آریا خلیق
۹۵۲۴۰۱۴

تمرین شماره ۵ درس سیستم‌عامل!

بهار ۹۸

۱)

زمانی از رجیسترها برای page table استفاده می‌کنیم که PT کوچک باشد. زمانی که PT بزرگ باشد استفاده از رجیسترهای سریع منطقی و ممکن نیست.

در این حالت PT در حافظه اصلی نگه‌داری شده و PTBR به این PT اشاره می‌کند (به جای اینکه هر بار برای یک سطر کل PT را بگردیم). تغییر PT فقط نیاز به تغییر این رجیستر دارد که در زمان بسیار صرفه‌جویی می‌کند (مانند زمان تعویض متن).

۲)
الف)
در هر سیستم paging دو مشکل اساسی وجود دارد:
۱- تبدیل آدرس مجازی به فیزیکی باید سریع باشد.
۲- اگر فضای حافظه مجازی بزرگ است، PT هم بزرگ خواهد بود.

دلیل مورد اول این است که تبدیل آدرس مجازی به فیزیکی باید روی هر نگاشت حافظه انجام شود. همه دستورات باید از حافظه بیایند و خیلی از آن‌ها یک refrence به یک متغیر دیگر در حافظه هم دارند. به خاطر همین باید یک یا چند ارجاع به PT در هر دستور داشته باشیم.

دلیل مشکل دوم هم این است که همه کامپیوترهای مدرن فضای آدرس حافظه حداقل ۳۲ بیتی دارند. اگر هر صفحه سایز ۴ کیلو بایت داشته باشد یک فضای آدرس ۳۲ بیتی باید ۱ میلیون صفحه داشته باشد. PT هم ۱ میلیون سطر دارد و همچنین هر فرایند هم نیاز به PT خود دارد.

به خاطر همین نیاز به یک page mapping سریع و بزرگ نیاز داریم. یک راه حل ساده این است که یک PT شامل آرایه‌ای از ثبات‌ها با یک ورودی برای هر صفحه مجازی داشته باشیم. زمانی که فرایند شروع به کار کرد، OS ثبات‌ها را به وسیله PT هر فرایند که به وسیله یک کپی که در حافظه اصلی نگه داشته می‌شود لود کند.

ب)
0.8 * (5+60) + 0.2 * (5+60+60) = 69.8

۳)

یک راه حل برای استفاده از PT های یک یا چند لایه‌ای استفاده از inverted PT است. در این روش شماره صفحه آدرس مجازی به یک مقدار hash نگاشت می‌شود (به وسیله یک تابه hash ساده). این مقدار hash یک اشاره‌گر به inverted PT است که دارای entryهای PT است. برای هر صفحه حافظه، یک entry در جدول وجود دارد (در روش‌های قبلی هر صفحه مجازی حافظه این‌گونه بود)

به خاطر خاصیت hashing بیش از یک چند آدرس مجازی می‌توانند به یک سطر hash table اشاره کنند روش chaining این نوع سرریز(به عبارتی مشکل) را رفع می‌کند.

مزایا و معایب:

استفاده از حافظه بسیار کوچک و کم است ولی برای رفع مشکل hashing و یکسان بودن باید از روش chaining استفاده کنیم، این مشکل به میزان قابل توجهی سربار محاسباتی تولید می‌کند ولی در عوض حافظه کم‌تری مصرف می‌کنیم.

۴)

یکی از راه‌های پیاده‌سازی فضاهای آدرس بیش از ۳۲ بیتی استفاده از hashed page table است که متغیری که hash می‌شود شماره صفحه مجازی است. هر ورودی در hash table دارای یک لینک لیست از المان‌هایی هستند که به همان مقدار hash می‌شوند (یک راهکار برای جلوگیری از collision است). هر آلمان از ۳ فیلد تشکیل شده است:

۱- شماره صفحه مجازی

۲- مقدار page frameای که آن را نگاشت می‌کنیم.

۳- یک اشاره‌گر به المان بعدی در همان لینک لیستی که برای جلوگیری از collision ایجاد شده بود.

الگوریتم اینگونه است:

شماره صفحه مجازی داخل یک hash table، هش می‌شود. این مقدار با اولین فیلد از اولین المان لینک لیست مقایسه می‌شود اگر برابر بود فیلد دوم استفاده می‌شود. اگر مساوی نبود المان‌های دیگر لینک‌لیست برای تساوی با فیلد اول این ورودی چک می‌شوند.

۵)

هم قطعه‌بندی و هم صفحه‌بندی مزایای خودشان را دارند.

صفحه‌بندی از تکه‌تکه کردن خارجی و بیشتر جلوگیری می‌کند که برای استفاده از main memory خوب است. الگوریتم‌های این حالت هم باید به گونه‌ای باشند که در رفتار برنامه‌ها مشکل ایجاد نکنند چون همواره بخش‌هایی داخل و خارج می‌شوند. حالت قطعه‌بندی علاوه بر فایده مورد قبل توانایی‌های بیشتری به برنامه‌نویس درمورد استفاده بهینه می‌دهد(مانند استفاده از ساختمان‌داده‌های خاص و ...)

در حالتی از هردوی این موارد را به وسیله ساپورت OS و سخت‌افزار پیاده‌سازی می‌کنیم. در این حالت فضای آدرس کاربر به تعدادی قطعه تقسیم می‌شود که در اختیار برنامه‌نویس است. هر قطعه هم به تعدادی صفحه با سایز یکسان که برابر با همان فریم حافظه اصلی است شکسته می‌شود (مانند صفحه‌بندی).

از دید برنامه‌نویس یک آدرس منطقی از شماره قطعه و offset قطعه تشکیل شده است. در حقیقت آدرس مجازی به ۳ قسمت تقسیم می‌شود: شماره قطعه، شماره صفحه و افست درون آن page و آدرس فیزیکی هم از شماره قاب صفحه و افست درون صفحه تشکیل می‌شود.

۶)

الف)

سیستم‌عامل‌های جدید دارای آدرس منطقی بسیار بزرگی هستند. این بزرگی باعث بزرگ شدن بیش از حد PT می‌شود. در سؤالات قبل علت بزرگی آن برای یک سیستم ۳۲ بیتی را نوشتم.

برای رفع مشکل تبدیل آدرس منطقی به فیزیکی از صفحه‌بندی چندسطحی استفاده می‌کنیم. تعدادی از صفحه‌ها را داخل یک جدول قرار می‌دهیم و دوباره یک سری از همان جداول را داخل جدول دیگری قرار می‌دهیم و این کار را به میزانی که مورد نیاز است انجام می‌دهیم.

ب)

۱-

$32 - 8 - 10 = 14$

۲-

جدول اول دارای ۸ بیت که می‌شود و جدول دوم دارای ۱۰ بیت است که تعداد صفحات برابر با ۲ به توان ۱۸ خواهد بود.

۳-

جدول اول ۸ بیتی است پس ۲ به توان ۸ درایه دارد که ۲۵۶ درایه دارد.

جدول دوم ۱۰ بیتی است پس ۱۰۲۴ درایه دارد.

۷)

1)

110 + 330 = 440

2)

80 + 111 = 191

3)

خطای دسترسی، آدرس درست نیست چون ۲۳۱ از limit و طول آن قطعه بیشتر است!

4)

210 + 498 = 708

5)

خطای دسترسی، آدرس درست نیست چون ۱۱۰ بیشتر از ۹۹ است.

۸)

سیستم‌عامل‌های کامپیوترهای خانگی و شخصی از swapping استفاده می‌کنند ولی این در مورد سیستم‌عامل‌های موبایل صادق نیست. این سیستم‌ها از فلش استفاده می‌کنند. محدودیت فضا یکی از دلایلی است که در این نوع سیستم‌عامل‌ها از swapping استفاده نمی‌شود.

دلایل دیگر این کار:

- محدودیت نوشتن روی حافظه قبل از اینکه دیگر غیر قابل اطمینان شود.
- سرعت کم بین حافظه اصلی و حافظه فلش در این موبایل

سیستم‌عامل IOS از برنامه‌ها به صورت داوطلبانه درخواست می‌کند که حافظه اصلی را رها کنند. کدهای برنامه که read-only هستند از سیستم پاک می‌شوند و در صورت نیاز دوباره لود می‌شوند. داده‌هایی که تغییر داده شده‌اند حذف نمی‌شوند. هر اپلیکیشنی که نتواند حافظه را به صورت مناسبی خالی کند و کم مصرف کند توسط سیستم‌عامل خاتمه داده می‌شود.

Android هم از روشی همانند IOS استفاده می‌کند. در این سیستم‌عامل هم در صورتی که حافظه اصلی غیر کافی باشد برنامه به وسیله OS خاتمه پیدا می‌کند. فقط قبل از خاتمه state و حالت برنامه را ذخیره می‌کند تا در صورت restart برنامه از آن استفاده شود.