سیستم‌های عامل - دکتر ابراهیمی‌مقدم

امیرحسین منصوری - ۹۹۲۴۳۰۶۹

تمرین سری اول

سوال ۱

الف) جمله درست است.

ب) نادرست؛ *سیاست* (یا Policy) مشخص می‌کند که چه کاری انجام شود. *مکانیزم* نحوه انجام کار را مشخص می‌کند.

پ) نادرست؛ در Asymmetric Multiprocessing، حافظه مشترکی وجود ندارد.

ت) نادرست؛ اگر تعداد پارامترها زیاد باشد (بیشتر از تعداد رجیسترهای موجود)، پارامترهای دیگر را در مکانی در حافظه قرار می‌دهیم (مثلا پشته سیستم عامل)، و آدرس آن مکان از حافظه را در به عنوان پارامتر در یک رجیستر قرار می‌دهیم.

ث) نادرست؛ سرویس‌های حیاتی‌تر مانند مدیریت حافظه یا CPU Scheduler می‌توانند همچنان در هسته قرار بگیرند؛ هرچند که روی این موضوع که دقیقا چه سرویس‌هایی داخل کرنل اجرا می‌شود و چه سرویس‌های بیرون از آن، اتفاق نظر وجود ندارد.

سوال ۲

الف) کاربرد اصلی آن، جلوگیری از اختلال در اجرای برنامه‌های دیگر، یا اجرای خود سیستم‌عامل، توسط یک برنامه مخرب یا نادرست است.

ب) هنگام اجرای یک برنامه کاربری (User Program) و هر زمان که یک System Call یا وقفه رخ می‌دهد، با تنظیم mode bit به Kernel Mode می‌رویم و کدهای سیستم‌عامل در این حالت اجرا می‌شوند. پس از اتمام کار سیستم‌عامل، دوباره با تنظیم mode bit، به User Mode می‌رویم و اجرای برنامه‌های کاربری ادامه پیدا می‌کنند.

پ) این دو حالت عبارت‌اند از Kernel Mode و User Mode. تفاوت بین آن‌ها به صورت زیر است:

۱. در حالت Kernel mode، بیشترین سطح دسترسی وجود دارد؛ به طوری که اجازه دسترسی مستقیم به سخت‌افزار نیز وجود دارد. اما در حالت User mode، دسترسی برنامه در حال اجرا توسط سیستم‌عامل کنترل می‌شود و دسترسی به سخت‌افزار تنها از راه صدازدن سیستم‌کال‌های سیستم‌عامل ممکن است.

۲. تغییر حالت از Kernel mode به User mode همیشه ممکن است، اما برعکس آن همیشه امکان‌پذیر نیست.

۳. در Kernel mode فضای آدرس‌دهی برنامه با هسته سیستم‌عامل مشترک است؛ ولی برای برنامه‌های User mode از حافظه مجازی (Virtual Memory) استفاده می‌شود.

سوال ۳

الف) وقفه یا Interrupt روشی برای اطلاع دادن به پردازنده درباره یک رویداد است، که باعث توقف عملکرد عادی پردازنده و اجرای یک کد خاص برای پاسخ به این رویداد می‌شود. مثلا هنگام انجام یک عملیات ورودی/خروجی توسط پردازنده، که معمولا زمان‌بر است، پردازنده با اعمال یک وقفه سخت‌افزاری توسط ماژول ورودی/خروجی از اتمام عملیات آگاه می‌شود و می‌تواند نتیجه عملیات را بخواند و روی آن پردازش انجام دهد. به این ترتیب، پردازنده نیاز به صبر کردن برای اتمام کارهای زمان‌بر (مثلا عملیات ورودی/خروجی) ندارد و می‌تواند در این حین، کارهای دیگر را انجام دهد.

وقفه می‌تواند سخت‌افزاری یا نرم‌افزاری باشد. همچنین می‌توان برای هر رویداد یک وقفه جداگانه تعریف کرد و به آن یک شماره خاص اختصاص داد. همچنین می‌توان برخی وقفه‌ها را mask کرد؛ به این ترتیب، این وقفه‌ها نادیده گرفته می‌شوند و در صورت آمدن آن‌ها، تاثیری در عملکرد پردازنده نخواهند داشت. برای وقفه‌ها نیز می‌توان *اولویت* در نظر گرفت، تا در صورت آمدن همزمان چندین وقفه بتوان درباره پذیرش آن‌ها تصمیم‌گیری کرد.

ب) سرکشی یا Polling روش دیگری برای بررسی رویدادهای خارجی (مثل عملیات ورودی/خروجی) است. در این روش، یک کد نرم‌افزاری رخ دادن رویداد را به طور مداوم بررسی می‌کند و به این ترتیب، تا زمان رخ‌دادن آن صبر می‌کند و سپس به ادامه کار عادی خود می‌پردازد.

این روش، برای زمانی که بخواهیم با سرعت زیاد رویدادهای خارجی را پردازش کنیم مناسب نیست و سربار بالایی دارد.

پ) بردار وقفه یا Interrupt Vector جدولی شامل آدرس مربوط به ابتدای کد Interrupt Handler هر وقفه است. هنگام آمدن یک وقفه که mask نشده است، پردازنده رجیستر Program Counter خود را روی آدرس موجود در این جدول تنظیم می‌کند و در نتیجه کد Interrupt Handler مربوطه اجرا می‌شود.

سوال ۴

سیستم *موازی*، قابلیت اجرای چند وظیفه به صورت همزمان را دارد (مثلا با به‌کارگیری چندین پردازنده). اما سیستم *همروند* صرفا می‌تواند چندین وظیفه را طوری مدیریت کند که با هم (و نه الزاما در آن واحد و همزمان) پیشروی کنند؛ به طور مثال، با انجام قسمت کوچکی از هر وظیفه و جابه‌جا شدن بین وظایف، می‌توان توهمی از همروندی را حتی در یک سیستم با یک پردازنده ایجاد کرد. درنتیجه سیستم همروند، الزاما موازی نیست.

سوال ۵

الف) به سیستمی که دارای یک پردازنده با چند هسته پردازشی است، سیستم Multi-core گفته می‌شود. همچنین به سیستمی که دارای چندین پردازنده جداگانه است، سیستم Multi-processor گفته می‌شود.

ب)

مزایای Multi-core

۱. هسته‌ها معمولا روی یک تراشه یکسان قرار می‌گیرند. در نتیجه ارتباط بین هسته‌ها سریع‌تر از حالتی است که هسته‌ها روی تراشه‌های متفاوت قرار بگیرند.

۲. سیستم‌های چندهسته‌ای توان مصرفی بهینه‌ای دارند.

معایب Multi-core

۱. به اندازه هسته‌های اضافه‌شده، کارایی بیشتر دریافت نمی‌کنیم. مثلا نمی‌توان گفت که سیستم دوهسته‌ای دوبرابر سیستم تک‌هسته‌ای کارایی دارد.

۲. همه سیستم‌عامل‌ها از پردازنده‌های چندهسته‌ای پشتیبانی نمی‌کنند.

مزایای Multi-processor

۱. تاب‌آوری بالاتری نسبت به سیستم‌های تک‌پردازنده‌ای دارد؛ چون در صورت از کار افتادن یک پردازنده، پردازنده‌های دیگر می‌توانند همچنان عمل کنند.

۲. می‌توان چندین Process را به صورت موازی روی آن اجرا کرد.

معایب Multi-processor

۱. ارتباط بین پردازنده‌ها سنگین‌تر و کندتر است.

۲. هماهنگ کردن پردازنده‌ها طوری بتوانند باهم کار کنند می‌تواند دشوار و چالشی باشد.

ج) سیستم چندهسته‌ای معمولا شامل یک پردازنده است که چندین هسته پردازشی دارد و می‌تواند چندین Instruction را با هم اجرا کند. سیستم Multi-processor چندین پردازنده جدا از هم دارد که با هم کار می‌کنند و هرکدام می‌تواند یک Process را اجرا کند.

سوال ۶

در معماری Microkernel که در مقابل معماری Monolithic مطرح می‌شود، بسیاری از سرویس‌های سیستم‌عامل به جای پیاده‌شدن در هسته، در یک برنامه User-space پیاده می‌شوند؛ در هسته صرفا بعضی عملکردهای پایه‌ای مثل مدیریت حافظه یا ارتباط بین Processها پیاده می‌شود. به این ترتیب، هسته کوچک می‌شود.

یکی از مزایای این معماری، آسان بودن گسترش عملکردهای سیستم‌عامل است. برای اضافه کردن یا تغییردادن یکی از عملکردها، نیازی به تغییر خود هسته نیست و فقط کافیست این عملکرد را به صورت یک برنامه User-space پیاده کنیم و به هسته معرفی کنیم. همچنین به علت کوچک‌تر شدن هسته، ایجاد تغییرات در آن ساده‌تر است و مثلا تبدیل یا port کردن هسته برای اجرا بر سخت‌افزارهای متفاوت ساده‌تر می‌شود.

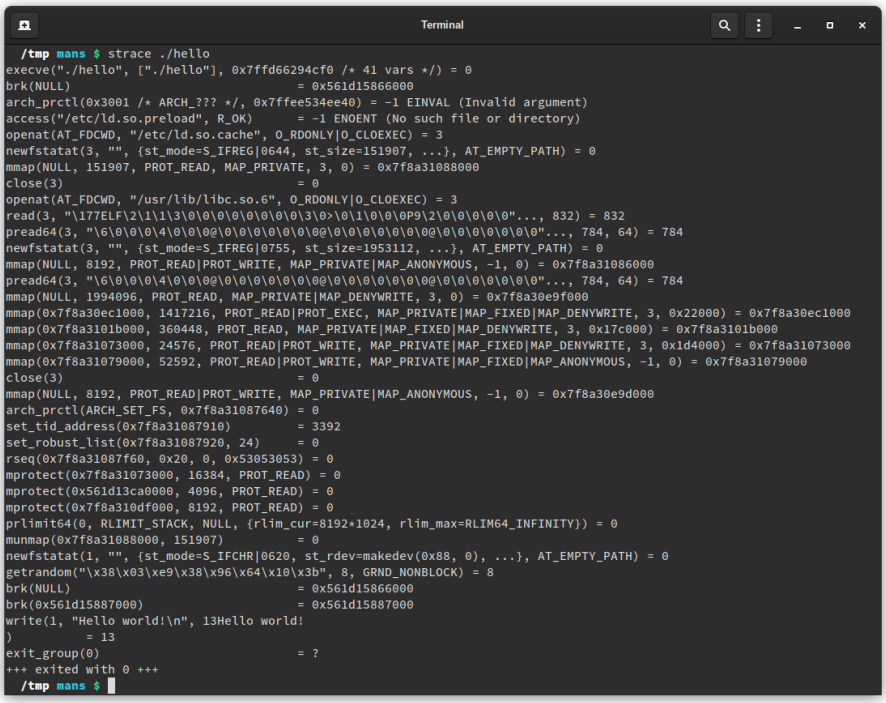
یک ایراد این معماری، افزایش سربار ارتباط بین قسمت‌های مختلفی است که دیگر در هسته قرار ندارند. چون عملکردهای مختلف، خودشان یک process هستند، برای ارتباط با هسته نیز از سازوکارهای ارتباط بین processها استفاده می‌کنند؛ در مقابل معماری Monolitic به دلیل اجرای بیشتر عملکردهای سیستم‌عامل در یک برنامه با فضای آدرس‌دهی یکسان، نیازی به ارتباط بین processها ندارد و به همین دلیل سربار کمتری دارد و سریع‌تر است.

سوال ۷

در این سوال، سیستم‌کال‌های سیستم‌عامل لینوکس را با استفاده از ابزار strace بررسی می‌کنیم. می‌توان گفت De-facto برنامه‌های ساده، همان برنامه‌ی Hello world مشهور است؛ و در اینجا سیستم‌کال‌های همین برنامه را بررسی می‌کنیم. کد این برنامه به زبان C در زیر آمده است.

```
1  #include <stdio.h>
2
3  int main() {
4      puts("Hello world!");
5      return 0;
6  }
```

خروجی اجرای این کد با strace به شکل زیر است.



حال چند مورد از سیستم‌کال‌های تصویر بالا را شرح می‌دهیم.

- `execve`: یک برنامه دیگر را اجرا می‌کند، و برنامه اجرا شده را جایگزین Process فعلی می‌کند. امضای تابع مربوط به این سیستم‌کال در glibc به صورت زیر است؛

```
int execve(const char *pathname, char *const _Nullable argv[],
           char *const _Nullable envp[]);
```

که در آن `pathname` آدرس فایل اجرایی برنامه (یا آدرس یک اسکریپت که با shebang آغاز می‌شود)، `argv` آرگومان‌های خط‌فرمان برنامه، و `envp` متغیرهای محیطی (Environment variables) برنامه هستند.

در اولین خط خروجی strace نیز می‌بینیم که این ابزار، با استفاده از همین سیستم‌کال، برنامه ما را اجرا کرده است؛ همچنین هیچ آرگومان خط‌فرمانی پاس داده نشده، و ۴۱ متغیر محیطی نیز به برنامه داده شده است.

- `brk`: این سیستم‌کال، آدرس انتهای Data Segment پروسه (قسمتی از حافظه پروسه که شامل متغیرهای global و static است) را تغییر می‌دهد. به این آدرس، break نیز گفته می‌شود. صدازدن این سیستم‌کال با آدرس صفر، آدرس فعلی break را برمی‌گرداند و چیزی را تغییر نمی‌دهد.
- `read`: از یک فایل، به مقدار مشخص شده داده می‌خواند و در یک بافر (فضایی در حافظه) می‌نویسد. امضای تابع مربوط به این سیستم‌کال در glibc به صورت زیر است؛

```
ssize_t read(int fd, void buf[.count], size_t count);
```

که در آن `fd` شماره file descriptor مربوط به یک فایل باز شده توسط این پروسه، `buf` آدرس بافر، و `count` تعداد بایت‌هایی است که می‌خواهیم از فایل بخوانیم. در صورت موفقیت، این تابع علاوه بر نوشتن در بافر، تعداد بایت‌های خوانده شده را نیز برمی‌گرداند.

- `write`: از یک بافر در حافظه، به مقدار مشخص شده داده می‌خواند و در یک فایل می‌نویسد. امضای تابع مربوط به این سیستم‌کال در glibc به شکل زیر است؛ که بسیار مشابه سیستم‌کال `read` است.

```
ssize_t write(int fd, const void buf[.count], size_t count);
```

در خروجی strace مشاهده می‌کنیم که متن `"Hello world!"` در فایلی با file descriptor برابر ۱ نوشته می‌شود و در نتیجه آن، این متن در خروجی استاندارد چاپ می‌شود. در لینوکس، file descriptor شماره ۱ همواره مربوط به خروجی استاندارد (یا stdout) است، و در واقع با نوشتن در این فایل می‌توان متن را در خروجی چاپ کرد. (همچنین fd شماره صفر و ۲ نیز به ترتیب مربوط به stdin و stderr هستند.)

- `close`: یک فایل باز را می‌بندد. تنها پارامتر آن نیز fd فایل مربوطه است.

سوال ۸

در مباحثه معروف بین Torvalds و AST به بسیاری از این دلایل اشاره شده است. AST معتقد بود که معماری monolithic منسوخ شده و متعلق به گذشته است، و معماری micro-kernel آینده معماری سیستم‌عامل‌ها است. خود او نیز سیستم‌عامل Minix را به عنوان یک سیستم‌عامل با قابلیت اجرا روی پردازنده‌های ارزان‌قیمت‌تر آن دوران و با اهداف آموزشی طراحی کرد که با معماری micro-kernel طراحی شده بود.

در مقابل، Torvalds لینوکس را برای اجرا روی پردازنده ۸۰۳۸۶ اینتل و با معماری monolithic طراحی کرده بود. این پردازنده، نسبتا جدید و گران‌قیمت بود و حتی برای خود Torvalds هم مدتی طول کشید تا بتواند یکی از این پردازنده‌ها را برای خودش تهیه کند.

AST دو اشکال عمده به لینوکس وارد می‌کرد، و به نظر او به همین دو دلیل کافی بود تا لینوکس «منسوخ‌شده» باشد:

۱. معماری monolithic: AST طرفدار micro-kernel بود و این معماری لینوکس را بازگشتی به دهه ۷۰ میلادی می‌دانست.

۲. Portable نبودن: از دید AST، معماری پردازنده‌ها به سرعت در حال تغییر و تحول بودند و حتی پیش‌بینی می‌کرد که سری ۸۰x۸۶ اینتل کنار خواهد رفت و معماری‌های دیگر جایگزین آن خواهند شد؛ در نتیجه او طراحی سیستم‌عامل برای یک معماری خاص، به خصوص معماری عجیب و غریبی مثل ۸۰x۸۶ را اشتباه می‌دانست. او Portability را مهم می‌دانست و لینوکس را برای استفاده از قابلیت‌های خاص سخت‌افزاری ۸۰۳۸۶ که الزاما روی معماری‌های دیگر موجود نیست سرزنش می‌کرد، و معتقد بود به همین دلیل لینوکس به زودی فراموش خواهد شد.
(به نظر می‌رسد پیش‌بینی او زیاد درست از آب در نیامده باشد...!)

برخی پاسخ‌های Torvalds به این انتقادات، چنین بود:

۱. او معماری micro-kernel را در مواردی مثل همروندی دچار مشکل می‌دانست؛ مثلا در حالی که لینوکس قابلیت Multithreaded Filesystem داشت و امکان کار با چندین فایل به طور همزمان را فراهم می‌کرد، در minix چنین قابلیتی موجود نبود.

۲. او معتقد بود که micro-kernel در نهایت کار طراحی سیستم‌عامل را پیچیده‌تر می‌کند و همان ارتباط مستقیم اجزای سیستم‌عامل با هم در یک کرنل monolithic سادگی بیشتری در طراحی دارد.

۳. از دید Torvalds، یک سیستم‌عامل نباید بیش از حد به Portablity اهمیت بدهد. چون سیستم‌عامل در نهایت لایه‌ای سطح‌بالاتر از سخت‌افزار است و مستقیم با سخت‌افزار ارتباط دارد و منطقی است که تا حدی از امکانات ویژه معماری‌های مختلف بهره ببرد.