به پرسش های زیر با ذکر تمام جزئیات خواسته شده پاسخ کامل بدهید.

(آ) سیستم عاملی را فرض کنید که در آن تنها یک فرآیند در کرنل وظیفه ی نظارت بر امنیت و صحت اجرای نخ های سطح کرنل را بر عهده دارد. حال فرض کنید در کنار چند نخ دیگر که از پیش در این سیستم موجود بودند، یک نخ جدید ایجاد می شود و کرنل برای ساخت نخ ها از معماری یک به یک تبعیت می کند. اکنون اگر نخ جدید نیاز به فراخواندن یک فراخوان سیستمی پیدا کند که نیاز است فرآیند نظارتی نام برده شده در بالا بر روی آن نظارت کند، به پرسش های کار پاسخ دهید:

i. سیستم برای نظارت بر نخ جدید باید به چه شکل عمل کند؟ معماری تولید نخ یک‌به‌یک است. این یعنی به ازای هر نخ تولید شده در فضای کاربر، یک نخ متناظر در فضای کرنلی تولید می‌شود. پس فرآیند نظارتی باید بتواند روی تک‌تک نخ‌های کرنل نظارت داشته باشد. برای نظارت داشتن و رصد کردن فراخوان‌های سیستمی باید یک الگو برای مدیریت انتخاب کنیم. سناریوهای گوناگونی می توان متصور شد.

**صد کردن مستقیم (Direct Monitoring)** فرآیند نظارتی با استفاده از system hooks تمامی فراخوانی‌های سیستمی از جانب نخ‌های کرنل را رصد می‌کند. باید ساختار هندلر فراخوانی‌های سیستمی را تغییر داد. به گونه‌ای که به فرآیند نظارتی قبل از اینکه روال مناسب را اجرا کند خبر داده شود. تا بعد از تایید این فرآیند نظارتی اجرای روال صورت گیرد.
**مزیت :** تاخیر ندارد و به صورت Real-Time رصد کردن را در اختیار دارد.
**عیب :** سربار زیاد به دلیل اینکه هر فراخوان سیتمی نیاز به دخالت این فرآیند نظارتی را می‌طلبد.

**مکانیزم اعلان (Notification Mechanism)** در این روش، وظیفه‌ی باخبر کردن فرآیند نظارتی، هنگام رخ دادن فراخوان سیستمی را به کرنل واگذار می‌کنیم. ر این روشی بعضی از فرآیندّا به خود کرنل واگذار می‌شوند. یعنی یک رصد اولیه انجام بدهد و اگر پتانسیل وجود یک خطر امنیتی را حس کرد به فرآیند نظارتی اعلان بفرستد.
**مزیت :** کرنل برای فرآیندهای مربوط به فرآیند نظارتی آن را باخبر می‌کند. در نتیجه سربار کمتری برای فرآیند نظارتی داریم.
**عیب :** تاخیر به دلیل این که سیستم اعلان، پتانسیل تأخیر را داراست.

**استفاده از صف (Queue-Based Monitoring)** فراخوان‌های سیستمی که نیاز به نظارت دارند به ترتیب در یک صف نگهداری می‌شوند. پیاده سازی آن میتواند از طریق یک صف اشتراکی بین کرنل و فرآیند نظارتی صورت گیرد.
**مزایا :**
- رصد کردن را ساده‌تر می‌کند و پیچیدگی روش‌های قبل را ندارد.
- اجازه می‌دهد که فرآیند نظارتی با سرعت خودش این فراخوان‌های سیستمی را مدیریت کند.

**عیب :** استفاده از صف نیاز به همگام‌سازی دارد وکرنه باعث وضعیت رقابتی می‌شود.

ii. این سازوکار چه تاثیری بر چندوظیفگی سیستم عامل خواهد گذاشت؟
- منطقا یک سربار اضافه میشود. چون باید نخ‌های اجرایی پیوسته رصد شوند.
- فرآیند نظارتی، یک منبع است. فراخوان‌های سیستمی باید از دروازه‌ی این فرآنید رد شوند. همین رقابت بر سر منبع نیز میتواند یک نقطه ضعف باشد.
- برای جلوگیری از بن‌بست‌ها (Deadlocks) و شرایط رقابتی (Race Conditions) برای وقتی که با استفاده از صف، فرآیند نظارتی را پیاده‌سازی میکنیم، باید بتوانیم همگام‌سازی لازم را صورت بدهیم.
- یک مقوله برای مدیریت به سیستم کامپیوتری اضافه شده است. همین موضوع مقیاس‌پذیری سیستم کامپیوتری را کاهش می‌دهد.

iii. راهکار شما برای حل مشکلات به وجود آمده چیست؟
- به جای بررسی کردن هر فراخوان سیستمی، اطلاعات که مربوط به نخ را کش کنیم و برای بررسی‌های آتی آن‌ها را ملاک قرار بدهیم.

- شدت رصد کردن را تغییر بدهیم. یک سامانه‌ی اولویتی به فراخوان‌های سیستمی بدهیم تا فقط آن دسته از فراخوان‌های سیستمی که احتمال خطای امنیتی زیادی دارند، تماما رصد شوند و این رصد برای بقیه‌ی فرآیندها، با احتمال خطای امنیتی کمتر، کمتر باشد و یا کلاً نباشد(یعنی فرآیند را امن تشخیص دهیم).
- از موازی سازی استفاده کنیم. به گونه‌ای که برای رصد کردن، اجرای نخ‌ها متوقف نشود. و اجرای فرآیند نظارتی در پیش‌زمینه صورت بگیرد.

(ب) فرض کنید داده ای در یک بخش از حافظه (یک فایل) وجود دارد و چند فرآیند نیاز به دسترسی دقیقا همزمان به آن دارند، همچنین همچنین فرض کنید سیستم عامل هیچ تضمینی بر روی کنترل دسترسی همزمان و عدم خرابی داده های این بخش از حافظه (فایل) نمی دهد. سازوکاری طراحی کنید که با استفاده از آن بتوان برنامه ای نوشت که در چنین سیستمی به هنگام نیاز به دسترسی به یک حافظه (فایل) هیچ خرابی رخ ندهد (امکان استفاده از تابع ()print و همچنین ساختارهای mutex و سمافور را نخواهید داشت)
سیستم عامل هیچ تضمینی برای مدیریت و هماهنگی فایل‌ها به ما نمی‌دهد و از ابزارهای mutex و semaphore بعنوان ابزارهای نرم‌افزاری شایع برای این کنترل نمیتوانیم استفاده کنیم. از تابع sleep هم نمی‌توان استفاده کرد. میتوانیم از فایل‌های قفل‌کننده (Lock Files) استفاده کنیم. فرآیندها برای دسترسی به حافظه‌ی اشتراکی ابتدا وجود فایل قفل‌کننده‌ی مربوطه به آن حافظه‌ی اشتراکی را بررسی میکنند. این فایل به عنوان نشانگر یک قفل عمل میکند.

- اگر فایل قفل‌کننده وجود داشته باشد، فرآیند صبر می‌کند تا قفل آزاد شود. در مدتی که فرآیند صبر می‌کند تا فایل قفل آزاد شود، باید در busy waiting بماند که بهینه نیست.
- اگر فایل قفل‌کننده وجود نداشته باشد، فرآیند فایل قفل‌کننده را ایجاد می‌کند.
  - در ایجاد این قفل باید از یک عملیات *اتمیک* استفاده کنیم.
  - اتمیک بودن سبب می‌شود تا هنگامی که چند فرآیند بخواهند همزمان یک فایل قفل ایجاد کنند فقط یکی از آنان موفق بشود.
  - مثل تابع open با فلگ O_CREATE | O_EXCL در سیستم‌های POSIX
  - یا با directory creation که در بیشتر فایل سیستم‌ها اتمیک است.
- پس از ایجاد حافظه، فرآیند می‌تواند عملیات مورد نظرش را (خواندن یا نوشتن) در حافظه‌ی اشتراکی پیاده کند.
- پس از اتمام کار فرآیند فایل قفل‌کننده را حذف می‌کند تا بقیه‌ی فرآیندها هم بتوانند به آن حافظه‌ی اشتراکی دسترسی پیدا کنند.

(ج) در مدل تبادل پیام با توجه به اینکه حافظه برای نگهداری پیام محدود است، سازوکاری ارائه دهید که بتوانیم (با تقریب خوبی) تعداد نامحدودی پیام در صندوق پیام بفرستیم. میتوانیم از مفهوم بافر نامحدود بهره ببریم. این بافر پیاده‌سازی‌های گوناگونی دارد. از جمله پیاده‌سازی با لیست پیوندی یا با لیست‌های داینامیک. در تئوری می‌توانیم تا بی‌نهایت در این بافر پیام درج کنیم. اما نکته‌ی مهم اینست که در عمل حافظه محدودیت دارد. لیست پیوندی تا بینهایت پیش نمی‌رود. برای همین میگوییم با تقریب خوبی این حافظه نامحدود است.
ایده‌ی دیگر برای بهبود حافظه اینست که در این پیاده‌سازی یک بیشینه حافظه تعریف کنیم و پس از اینکه بافر پر شد با یک سیاست مشخص شروع به جایگزینی یک پیام انتخابی با پیام جدید کنیم. مثلا سیاست انتخابی می‌تواند این باشد: پیامی که از همه زودتر به بافر اضافه شده است از همه زودتر هم خارج شود(مانند یک صف سیاست FIFO داشته باشد). میتوانیم از یک صف حلقوی (یا لیست پیوندی حلقوی) برای این پیاده‌سازی استفاده کنیم. راه‌های زیادی برای این پیاده‌سازی پیش روی ماست.
هر کدام ممکن است نکات مثبت و منفی گوناگونی داشته باشند. مثلا یک پیاده‌سازی ممکن است سرعت دسترسی به پیام بیشتری داشته باشد و در ازای آن مجبور به حذف پیام‌های بیشتری شود.
حتی میتوانیم یک قدم جلوتر برویم و از یک حافظه‌ی جانبی (مانند swap در فرآیندها) برای افزایش گنجایش بافر استفاده کنیم. این فیچر پیچیدگی را منطقا زیاد می‌کند اما در تئوری با این روش می‌تونیم پیام‌های بیشتری را به صورت همزمان ذخیره کنیم.

(د) شرایطی را بیان کنید که در آن موازی سازی به هیچ وجه نمی تواند به تسریع محاسبات کمک کند.

- فرض کنید که میخواهید یک فرآیند بسیار سبک را اجرا کنید. در این حالت سربار اجرای موازی آن از سودی که دارد بیشتر می‌شود.
- فرض کنید چند تسک که میخواهید اجرا کنید به هم وابستگی زیادی داشته باشند. این شرایط باعث محدودیت در اجرای موازی میشود. و مانند مورد بالا در نهایت ممکن است سربار آن بسیار بالا برود.
- الگوریتم‌هایی وجود دارند که مانند یک دنباله و به هم متصل اجرا می‌شوند. در این الگوریتم اجرای موازی و شکستن فرآیند به چند تسک ممکن نیست.

(ه) تحقیق کنید که در کدام سیستم عامل ها مدل های چندنخی دو سطحی و چند به چند پیاده سازی شده است و همچنین آیا هنوز استفاده ای از این مدل ها صورت می گیرد یا خبر؟ علت آن را نیز بیان کنید.

**مدل چند به چند (Many-to-Many)**

- این مدل در برخی نسخه‌های قدیمی‌تر از Solaris (مانند Solaris 2.x) و IRIX پیاده‌سازی شده است.
- در برخی از سیستم‌های قدیمی‌تر یونیکس از این مدل استفاده می‌شد.
- این مدل به تدریج کنار گذاشته شد و امروز به ندرت از آن استفاده می‌شود

**مدل دو سطحی (Two-Level)**

- در نسخه‌های قدیمی‌تر FreeBSD از این مدل استفاده میشد.
- امروزه به ندرت استفاده می‌شود.

**چرا این مدل‌ها دیگر رایج نیستند ؟**

۱. پیچیدگی مدیریت: نگاشت نخ‌های کاربر به نخ‌های هسته نیازمند الگوریتم‌های پیچیده است که می‌تواند مشکلاتی مانند گرسنگی منابع (Resource Starvation) ایجاد کند.

۲. بهبود سخت‌افزار: پردازنده‌های چند هسته‌ای مدرن و معماری‌های جدید به مدل‌های ساده‌تر (مانند یک به یک) اجازه می‌دهند به طور کارآمدتری عمل کنند

۳. بهبود سیستم‌های عامل: هسته سیستم‌عامل‌های مدرن قابلیت مدیریت کارآمد نخ‌ها را در مدل یک به یک فراهم کرده است.

(و) سیستم عاملی را فرض کنید که در آن هیچ امکان اشتراک گذاری داده ای میان فرآیند ها وجود نداشته باشد. در این صورت بگویید چگونه می توان داده ای را میان چند فرآیند به اشتراک گذاشت.
اگر نتوان از روش‌های مستقیم برای اشتراک‌گذاری اطلاعات بین فرآیندها استفاده کرد، باید برای انتقال داده از روش‌های غیرمستقیم استفاده کنیم.

- فایل‌های موقتی در دیسک ذخیره شده باشند و فرآیندها بتوانند از طریق آن با هم داده رد و بدل کنند یا میتوانیم سیستم‌عامل را قاطی این مکانیزم کنیم و بخشی از حافظه‌ی اصلی که به یک فایل مرتبط می‌شود در اختیار فرآیندها قرار بگیرد.
- یا مثلا با استفاده از پایگاه‌های داده این عمل صورت گیرد. فرآیندها اطلاعات را در یک پایگاه داده ذخیره و بازیابی کنند.

(ز) فرض کنید در یک سیستم عامل هیچ تضمینی بر روی عدم هم پوشانی بخش داده و هرم نباشد. سازوکاری ارائه کنید که با استفاده از آن در برنامه ی خود بتوانیم از همپوشانی جلوگیری کنیم.
اگر سیستم‌عامل این عدم هم‌پوشانی را مدیریت نمی‌کند، به این معنی است که ما باید به صورت دستی این محدودیت و مرز را مدیریت کنیم. اگر مطمین باشیم که حافظه‌ی هرم به صورت کاملا ایزوله از دیگر بلاک‌های حافظه استفاده می‌شود، ی‌توانیم مطمین باشیم که با بخش داده نیز هم‌پوشانی و تداخلی ندارد. اول از همه یه بخشی از حافظه را به هرم تخصیص می‌دهیم. حال توابع شخصی‌سازی شده‌ای برای تخصیص حافظه از هرم یا آزاد کردن از هرم بنویسیم. با این اقدامات می توانیم مطمین شویم که هرم هیچ تصرفی در بخش داده نخواهد کرد.

- می‌توانیم یک گارد محافظتی نیز اطراف حافظه‌ی هرم قرار دهیم تا مطمین شویم از بخش داده تصرفی در حافظه‌ی هرم صورت نمی‌گرد. . به این گارد صفحات حافظه‌ای نگهبان (Memory Guard Pages) می‌گویند
- تعریف: صفحاتی از حافظه که غیرقابل‌دسترسی‌اند(برای مثال نه می‌توان در آن‌ها نوشت و نه از آن‌ها خواند). اگر برنامه‌ای بخواهد برای ذخیره یا بازیابی به آن‌ها دسترسی بیابد با خطا مواجه خواهد شد
- می‌توانیم قبل از حافظه‌ی هرم و بعد از آن از این گاردهای محافظتی استفاده کنیم. به این طریق می‌توانیم مطمین شویم که حافظه‌ی هرم از محدوده‌ی تعیین شده به آن‌طرف‌تر دسترسی نخواهد داشت.