به نام خدا

محمدمهدی آقاجانی

# تمرین سری اول

## سیستم عامل

استاد : استاد طاهری جوان

تمرین اول :

برای اینکه پردازنده بتواند با دستگاه های ورودی و خروجی کار کند هر دستگاه وقتی کارش به پایان رسید وقفه مربوط به خود را صدا میزند سپس پردازنده کار جاری را رها میکند و به جدول وقفه ها نگاه میکند که ببیند وقفه مربوط به کدام دستگاه صدا زده شده است سپس روتین وقفه اجرا میشود تا وقفه هندل شود بعد از اتمام آن وقفه دوباره پردازنده به کار قبلی خود برمیگردد .

تمرین دوم :

روش اول : programmed I/O

در این حالت وقتی ورودی خروجی میخواهد کار کند پردازنده منتظر آن میماند و وقتی کار دستگاه ورودی خروجی تمام شد بیت مربوط به اتمام کار را فعال میکند در این روش پردازنده باید مدت زیادی منتظر دستگاه ورودی خزوجی بماند و همچنین به صورت متناوب بیت اتمام را چک کند که به این کار polling نیز میگویند.

روش دوم : interrupt-driven I/O

دستگاه ورودی خروجی یک سیگنال وقفه را صدا میزند و پردازنده کار جاری را رها میکند و به جدول وقفه ها نگاه میکند تا ببیند برای وقفه به صدا درآمده با شماره خاص باید چه روتین وقفه ای را اجرا کند پس از اجرای روتین وقفه مورد نظر دوباره پردازنده به کار قبلی خود بازمیگردد .

روش سوم : DMA

وقتی میخواهیم انتقال اطلاعات بسیار زیادی بین دستگاه ورودی خروجی و مموری داشته باشیم برای استفاده از گذرگاه آدرس و داده باید همواره پردازنده را آگاه کنیم و اینکار خود زمان بر میباشد در این حالت یک سخت افزاری به نام DMA وجود دارد که میتواند این کار را راحت تر کند بدین صورت که در صورت انتقال اطلاعات حجیم بین مموری و دستگاه ورودی خروجی میتواند کنترل گذرگاه داده و آدرس را از پردازنده بگیرد ( که البته برای اینکار از خود پردازنده اجازه میگیرد ) برای دریافت اجازه باید DMA به پردازنده بگوید که تعداد بایت مورد انتقال چه مقدار است و همچنین آدرس شروع حافظه را نیز به پردازنده باید بدهد بعد از اینکار در صورتی که پردازنده اجازه داد گذرگاه در اختیار DMA قرار میگیرد و در این هنگام پردازنده اتصال خود را از bus قطع میکند ولی همواره در حال شنود است تا انتقال اطلاعات به پایان برسد .

تمرین سوم :

دسته بندی کامپیوتر ها از نگاه Flynn به صورت زیر میباشد :

۱- SISD ( single instruction stream single data stream )
در این دسته در هر سیکل پردازنده یک دستور اجرا شده و یک داده از مموری خوانده میشود که همان single processing میباشد.
۲- SIMD ( single instruction stream multiple data stream )
در این دسته در هر سیکل یک دستور بر روی چندین داده اجرا میشود که GPU ها از این نوع میباشد.
۳- MISD ( multiple instruction stream single data stream )
در این دسته چندین دستور بر روی یک داده اجرا میشود . برای سیستم های امن استفاده میشود به خاطر اینکه از صحت کار اطیمینان حاصل کنند
٤- MIMD ( multiple instruction stream multiple data stream )
در این حالت در هر سیکل چندین دستور بر روی چندین داده اجرا میشود که به معنای همان multi processing میباشد .

تمرین چهارم :

رجیستر ها به دو نوع کلی عام منظوره و خاص منظوره تقسیم میشوند رجیستر های عام منظوره در رجیستر فایل وجود دارند که برای محاسبات جاری پردازنده از آن ها استفاده میشود و میتوان مقادیری را در آن ها ذخیره کرد تا در محاسبات مربوطه از آن مقادیر استفاده نمود اما رجیستر های خاص منظوره شامل موارد زیر میشوند که خود میتواند به دو دسته رجیستر های داده و رجیستر های آدرس تقسیم شود :

۱- Address register : مخصوص ذخیره کردن آدرس برای خواندن از مموری
۲- Program counter : شماره خط خوانده شده برنامه از مموری را دارد
۳- Instruction register : آدرس دستور خوانده شده از مموری را دارد
٤- Data register : داده خوانده شده از مموری را نگه داری میکند .

اما از دید دیگر نیز میتوان رجیستر ها را دسته بندی کرد :

۱- رجبیستر های قابل رویت برای کاربر : که در واقع کامپایلر ها به طور هوشمند به آن دسترسی دارند و حتی به برخی به صورت مستقیم برنامه نویس در برخی زبان ها دسترسی دارد مانند AR , DR
۲- رجبیستر های غیر قابل رویت ( کنترل و وضعیت ) : این رجیستر ها برای انجام کار های داخلی پردازنده ها میباشد مانند flag ها و PC , IR . به این دسته از رجیستر ها فقط سیستم عامل دسترسی دارد

تمرین پنجم :

۱- Monolithic :
در این حالت به دلیل بهره وری از حداقل فضا و حداکثر کارایی و از طرفی به دلیل محدود بودن سخت افزار هیچ نوع عملکرد ماژولاری برای سیستم عامل وجود ندارد مانند DOS . سیستم عامل به صورت مجموعه ای از رویه ها نوشته شده که هر یک میتوانددیکری را فراخوانی کند و هیچ حفاظتی وجود ندارد . برای فراخوانی های سیستمی ابتدا پارامتر های لازم این فراخوانی ها را در مکان های مشخص قرار میدهند و سپس با یک دستور العمل trap instruction مخصوص انجام میشود که به kernel call معروف است که یان دستور العمل ماشین را از مد کاربر به مد سیستم عامل قرار میدهد و کنترلش در اختیار سیستم عامل قرار میگیرد .
۲- Layered :
در این حالت سیستم عامل به چند سطح تقسیم می شود و هر لایه یا سطح فقط میتواند از سرویس ها و توابع سطوح ئایین تر استفاده کند . به این صورت میتوان هر لایه را به صورت مستقل ارزیابی و خطایابی کرد . از مشکلات این روش همین تعریف دقیق لایه هاست و اینکه هر درخواست باید لایه هایی را طی کند ایجاد سربار میشود . اولین سیستمی که با این روش طراحی شد THE ساخته داکسترا و شاگردانش بود که لایه بندی زیر را داشت :
لایه صفر : تعیین میکند cpu هر لحظه در اختیار کدام فرآیند باشد
لایه یک : مدیریت حافظه اصلی و جانبی را برعهده دارد
لایه دو : ارتباط کنسول اپراتور و عر فرآیند را مشخص میکند .
لایه سه : مدیریت دستگاه های ورودی و خروجی و بافر کردن اطلاعات
لایه چهار : برنامه های کاربران را اجرا میکند
لایه پنج : در این لایه فرایند اپراتور سیستم قرار دارد .
۳- Virtual machine
این سیستم عامل به محض نصب بر روی یک سخت افزار آن را شبیه سازی میکند و میتوان سیستم عامل های دیگر را بر روی آن سوار نمود . به عبارتی هر کاربر با یک سیستم عامل به صورت جدا ارتباط دارد و هر ماشین مجازی از دیگر ماشین های مجازی کاملا جداست و در نتیجه ایرادات امنیتی ندارد .
٤- Client server
ایده این نوع طراحی این است که تا حد مکن کد ها و سرویس ها را به لایه های بالاتر منتقل کرد و فقط کار های حساس و خالص سخت افزاری را به عهده هسته گذاشتدر این حالت چون هسته سیستم عامل بسیار کوچک میشود به این نوع طراحی ریز هسته نیز میگویند . در این حالت اکثر وظایف سیستم عامل در سطح فرآیند های کاربر پیاده سازی میشود و به عنوان سرویس دهنده شناخته میشوند و فرآیند های خود کاربر به عنوان سرویس گیرنده . در این حالت فرآیند های کاربر برای انجام کاری مانند خواندن فایل به سرویس مربوط به خواندن فایل در خواست میدهد و جواب میگیرد . وظیفه هست در این بین ارتباط بین سرویس دهنده ها و سرویس گیرنده ها میباشد البته هسته در این حالت دارای وظایف دیگری نیز میباشد از قبیل :
۱- زمان بندی فرآیند ها
۲- مدیریت حافظه در سطح پایین مانند برنامه ریزی برای ثبات ها
۳- مدیریت I/O

و اموری که پیاده سازی آن ها در سطح سرویس های عادی امنیت سیستم را به خطر خواهد انداخت

در این حالت سیستم عامل به سرویس ها یی تبدیل شده که هر یک وجوهی از سیستم عامل را پیاده سازی میکنند و در نتیجه هر بخش میتواند مستقل از دیگری عمل کند و احیانا اگر در کار سرویس خللی ایجاد شود کل سیستم مختل نمیگردد.

تمرین ششم :

مود دوگانه کاربر – کرنل به این صورت است که سطح دسترسی در سیستم را تقسیم به دو بخش میکنیم یک بخش کاربر و دیگری بخش کرنل . سیستم عامل در بخش کرنل اجرا میشود و برنامه های کاربران در بخش کاربر . ممکن است در برنامه های کاربران نیاز باشد تا سرویسی از سیستم عامل درخواست شود در این صوت سیستم از حالت کاربر به حالت کرنل سوییچ میکند . در مود کرنل سیستم عامل کاملا بر روی سخت افزار سوار است . این کار باعث میشود که کاربران را از پیچیدگی های سخت افزاری دور نگه داریم و مدیریت سخت افزار را کاملا در اختیار سیستم عامل قرار دهیم که در نتیجه از آسیب رسیدن به سخت افزار توسط کاربر های ناآگاه جلوگیری میشود . در واقع سطح کرنل برای این است که سیستم عامل بر روی آن سوار شود و بین ما که در سطح کاربر هستیم و سخت افزار ارتباط برقرار کند و در این بین این ارتباط را منترل نیز بکند تا به سخت افزار آسیب نرسد . این سطح کاربر میتواند به صورت های مختلف کامند یا گرافیکی پیاده سازی بشود . مثلا در ویندوز که به صورت گرافیکی پیاده سازی شده شما به عنوان کاربر نمیتوانید برخی دستورات سیستمی را اجرا کنید و این به خاطر این است که سیستم عامل در مود کرنل این دستورات را reject میکند

تمرین هفتم :

از اصلی ترین وظایف سیستم عامل است . این وظیفه شامل ایجاد و حذف فرآیند ها ، زمان بندی فرآیندها برای دریافت منابع ، مدیریت همزمانی و همگام سازی فرآند ها ، و جلوگیری از ایجاد deadlock

تمرین هشتم :

تایمر پردازنده به منظور تعیین زمان اجرای پردازنده در هر برهه زمانی استفاده میشود در سیستم های time sharing سیستم عامل یک تایمر تنظیم کرده و ضمن سپردن پردازنده به فرآیند خود کنار میرودپس از طی زمان تعیین شده رد تایمر یک وقفه به صدا در می آید که در نتیجه پردازنده به سیستم عامل باز میگردد . همچنین برخی از کار ها شبیه تست سخت افزار یا حافظه ها باید در دوره های زمانی به طور مرتب انجام شوند که در مجموع سیستم عامل باید از ساعت داخلی دستگاه کمک بگیرد .

تمرین نهم :

بافر در واقع یک حافظه میانی و موقتی ست که میتواند به صورت فیزیکی بخشی از مموری را دربربگیرد . بافر ها در کامپیوتر کاربرد فراوانی دارند . بافر ها از به روش مختلف میتوانند ایجاد شوند هم ازطریق درخواست سیستم عامل . هم از طریق درخواست خود برنامه نویس میتواند ایجاد شود . اما بافر ها از لحاظ محل ایجاد دو نوع هستند :

۱- بافر سخت افزاری : که در واقع برای انتقال اطلاعات دستگاه ها ی ورودی کند به داخل کامپیوتر میباشد . مثلا برای چاپگر ها که کند هستند حافظه بافری اختصاص میدهند و بعد از اینکه بافر آن ها پر شد پردازنده اطلاعات درون بافر به کامپیوتر انتقال ئیدا میکند تا بی جهت وقت پردازنده برای کار با دستگاه های کند گرفته نشود
۲- بافر نرم افزاری : که در واقع بخشی از حافظه اصلی توسط سیستم عامل به بافر تخصیصی میابد و سیستم توسط آن بخش عملیات بافرینگ را انجام میدهد.

در هر دوی موارد بالا این امکان به وجود می آید که بتوان عملیات خواندن و نوشتن از ورودی را به همپوشانی زمانی انجام داد و زمان را کاهش داد.

انواع بافرینگ وجود دارد :

۱- Simple buffering :در این حالت با یک بافر پردازنده باید منتظر بماند تا بافر پر شود سپس به سراغ بافر رفته و اطلاعات آن را به درون سیستم آورده و پرازش کند . در سیستم های single processing این کار باعث معطل ماندن پردازنده میشود و لی اگر multi process باشد ، پردازنده میتواند در زمان پر شدن بافر به کار های دیگر بپردازد .
۲- Double buffering : در این حالت از دو بافر استفاده میشود و هنگامی که یک بافر در حال پر شدن است میتوان بافر پرشده دیگری را توسط پردازنده ، پردازش کرد . در این حالت بدیهی ست که برای کارا بودن این روش باید سرعت پر

شدن بافر کند تر از پردازش پردازنده باشد تا سرعت افزایش پیدا کند ( که عموما همین گونه هم هست و پردازنده ها سریع هستند )

۳- Multi buffering : همانند بالا ست ولی از چندین بافر استفاده میکند .

تمرین دهم :

واسط بین برنامه های کاربردی در حال اجرا و هسته هستند . به این صورت که وقتی برنامه ها در مد کاربر در حال اجرا هستند اگر نیاز به یک فعالیت سیستمی داشته باشند مانند خواندن فایل باید یک فراخوان سیستمی آن را صدا کنند . آن گاه این فراخوان سیستمی که به صورت یک رویه پیاده سازی شده دستگاه را به مد هسته میبرد و کنترل را در اختیار سیستم عامل قرار میدهد و بعد از انجام کار دوباره به حالت قبلی باز میگردد . در نتیجه system calls شبیه رویه های عادی هستند با این تفاوت که دستگاه را از مد کاربر به مد هسته میبرند.

تمرین یازدهم :

در حالت عادی سیستم در چرخه فون نیومن میچرخد . اما برای پردازش وقفه ها که اصولا برای انتقال کنترل از برنامه جاری به برنامه دیگر طراحی شده اند ، در انتهای هر چرخه پردازنده بررسی میکند تا ببیند آیا وقفه ای به صدا در آمده است یا خیر . اگر وقفه ای به صدا درآمده بود آدرس روتین مربوطه آن را از روی جدول از پیش طراحی شده که در حافظه و به صورت ثابت قرار دارد میخواند سپس ثبات pc را به آدرس آن روتین مربوطه انتقال میدهد تا آن روتین را اجرا کند برای اجرا آن محتویات فعلی ثبات ها را در پشته ذخیره میکند تا بعد از اتمام روتین وقفه دوباره کنترل را به برنامه قبلی باز گرداند. البته اجرای وقفه به صدا درآمده در انتهای هر چرخه بستگی به اولویت مربوط به وقفه دارد و اگر اولویتش بالاتر از پردازنده بود آن را اجرا میکند و الا آن را کنار میگذارد برای تغییر اولویت پردازنده میتوان بیت های PSW را تغییر داد.

تمرین دوازدهم :

PSW یک سری ثبات های وضعیتی پردازنده میباشد که بسیار مهم هستند . ثبات های پرچم در PSW هستند همچنین ثبات های مد سیستم ( کاربر یا هسته ) یا کنترل وقفه نیز در PSW وجود دارند . همچنین نکته مهم دیگر این است که پردازنده هنگام وقفه ها PSW را در پشته ذخیره میکند.

تمرین دوازدهم :

الف ) در این شکل hardware مربوط به بخش سخت افزار است یعنی پردازنده و مموری و دستگاه های I/O . همچنین بخش kernel که در واقع همان هسته میباشد برای ارتباط صحیح کاربر با سخت افزار طراحی شده که سیستم عامل در این بخش قرار میگیرد و درخواست های کاربر برای استفاده از سخت افزار را کنترل می کند تا کاربر خواسته یا ناخواسته موجب آسیب رساندن به سخت افزار نشود . بخش shell در واقع پوسته ای ست که برای راحتی کار کاربر در تعامل با سیستم عامل ایجاد گردیده است . shell میتواند پیاده سازی های دلخواهی دستوری ( مانند unix ) و یا گرافیکی مانند ویندوز داشته باشد . لایه بعدی نوعی پیاده سازی همان shell میباشد که به صورت دستوری است و در لایه آخر که user میباشد کاربر شمای فعلی سیستم عامل ها را مشاهده میکند.

ب ) برنامه های کاربردی برای استفاده های شخصی کاربر است که میتواند در لایه command باشد زیرا کاربر آن ها را کاملا میبیند و حس میکند و آن برنامه ها هستند که خود با سیتم عامل ارتباط دارند اما برنامه های سیستمی در لایه shell هستند که کاربر آن ها را نمبیند ولی آن ها در حال تعامل با سیستم عامل هستند . فراخوان های سیستمی هم در لایه kernel هستند تا بتوانند از طریق سیتم عامل با سخت افزار در ارتباط باشند.

ج ) با کلیتش موافقم و لی بهتر است لایه های command و user به عنوان بخشی از لایه shell در نظر گرفته شود

تمرین سیزدهم :

در واقع مجموعه ای از function ها و ورودی و و نوع خروجی آن هاست . برنامه نویسان سیستم عامل های مختلف از API مربوط به سیستم عامل استفاده میکنند تا بتوانند فراخوان های سیستمی را فراخوانی کنند . این روش فواید زیادی دارند یکی از آنها این است که برنامه های نوشته شده میتوانند قابل حمل باشند یعنی بر روی هر سیستمی که از API مورد نظر پشتیبانی میکند قابل اجرا هستند ( البته این موضوع به علت پیچیدگیهای دیگر خیلی قابل دسترس نیست ) از طرفی دیگر کار کردن مستقیم برنامه نویسان با فراخوان های سیستمی سخت میباشد و با طراحی این API ها میتوانند کار با فراخوان سیستمی را ساده نمایند . از انواع API ها

میتوان به Windows API برای سیستم عامل windows و POSIX برای لینوکس و مکینتاش و یونیکس و JAVA API برای JVM نام برد .

تمرین چهاردهم :

وقتی سیستم میخواهد روشن شود و بالا بیاید باید برنامه ای در ابتدا اجرا گردد تا کارهای جاری پردازنده و دستگاه های دیگر به راه بیفتد این برنامه میتواند خیلی ساده باشد . این برنامه که bootstrap program نامیده میشود باید بروی ROM یا EEPROM ذخیره گردد که حافظه ای از پیش گرفته شده برای سیستم است .

تمرین پانزدهم :

در حالت نامتقارن یک پردازنده اجرای سیستم عامل را بر عهده داشته و بقیه به اجرای برنامه های کاربر می پردازند ولی در حالت متقارن سیستم عامل میتواند به صورت همزمان بر روی چند پردازنده و یا بر روی پردازنده دلخواه اجرا شود