیچیده‌تر شود. از طرفی منابع موردنیاز یک کاربر در کلاد

ممکن است روی سیستم‌های فیزیکی متنوعی قرار گرفته باشد در صورتی که در سیستم عامل، از منابع فیزیکی یک ماشین منفرد استفاده می‌شود.

از آنجا که کامپیوترهای ارائه دهنده‌ی سرویس Cloud در هر صورت منابع محدودی دارند، بنابراین باید درخواست‌های فراوان کاربران را میان منابع موجود خود توزیع نمایند و برای اجرای آنها واحدهای پردازشی را allocate نمایند.

جواب سوال ۳:

بسته به نوع سیستم عامل ممکن است از هرکدام از این روش‌ها برای پیاده‌سازی درایورها استفاده شده باشد. برای مثال در سیستم های یکپارچه، درایورها به عنوان بخشی از کرنل پیاده می‌شوند.

در سیستم های امروزی و به خصوص سیستم‌های مبتنی بر لینوکس، درایورها به صورت یک kernel module پیاده‌سازی می‌شوند.

این ماژول در صورت نیاز، Load شده و به کرنل اضافه می‌شود.

از طرفی با فراخوانی system call ها مانند open, read, write برای هر دیوایس، توابع موجود در ماژول درایور آن دیوایس اجرا می‌شوند. مثل این است که فراخوانی‌های سیستمی برای هر دیوایس را در درایور مربوطه‌اش پیاده‌سازی کرده باشیم.

جواب سوال ۴:

چنانچه در سیستم از معماری سوئیچ استفاده شده باشد، تداخلی به وجود نمی‌آید زیرا در این معماری اجزای مختلف می‌توانند به صورت همزمان با هم ارتباط برقرار نمایند و رقابتی برای اختصاص یک باس برای انتقال اطلاعات وجود ندارد.

اما در یک سیستم با معماری bus، ممکن است تداخل ایجاد شود:

در این حالت اگر CPU حین اجرای دستوری به دسترسی به حافظه (جهت fetch دستور یا داده) نیاز داشته باشد، می‌تواند تا اتمام عملیات DMA صبر کند که در این حالت فقط کمی زمان از دست می‌دهد ولی ارسال اطلاعات از DMA را نباید متوقف کنیم زیرا ممکن است اطلاعاتی حین ارسال از دست برود.

از طرفی نیز بهتر است DMA اطلاعات نهایی را برای RAM ارسال کرده تا در صورت نیاز، CPU بتواند به آخرین اطلاعات به روز شده دست یابد.

جواب سوال ۵:

کارایی:

در مدل یکپارچه با توجه به یکپارچه بودن سیستم عموما کارایی بالاتری نسبت به دو مدل دیگر وجود دارد. در مدل میکرو کرنل به علت overhead بالا ارتباط بین دو فرآیند یا سرویس‌های کاربر، سرعت و کارایی پایین می‌آید زیرا برای هر گونه ارتباط نیاز به ارسال پیام از طریق کرنل وجود دارد که این عملیات زمان‌بر و هزینه‌بر است. از طرفی در مدل لایه‌ای نیز کارایی ضعیف است زیرا هر برنامه‌ی کاربر باید به ترتیب لایه‌های زیادی را تا کرنل طی کند.

نحوه ارتباط برنامه های سطح کاربر با خدمات سیستم عامل:

در مدل یکپارچه عموما برنامه ها در یک سطح اجرا می شوند و بدین ترتیب برنامه های سطح کاربر مستقیما با کرنل در ارتباط هستند.

در مدل لایه‌ای ارتباط سطح کاربر از طریق لایه‌ی زیرین خود انجام می‌شود که این مساله Debugging و verification را به مراتب ساده‌تر می‌کند ولی overhead را بالا می‌برد.

در مدل میکرو کرنل نیز برنامه‌های سطح کاربر با خدمات سیستم عامل از طریق MicroKernel Message Passing در ارتباط می‌باشند و هیچ دو فرآیندی به صورت مستقیم ارتباط ندارند.

اشکال‌زدایی:

در مدل لایه‌ای، به دلیل لایه ای بودن ساختار کرنل و عدم وابستگی لایه های متخلف به هم، اشکال زدایی به مراتب ساده‌تر از روش‌های دیگر است زیرا اشکال و خطای احتمالی موجود در یک لایه مستقل از لایه‌های دیگر است و لایه‌ها به خوبی از هم مجزا شده‌اند.

در مدل یکپارچه به علت یکپارچه بودن سیستم و وابستگی بالای قسمت های مختلف سیستم به هم، عموما عملیات اشکال یابی و اشکال زدایی به مراتب از دو مدل دیگر سخت تر است.

در مدل میکرو کرنل هم به توجه به کوچک بودن کرنل و واگذاری بسیاری از به فضای کاربر، اشکال زدایی نسبت به حالت یکپارچه ساده تر است و خرابی یک سرویس تاثیری روی بقیه سرویس ها ندارد زیرا اکثر عملیات به صورت سرویس به بخش کاربر منتقل شده است.

جواب سوال ۶:

در برنامه نویسی از هر دو رویکرد(built-in و غیر built-in) استفاده شده است.

برای مثال به منظور پیاده‌سازی دستور cd از رویکرد built-in استفاده شده است اما برای دستورات دیگر همانند ls و top و ... دستوراتی هستند که با رویکرد غیر built-in پیاده شده‌اند.

دلیل این انتخاب: اگر از رویکرد غیر built-in استفاده کنیم آنگاه CLI منعطف خواهد بود و به سادگی می‌توان آن را توسعه داد (برای اضافه کردن ویژگی و دستورات جدید نیاز به

کامپایل دوباره نیست) اما اگر با رویکرد built-in نوشته شده باشد آنگاه سرعت اجرای دستورات نسبت به حالت قبل بیشتر است ولی توسعه‌ی آن سخت خواهد بود.

در CLI برای آنکه هم انعطاف و توسعه‌پذیری و هم سرعت اجرا را داشته باشیم، دستورات کم و پایه‌ای به صورت built-in پیاده شده است ولی بقیه دستورات به صورت غیر built-in استفاده شده است.

نکته: تمام دستورات غیر built-in دارای یک فایل باینری هستند که CLI با بارگذاری و اجرای فایل باینری، آن دستور را انجام می‌دهد. همان گونه که در آزمایشگاه گفته شد، شما می‌توانید با استفاده از دستور which فایل باینری دستورات را پیدا کنید، بنابراین اگر دستور which بتواند فایل باینری یک دستور را پیدا کند یعنی آن بخش به صورت غیر built-in پیاده شده است و در غیر این صورت یعنی به صورت built-in پیاده سازی شده است. از طرفی دستورات built-in چون همراه با CLI بارگذاری می‌شوند به دلیل عدم وجود فایل باینری، امکان بارگذاری آنها با سطح دسترسی متفاوت وجود ندارد(با همان سطح دسترسی bash اجرا می‌شوند.) برای درک بهتر دستور زیر را امتحان کنید:

Sudo cd /home

جواب سوال ۷:

چند مورد از توابع System Call API:

Close, open, read, write, …

یکی از مبانی مهم Unix و مشتقات آن "Everything is a file" است.

به این معنی که سیستم عامل‌های Unix Based یک فایل سیستم از طیف وسیعی از اسناد، دایرکتوری‌ها، هارد درایوها، کیبوردها و ... دارند.

به عنوان مزیت این ویژگی باعث می‌شود که برنامه‌ها و API های یکسانی برای طیف وسیعی از منابع استفاده شود که باعث سادگی کار با System Call ها می‌شود و برای یک توسعه-دهنده استاندارد خوب و ساده‌ای از این طریق برای نوشتن درایورها و استفاده از آنها فراهم

می‌شود. برای مثال برای دیوایس‌ها هم همان توابع read, write, open, ... که در فایل‌ها استفاده می‌شود به کار می‌رود و توسعه‌ی هر درایور به معنی پیاده‌سازی بدنه‌ی این توابع است.

عیب این کار نیز پیچیده‌تر شدن ساختار فایل‌سیستم‌ها برای پوشش دادن به پیاده سازی درایور جهت انواع دیوایسها می‌باشد.