سیستم‌های عامل - دکتر ابراهیمی‌مقدم

امیرحسین منصوری - ۹۹۲۴۳۰۶۹

تمرین سری هفتم

سوال ۱

Internal/External Fragmentation

پدیدهٔ Internal Fragmentation وقتی رخ می دهد که حافظه را به تکه‌های کوچکی تقسیم کنیم، و هنگام اختصاص حافظه، صرفا بتوانیم یک یا چندتا از این تکه‌های کوچک را اختصاص دهیم (یعنی نتوان تکهٔ کوچکتری را اختصاص داد). در این حالت، اگر به حافظه‌ای کمتر از اندازه این تکه‌ها نیاز داشته باشیم، یک تکهٔ کامل اختصاص می‌یابد و عملا قسمتی از فضای این تکه هدر می‌رود و استفاده نمی‌شود.

در مقابل، External Fragmentation وقتی رخ می‌دهد که حافظه را به صورت غیر پیوسته اختصاص بدیم. در این صورت بین فضاهای اختصاص داده شده، حفره‌های خالی وجود خواهند داشت و ممکن است این حفره‌ها آنقدر کوچک باشند که قابل استفاده نباشند و نتوان آن‌ها را اختصاص داد. در نتیجه ممکن است در مجموع فضای خالی به اندازهٔ مورد نیاز وجود داشته باشد؛ اما این فضای خالی به صورت حفره‌های کوچک و تکه‌تکه شده پراکنده در حافظه وجود دارد و قابل استفاده نیست.

Static/Dynamic Linking

در روش Static Linking، تمامی کدهای برنامه که در Object fileهای حاصل از کامپایل موجود هستند، در فایل اجرایی نهایی قرار می‌گیرند. یک مزیت این روش این است که نیازی به نصب بودن کتابخانه یا وابستگی خاصی برای اجرای برنامه نیست؛ چون تمامی کدهای مورد نیاز در فایل اجرایی موجود است. یک ایراد این روش این است که اگر دو برنامه داشته باشیم که کد های مشترکی داشته باشند، این کدهای مشترک حین اجرای این دو برنامه، دوبار در حافظه قرار می‌گیرند؛ که چندان بهینه نیست.

اما در مقابل، در روش Dynamic Linking تنها برخی کدهای موجود در Object Fileها در فایل اجرایی برنامه قرار می‌گیرند و فقط ارجاعاتی به کدهای دیگر در برنامه نهایی باقی می‌ماند؛ کدهای دیگر که معمولاً مربوط به کتاب‌خانه‌های استفاده شده است، در فایل‌های جداگانه قرار می‌گیرند (مثلاً در ویندوز، این فایل‌ها به فایل‌های DLL مشهورند). سپس در زمان اجرای برنامه، سیستم‌عامل این کدهای دیگر را در حافظه قرار می‌دهد و عمل Linking را حین اجرا و در زمان نیاز انجام می‌دهد. یک مزیت این روش، این است که اگر دو برنامه در حال اجرا، از کتاب‌خانه‌های مشترکی استفاده کرده باشند، سیستم‌عامل می‌تواند این کتاب‌خانه را تنها یک‌بار در حافظه قرار دهد و هر دو برنامه را به قسمت یکسانی از حافظه که کد در آن قرار گرفته ارجاع دهد. مزیت دیگر، سبک‌شدن حجم فایل اجرایی نهایی است. از معایب این روش نیز نیازمند بودن به نصب بودن کتاب‌خانه‌های موردنیاز بر روی سیستم موردنظر است.

Logical/Physical Address

Logical Address، آدرسی است که توسط پردازنده حین اجرای برنامه تولید می‌شود؛ و نسبت به فضای آدرس دهی هر پروسه مشخص می‌شود. این آدرس باید بعداً به یک Physical Address ترجمه شود. Physical Address آدرسی است که در نهایت به حافظه اصلی داده می‌شود؛ و در واقع از دید سخت‌افزار، همان عددی است که به عنوان آدرس روی پایه‌های چیپ حافظه می‌رود.

سوال ۲

First-Fit

Allocation request	Allocated Hole	State of holes
5k	10k	**5k**, 12k, 4k, 8k, 6k
8k	12k	5k, **4k**, 4k, 8k, 6k
3k	5k	**2k**, 4k, 4k, 8k, 6k
6k	8k	2k, 4k, 4k, **2k**, 6k
10k	-	2k, 4k, 4k, 2k, 6k

Best-Fit

Allocation request	Allocated Hole	State of holes
5k	6k	10k, 12k, 4k, 8k, **1k**
8k	8k	10k, 12k, 4k, **0k**, 1k
3k	4k	10k, 12k, **1k**, 1k
6k	10k	**4k**, 12k, 1k, 1k
10k	12k	4k, **2k**, 1k, 1k

Best-Fit

Allocation request	Allocated Hole	State of holes
5k	12k	10k, **7k**, 4k, 8k, 6k
8k	10k	**2k**, 7k, 4k, 8k, 6k
3k	7k	2k, **4k**, 4k, 8k, 6k
6k	8k	2k, 4k, 4k, **2k**, 6k
10k	-	2k, 4k, 4k, 2k, 6k

در نهایت، تنها الگوریتم Best-Fit موفق شده تمام درخواست‌ها را پاسخ دهد؛ و حتی نیز موفق شده یکی از حفره‌ها را از بین ببرد. الگوریتم‌های دیگر در پاسخ به درخواست آخر موفق نبوده‌اند؛ حتی با این که حافظه کافی در سیستم موجود است.

سوال ۳

الف)

یک دسترسی برای پیدا کردن Frame number مربوطه، و یک دسترسی برای خواندن صفحه مربوطه نیاز داریم؛ پس در کل ۲ دسترسی به حافظه اصلی نیاز است و این دو دسترسی $80ns \times 2 = 160ns$ زمان خواهد برد.

ب)

اگر دسترسی به TLB به صورت سریال باشد:

$$\text{ETA} = 0.7(10 + 80) + 0.3(10 + 80 + 80) = 114ns$$

اگر دسترسی به صورت موازی باشد:

$$\text{ETA} = 0.7(10 + 80) + 0.3(80 + 80) = 111ns$$

سوال ۴

با توجه به اعداد داده شده، Page table ما دارای $\frac{2^3 \times 2^{30}}{2^4} = 2^{29}$ سطر خواهد بود که این عدد، متناظر با تعداد کل صفحات ممکن در فضای آدرس‌دهی Logical خواهد بود و برای نمایش آن، به ۲۹ بیت نیاز داریم. همچنین چون اندازه هر صفحه $2^3 \times 2^{10} = 2^{13}$ است، Page offset ما نیز ۱۳ بیتی خواهد بود. در نتیجه فضای آدرس‌دهی Logical ما $29 + 13 = 42$ بیت خواهد داشت.

سوال ۵

اگر اندازه صفحه بیش از حد بزرگ باشد، میزان و شدت Internal fragementation بالا می‌رود. اگر هم اندازه صفحه بسیار کوچک باشد، اندازهٔ Page table بزرگ می‌شود؛ تا جایی که نمی‌توان آن را مدیریت کرد؛ یا مثلا در حافظه اصلی جا نمی‌شود.

سوال ۶

الف)

پدیدهٔ thrashing وقتی رخ می‌دهد که یک پروسه، بیشتر زمان اجرایش را صرف swap کردن pageها بین حافظه اصلی و حافظه مجازی می‌کند؛ معمولا به این علت که به اندازه کافی و مورد نیاز frame ندارد. این پدیده باعث کاهش Utilization می‌شود و به همین دلیل اگر درست تشخیص داده نشود، ممکن است سیستم‌عامل را وادار کند درجه Multiprogramming را بالا ببرد که اوضاع را بهتر نمی‌کند (و حتی می‌تواند بدتر کند).

ب)

به مجموعهٔ تمام آدرس‌های Virtual که پردازنده می‌تواند بسازد، فضای آدرس‌دهی مجازی گفته می‌شود.

سوال ۷

روند دسترسی به صفحات به صورت زیر است.

Requested Page	Status	Frame 0	Frame 1	Frame 2	Frame 3
5	Page Fault	**5**	-	-	-
4	Page Fault	5	**4**	-	-
2	Page Fault	5	4	**2**	-
7	Page Fault	5	4	2	**7**
5	Hit	5	4	2	7
3	Page Fault	5	**3**	2	7
5	Hit	5	3	2	7
2	Hit	5	3	2	7
7	Hit	5	3	2	7
4	Page Fault	5	**4**	2	7

پس در ۶۰٪ مواقع دچار Page Fault شده‌ایم. بنابراین

$$\text{EAT} = (1 - 0.6) \times (100ns) + (0.6)(1000000ns) = 600040ns$$

سوال ۸

روند اجرای FIFO به صورت زیر است.

Requested Page	Status	Frame 0	Frame 1	Frame 2
9	Page Fault	**9**	-	-
1	Page Fault	9	**1**	-
8	Page Fault	9	1	**8**
9	Hit	9	1	8
2	Page Fault	**2**	1	8
1	Hit	2	1	8
3	Page Fault	2	**3**	8
0	Page Fault	2	3	**0**
1	Page Fault	**1**	3	0
3	Hit	1	3	0
2	Page Fault	1	**2**	0

در نهایت ۸ Page Fault داشتیم. همچنین روند الگوریتم Optimal به صورت زیر است.

Requested Page	Status	Frame 0	Frame 1	Frame 2
9	Page Fault	**9**	-	-
1	Page Fault	9	**1**	-
8	Page Fault	9	1	**8**
9	Hit	9	1	8
2	Page Fault	**2**	1	8
1	Hit	2	1	8
3	Page Fault	2	1	**3**
0	Page Fault	**0**	1	3
1	Hit	0	1	3
3	Hit	0	1	3
2	Page Fault	**2**	1	3

در اینجا ۷ Page Fault رخ داد. بنابراین FIFO به اندازه یک Page Fault ضعیف‌تر عمل کرده است.

سوال ۱۱

چون $512 = 2^9$ فریم داریم، پس ۹ بیت برای مشخص کردن هر فریم کافی است. به طور مشابه، چون $2048 = 2^{11}$ صفحه داریم، پس ۱۱ بیت هم برای مشخص کردن هر صفحه کافی است. در نهایت چون اندازه صفحه $4\text{kb} = 2^{12}\text{bytes}$ است، ۱۲ بیت برای Page offset می‌خواهیم.

با تفاسیر بالا، به $21 = 12 + 9$ بیت برای فضای آدرس‌دهی فیزیکی، و $23 = 12 + 11$ بیت برای فضای آدرس‌دهی مجازی نیاز داریم.