پاسخ‌نامه تمرین سوم درس سیستم‌های عامل

دکتر زرندی

پاییز ۹۹

۱- همانطور که می‌دانید، از دو روش بر مبنای کرنل (*Kernel-based*) و برنامه‌های سیستمی (*System Program*) برای پیاده‌سازی مفسر دستورات (*Command Interpreter*) استفاده می‌شود. ضمن توضیح نحوه پیاده‌سازی هر کدام از این دو روش، استفاده از آن‌ها را با ذکر دلیل توجیه کنید.

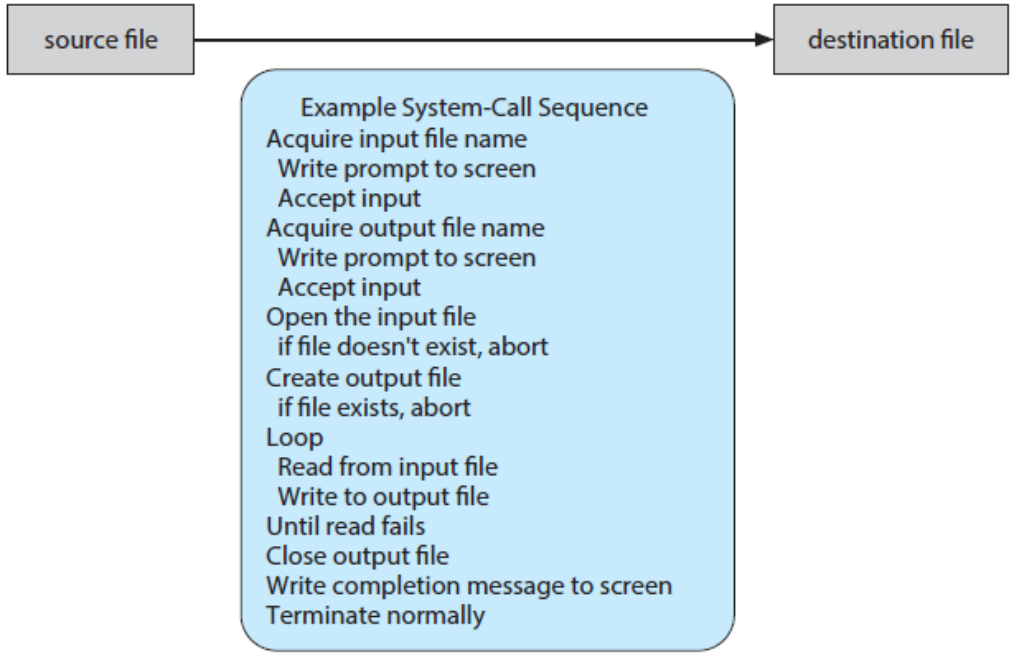
در روش بر مبنای کرنل، مفسر دستور خود شامل کدهای لازم برای اجرای دستورات می‌باشد. برای مثال، دستور پاک کردن فایل باعث می‌شود که مفسر دستور به یک قسمت از کد خودش بپرد که پارامترها را تنظیم کرده و *system call* مناسب را انجام می‌دهد. دلیل استفاده از این روش سرعت بالای آن می‌باشد.

در روش دیگر (که در *UNIX* از آن استفاده می‌شود)، بیشتر دستورات توسط برنامه‌های سیستمی پیاده‌سازی می‌شوند. در این حالت، مفسر دستور خود هیچ دستوری را نمی‌فهمد، بلکه از دستور استفاده می‌کند تا فایلی که باید در حافظه بارگذاری و اجرا شود را شناسایی کند. در این روش، برنامه‌نویسان به راحتی می‌توانند دستورات جدید را به سیستم اضافه کنند. همچنین اندازه مفسر دستور کوچک باقی مانده و برای اضافه شدن دستورات جدید نیازی به ایجاد تغییرات در آن نیست.

۲- امروزه توسعه‌دهندگان به طور عمده از رابط‌های برنامه‌نویسی (*API*) استفاده می‌کنند.
الف) مزایای استفاده از این روش را بیان کنید.

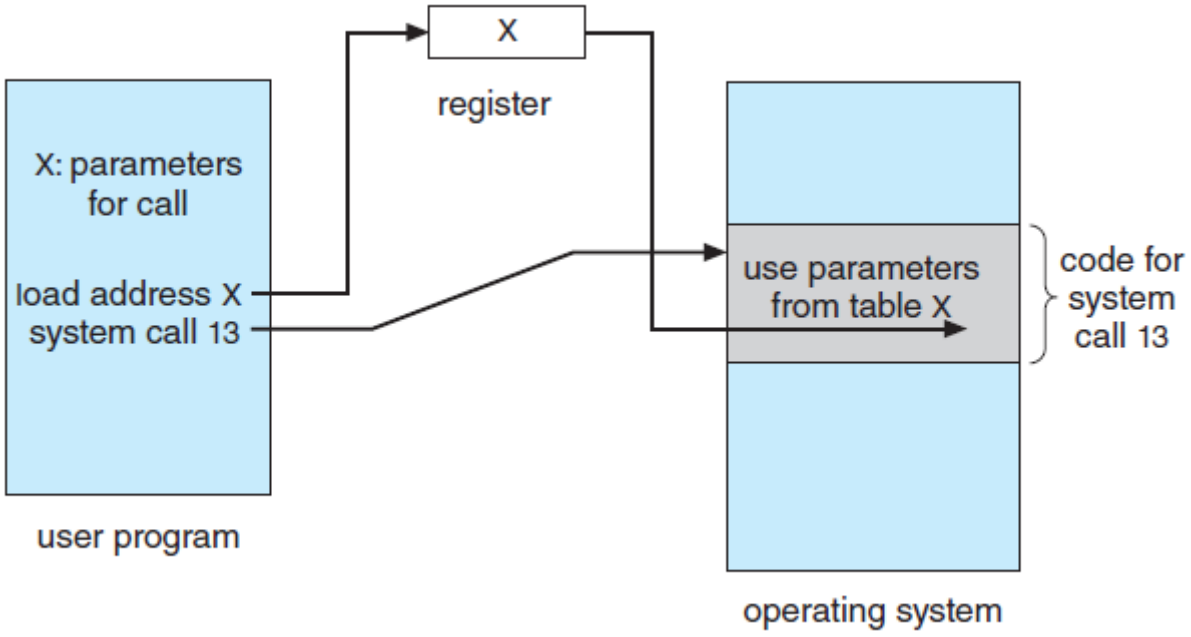
۱) قابل حمل بودن: برنامه‌نویسی که یک برنامه را با استفاده از رابط برنامه‌نویسی طراحی می‌کند، انتظار دارد که برنامه‌اش در هر سیستم دیگری که از آن رابط برنامه‌نویسی پشتیبانی می‌کند، اجرا شود.

۲) سادگی: رابط‌های برنامه‌نویسی معمولا بسیار ساده‌تر از *system call*ها بوده و جزئیات کمتری نسبت به آن‌ها دارند.



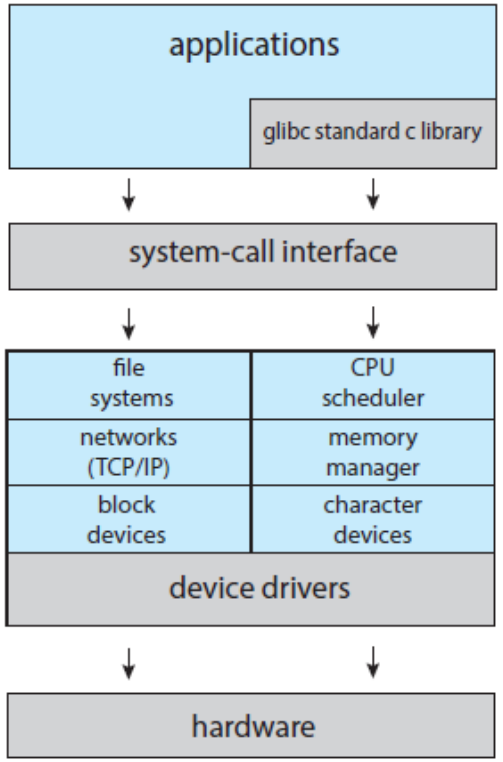
ب) سیستم‌عامل لینوکس از چند راه برای عبور متغیرها (*Parameter Passing*) استفاده می‌کند. آن‌ها ذکر کرده و توضیح دهید که استفاده ترکیبی از آن‌ها چه فایده‌ای دارد.

سیستم‌عامل لینوکس برای عبور متغیرها از روش‌های بر مبنای رجیستر، بلوک حافظه و پشته استفاده می‌کند. روش مبتنی بر رجیستر سریعتر بوده اما تعداد کمی از متغیرها را می‌تواند ذخیره کند. از طرف دیگر روش‌های بر پایه بلوک حافظه و پشته توانایی ذخیره تعداد بیشتری متغیر را دارند اما کند هستند. بنابراین سیستم‌عامل لینوکس برای تعداد ۵ متغیر یا کمتر از آن از روش بر مبنای رجیستر و برای بیشتر از ۵ متغیر از روش مبتنی بر بلوک حافظه استفاده می‌کند.



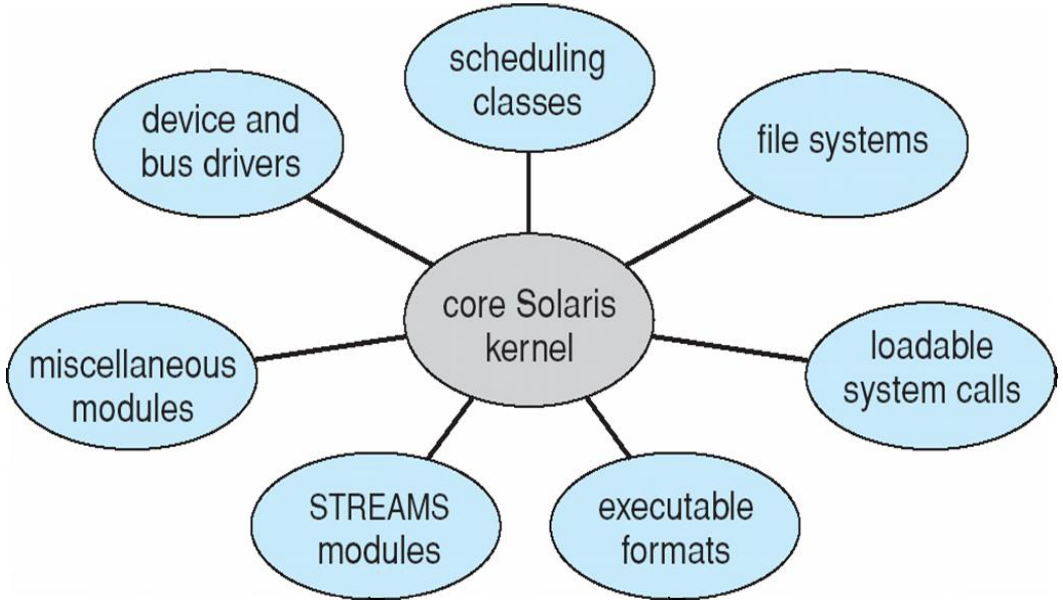
۳- در رابطه با ساختار سیستم‌های عامل به سوالات زیر پاسخ دهید.
الف) برای طراحی یک سیستم‌عامل با بیشترین سرعت و کارایی باید از چه ساختاری استفاده کنیم؟ چرا؟

از ساختار یکپارچه (*Monolithic*) استفاده می‌کنیم. زیرا سربار بسیار کمی در رابطهای *system call* آن وجود دارد و ارتباطات درون کرنل آن سریع است. به همین جهت دارای کارایی و سرعت بالایی می‌باشد.

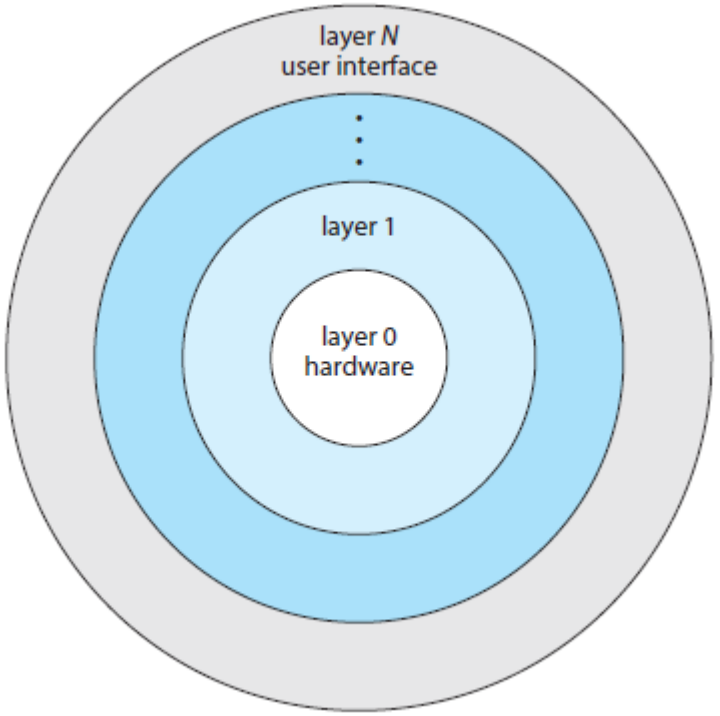


ب) فرض کنید می‌خواهیم تغییراتی که یک تیم توسعه سیستم‌عامل در آن ایجاد می‌کند، تاثیری بر کار دیگر تیم‌ها نداشته باشد. در این صورت چه رویکردی را برای طراحی اجزاء این سیستم‌عامل پیشنهاد می‌کنید؟ چه راه‌هایی برای پیاده سازی آن وجود دارد؟

برای این منظور باید از رویکرد اجزاء ***loosely coupled*** استفاده کنیم. به این ترتیب تغییرات در یک جزء تنها همان جزء را تحت تاثیر قرار می‌دهد و به پیاده‌سازان سیستم اجازه می‌دهد که با آزادی عمل بیشتری به ساخت و تغییر در اجزاء سیستم بپردازند.
برای این نوع پیاده‌سازی می‌توان از ساختارهای لایه‌ای (***Layered Approach***) و ماژولار (***Modules***) استفاده کرد.



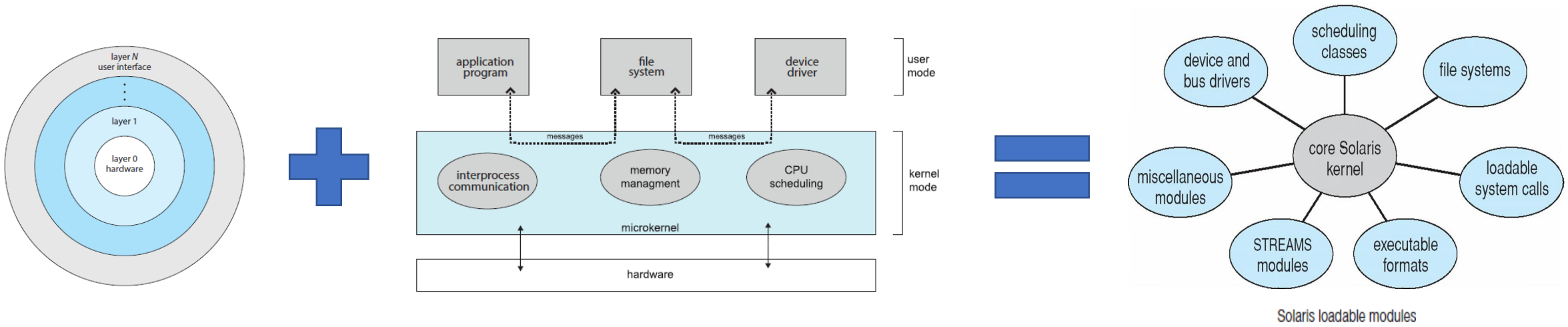
Solaris loadable modules



ج) ساختار ماژولار (*Modules*) به چه ساختارهای دیگری شباهت دارد؟ چگونه معایب آن‌ها را برطرف کرده است؟

ساختار *Modules* مانند یک سیستم لایه‌ای است که هر بخش آن *interface* حفاظت شده‌ای دارد. اما از ساختار لایه‌ای منعطف‌تر است زیرا هر ماژول آن می‌تواند هر ماژول دیگری را صدا کند.

این ساختار همچنین به رویکرد *Microkernel* نیز شباهت دارد، زیرا ماژول اولیه تنها شامل کارکردهای اصلی و نحوه بارگذاری و ارتباطات با دیگر ماژول‌ها می‌باشد. اما کارایی آن از *Microkernel* بیشتر است، چون برای ارتباطات میان ماژول‌ها نیازی به *message passing* نیست.



Solaris loadable modules

۴- در درس با معایب ساختار *Microkernel* از جمله سربار ارتباطات میان سرویس‌های مختلف آشنا شدید. راجع به چگونگی برطرف شدن این مشکل در سیستم‌عامل *macOS* تحقیق کنید.

ارسال پیام میان سرویس‌های مختلفی که در فضای آدرس‌دهی کاربر قرار دارند، باعث افت کارایی در ساختار *Microkernel* می‌شود. برای رفع این مشکل در سیستم‌عامل *macOS*، سرویس‌های مختلف کرنل مانند *Mach*، *BSD* و *I/O kit* در یک فضای واحد آدرس‌دهی ترکیب شده‌اند. به این ترتیب *message passing* همچنان انجام می‌شود اما چون همه سرویس‌ها در یک فضای آدرس‌دهی قرار دارند، نیازی به کپی کردن اطلاعات نیست.