سیستم‌های عامل

نیم‌سال اول ۰۳-۰۲

دانشکده‌ی مهندسی کامپیوتر

شماره دانشجویی: ۹۹۱۰۵۵۶۱

شایان صالحی

تمرین چهارم

پاسخ مسئله‌ی ۱.

در جواب به این سوال می‌توان گفت که پیج‌های حافظه به یک اندازه نیستند و به یک اندازه بودن آنها می‌تواند مشکلاتی همانند مشکلات زیر به وجود بیارن:

- بهره‌وری ناکارآمد حافظه (تکه‌تکه شدن داخلی): وقتی فرآیندها به کل اندازه استاندارد صفحه نیاز ندارن، بخش استفاده نشده صفحه هدر می‌ره. به این موضوع تکه‌تکه شدن داخلی می‌گن.
- مشکلات عملکرد برای داده‌های حجیم: برای فرآیندهایی که با مجموعه‌های داده بزرگ سروکار دارن، استفاده از صفحات با اندازه استاندارد می‌تونه منجر به افزایش بار مدیریت صفحات شه و دسترسی به حافظه رو کند کنه.
- انعطاف‌پذیری محدود: صفحات با اندازه ثابت ممکنه برای همه نوع برنامه‌ها، به خصوص اونهایی که نیازهای متغیر حافظه‌ای دارن، بهینه نباشن.

حالا راه‌هایی که می‌تونیم این مشکل رو حل کنیم به این صورته:

- اندازه‌های متفاوت صفحه (ترکیبی از اندازه‌های صفحه): بعضی از سیستم‌های مدرن از بیشتر از یک اندازه صفحه پشتیبانی می‌کنن. این روش به سیستم عامل اجازه می‌ده تا حافظه رو به شکلی تخصیص بده که بیشتر به اندازه مجموعه داده‌های یک فرآیند مناسب باشه. مثلاً، صفحات بزرگ می‌تونن برای برنامه‌هایی که نیاز به حافظه زیادی دارن استفاده شن، در حالی که صفحات کوچکتر برای فرآیندهای کم‌تقاضا به کار می‌رن.
- صفحات بزرگ و صفحات عظیم (HugePages): تو لینوکس و بعضی از سیستم‌های عامل شبه یونیکس، یه ویژگی به نام HugePages وجود داره که اجازه استفاده از اندازه‌های بزرگتر صفحه (مثل ۲ مگابایت یا ۱ گیگابایت) رو برای برنامه‌های خاص می‌ده، که باعث کاهش بار مدیریت تعداد زیادی صفحات می‌شه.
- صفحه‌بندی تقاضامحور و جابجایی (Swapping): اگرچه راه‌حل مستقیمی برای مشکل اندازه صفحه نیست، صفحه‌بندی تقاضامحور (بارگذاری صفحات به حافظه فقط زمانی که نیاز هستند) و جابجایی (انتقال صفحات بین رم و دیسک) می‌تونه تاثیر اندازه‌های ناکارآمد صفحه رو با بهینه‌سازی نحوه استفاده از حافظه کاهش بده.

هر کدوم از این راه‌حل‌ها با معایب خودشون همراه هستن. مثلاً، در حالی که اندازه‌های بزرگتر صفحه می‌تونه بار مدیریت تعداد زیادی صفحه رو کم کنه، می‌تونه منجر به تکه‌تکه شدن داخلی بیشتر شه اگر درست مدیریت نشه. به همین دلیل، سیستم‌های عامل اغلب مکانیزم‌هایی رو فراهم می‌کنن تا این عوامل رو بر اساس نیازهای سیستم و برنامه‌هایش تعادل ببخشن.

پاسخ مسئله‌ی ۲.

پاسخ مسئله‌ی ۳.

در اینجا می‌دانیم که به صورت کلی در سیستم‌های ۳۲ بیتی PTE size ۴ بایت یا همان ۳۲ بیت است. حال برای پیدا کردن تعداد PTEها خواهیم داشت:

$$Number\,of\,PTEs = \frac{Page\,Size}{PTE\,Size} = \frac{۴۰۹۶}{۴} = ۱۰۲۴$$

در مورد استفاده از حافظه واقعی برای کمتر کردن سایز Page Table، سیستم عامل بخشی از حافظه فیزیکی (واقعی) رو برای ذخیره جدول‌های صفحه استفاده می‌کنه. اما، لازم نیست همه جدول‌های صفحه کاملاً پر شده باشند یا حتی همیشه در حافظه باشند.

همچنین روش Demand Paging سیستم عامل اجازه می‌ده فقط اون صفحاتی (و در نتیجه جدول‌های صفحه) رو به حافظه بارگذاری کنه که در حال حاضر نیازه، و اینکار اثر حافظه‌ای جدول‌های صفحه رو کاهش می‌ده.

تو سیستم جدول صفحه چند سطحی، فقط جدول صفحه سطح بالاتر (سطح_۲ در این مورد) باید کاملاً در حافظه حضور داشته باشه. جدول‌های صفحه سطح پایین‌تر (سطح_۱) به موقع بارگذاری می‌شن. این کار به شدت میزان حافظه واقعی لازم برای جدول‌های صفحه رو کاهش می‌ده.

در مورد آدرس ۳۲ بیتی هم ما سه بخش زیر را داریم:

- آفست ۱۲ بیتی: مکان دقیق درون یک صفحه رو مشخص می‌کنه (چون$۴۰۹۶ = ۲^{۱۲}$، که اندازه صفحه است). آفست ۱۲ بیتی برای پیدا کردن بایت دقیق در صفحه ۴ کیلوبایتی که فرآیند می‌خواد بهش دسترسی داشته باشه، استفاده می‌شه.
- جدول صفحه سطح_۱ ۱۰ بیتی: این قسمت آدرس برای ایندکس کردن در جدول صفحه سطح_۱ استفاده می‌شه. ۱۰ بیت برای جدول صفحه سطح_۱ استفاده می‌شه تا در جدول صفحه سطح_۱ مشخص شده (که توسط PTE سطح_۲ اشاره شده) ایندکس بشه، که دوباره می‌تونه ۱۰۲۴ ورودی داشته باشه.
- جدول صفحه سطح_۲ ۱۰ بیتی: این قسمت آدرس برای ایندکس کردن در جدول صفحه سطح_۲ استفاده می‌شه. ۱۰ بیت برای جدول صفحه سطح_۲، یکی از ۱۰۲۴ ($۲^{۱۰}$) ورودی‌های موجود در جدول صفحه سطح_۲ رو ایندکس می‌کنه. این ورودی به یک جدول صفحه سطح_۱ در حافظه اشاره می‌کنه.

این ساختار اجازه می‌ده تا مدیریت یک فضای آدرس مجازی بزرگ (۴ گیگابایت در یک سیستم ۳۲ بیتی) انجام بشه، در حالی که اثر حافظه‌ای جدول‌های صفحه رو با استفاده از تکنیک‌های صفحه‌بندی سلسله مراتبی و صفحه‌بندی تقاضامحور، قابل مدیریت نگه می‌داره.

پاسخ مسئله‌ی ۴.

الف

در اینجا از آنجایی که میانگین زمان دسترسی به حافظه $۵۰ ns$ بوده و دسترسی TLB هم $۱۰ ns$ است. حال دو حالت داریم:

- در حالت TLB hit زمان دسترسی به این صورت است:

$$time\,to\,access\,memory = ۱۰\ ns(TLB) + ۵۰\ ns(Memory) = ۶۰\ ns$$

- در حالت TLB miss داریم:

$$time\,to\,access\,memory = ۱۰\ ns(TLB) + ۵۰\ ns(PageTable) + ۵۰\ ns(Memory) = ۱۱۰\ ns$$

حال با توجه به ضریب TLB ۵۰ درصد خواهیم داشت:

$$Avarage\,Time = \frac{۱}{۲} \times ۶۰\ ns + \frac{۱}{۲} \times ۱۱۰\ ns = ۸۵\,ns$$

حال درصورتی که TLB نداشته باشیم داریم:

$$Avarage\,Time = ۵۰\ ns(PageTable) + ۵۰\ ns(Memory) = ۱۰۰\ ns$$

با قیاس کردن این دو حالت می‌توان فهمید که روش TLB با ضریب ۵۰ درصد می‌تواند میانگین زمان دسترسی را ۱۵ نانو ثانیه کاهش دهد.

ب

اگر ضریب hit rate را H در نظر گرفته و زمان میانگین را T در نظر بگیریم داریم:

$$T = H \times (Time\,for\,TLB\,hit) + (۱ - H) \times (Time\,for\,TLB\,miss)$$

پس داریم:

$$T = ۶۱\ ns, \quad Time\,for\,TLB\,hit = ۶۰\ ns, \quad Time\,for\,TLB\,miss = ۱۱۰\ ns$$

$$\implies ۶۱ = H \times ۶۰ + (۱ - H) \times ۱۱۰ \implies ۵۰H = ۴۹ \implies H = ۰/۹۸$$

بنابراین ضریب TLB می‌بایست ۹۸ درصد باشد تا میانگین زمان ۶۱ نانو ثانیه شود..

پاسخ مسئله‌ی ۵.

پاسخ مسئله‌ی ۶.

پاسخ مسئله‌ی ۷.