تمرین اول

|  |  |
| --- | --- |
| علی علی‌محمّدی | 9613027 |

|  |  |
| --- | --- |
| 1) | هر چند که نوار مغناطیسی از ارزان‌ترین حافظه‌های ممکن است، سرعت خواندن و نوشتن اطلاعات بر روی آن در مقایسه با انواع دیگر حافظه، بسیار پایین است. این در حالی است که در کامپیوترهای امروزی گلوگاه عملکرد، سرعت حافظه است و هر چه از حافظه‌های سریع‌تری استفاده شود، عملکرد سیستم بهبود خواهد یافت. به عبارتی دیگر می‌توان گفت که هزینه‌ی اضافی که بابت حافظه‌های سریع‌تر پرداخت می‌شود، ارزش اقتصادی دارد؛ زیرا عملکرد سیستم را در یک بازه‌ی زمانی طولانی تحت تأثیر قرار می‌دهد. موردی که کمتر مورد توجه است، نحوه‌ی دسترسی به اطلاعات در نوار مغناطیسی است؛ در نوار مغناطیسی، دسترسی به اطلاعات به صورت Sequential است و باید نوار را تا جایی که به اطلاعات مورد نظر نرسیده‌ایم، بخوانیم و جلو برویم. این در حالی است که سیستم‌های حافظه‌ای که اکنون مورد استفاده قرار می‌گیرند، امکان دسترسی بسیار سریع را به داده‌های Fragmentشده (قطعه‌قطعه شده) فراهم می‌سازد. |
| 2) | زیرا خواندن یا نوشتن در یک حافظه با فرایندهای دیگر تداخل ایجاد می‌کند و یا حتی ممکن است به برنامه‌های مخرب اجازه‌ی فعالیت بدهد؛ مثلاً ممکن است یک فرایند در حالی که کاربر دارد گذرواژه‌ی خود را وارد می‌کند، شروع به خواندن از کیبرد کند که اطلاعات حساس کاربر را به خطر می‌اندازد. در واقع، اگر مدیریت دستورالعمل‌های I/O را به کاربر واگذار کنیم، ممکن است به نحوی از آن‌ها سوءاستفاده کند؛ مثلاً ممکن است داده‌های جدید را روی داده‌های قبلی بنویسد که باعث از دست رفتن داده‌ها می‌شود.  بنابراین دستورالعمل‌های I/O ممتاز هستند تا سیستم‌عامل بتواند بر روی آن‌ها نظارت داشته باشد و قبل از اجرای هر دستور I/O بررسی کند که آیا انجام این دستورالعمل مجاز است یا خیر! و تنها در صورتی که مجاز بود، اجازه‌ی اجرای آن را صادر کند. |
| 3) | اجرای حدسی یک تکنیک بهینه‌سازی است که در آن، یک سیستم کامپیوتری عملیاتی را انجام می‌دهد که ممکن است هیچ‌وقت مورد استفاده قرار نگیرند. در واقع سیستم، مجموعه‌ای از دستورات را قبل از این که بداند مورد استفاده قرار می‌گیرند یا خیر، اجرا می‌کند تا از تأخیر احتمالی در اجرای دستورات در زمانی که واقعاً به نتیجه‌ی آن‌ها احتیاج دارد، جلوگیری کند. در صورتی که نتیجه‌ی اجرای این دستورات مورد نیاز نباشد، تغییرات انجام شده توسط این دستورات، به حالت اولیه بازگردانده می‌شوند و نتایج آن دستورات، نادیده گرفته می‌شود. مزیت آن این است که امکان داشتن هم‌زمانی (Concurrency) بیشتر را فراهم می‌کند و تأخیر در اجرا را پایین می‌آورد. از طرفی انجام این کار منابع سیستم را مصرف می‌کند و اگر تعداد این دستورالعمل‌ها زیاد باشد، میزان منابع مورد نیاز به صورت توانی (exponential) افزایش می‌یابد و در نهایت نیز ممکن است از این پیش‌پردازش انجام شده، استفاده نشود و نتایج آن دور ریخته شود که در این صورت، تأخیر در اجرای دستورالعمل‌های فرایندهای دیگر افزایش پیدا می‌کند. |
| 4) | 1- این آسیب‌پذیری یک حمله‌ی اجرای حدسی به پردازنده‌های اینتل است که ممکن است منجر به افشای اطلاعات حساس ذخیره شده در کامپیوترهای شخصی و سرورهای اَبری شخص سوم شود. دو نسخه از آن وجود دارد که نسخه‌ی اول، داده‌های محدوده‌های SGX و نسخه‌ی دوم، ماشین‌های مجازی، hypervisorها، حافظه‌ی هسته‌ی سیستم‌عامل‌ها، حافظه‌ی حالت مدیریت سیستم را مورد حمله قرار می‌دهد. استفاده از وصله‌های نرم‌افزاری به‌روزرسانی‌شده می‌تواند مقداری از نگرانی‌های مربوط به این موضوع را مرتفع سازد اما باعث کاهش توان پردازشی پردازنده می‌شود. اینتل تنها راه برطرف‌کردن این آسیب‌پذیری را جایگزینی و تعویض سخت‌افزار و استفاده از سخت‌افزار جدید است که نسبت به این آسیب‌پذیری ایمن شده‌اند.  2- یک آسیب‌پذیری امنیتی پردازنده‌های سریِ Core™ اینتل است که از ترکیب تعدادی آسیب‌پذیری در فناوری اجرای حدسی و نحوه‌ی کنترل Context Switching بر روی واحد نقطه‌ی (ممیز) شناور (FPU) توسط برخی سیستم‌عامل‌ها حاصل می‌شود. با بهره‌گیری از این آسیب‌پذیری، یک فرایند محلی می‌تواند محتوای ثبات‌های FPU را که به فرآیند دیگری تعلق دارند، فاش کند. برای رفع این آسیب‌پذیری می‌توان با بازیابی همیشگی وضعیت FPU در هنگام Context Switching فرایندها، آسیب‌پذیری را در سطح سیستم‌عامل و hypervisor کاهش داد.  3- این آسیب‌پذیری یک حمله‌ی اجرای حدسی به پردازنده‌های X86 اینتل است که از ویژگیِ hyper-threading استفاده می‌کنند و از امکان خواندن بافرهای داده‌ای که بین قسمت‌های مختلف پردازنده وجود دارند، سوءاستفاده می‌کند. Fallout، RIDL و ZombieLoad از این دسته آسیب‌پذیری‌ها هستند. برای مقابله با این دسته از آسیب‌پذیری‌ها اعمال رفع اشکال نرم‌افزاری در سیستم‌عامل‌ها، مکانیزم‌های مجازی‌سازی، مرورگرهای وب و میکروکُد (microcode) لازم است. نصب به‌روزرسانی‌های ارائه شده در تاریخ 14اُم ماه مِی سال 2019 نیز برای مرتفع ساختن این مشکل توصیه می‌شود.  4- یک آسیب‌پذیری پردازنده‌های اینتل است که از عوارض جانبی ناشی از اجرای حدسی، برای بهبود عملکرد حمله‌ی Rowhammer و انواع دیگر حملات به حافظه و cache استفاده می‌کند. تا به حال هیچ‌گونه راهکار نرم‌افزاری یا سخت‌افزاری برای مقابله با این آسیب‌پذیری ارائه نشده است. |
| 5) | مزیت اصلی استفاده از ساختار ریزهسته، سهولت گسترش سیستم‌عامل است. در این رویکرد، همه‌ی سرویس‌های جدید به فضای کاربر اضافه می‌شوند و در نتیجه نیازی به تغییر و دست‌کاری هسته وجود ندارد. از آن‌جا که اکثر سرویس‌ها به جای آن که به عنوان فرایندهای هسته در حال اجرا باشند، به عنوان فرایندهای کاربر در حال اجرا هستند، ساختار ریزهسته امنیت و اطمینان بیشتری را فراهم می‌کند. این در حالی است که ریزهسته‌ها به دلیل افزایش کارکرد سربار سیستم (System Function Overhead)، ممکن است دچار کاهش عملکرد شوند. |
| 6) | Genode یک معماری سیستم‌عامل جدید است که با استفاده از یک ساختار سازمانی سخت‌گیرانه بر روی تمام اجزای نرم‌افزاری از جمله درایورهای دستگاه، سرویس‌های سیستم و برنامه‌های کاربردی قادر است بر مشکلات مربوط به پیچیدگی سیستم تسلط یابد. این سیستم‌عامل به عنوان یکی از معدود سیستم‌عامل‌های منبع‌باز قابل توجه است که از یک سیستم‌عامل اختصاصی مانند Unix مشتق نشده است. این سیستم‌عامل از ساختار ریزهسته استفاده می‌کند که در نهایت مجموعه‌ای از اجزای فضای کاربری بر روی آن قرار گرفته است. فلسفه طراحی اصلی این سیستم‌عامل بر پایه‌ی داشتن یک پایگاه محاسباتی کوچک قابل اعتماد در سیستم‌عاملی امنیت‌گرا است. فریم‌وُرکِ Genode OS یک جعبه‌ی ابزار منبع‌باز برای ساخت سیستم‌های عامل Component-based بسیار امن است. این سیستم‌عامل بر روی پردازنده‌های X86 و ARM قابل اجرا است. حجم فعلی کد هسته‌ی این سیستم‌عامل حدود 406هزار خط است.  میکروکرنل (ساختار ریزهسته) seL4 اولین هسته‌ی سیستم‌عامل جهان است که دارای قابلیت end-to-end اثبات صحت پیاده‌سازی و اِعمال امنیت است که عموماً از آن برای پیاده‌سازی سیستم‌عامل‌های شبهِ‌یونیکس (Unix-like) استفاده می‌شود اما کاربردهای دیگری نیز دارد. این سیستم‌عامل به جهت استقلال از سکو (Platform Independence)، افزایش امنیت، انزوا، قابلیت اطمینان و استحکام طراحی شده است. این سیستم‌عامل بر روی پردازنده‌های X86 و ARM قابل اجرا است. حجم فعلی کد هسته‌ی این سیستم‌عامل حدود 972هزار خط است. |
| 7) | الف) برنامه‌های سیستمی، برنامه‌هایی هستند که با همراه سیستم‌عامل هستند اما لزوماً بخشی از هسته‌ی آن نیستند و با عملکرد سیستم ارتباط دارند و در مدیریت سیستم به آن کمک می‌کنند.  ب) در سیستم‌عامل‌های کامپیوتری چندوظیفه‌ای، daemon یک برنامه کامپیوتری است که به جای این که تحت کنترل مستقیم کاربر تعاملی باشد، به عنوان یک فرایند در پس‌زمینه اجرا می‌شود. سرویس‌های سیستم که به عنوان برنامه‌های کمکی سیستم نیز شناخته می‌شوند، محیط مناسبی را برای توسعه و اجرای برنامه‌های کاربردی فراهم می‌کنند؛ مانند سرویس‌های مدیریت فایل، شرح وضعیت سیستم و ارتباطات.  ج) API مجموعه‌ای از توابع را که برای برنامه‌نویس‌های برنامه‌های کاربردی در دسترس است، مشخص می‌کند؛ از جمله پارامترهایی که به هر تابع به عنوان ورودی داده می‌شود و مقادیر بازگشتی که برنامه‌نویس می‌تواند انتظار داشته باشد. سه مورد از رایج‌ترین APIهای موجود برای برنامه‌نویس‌های برنامه‌های کاربردی عبارتند از: Windows API برای سیستم‌های ویندوز، POSIX API برای سیستم‌های مبتنی بر POSIX (که تقریباً تمام نسخه‌های UNIX ، Linux و macOS را شامل می‌شود) و Java API برای برنامه‌های جاوایی که روی ماشین مجازی جاوا اجرا می‌شوند.  د) فراخوانی سیستم یک رابط اساسی بین یک فرایند و سیستم‌عامل فراهم می‌سازد. فراخوانی سیستم روشی برنامه‌گونه است که در آن یک برنامه‌ی کامپیوتری از هسته‌ی سیستم‌عاملی که بر روی آن اجرا می شود، سرویسی را درخواست می‌کند که ممکن است شامل خدمات مرتبط با سخت افزار (به عنوان مثال، دسترسی به درایو دیسک سخت)، ایجاد و اجرای فرایندهای جدید و ارتباط با سرویس‌های حیاتی هسته، مانند سرویس زمانبندی فرآیند باشد. |
| 8) | دو مدل اساسی ارتباط درون پردازشی وجود دارد: حافظه‌ی مشترک و ارسال پیام. در مدل حافظه‌ی مشترک، قسمتی از حافظه که توسط فرایندهای همکار به اشتراک گذاشته می‌شود، ایجاد می‌شود. سپس فرایندها می‌توانند با خواندن و نوشتن داده‌ها در منظقه‌ی مشترک، اطلاعات را تبادل کنند. در مدل ارسال پیام، ارتباط با استفاده از پیام‌هایی که بین فرایندهای همکار رد و بدل می‌شود، صورت می‌گیرد. ارسال پیام برای مبادله مقادیر کم از داده‌ها مناسب است؛ زیرا نیازی به جلوگیری از هیچ تداخلی (Conflict) نیست. علاوه بر آن، پیاده‌سازی مکانیزم ارسال پیام در سیستم‌های توزیع‌شده نسبت به مکانیزم حافظه‌ی مشترک آسان‌تر است. حافظه‌ی مشترک سریع‌تر از ارسال پیام است؛ زیرا سیستم‌های انتقال پیام معمولاً با استفاده از فراخوانی‌های سیستم پیاده‌سازی می‌شوند و بنابراین به دخالت بیشتر هسته نیاز دارند که باعث زمان‎بر شدن کل این مکانیزم گردد. در سیستم‌های حافظه‌ی مشترک، فراخوانی‌های سیستم فقط برای ایجاد مناطق حافظه‌ی مشترک مورد نیاز است. پس از ایجاد منطقه‌ی حافظه‌ی مشترک، کلیه‌ی دسترسی‌ها به عنوان دسترسی معمول حافظه تلقی می‌شوند و نیازی به کمک گرفتن از هسته وجود ندارد. به این ترتیب، مدل حافظه‌ی مشترک یک روش مستقیم ارتباطی است؛ زیرا هسته‌ی سیستم‌عامل در آن نقشی ندارد و مدیریت ارتباط به توافق انجام شده میان دو فرایند بستگی دارد. این در حالی است که مدل ارسال پیام، یک روش غیرمستقیم ارتباطی است؛ زیرا از امکانات هسته‎ی سیستم‌عامل برای تبادل اطلاعات میان فرایندها استفاده می‌شود. |