پاسخ تمرینات سری هشتم درس سیستم های عامل

محمدرضا غفرانی ۹۶۳۱۰۵۳

# سوال ۲

درسیستم عامل ویندوز ۱۰ حافظه مجازی با استفاده از demand paging و با استفاده از استراتزی clustering پیاده سازی می‌شود. در استراتژی clustering هنگامی که یک page fault رخ می‌دهد، نه تنها آن صفحه‌ای که درخواست داده شده، به حافظه آورده می‌شود، بلکه جند صفحه بعد‌تر از آن صفحه و چند صفحه قبل‌تر از آن به حافظه آورده می‌شود. اندازه cluster بستگی به نوع صفحه دارد. برای صفحاتی که شامل داده[[1]](#footnote-1) هستند و نه دستور[[2]](#footnote-2)، اندازه cluster برابر ۳ است یعنی موقع آوردن یک صفحه درخواست داده شده یک صفحه بعدتر و یک صفحه قبل‌تر از آن نیز به حافظه دعوت می‌شود. سایر انواع صفحات دارای cluster با اندازه هفت هستند.

# سوال ۳

# 

همانطور که در تصویر نیز دیده می‌شود سیستم‌عامل Linux برای آن که بتواند از آدرس دهی سه سطحی پشتیبانی کند، حافظه را به چهار قسمت تقسیم می‌کند. شیوه دستیابی به یک صفحه به این شکل است که ابتدا با استفاده از ثبات cr3 به محل Page directory در حافظه رم پی می‌بریم، هر برنامه در حال اجرا یک Page directory در حافظه رم دارد. اندازه Page directory به اندازه اندازه یک صفحه است. هر یک از مدخل های Page directory به یک صفحه در Page middle directory اشاره می‌کند. حال با استفاده از Page directory به آدرس Page middle directory پی می‌‌بریم. طول Page middle directory می‌تواند به اندازه چندین صفحه باشد. هر مدخل در Page middle directory به یک صفحه در Page table اشاره می‌کند. سپس با استفاده از مدخل مورد نظر در Page middle directory به آدرس Page table دسترسی پیدا می‌کنیم. طول Page table نیز می‌تواند به اندازه جندین صفحه باشد. هر مدخل در Page table به یک فریم در virtual page اشاره می‌کند. بنابراین می‌توانیم با استفاده از Page table به آدرسی از حافظه که صفحه مورد نظر ما در آن ذخیره شده است، دسترسی پیدا کنیم.

# سوال ۵

در واقعیت هیچ مقدار بهینه‌ای برای اندازه صفحه در حافظه وجود ندارد. اگر اندازه صفحه را برزگ در نظر بگیریم در آخرین صفحه‌ای که به یک فرآیند داده می‌شود به طور میانگین نصف آن خالی می‌ماند. و همچنین اگر اندازه صفحه را برزگ در نظر بگیریم، هنگامی که آن صفحه وارد حافظه می‌شود حافظه را اشغال می‌کند اما تنها بخشی از این حافظه را که اشغال کرده است را استفاده می‌کند و بقیه حافظه را عملا بیهوده اشغال کرده است. در طرف دیگر اگر اندازه صفحات را کوچک در نظر بگیریم اندازه page table افزایش می‌یابد. همچنین در هنگامی که اندازه صفحات را کوچک در نظر بگیرم زمان زیادی صرف جابه‌جایی و آوردن این صفحات از حافظه دیسک به حافظه رم می‌شود. همانطور که گفتیم اگر اندازه صفحات رو کوچک در نظر بگیریم اندازه page table افزایش می‌یابد، این عامل باعث می‌شود تا اندازه TLB را زیاد در نظر بگیریم تا بتوانیم تمامی صفحات را در آن جا دهیم ولی باید اندازه TLBکوچک باقی بماند تا بتوان در سریعترین زمان ممکن به حافظه مورد نظر دست یافت. بنابراین از نظر TLB هر چه اندازه صفحات بزرگ‌تر باشد بهتر است. در عمل سیستم‌عامل‌ها برای قسمت kernel اندازه صفحه را بزرگ و برای فرآیند‌های کاربر اندازه صفحه را کوچک در نظر می‌گیرند. در برخی از ماشین‌ها هنگامی که پردازنده از یک فرآیند به فرآیند دیگر می‌رود باید page table در ثبات‌های پردازنده قرار گیرد. ازنظر این ماشین ها نیز هر چه اندازه صفحات بزرگ‌تر باشد بهتر است و زمان کمی نیاز است تا آدرس آن‌ها در ثبات پردازنده قرار گیرد.

حال می خواهیم از ریاضیات کمک بگیریم و با استفاده از آن محاسبه کنیم که حالت بهینه اندازه صفحه چقدر است. برای این کار فرض کنیم میانگین اندازه فرآیندها s بایت، ندازه صفحه p بایت باشد. فرض می‌کنیم هر مدخل در جدول page table نیاز به e بایت برای ذخیره سازی نیاز دارد. همچنین نصف آخرین صفحه تخصیص داده به فرآیند خالی می‌ماند. بنابراین overhead کلی برای یک فرآیند به صورت زیر است:

با کاهش اندازه صفحه عبارت اول زیاد می‌شود و با افزایش آن مقدار عبارت دوم زیاد می‌شود؛ بنابراین، به نظر می‌رسد که یک مقدار بهینه برای آن وجود داشته باشد. از عبارت نسبت به pمشتق گرفته و آن را مساوی صفر قرار می‌دهیم:

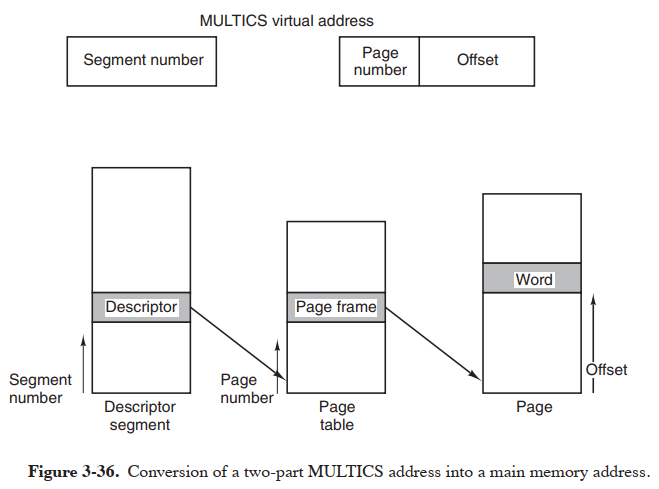
مقدار p برابر می‌شود با: بنابراین هنگامی که s = 1 MB باشد و e = 8 باشد بهترین اندازه صفحه 4 KB است. در عمل نیز معمولا اندازه صفحه را برابر 4 KB در نظر می‌گیرند.

# سوال ۷

در سیستم‌عامل ویندوز ۱۰ هنگامی که یک فرآیند به حافظه آورده می‌شود، حداقل ۵۰ صفحه و حداکثر ۳۴۵ صفحه به آن فرآیند اختصاص می‌دهد. به حداقل صفحه‌ای که سیستم‌عامل تضمین می‌کند که در اختیار قرآیند قرار دهد working set minimum و به حداکثر صفحه‌ای که سیستم‌عامل در اختیار فرآیند قرار می‌دهد working set maximum گفته می‌شود. باید توجه داشت که ویندوز ۱۰ تنها در صورتی به یک برنامه اجازه رسیدن به working set maximum را می‌دهد که فضای کافی در حافظه وجود داشته باشد. البته تا زمانی که محدودیت های سنگینی روی اندازه working set نباشد، اندازه working set یک فرآیند ممکن است از حداکثر مقدار آن بیشتر و یا حتی از حداقل میزان ممکن برای آن کمتر باشد. ویندوز از الگوریتم جایگزینی LRU-approximation clock برای جایگزینی صفحات استفاده می‌کند. همچنین این سیستم‌عامل هم از جایگزینی محلی و global نیز استفاده می‌کند. بخشی که مدیریت حافظه مجازی را به عهده دارد، لیستی از قسمت هایی از حافظه که خالی است را در اختیار دارد، همچنین یک مقدار threshold نیز دارد که با استفاده از آن مشخص می‌‌کند آیا در فضای خالی کافی در حافظه وجود دارد یا خیر. هنگامی که یک page fault رخ می‌دهد، اگر فضای خالی کافی در حافظه وجود داشته باشد، سیستم بدون توجه به این که آیا فرآیند از مقدار working set maximum عبور کرده یا خیر، یک فضای خالی جدید را از بین فضاهای‌ خالی موجود در سیستم به آن فرآیند اختصاص می‌دهد. هنگامی که فضای خالی کافی در حافظه وجود ندارد، Kernel وارد عمل شده و از فضایی که به فرآیند اختصاص داده شده طبق الگوریتم LRU یک صفحه را انتخاب کرده و عوض می‌کند.

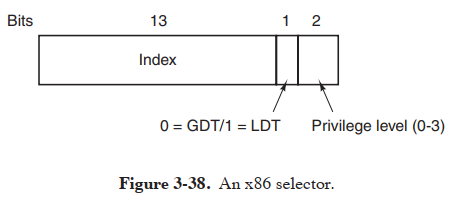
هنگامی که فضای خالی حافظه از مقدار threshold کمتر شود، automatic working-set trimming وارد عمل می‌شود تا حافظه را خالی کند. شیوه کار automatic working-set trimming به این شکل است که ابتدا فرآیندهای بزرگ و بیکار بررسی می شوند که آیا از میزان working-set minimum بیشتر فضا در اختیار گرفته‌اند یا خیر، در مرحله بعد فرآیندهای کوچک که مدام فعال بوده‌اند بررسی می‌شوند.Automatic working-set trimming کار خود را تا آنجا ادامه می‌دهد که فضای خالی حافظه به حد معینی برسد و اگر مجبور شود ممکن فضای تخصیص داده شده به برخی فرآیندها را به زیر working-set minimum نیز برساند.

# سوال ۹

تا پیش از ارائه پردازنده های x86\_64، رفتار حافظه مجازی در پردازنده‌های x86 هنگامی که هم می‌خواستیم از segmentation و هم از paging استفاده کنیم، شباهت زیادی یه پردازنده‌های MULTICS داشت. تفاوت پردازنده‌های x86 با پردازنده‌های MULTICSدر آن بود که پردازنده‌های MULTICS تعداد segmentationها زیاد بود ولی اندازه هر segment کوجک بود ولی این قضیه در مورد پردازنده‌های x86 برعکس بود در این پردازنده تعداد segmentationها نسبت به پردازنده‌های MULTICS کمتر بود ولی اندازه هر segment بزرگ بود. مشی x86 به نظر منطقی‌تر می‌رسد چون تعداد برنامه هایی که تعداد زیادی segment بخواهند می‌شود گفت بسیار کم است ولی تعداد برنامه‌هایی که نیاز به segmentهای بزرگ‌تری دارند، خیلی زیاد است.

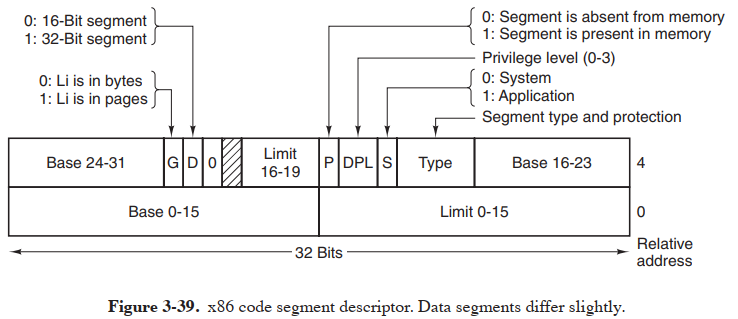
با ارائه پردازنده‌های x86\_64 از روش segmentation دیگر پشتیانی نشد. گرچه می‌توان در حالت legacy از روش segmentation بهره برد. البته هنوز هم به خاطر برنامه‌های قدیمی از این روش پشتیبانی می‌شود ولی دیگر همانند سابق کار نمی‌کند و segmentation واقعی را پشتیبانی نمی‌کند.

قلب حافظه مجازی پردازنده x86 از دو جدول به نام های LDT(Local Descriptor Table) و GDT(Global Descriptor Table) تشکیل شده است. هر برنامه LDT مخصوص به خود را دارد ولی GDT بین همه برنامه‌ها به اشتراک گذاشته می‌شود.LDT اطلاعات مربوط به segment های آن برنامه را که شامل کد برنامه، داده‌های مربوط به آن برنامه و همجنین پشته آن برنامه را نگهداری می‌کند. GDT اطلاعات مربوط به system segment که شامل خود سیستم‌عامل نیز می‌شود را نگهداری می‌کند.

برای دسترسی به یک segment، بک برنامه ابتدا selector مربوط به آن segment را در یکی از شش ثبات segment register قرار می‌دهد. در هنگام اجرا ثبات CS مقدار selector مربوط به code segment و ثبات DS مقدار selector مربوط به data segment را نگه می‌دارد. هر selector یک عدد ۱۶بیتی است که در شکل زیر نشان داده شده‌است.

یک selector مشخص می‌کند که آیا آن segment یک segment محلی و یا global است. سیزده بیت دیگر مدخل آن در LDT و یا GDT را مشخص می‌کند. با توجه به این که این قسمت ۱۳ بیت در نظر گرفته شده است بنابراین طول segment descriptor حداکثر 8 KB می‌تواند باشد. دو بیت دیگر مربوط به حفاظت است. همچنین descriptor صفر غیرمجاز تلقی می‌شود. این مقدار ممکن است به یک segment register داده شود تا مشخص کند که این ثبات فعلا در دسترس نیست.

هنگامی که یک selector در یک segment register بارگزاری می‌شود، descriptor مربوط به آن از LDT و یا GDT خوانده می‌شود و در ثبات‌های microprogram ذخیره می‌شود. همان طور که در شکل زیر دیده می‌شود، descriptor از ۸ بایت تشکیل شده است و شامل آدرس پایه segment، اندازه و سایر اطلاعات است.

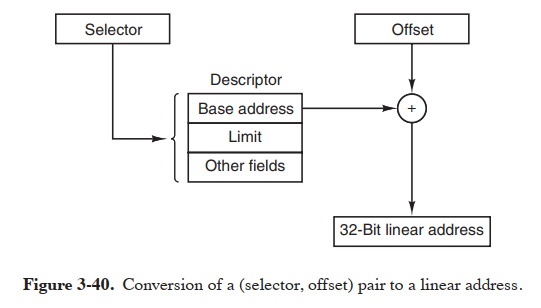


فرمت selector به این شکل است که ابتدا بر اساس بیت دوم selector یکی از GDT و یا LDT انتخاب ‌می‌شود. سپس مقدار selector به یک ثبات internal scratch register منتقل می‌شود و سه بیت کم ارزش آن صفر می‌شود. درنهایت، آدرس هر یک از جدول LDT و یا GDT با آن جمع می‌شود تا آدرس مربوط به descriptor به دست آید.

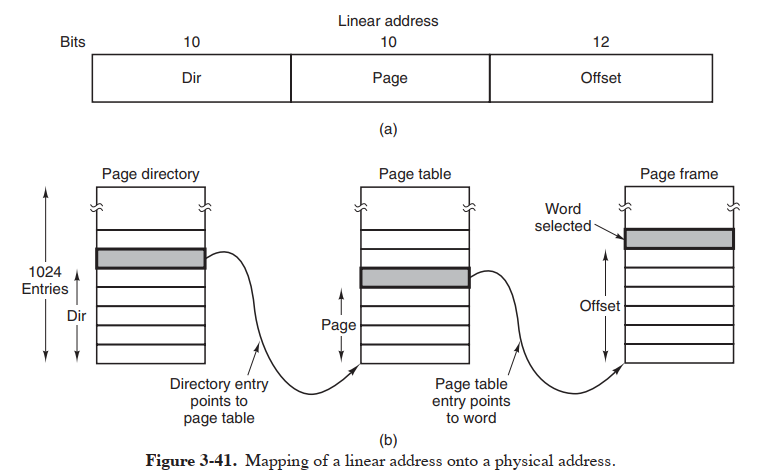
حال می‌خواهیم مرحله‌ای که جفت (selector, offset) به آدرس فیزیکی تبدیل شده‌اند را بررسی کنیم. هنگامی که microprogram می‌فهمد کدام segment register باید استفاده شود، می‌تواند descriptor مربوط به آن را در internal register خود بیاید. هنگامی که segment وجود نداشته باشد ( مقدار selector صفر باشد) و یا در حافظه رم وجود نداشته باشد، یک trap رخ می‌دهد.

سپس با استفاده از Limit چک می‌‌شود که آیا offset فراتر از آخر segment است. در صورتی که offset فراتر از آخر segment باشد یک trap رخ می‌دهد. منطقا، باید یک قسمت ۳۲ بیتی در descriptor وجود داشته باشد که اندازه segment را به ما بدهد ولی تنها ۲۰ بیت برای این کار در اختیار ما است. بنابراین روش دیگری برای این کار استفاده می‌شود و آن استفاده از Gbit (Granularity) است. اگر این بیت صفر باشد اندازه Limit دقیقا برابر اندازه segment است. ولی اگر مقدار آن یک باشد مقدار Limit طول segment را بر حسب طول page می‌دهد. اگر اندازه صفحات را 4 KB در نظر بگیریم ۲۰ بیت برای اندازه segment ها تا ۲۳۲ بایت را کفایت می‌کند.

فرض کنیم که segment در حافظه باشد و مقدار offset نیز معتبر باشد، در این مرحله پردازنده ۳۲ بیت آدرس base که در descriptor است را با offset جمع می‌کند تا آدرسی به نام linear address (آدرس خطی) به دست آید. همانطور که در تصویر نشان داده شده است آدرس base خود به سه قسمت تقسیم شده‌است و به منظور سازگاری با 286 در تمام طول descriptor پخش شده‌‌است. در نتیجه مقدار Base اجازه می‌دهد که هر segment در یک مکان اختیاری از حافظه در طول آدرس خطی ۳۲ بیت آغاز شود.



اگر paging غیرفعال شده باشد، آدرس خطی به صورت آدرس فیزیکی تعبیر می‌شود. بنابراین اگر segmentation غیرفعال باشد، ما یک سیستم مبتنی بر segmentation در اختیار داریم، که آدرس هر segment در descriptor مربوط به آن نگهداری می‌شود. نکته‌ای که باید به آن توجه کرد این است که segment ها از overlapping جلوگیری نمی‌کنند، شاید به این دلیل که هزینه پیاده سازی آن فزاینده خواهد بود.

در طرف دیگر، اگر paging فعال باشد، آدرس خطی یک آدرس مجازی تعیبر می‌شود و از طریق page table ها به صورت زیر به آدرس فیزیکی تبدیل ‌می‌شود. در این جا از page table های دو مرحله‌ای استفاده می‌شود. هر برنامه در حال اجرا یک page directory دارد که شامل ۱۰۲۴ مدخل ۳۲ بیتی است. محل این page directory با استفاده از global register مشخص شده است. هر یک از مدخل های این page directory به یک page frame اشاره می‌کند.

همانطور که در شکل بالا دیده می‌شود، یک آدرس خطی به سه بخش تقسیم می شود، Dir، Page و offset. مقدار Dir برای این استفاده می‌شود که محل اشاره‌گر به page table مناسب به دست آید. سپس با استفاده از مقدار قسمت Page به محل page frame پی می‌بریم. درنهایت، با استفاده از مقدار Offset به محل واقعی آدرس مجازی در حافظه پی می‌بریم.

گرچه هر یک از مدخل های page table از ۳۲ بیت تشکیل شده‌است اما تنها ۲۰ یت آن مربوط به page frame است. باقی بیت‌ها شامل contain access bit، dirty bit، بیت هایی که سخت‌افزار برای بهره‌بردن بیشتر سیستم‌عامل قرار داده است، بیت های کنترلی و سایر بیت ها می‌شود.

هر page table شامل مدخل‌هایی برای ۱۰۲۴ صفحه با اندازه 4 KB است؛ بنابراین هر page table ۴ مگابایت از حافظه را مدیریت می‌کند. segment هایی که کمتر از 4MB هستند یک page directory با یک مدخل خواهند داشت که یک اشاره‌گر به تنها page table آنهاست. با این روش overhead مربوط به segment های کوتاه تنها دو صحفه خواهد بود،

برای جلوگیری از مراجعه مکرر پردازنده به حافظه برای تبدیل آدرس از TLB نیز استفاده می‌شود. این حافظه مقدار آخرین Dir-Page را به همراه آدرس فیزیکی که به آن نگاشت شده را نگه می‌دارد. هنگامی که این داده در TLB نباشد اطلاعات از حافظه خوانده می‌شود.

همچنین اگر بعضی از برنامه‌ها نیازی به paging نداشته باشند و تنها یک صفحه‌ای باشند ولی به تمامی ۳۲ بیت آن نیاز داشته باشند نیز به خوبی در اینجا مدیریت می‌شود ، در این حالت می‌توان مقدار segment register را به selector هم داد و همچنین در descriptor برنامه مقدار Base را صفر و مقدار Limit را هم برابر ماکزیمم میزان ممکن قرار می‌دهیم. Offset دستورالعمل در این حالت تنها با یک فضای خالی آدرس خطی خواهد بود.

1. data [↑](#footnote-ref-1)
2. instruction [↑](#footnote-ref-2)