



پاسخ مسئله‌ی ۱.

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

- بهره‌وری ناکارآمد حافظه (تکه‌تکه شدن داخلی): وقتی فرایندها به کل اندازه استاندارد صفحه نیاز ندارند، بخش استفاده نشده صفحه هدر می‌رود. به این موضوع تکه‌تکه شدن داخلی می‌گویند.
- مشکلات عملکرد برای داده‌های حجیم: برای فرایندهایی که با مجموعه‌های داده بزرگ سروکار دارند، استفاده از صفحات با اندازه استاندارد می‌تونه منجر به افزایش بار مدیریت صفحات شده و دسترسی به حافظه رو کند کند.
- انعطاف‌پذیری محدود: صفحات با اندازه ثابت ممکنه برای همه نوع برنامه‌ها، به خصوص اونهایی که نیازهای متغیر حافظه‌ای دارند، 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 فرایند مناسب باشه. مثلاً، صفحات بزرگ می‌تونن برای برنامه‌هایی که نیاز به حافظه زیادی دارند استفاده شن، در حالی که صفحات کوچکتر برای فرایندهای 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 (مثلاً 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 هستن) 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 PTE ها خواهیم داشت:

$$\text{Number of PTEs} = \frac{\text{Page Size}}{\text{PTE Size}} = \frac{4096}{4} = 1024$$

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 می‌کند. 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 $4096 = 2^{12}$ ، 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 سلسله‌مراتبی و صفحه‌بندی تقاضامحور، قابل مدیریت نگه می‌داره.

پاسخ مسئله‌ی ۴.

الف

در اینجا از آنجایی که میانگین زمان دسترسی به حافظه 50 ns بوده و دسترسی TLB هم 10 ns است. حال دو حالت داریم:

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

$$\text{time to access memory} = 10\text{ ns(TLB)} + 50\text{ ns(Memory)} = 60\text{ ns}$$

- در حالت TLB miss داریم:

$$\text{time to access memory} = 10\text{ ns(TLB)} + 50\text{ ns(PageTable)} + 50\text{ ns(Memory)} = 110\text{ ns}$$

حال با توجه به ضریب TLB ۵۰ درصد خواهیم داشت:

$$\text{Avarage Time} = \frac{1}{4} \times 60\text{ ns} + \frac{1}{4} \times 110\text{ ns} = 85\text{ ns}$$

حال در صورتی که TLB نداشته باشیم داریم:

$$\text{Avarage Time} = 50\text{ ns(PageTable)} + 50\text{ ns(Memory)} = 100\text{ ns}$$

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:

$$T = H \times (\text{Time for TLB hit}) + (1 - H) \times (\text{Time for TLB miss})$$

پس داریم:

$$T = 61\text{ ns}, \quad \text{Time for TLB hit} = 60\text{ ns}, \quad \text{Time for TLB miss} = 110\text{ ns}$$

$$\Rightarrow 61 = H \times 60 + (1 - H) \times 110 \Rightarrow 50H = 49 \Rightarrow H = 0.98$$

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

پاسخ مسئله‌ی ۵.

الف

برای آنکه کمترین تعداد جدول صفحات را پیدا کنیم می‌دانیم می‌دانیم که Page Size ما ۶۴ کیلوبایت بوده و سائز Entry ما ۴ بایت بوده، به این ترتیب داریم:

$$\text{Entries of Number} = \frac{\text{Size Page}}{\text{Size Entry Table Page}} = \frac{65536 \text{ bytes}}{4 \text{ bytes/entry}} = 16,384$$

هر جدول صفحه می‌تونه ۱۶۳۸۴ ورودی داشته باشه (همانطور که محاسبه شد). چون هر جدول صفحه در یک صفحه جا می‌گیره، حداقل اندازه لازم برای هر جدول صفحه، اندازه یک صفحه است، که ۶۴ کیلوبایت است.

ب

در ابتدا می‌بایست بیت‌های آفست را تعیین کنیم، به این ترتیب داریم:

$$\text{Bits Offset} = \log_2(65536) = 16 \text{ bits}$$

از آنجایی که آدرس مجازی ما ۲۴ بیت بوده خواهیم داشت:

$$\text{Bits Table Page} = \text{Bits Address Virtual Total} - \text{Bits Offset} = 24 - 16 = 8 \text{ bits}$$

چون ۸ بیت برای جدول صفحه استفاده می‌شود، و هر ورودی مربوط به یک صفحه است، یک جدول صفحه سطح یک برای آدرس دهی کل فضای آدرس مجازی کافی است.

ج

چون فضای آدرس فیزیکی ۳۲-بیتی است، ورودی‌های جدول صفحه باید شامل شماره صفحه فیزیکی به علاوه هر متادیتا دیگری باشند. با فرض اینکه کل آدرس فیزیکی ۳۲-بیتی برای آدرس دهی استفاده می‌شه (که شامل شماره صفحه فیزیکی و آفست درون صفحه می‌شه)، تعداد بیت‌های موجود برای متادیتا بستگی به این داره که چند بیت برای نمایش شماره صفحه فیزیکی لازمه.

شماره صفحه فیزیکی توسط فضای آدرس فیزیکی منهای بیت‌های آفست (که در این مورد برای صفحات فیزیکی و مجازی یکسان است) تعیین می‌شود:

$$\text{Bits Number Page Physical} = \text{Space Address Physical} - \text{Bits Offset} = 32 - 16 = 16 \text{ bits}$$

بنابراین، تعداد بیت‌های موجود برای متادیتا در هر ورودی:

$$\text{Bits Metadata} = \text{Size Entry Table Page} - \text{Bits Number Page Physical} = 32 - 16 = 16 \text{ bits}$$

پس، هر ورودی در جدول صفحه می‌تونه ۱۶ بیت متادیتا داشته باشد.

پاسخ مسئله‌ی ۶.

پاسخ مسئله‌ی ۷.