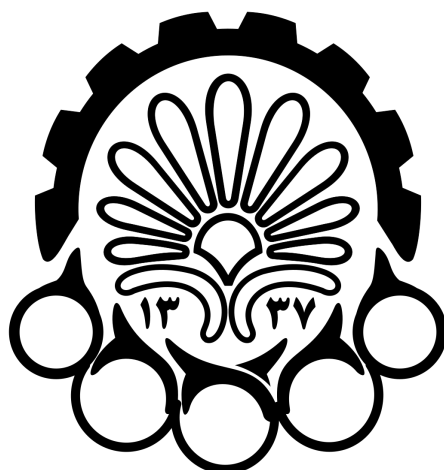


سیستم‌های عامل
دکتر جوادی



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

تمرین سری پنجم

۲۱ دی ۱۴۰۲



سوال اول

در یک سیستم صفحه‌بندی، Page table در حافظه اصلی قرار دارد.

۱. اگر مراجعه به حافظه ۵۰ نانو ثانیه زمان ببرد، چقدر طول می‌کشد که در قالب سیستم صفحه‌بندی به داده یا دستور مورد نظر خود دسترسی پیدا کنیم؟
۲. فرض کنید که TLB را نیز به سیستم اضافه می‌کنیم و پیدا کردن یک مدخل جدول صفحات در ۲ نانو ثانیه زمان می‌برد. اگر ۷۵ درصد از مراجعات جدول صفحات در TLB نیز یافت شود، زمان موثر دسترسی چقدر خواهد بود؟

پاسخ

۱. زمانی که باید به داده یا دستور مورد نظر دسترسی پیدا کنیم، به توجه به مفهوم صفحه‌بندی، از فرمول زیر می‌توان استفاده کرد:

زمان دسترسی = زمان مراجعه به حافظه + زمان پیدا کردن داده در حافظه

معمولاً زمان مراجعه به حافظه در مرتبه نانو ثانیه است. زمان پیدا کردن داده در صفحه نیز بستگی به سازماندهی و سیاست‌های صفحه‌بندی دارد.

۲. زمان موثر دسترسی از رابطه زیر محاسبه می‌شود: زمان موثر دسترسی = زمان مراجعه به TLB + زمان پیدا کردن داده در صفحه

در اینجا فرض کردیم که زمان مراجعه به TLB ۲ نانو ثانیه است. مقدار زمان پیدا کردن داده در صفحه همانند قسمت الف است و به سیاست‌های صفحه‌بندی و سازماندهی حافظه بستگی دارد.

سوال دوم

به سوالات زیر پاسخ دهید:

۱. تعدادی برنامه داریم که به ۳۵۰ مگابایت حافظه برای اجرا نیاز دارند و از روش اختصاصی پیوسته استفاده کرده‌ایم. اگر سیاست First Fit را به کار ببریم و همچنین بخواهیم که به طور ۷۹ درصد از حجم فرآیندها در حافظه اصلی باشد، پیشنهاد می‌دهید که حافظه اصلی با چه ظرفیتی را تهیه کنیم؟ چرا؟

۲. فرض کنید فضای آدرس منطقی ۱۲۸ کیلوبایت، آدرس فیزیکی ۵۱۲ کیلوبایت و Page Size برابر با ۱۶ کیلو بایت باشد. موارد زیر را حساب کنید:

(آ) بیت‌های لازم برای آدرس دهی منطقی

(ب) بیت‌های لازم برای آدرس دهی فیزیکی

(ج) تعداد Page های آدرس منطقی

(د) تعداد فریم‌ها در آدرس فیزیکی

(ه) اندازه Page Table

پاسخ

۱. حجم حافظه مورد نیاز = (حجم برنامه‌ها ÷ درصد استفاده) $\times 100$
- حجم حافظه مورد نیاز = (۳۵۰ مگابایت ÷ ۷۹ درصد) $\times 100 = 443/7$ مگابایت
- پس، برای استفاده از سیاست FirstFit و داشتن حداقل ۷۹ درصد حافظه‌ای پر، می‌توانیم حافظه‌ای با ظرفیت حداقل ۴۴۳/۷ مگابایت تهیه کنیم.
۲. در ادامه هر مورد را بررسی می‌کنیم:

(آ) بیت‌های لازم برای آدرس دهی منطقی: $\log_2(128KB) = 17 \text{ bit}$

(ب) بیت‌های لازم برای آدرس دهی فیزیکی: $\log_2(512KB) = 19 \text{ bit}$

(ج) تعداد Page های آدرس منطقی: تعداد صفحات = آدرس منطقی ÷ اندازه صفحه = $128 \div 16 = 8$ پیج

(د) تعداد فریم‌ها در آدرس فیزیکی: تعداد فریم‌ها = آدرس فیزیکی ÷ اندازه صفحه = $512 \div 16 = 32$ فریم

(ه) اندازه PageTable: $\text{TablePage} = \text{تعداد صفحات} \div \text{تعداد فریم‌ها در هر جدول} = 32 \div 8 = 4$ پیج

سوال سوم

یک سیستم که از MultiLevel Paging استفاده می‌کند را در نظر بگیرید. اندازه هر صفحه ۳۲ کیلو بایت است. حافظه با بایت آدرس‌دهی می‌شود و اندازه آدرس مجازی ۴۸ بیت است. اندازه ورودی Page Table هم ۴ بایت است. چند سطح از Page Table اتخاذ می‌شود؟

پاسخ

برای محاسبه تعداد سطوح TablePage در PagingMultiLevel می‌توانیم از اطلاعات زیر استفاده کنیم:

۱. اندازه آدرس مجازی: ۴۸ بیت

۲. اندازه صفحه: ۳۲ کیلوبایت

۳. اندازه ورودی PageTable: ۴ بایت

حالا می‌توانیم محاسبه کنیم:

۱. تعداد بیت‌ها برای آدرس مجازی: $\log_2(\text{size of virtual address}) = \log_2(2^{48}) = 48\text{bit}$

۲. تعداد بیت‌ها برای آدرس صفحه (Offset): $\log_2(\text{size of page}) = \log_2(2^{15}) = 15\text{bit}$

۳. تعداد بیت‌ها برای آدرس مجازی با توجه به Offset: $48 - 15 = 33\text{bit}$

۴. تعداد بیت‌ها برای هر ورودی TablePage: $\log_2(\text{size of input page table}) = \log_2(2^4) = 4\text{bit}$

۵. تعداد سطوح TablePage: $\frac{33}{4} = 8.25$

اما چون تعداد سطوح باید صحیح باشد، ما به بالا گرد می‌کنیم. بنابراین، از ۹ سطح از TablePage استفاده می‌شود.

سوال چهارم

یک سیستم تقاضا Paging ۱۰۰ واحد زمانی برای هر Page Fault و ۳۰۰ واحد زمانی برای جابه‌جایی یک Dirty Page نیاز دارد و زمان دسترسی به حافظه اصلی ۱ واحد زمانی می‌باشد. اگر احتمال رخداد Page Fault برابر با P باشد و همچنین احتمال Dirty Page بودن به شرط Page Fault نیز برابر با P باشد و میانگین زمان دسترسی برابر با ۳ واحد زمانی، مقدار P را بیابید.

پاسخ

زمان میانگین دسترسی به حافظه = احتمال رخداد پیج‌فالت \times (زمان پیج‌فالت + احتمال پیج درتی \times زمان جابه‌جایی پیج درتی) + (۱ - احتمال رخداد پیج‌فالت) \times زمان دسترسی به حافظه اصلی

۱. زمان میانگین دسترسی به حافظه زمان میانگین دسترسی به حافظه برابر با ۳ واحد زمانی است.

۲. احتمال رخداد احتمال FaultPage رخداد FaultPage با PP نشان داده شده است.

۳. زمان Fault زمان FaultPage برابر با ۱۰۰ واحد زمانی است.

۴. احتمال Page احتمال Dirty Dirty با PP نشان داده شده است.

۵. زمان جابه‌جایی Page زمان Dirty جابه‌جایی PageDirty برابر با ۳۰۰ واحد زمانی است.

۶. زمان دسترسی به حافظه اصلیزمان دسترسی به حافظه اصلی برابر با ۱ واحد زمانی است.

با جای‌گذاری مقادیر بالا در فرمول داریم:

$$3 = P \times (100 + P \times 300) + (1 - P) \times 1$$

با حل معادله بالا در پایتون بازه P به صورت زیر بدست می‌آید:

$$P: [-0.00990243 \ 0.67323576]$$

سوال پنجم

به سوالات زیر پاسخ کامل دهید.

۱. زمان دسترسی موثر را برای رشته ارجاعات زیر برای هر یک از الگوریتم‌های بهینه و FIFO و LRU را بدست آورید.
(تعداد فریم‌ها ۴، سربار خطای صفحه ۵ میلی‌ثانیه و زمان دسترسی به RAM ۵۰۰ نانوثانیه است)

0, 3, 1, 4, 0, 5, 2, 1, 4, 5, 4, 5, 0

۲. با فرض موجود بودن r فریم در حافظه اصلی و n صفحه $r < n$ برای رشته مراجعات زیر، تعداد خطاهای صفحه را برای الگوریتم LRU مشخص کنید.

1, 1, 2, 1, 2, 3, 1, 2, 3, 4, ..., 1, 2, 3, 4, 5, ..., n

پاسخ

۱. برای محاسبه زمان دسترسی موثر به حافظه برای هر الگوریتم، از فرمول زیر استفاده می‌شود:

زمان دسترسی موثر = تعداد صفحات ارجاعی به حافظه \times زمان دسترسی به رم + تعداد خطاهای صفحه \times سربار خطای صفحه

(آ) الگوریتم Optimal: در الگوریتم Optimal، تعداد خطاهای صفحه مساوی با تعداد ارجاعات به صفحات منهای تعداد صفحات موجود در حافظه است. برای رشته داده شده:
تعداد خطاهای صفحه = تعداد ارجاعات - تعداد صفحات مختلف
 $13 - 6 = 7$

(ب) الگوریتم FIFO: در الگوریتم FIFO، هر زمان که یک صفحه جدید وارد حافظه می‌شود، صفحه‌ای که اولین بار وارد حافظه شده بود از حافظه حذف می‌شود.
تعداد خطاهای صفحه = تعداد ارجاعات + تعداد صفحات موجود در حافظه
 $13 + 4 = 17$

(ج) الگوریتم LRU: در الگوریتم LRU، صفحه‌ای که مدت زمان استفاده آن کمترین بوده، از حافظه حذف می‌شود.
تعداد خطاهای صفحه = تعداد ارجاعات + تعداد صفحات موجود در حافظه \times (سربار خطای صفحه - ۱)
 $13 + 4 \times (5 - 1) = 33$

۲. برای محاسبه تعداد خطاهای صفحه در الگوریتم می‌توان از فرمول زیر استفاده کرد.

$$n - r + 1$$

$$n - r + 1 = n - 1$$

سوال ششم

در تمرین قبلی با مفهوم فایل سیستم‌ها در لینوکس آشنا شدیم. برای بررسی فایل سیستم proc می‌توانید از توضیحات موجود در دستور man استفاده کنید:

```
$ man proc
```

در دایرکتوری هر پردازش در فایل سیستم Proc یک فایل به نام maps وجود دارد که اطلاعات مربوط به memory map هر پردازش را در آن نگه می‌دارد.

امتیازی: با استفاده از man proc یکی از این فایل‌ها را مثلاً

```
proc/1/maps
```

را بررسی کنید و هر یک از ستون‌های آن را توضیح دهید که چه چیزی را نمایش می‌دهد.

پاسخ

با دستور زیر فایل maps یکی از PID های دلخواه‌مان را باز می‌کنیم:

```
$ sudo vim /proc/1/maps
```

خروجی فایل به صورت زیر است:

```

7fae0f747000-7fae0f749000 r--p 00000000 fc:01 13135771 /usr/lib/x86_64-linux-gnu/libacl.so.1.1.2301
7fae0f749000-7fae0f74d000 r-xp 00002000 fc:01 13135771 /usr/lib/x86_64-linux-gnu/libacl.so.1.1.2301
7fae0f74d000-7fae0f74e000 r--p 00006000 fc:01 13135771 /usr/lib/x86_64-linux-gnu/libacl.so.1.1.2301
7fae0f74e000-7fae0f74f000 r--p 00007000 fc:01 13135771 /usr/lib/x86_64-linux-gnu/libacl.so.1.1.2301
7fae0f74f000-7fae0f750000 rw-p 00008000 fc:01 13135771 /usr/lib/x86_64-linux-gnu/libacl.so.1.1.2301
7fae0f750000-7fae0f753000 r--p 00000000 fc:01 13109596 /usr/lib/x86_64-linux-gnu/libapparmor.so.1.17.0
7fae0f753000-7fae0f75c000 r-xp 00003000 fc:01 13109596 /usr/lib/x86_64-linux-gnu/libapparmor.so.1.17.0
7fae0f75c000-7fae0f763000 r--p 0000c000 fc:01 13109596 /usr/lib/x86_64-linux-gnu/libapparmor.so.1.17.0
7fae0f763000-7fae0f764000 r--p 00012000 fc:01 13109596 /usr/lib/x86_64-linux-gnu/libapparmor.so.1.17.0
7fae0f764000-7fae0f765000 rw-p 00013000 fc:01 13109596 /usr/lib/x86_64-linux-gnu/libapparmor.so.1.17.0
7fae0f765000-7fae0f769000 r--p 00000000 fc:01 13107300 /usr/lib/x86_64-linux-gnu/libkmod.so.2.4.0
7fae0f769000-7fae0f77a000 r-xp 00004000 fc:01 13107300 /usr/lib/x86_64-linux-gnu/libkmod.so.2.4.0
7fae0f77a000-7fae0f77f000 r--p 00015000 fc:01 13107300 /usr/lib/x86_64-linux-gnu/libkmod.so.2.4.0
7fae0f77f000-7fae0f780000 r--p 00019000 fc:01 13107300 /usr/lib/x86_64-linux-gnu/libkmod.so.2.4.0
7fae0f780000-7fae0f781000 rw-p 0001a000 fc:01 13107300 /usr/lib/x86_64-linux-gnu/libkmod.so.2.4.0
7fae0f781000-7fae0f784000 r--p 00000000 fc:01 13107737 /usr/lib/x86_64-linux-gnu/libaudit.so.1.0.0
7fae0f784000-7fae0f78c000 r-xp 00003000 fc:01 13107737 /usr/lib/x86_64-linux-gnu/libaudit.so.1.0.0
7fae0f78c000-7fae0f7a1000 r--p 0000b000 fc:01 13107737 /usr/lib/x86_64-linux-gnu/libaudit.so.1.0.0
7fae0f7a1000-7fae0f7a2000 r--p 0001f000 fc:01 13107737 /usr/lib/x86_64-linux-gnu/libaudit.so.1.0.0
7fae0f7a2000-7fae0f7a3000 rw-p 00020000 fc:01 13107737 /usr/lib/x86_64-linux-gnu/libaudit.so.1.0.0
7fae0f7a3000-7fae0f7b1000 rw-p 00000000 00:00 0
7fae0f7b1000-7fae0f7b4000 r--p 00000000 fc:01 13107746 /usr/lib/x86_64-linux-gnu/libpan.so.0.85.1
7fae0f7b4000-7fae0f7bd000 r-xp 00003000 fc:01 13107746 /usr/lib/x86_64-linux-gnu/libpan.so.0.85.1
7fae0f7bd000-7fae0f7c0000 r--p 0000c000 fc:01 13107746 /usr/lib/x86_64-linux-gnu/libpan.so.0.85.1
7fae0f7c0000-7fae0f7c1000 r--p 0000f000 fc:01 13107746 /usr/lib/x86_64-linux-gnu/libpan.so.0.85.1
7fae0f7c1000-7fae0f7c2000 rw-p 00010000 fc:01 13107746 /usr/lib/x86_64-linux-gnu/libpan.so.0.85.1
7fae0f7c2000-7fae0f7c4000 r--p 00000000 fc:01 13136378 /usr/lib/x86_64-linux-gnu/libseccomp.so.2.5.4
7fae0f7c4000-7fae0f7d2000 r-xp 00002000 fc:01 13136378 /usr/lib/x86_64-linux-gnu/libseccomp.so.2.5.4
7fae0f7d2000-7fae0f7e0000 r--p 0001e000 fc:01 13136378 /usr/lib/x86_64-linux-gnu/libseccomp.so.2.5.4
7fae0f7e0000-7fae0f7e1000 r--p 0001d000 fc:01 13136378 /usr/lib/x86_64-linux-gnu/libseccomp.so.2.5.4
7fae0f7e1000-7fae0f7e2000 rw-p 0001e000 fc:01 13136378 /usr/lib/x86_64-linux-gnu/libseccomp.so.2.5.4
7fae0f7e2000-7fae0f842000 r--p 00000000 fc:01 13109726 /usr/lib/x86_64-linux-gnu/systemd/libsystemd-core-253.so
7fae0f842000-7fae0f92e000 r-xp 00045000 fc:01 13109726 /usr/lib/x86_64-linux-gnu/systemd/libsystemd-core-253.so
7fae0f92e000-7fae0f92d000 r--p 00131000 fc:01 13109726 /usr/lib/x86_64-linux-gnu/systemd/libsystemd-core-253.so
7fae0f92d000-7fae0f9e1000 rw-p 00150000 fc:01 13109726 /usr/lib/x86_64-linux-gnu/systemd/libsystemd-core-253.so
7fae0f9e1000-7fae0f9e2000 r-xp 001e4000 fc:01 13109726 /usr/lib/x86_64-linux-gnu/systemd/libsystemd-core-253.so
7fae0f9e2000-7fae0f9e4000 rw-p 00000000 00:00 0
7fae0f9e4000-7fae0f9e5000 r--p 00000000 fc:01 13108459 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7fae0f9e5000-7fae0fa0f000 r-xp 00001000 fc:01 13108459 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7fae0fa0f000-7fae0fa10000 r--p 0002b000 fc:01 13108459 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7fae0fa10000-7fae0fa1b000 r--p 00035000 fc:01 13108459 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7fae0fa1b000-7fae0fa1d000 rw-p 00037000 fc:01 13108459 /usr/lib/x86_64-linux-gnu/ld-linux-x86-64.so.2
7ffd02def000-7ffd02e10000 rw-p 00000000 00:00 0 [stack]
7ffd02ec0000-7ffd02ec4000 r--p 00000000 00:00 0 [vvar]
7ffd02ec4000-7ffd02ec6000 r-xp 00000000 00:00 0 [vdso]
7ffd02ec6000-7ffd02ec61000 --xp 00000000 00:00 0 [vsyscall]

```

ساختار هر خط به صورت زیر است:

۱. آدرس شروع-آدرس پایان: نمایانگر محدوده آدرس حافظه است که این ناحیه در آن قرار دارد. آدرس شروع نشان‌دهنده ابتدای ناحیه و آدرس پایان نشان‌دهنده انتهای آن است.

۲. دسترسی‌ها: نشان‌دهنده میزان دسترسی‌ها به حافظه است. به عنوان مثال، r-x نشان‌دهنده دسترسی خواندن و اجرا (ولی نه نوشتن) به حافظه است.

۳. آفست: نشان‌دهنده offset نسبت به نقطه شروع فایل (اگر موجود باشد) یا offset نسبت به ابتدای ناحیه حافظه است.

۴. دسته حافظه: نشان‌دهنده نوع حافظه استفاده شده، مثل heap stack و ... است.

۵. حقوق: نشان‌دهنده حقوق دسترسی به فایل یا ناحیه حافظه است.

۶. دستگاه minumum: در صورتی که ناحیه حافظه minumum باشد، این ستون نشان‌دهنده minor number دستگاهی است که minumum به آن متصل شده است.

۷. شماره minumum:

۸. فایلی که متعلق به حافظه است: اگر ناحیه حافظه به یک فایل متصل باشد، این ستون نشان‌دهنده مسیر فایلی است.