بسم تعالی

سیستم‌های عامل

تمرین هشتم

استاد:

دکتر حسین اسدی

نویسنده :

محمدهومان کشوری

شماره دانشجویی :

99105667

تمرینات تئوری

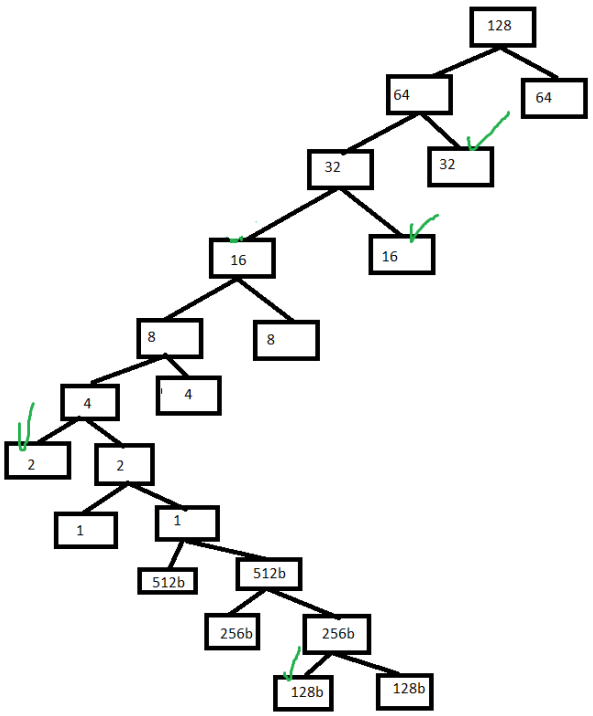
سوال 1.

الف)

می‌دانیم طریقه کار buddy system به این صورت است که حافظه را هی به دوقسمت تقسیم کرده و جلو می‌رود تا در نهایت به کوچک‌ترین قطعه‌ای برسد که در آن بتواند حافظه را تخصیص دهد.

1. ابتدا درخواست 2KB می‌آید و کوچک‌ترین قسمتی که می‌تواند به آن تخصیص دهد را انتخاب می‌کند که یعنی 2KB. پس یک چانک **2KB** به این درخواست تخصیص می‌دهد.
2. این درخواست می‌آید و 13KB درخواست جا می‌کند پس کوچک‌ترین قسمتی که به آن می‌توان تخصیص دهد **16KB** است.
3. با آمدن این درخواست چانک دیگر 2 کیلوبایت را تا جایی تقسیم می‌کند که بتواند 125 بایت را در آن جا دهد یعنی تا **128 بایت**
4. کوچک‌ترین قسمتی که می‌تواند به این درخواست تخصیص دهد **32KB** است.

عملا حافظه به صورت روبه‌رو تقسیم شده است.



2 * 64 ⇒ each 64 = 2 * 32 ⇒ each 32 = 2 * 16 ⇒ each 16 = 2 * 8 ⇒ each 8 = 2 * 4 ⇒ each 4 = 2 * 2

در عکس نیز وضعیت حافظه مشخص است که به تعدادی چانک تقسیم شده که تعدادی از آنها به پردازه‌هایی تخصیص داده شده است.

ب)

طبق شکل متوجه می‌شویم که در کل **10 عملیات split** انجام شده است.

ج)

پس از آزادسازی بلوک 125 بایتی به این دلیل که تا بلوک 2 کیلوبایتی به هیچ پردازه دیگری حافظه تخصیص داده نشده، تا آنجا در هم آمیختن انجام می‌شود پس طبق درخت :

128B + 128B = 256B ⇒ 256B + 256B = 512B ⇒ 512B + 512B = 1KB ⇒ 1KB + 1KB = 2KB

حال دیگر جلو نمی‌رویم چرا که تکه دیگر 2KB هنوز آزاد نشده است!!!.

پس مشاهده می‌کنیم در کل **4 عملیات آمیختن** داریم.

د)

منظور از این سیستم این است که از هر بلوک یکی داریم (که البته فرض می‌کنیم تقسیم‌بندی از قبل انجام شده) پس با ترتیب ورودی‌های ما به بیشتر از یک بلوک نیاز نداریم. پس تعداد **تقسیم‌های ما 0 است.**

حال اگر بخواهیم **خودمان تقسیم را انجام دهیم** : در این حالت ابتدا برای تقسیم باید **4 عدد 32 کیلوبایت** داشته باشیم و سپس یکی را تقیسم می‌کنیم تا به **16B** برسیم سپس حافظه‌ها را تخصیص می‌دهیم یعنی عملا مانند قسمت الف است با این تفاوت که در ابتدا 4 تقسیم داریم و درنهایت با تقسیم تا رسیدن به اندازه بلوک 16B متوقف می‌شویم(بجای بلوک 128B).

پس یعنی عملا 12 عمل تقسیم انجام می‌دهیم (اولی از 128KB یک تقسیم به 4 قسمت 32KB ای و بعد تقسیم یک 32KB ای تا رسیدن به بلوک 16B)

ه)

اگر قرار باشد از هر سایز یک همواره یک بلاک داشته باشیم پس تعداد درهم آمیختن ما **0** می‌شود اما اگر نه، باید ابتدا 3 هم آمیتخن انجام دهیم تا از 16B به 128B برسیم و سپس از 128B هم آمیختن انجام دهیم تا به 1KB برسیم یعنی **4 + 3 = 7 هم آمیختن**.

سوال 2.

Hit rate = 85/100

Miss time = 100ns + 100ns

Hit time = 10ns + 100ns

Page fault = 2/100

Page change = 2ms

الف)

امکان دارد در tlb هیت د**اشته باشیم یا نداشته باشیم** و بعد از آن **page fault بخوریم یا نخوریم**(که مستقل از tlb hit است).

حال فرض می‌کنیم که 100ns نیاز است تا به مموری دسترسی پیدا کنیم و 100ns نیاز است تا داده را از آن برگردانیم، پس tlb زمان دسترسی را 0 می‌کند اما زمان بازگردادن داده همان 100ns می‌ماند.

EAT = 85/100 * 98/100 * (10ns + 100ns) + 85/100 * 2/100 * (10ns + 100ns + 2000ns) + 15/100 * 98/100 * (10ns + 100ns + 100ns) + 15/100 * 2/100 * (10ns + 100ns + 100ns + 2000ns) = **165ns**

ب)

در یکی حافظه چند سطحی، با افزایش تعداد سطح، زمان دسترسی به حافظه افزایش می‌یابد(ادامه سوال صفحه بعد)

یعنی

در حافظه 2 سطحی زمان دسترسی ما 200ns می‌شود و بازگردادن همان 100ns می‌ماند.

EAT = 85/100 * 98/100 * (10ns + 100ns) + 85/100 * 2/100 * (10ns + 100ns + 2000ns) + 15/100 * 98/100 * (10ns + 100ns + 100ns + 100ns) + 15/100 * 2/100 * (10ns + 100ns + 100ns + 100ns + 2000ns) = **180ns**

در حافظه 3 سطحی زمان دسترسی ما 300ns می‌شود و بازگردادن همان 100ns می‌ماند.

EAT = 85/100 * 98/100 * (10ns + 100ns) + 85/100 * 2/100 * (10ns + 100ns + 2000ns) + 15/100 * 98/100 * (10ns + 100ns + 100ns + 100ns + 100ns) + 15/100 * 2/100 * (10ns + 100ns + 100ns + 100ns + 100ns + 2000ns) = **195ns**

سوال 3.

الف)

در این سیستم هر page table به بعدی متصل است و عملا آدرس خود page tableها درون page table دیگر نگه داشته می‌شود.
پس یعنی در اولین سطح page table آدرس page table سطح دوم نگه داشته می‌شود که یعنی در سطح اول 210 آدرس برای page tableهای سطح دوم نگه داشته می‌شود.

210 Page table address for layer 2 ⇒ each layer 2 page table contains address of 210 page tables ⇒ 210 * 210 = number of total entries in page table layer 2

ب)

این allocator برای تخصیص حافظه به مموری کرنل استفاده می‌شود.
در این نوع تخصیص‌دهنده، هر Slab متشکل از چندین page است که به صورت **فیزیکی پیوسته** هستند.
کش سیستم حاوی **یک یا چند Slab است** و کش‌های زیر وجود دارند :

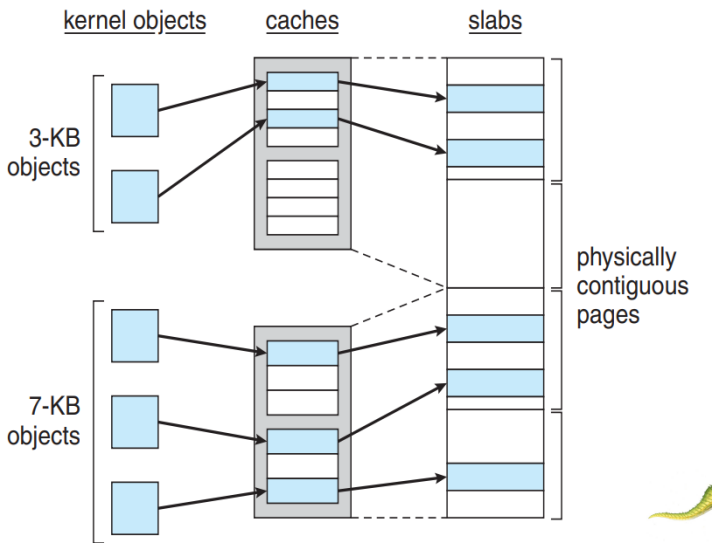
1. برای تمام داده‌ساختارها در kernel، یک کش وجود دارد.
2. برای تمامی file object ها یک کش
3. برای تمامی سمافورها یک کش

هر کش در زمان ایجاد با تعداد object ساخته می‌شود که روی هر کدام از آنها **لیبل free** را قرار می‌دهیم و بعد از استفاده از هر داده‌ساختار، لیبل object ها را به **used** تغییر می‌دهیم.
حال اگر slab با ابجکت‌های استفاده‌شده پر شد :

1. در صورت وجود یک slab خالی ⇐ از slab خالی به ابجکت تخصیص می‌یابد

2. در صورتی که هیچ slab خالی نبود ⇐ یکی slab جدید تخصیص می‌یابد

پس یعنی به صورت کلی در این تخصیص‌دهنده، یک slab pool از slab هایی وجود



دارد که از قبل تخصیص داده شده‌اند.

مزیت‌ها :

1. بخاطر این که slabها از قبل تخصیص داده شده‌اند، دیگر **سربار تخصیص و آزاد کردن حافظه را نداریم** و پاسخ به درخواست تخصیص حافظه **بسیار سریع** می‌شود.
2. همچنین چون تمامی حافظه به صورت تعدادی slab با حافظه مشخص در آمده، پس عملا از **memory fragmentation جلوگیری** کرده‌ایم.

ج)

احتمالا به این دلیل باشد که در سیستم‌های بی‌درنگ، **consistency** بشدت مهم است به این دلیل که یکی از 5 اصلی مهم در سیستم‌های بی‌درنگ این است که مطئمن شویم در طراحی ما Determinism وجود دارد بدین معنی که بتوانیم با قطعیت بگوییم کدام فرایند سیستمی در زمان معین انجام می‌شود که برای این کار نیاز به سیستمی با پایداری

یا همان consistency بالا داریم و می‌دانیم Local Replacement برای ما **پایداری بالایی** به ارمغان می‌آورد. همچینن اگر از global استفاده کنیم یک پردازه با اولویت پایین‌تر می‌تواند پیج پردازه با اولویت بالاتر را بگیر و به مشکل page fault بخوریم و ددلاین پردازه miss شود.

و نیز استفاده از global replacement می‌تواند **مشکلاتی در پیشبینی** بوجود بیاورد چرا که همانطور که از اسلایدها مشخص است زمان اجرای پردازه‌ها می‌تواند بسیار تفاوت داشته باشد.

Real-Time System Characteristics

5 CHARACTERISTICS OF AN RTOS

- **Determinism:** Repeating an input will result in the same output.
- **High performance:** RTOS systems are fast and responsive, often executing actions within a small fraction of the time needed by a general OS.
- **Safety and security:** RTOSes are frequently used in critical systems when failures can have catastrophic consequences, such as robotics or flight controllers. To protect those around them, they must have higher security standards and more reliable safety features.
- **Priority-based scheduling:** Priority scheduling means that actions assigned a high priority are executed first, and those with lower priority come after. This means that an RTOS will always execute the most important task.
- **Small footprint:** Versus their hefty general OS counterparts, RTOSes weigh in at just a fraction of the size. For example, Windows 10, with post-install updates, takes up approximately 20 GB. VxWorks®, on the other hand, is approximately 20,000 times smaller, measured in the low single-digit megabytes.

- **Global Replacement** – process selects a replacement frame from set of all frames; one process can take a frame from another
  - But then process execution time can vary greatly
  - But greater throughput so more common
- **Local Replacement** – each process selects from only its own set of allocated frames
  - More consistent per-process performance
  - But possibly underutilized memory

سوال 4.

سیاست Clock : در این سیاست در صورت استفاده از page، بیت second chance آن را به 1 تغییر می‌دهیم. در صورتی که تمامی خانه‌ها پر باشند، تا جایی که به اولین بیت second chance مساوی 0 برسیم جلو می‌رویم و 1ها را 0 می‌کنیم (اگر به اخر برسیم به اول لیست باز می‌گردیم). می‌دانیم که second chance هنگام ورود نیز 0 است.

حال سناریو آمدن زیر را در نظر بگیرید :

در CLOCK چون بعد از اولین C، دوباره پوینتر بر روی اولین page قرار می‌گیرد پس

Sequence : A,B,C,C,B,A,D

LRU

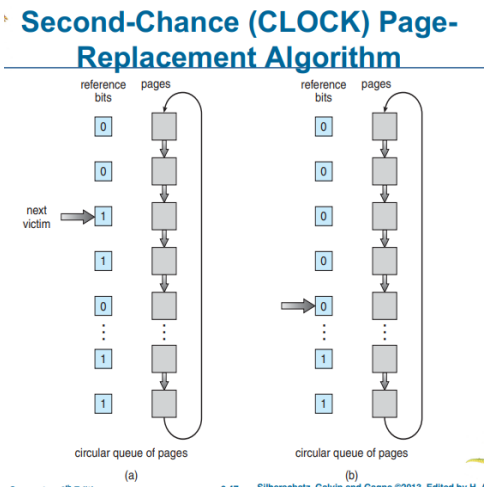
A	A	A	A	A	**A**	A
	B	B	B	**B**	B	B
		C	**C**	C	C	D

CLOCK

A	A	A	A	A	**A**	D
	B	B	B	**B**	B	B
		C	**C**	C	C	C

Second-Chance Algorithm

- Generally FIFO, plus HW-provided reference bit
- **Clock** replacement
- If page to be replaced has
  - Reference bit = 0 ➔ replace it
  - Reference bit = 1 then:
    - Set reference bit 0, leave page in memory
    - Replace next page, subject to same rules



سوال 5.

a)

Virtual Address	:	32 bits
Physical Address	:	30 bits
Virtual Page Count	:	20 bits (Virtual Address - Size of page)
Physical Page Count	:	18 bits (Physical - Size of Page)
Offset	:	12 bits (Size of each page)

b)

Virtual Address	:	32 bits
Physical Address	:	31 bits
Virtual Page Count	:	18 bits (Virtual Address - Size of page)
Physical Page Count	:	17 bits (Physical - Size of Page)
Offset	:	14 bits (Size of each page)

c)

Virtual Address	:	64 bits
Physical Address	:	34 bits
Virtual Page Count	:	50 bits (Virtual Address - Size of page)
Physical Page Count	:	20 bits (Physical - Size of Page)
Offset	:	14 bits (Size of each page)

سوال 6.

MIN (OPT)

	P1	P2	P3	P4	P5	P3	P4	P1	P6	P7	P8	P7	P8	P9	P7	P8	P9	P5	P4	P5	P4	P2
E1	P1	P1	P1	P1	P1	P1	P1	**P1**	P6	P6	P8	P8	**P8**	P8	P8	**P8**	P8	P8	P8	P8	P8	P8
E2		P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P2	P9	P9	P9	**P9**	P9	P9	P9	P9	P9
E3			P3	P3	P3	**P3**	P3	P3	P3	P7	P7	**P7**	P7	P7	**P7**	P7	P7	P7	P7	P7	P7	P7
E4				P4	P4	P4	**P4**	P4	P4	P4	P4	P4	P4	P4	P4	P4	P4	P4	**P4**	P4	**P4**	P2
E5					P5	P5	P5	P5	P5	P5	P5	P5	P5	P5	P5	P5	P5	**P5**	P5	**P5**	P5	P5

FIFO

	P1	P2	P3	P4	P5	P3	P4	P1	P6	P7	P8	P7	P8	P9	P7	P8	P9	P5	P4	P5	P4	P2
E1	P1	P1	P1	P1	P1	P1	P1	P1	P6	P6	P6	P6	P6	P6	P6	P6	P6	P6	P6	P5	P5	P5
E2		P2	P2	P2	P2	P2	P2	P2	P2	P7	P7	P7	P7	P7	P7	P7	P7	P7	P7	P7	P7	P2
E3			P3	P3	P3	P3	P3	P3	P3	P3	P8	P8	P8	P8	P8	P8	P8	P8	P8	P8	P8	P8
E4				P4	P4	P4	P4	P4	P4	P4	P4	P4	P4	P9	P9	P9	P9	P9	P9	P9	P9	P9
E5					P5	P5	P5	P5	P5	P5	P5	P5	P5	P5	P5	P5	P5	P5	P4	P4	P4	P4

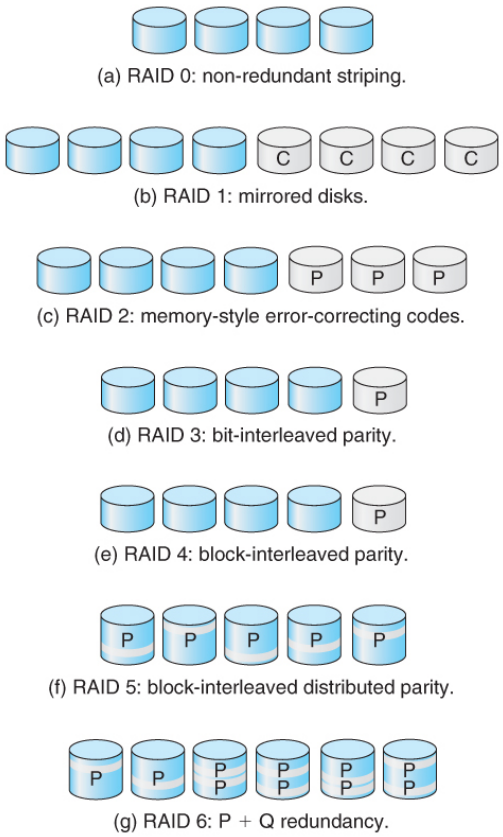
LRU

	P1	P2	P3	P4	P5	P3	P4	P1	P6	P7	P8	P7	P8	P9	P7	P8	P9	P5	P4	P5	P4	P2
E1	P1	P1	P1	P1	P1	P1	P1	**P1**	P1	P1	P1	P1	P1	P1	P1	P1	P1	P5	P5	**P5**	P5	P5
E2		P2	P2	P2	P2	P2	P2	P2	P6	P6	P6	P6	P6	P6	P6	P6	P6	P6	P4	P4	**P4**	P4
E3			P3	P3	P3	**P3**	P3	P3	P3	P3	P8	P8	**P8**	P8	P8	**P8**	P8	P8	P8	P8	P8	P8
E4				P4	P4	P4	**P4**	P4	P4	P4	P4	P4	P4	P9	P9	P9	**P9**	P9	P9	P9	P9	P9
E5					P5	P5	P5	P5	P5	P7	P7	**P7**	P7	P7	**P7**	P7	P7	P7	P7	P7	P7	P2

سوال 7.

به این دلیل که طبق **اصل locality**، درخواست‌های کاربر به یکدیگر از نظر مکانی نزدیک است پس سر نیاز به جابه‌جایی خیلی زیاد ندارد.
همچنین استفاده از این الگورتیم سربار خیلی کمی به نسبت باقی الگوریتم‌ها دارد و از نظر پیاده‌سازی نیز بسیار ساده‌تر است.
الگوریتم مناسب برای محیط تک کاربره الگوریتمی است که سربار کم و پیاده‌سازی ساده داشته باشد حال پیاده‌سازی SSTF می‌تواند گزینه خوبی باشد چرا که از نظر محاسباتی بخاطر تک کاربره بودن، نیاز به محاسبات کمتری دارد و نیز درخواست‌ها تقریبا طبق **locality** نزدیک یکدیگر هستند پس پیاده‌سازی آن نیز پیچیدگی زیادی ندارد.

سوال 8.



بله، مثلا سناریوی زیر را درنظر بگیرید :

فرض کنید چندین درخواست خواندن یک محتوا از قسمت‌هایی مختلفی از **یک دیسک** دریافت کنیم، حال در این صورت چون داده ما به صورت توزیع شده قرار دارد، پس سریع‌تر می‌توانیم به داده مورد نظر برسیم و آنرا بخوانیم.

همچنین در صورتی که خواندن ما همزمان باشد نیز می‌توانیم به صورت موازی از یک دیسک و بکاپ آن بخوانیم که باز سرعت ما را بسیار بالا می‌برد.

یا سناروی زیر که چندین درخواست خواندن از دیسک 1 امده که می‌توان آنها را به صورت توزیع شده به دیسک‌های mirror دیسک 1 فرستاد و عملا به صورت موازی از دیسک 1 خواند.

سوال 9.

از صورت سوال متوجه می‌شویم که جهت حرکت در ابتدا از چپ به راست بوده است.

FCFS : 322 → 1300 → 1750 → 102 → 2509 → 248 → 1334 → 1740 → 68

Total = (1300 - 322) + (1750 - 1300) + (1750 - 102) + (2509 - 102) + (2509 - 248) + (1334 - 248) + (1740 - 1334) + (1740 - 68) = **10898**

SSTF : 332 → 248 → 102 → 68 → 1300 → 1334 → 1740 → 1750 → 2509

Total = |332 - 248| + |248 - 102| + |102 - 68| + |68 - 1300| + |1300 - 1334| + |1334 -1740| + |1740 - 1750| + |1750 - 2509| = **2705**

SCAN : 332 → 1300 → 1334 → 1740 → 1750 → 2509 → 3999 → 248 → 102 → 68

Total = 3999 - 332 + 3999 - 68 = **7598**

C-SCAN : 332 → 1300 → 1334 → 1740 → 1750 → 2509 → 3999 → 0 → 68 → 102 → 248

Total = 3999 - 332 + 3999 + = **3999 - 332 + 3999 + 248= 7914**

سوال 10.

الف)

ازاده از اشاره‌گرهای مستقیم یعنی تعداد سرچ ما برای یافتن فایل مورد نظر کمتر می‌شود و زمان یافتن فایل کاهش می‌یابد اما از طرفی ارایه ما بسیار بزرگ شده و خود از نظر حافظه‌ای مقدار زیادی حافظه اشغال می‌کند و نیز حداکثر حجم هر فایل کاهش می‌یابد.

ب)

هر inode به یک بلوک n-بایتی اشاره می‌کند که خود ارایه اصلی c تا inode می‌تواند داشته باشد. پس یعنی عملا **در بیشترین حالت** یک جدول که به c جدول اشاره می‌کند که هرکدام به c جدول اشاره می‌کنند که در خود nc بلوک دارند.

$c * c * c * n = c^3 * n$ Byte = **MAX**

در کمترین حالت نیز یک جدول یک سطحی داریم که صرفا n * c ورودی دارد.

n * c = c * n Byte = **MIN**

پ) 3 برابر

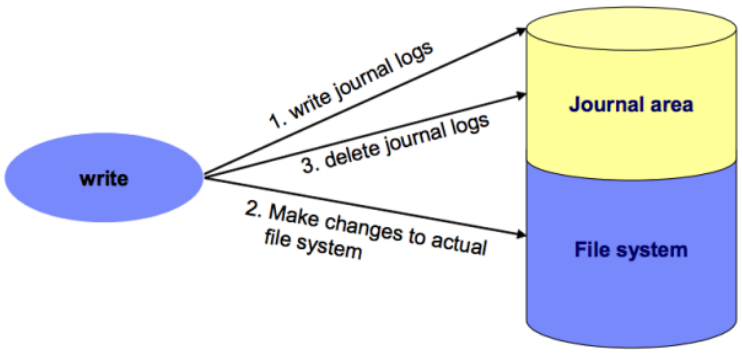
در بیشترین حالت عملا نیاز به **3 دسترسی** داریم و در کمترین حالت نیاز به **1 دسترسی** داریم، پس سرعت در حالت کمترین حجم، **3 برابر** سرعت در حالت بیشترین حجم است.

سوال 11.

الف)

این فایل سیستم، از یک journal برای نگه‌داری تغییرات اعمال شده بر file system نگه‌داری می‌کند که **log تمامی تراکنش‌های فایل سیستم** را دارد و این امکان را به فایل سیستم می‌دهد که پس از **بروز مشکل** (مثلا crash کردن یا خاموش شدن ناگهانی) به صورت **سریع و مطمئن** ریکاوری داشته باشد.

در این journal همچنین **تمامی تغییرات** بر فایل سیستم از جمله ساختن، پاک کردن و تغییرات (modification) فایل‌ها و directoryها **قبل از قرار گرفتن** بر روی فایل سیستم را ثبت می‌کند که مطمئن می‌شود فایل سیستم از **پایداری(consistancy) و تمامیت (integrity) خوبی** بهره‌مند است و در صورت بروز مشکل می‌تواند به **حالت پایدار قبلی** بازگردد(revert to previous commit).



ب)

حالت درونی : زمانی رخ می‌دهد که از نظر سیستم‌عامل فضای خالی وجود ندارد ولی در واقع اندازه یک فایل، کوچکتر از اندازه حافظه تخصیص داده شده به آن (مثلا بلوک) باشد که باعث می‌شود **درون بلوک تخصیص داده‌شده، حفره داشته باشیم**.

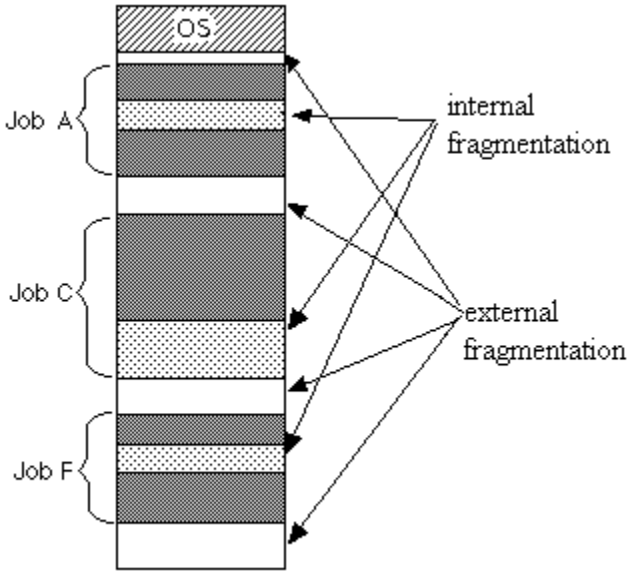
حالت بیرونی : زمانی رخ می‌دهد که به اندازه کافی به صورت سرجمع **فضای خالی داریم** اما این فضا یک **فضای پیوسته نیست** تا به یک فایل بزرگ تخصیص دهیم.

پس به صورت کلی حالت درونی **در یک allocation unit** اتفاق می‌افتد ولی حالت بیرون **بین چندین allocation unit** اتفاق می‌افتد.

به عنوان مثال، فرض کنید یک دیسک با سایز بلوک حافظه 5KB داشته باشیم و بخواهیم سه فایل A = 2KB و B = 4KB و C = 7KB را ذخیره کنیم.

حال درصورتی که به هر فایل یک بلوک اختصاص دهیم، internal fragmentation خواهیم داشت چرا که برای فایل‌های A و B، به ترتیب 3KB و 1KB درون بلوک حافظه استفاده نشده داریم.

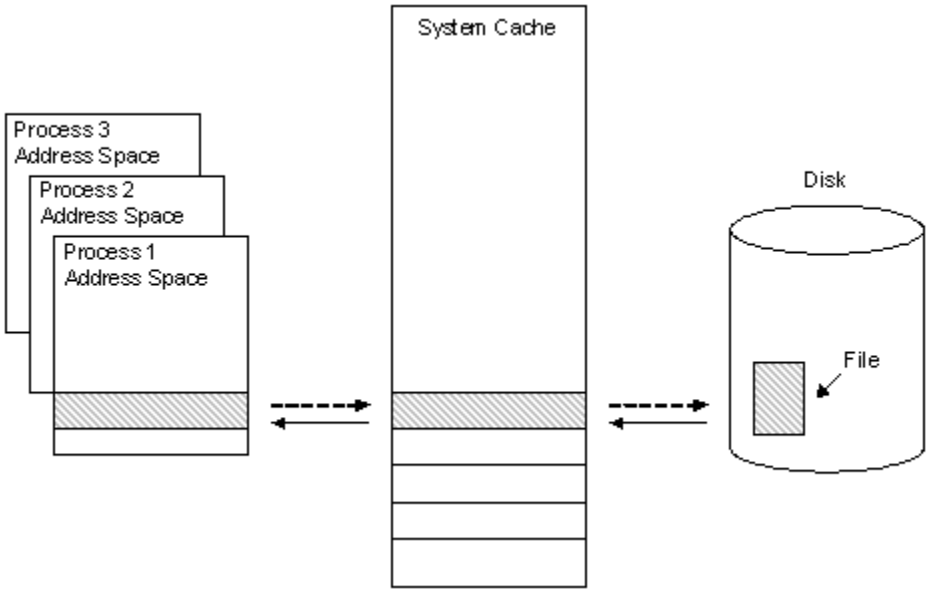
در صورتی که فرض کنیم کل حافظه ما متشکل از 4 بلوک 5KB است و به A و B دو بلوک اول و سوم را تخصیص داده‌ایم یعنی عملا الان 10KB فضای ازاد غیر پیوسته داریم که نمی‌توانیم به C تخصیص بدهیم پس به اندازه کافی بلوک داریم اما برای تخصیص پیوسته نیست پس دچار external fragmentation نیز شده‌ایم.



پ)

فایل کش تکنیکی است که در آن، سیستم‌عامل از حافظه استفاده می‌کند تا فایل‌ها و داده‌ای که **بیشترین استفاده** را از هارد دیسک **ذخیره کند** و عملا از تعداد عملیات **خواندن و بازگرداندن داده(data retrival) بکاهد**.

این مدل از کش از **زمان پاسخگویی (response time) بشدت می‌کاهد** و **عملکرد کلی سیستم را بالا** می‌برد.



سوال 12.

الف) خیر !!

فایل سیستم FAT : سیستمی که در آن از یک جدول برای نگه‌داری cluster allocation استفاده می‌شود و به دلیل **سادگی طراحی** آن در دستگاه‌های قابل حمل و نقل و کوچک استفاده می‌شود اما محدودیت‌هایی در **سایز** دارد.

به دلیل نوعی که این فایل سیستم کار می‌کند **نمی‌تواند** دچار external fragmentation شود چرا که با استفاده از تعدادی **پوینتر** فقط کلاسترهایی را به ما می‌دهد که برنامه نیاز دارد. یعنی عملا قسمت‌های خالی را **با پوینتر به هم لینک می‌کند** و هنگامی که نیاز به کلاستر جدید باشد، از قسمت قبلی به آن اشاره می‌کند.

درصورتی که تعداد زیادی **فایل کوچک پاک شوند**، در جدول FAT، به صورت **Free** در می‌آیند و پوینتر می‌تواند به آنها اشاره کند پس **external fragmentation نداریم**.

ب)

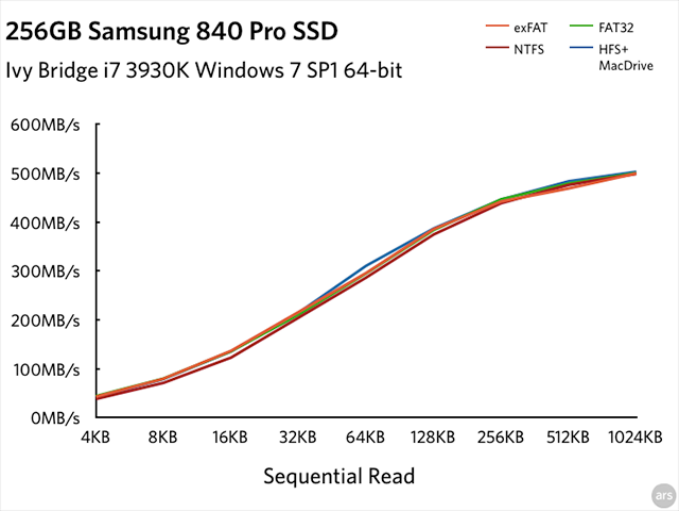
خود جدول FAT بر **روی Disk** و عملا **sector 0** که همان **اولین سکتور** است ذخیره می‌شود چرا که در صورتی که دیسک حتی به کامپیوتر **متصل نباشد**، فایل سیستم می‌تواند **تخصیص‌ها را دنبال کند** و عملا از **تخصیص‌های دیسک و اطلاعات مربوط به تخصیص‌ها** مطلع است همچنین یک دلیل دیگر می‌تواند این باشد که باید مکان این جدول مشخص باشد تا هنگام بوت بتواند عملیات تقسیم‌بندی را به درستی انجام دهد.

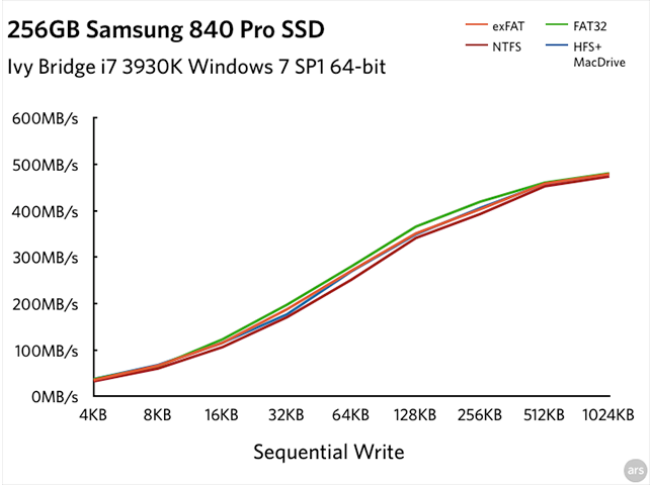
به همین دلیل FAT یک فایل سیستم بسیار **انعطاف‌پذیر و قابل حمل و نقل** (portable) است و برای ذخیر داده بر فلاپی دیسک‌ها و یا usb ها استفاده می‌شود.

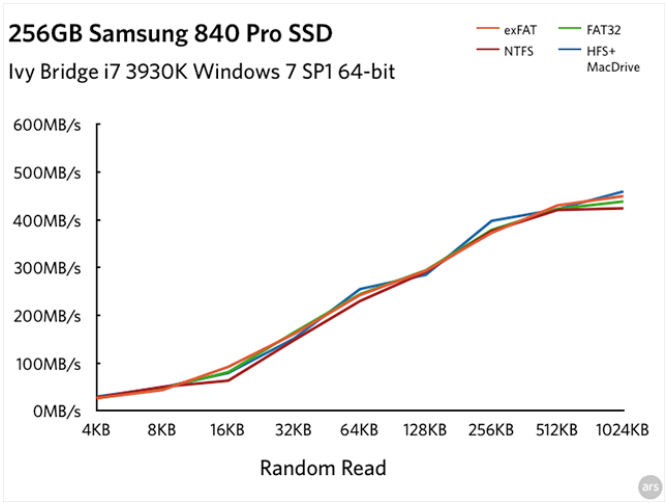
پ)

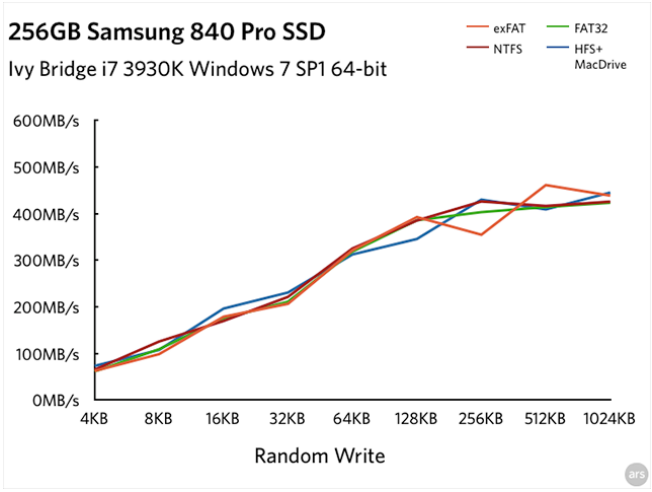
برای این مقایسه تعدادی بنچ‌مارک گرفته شده توسط دو سایت مختلف را در ادامه می‌آوریم.

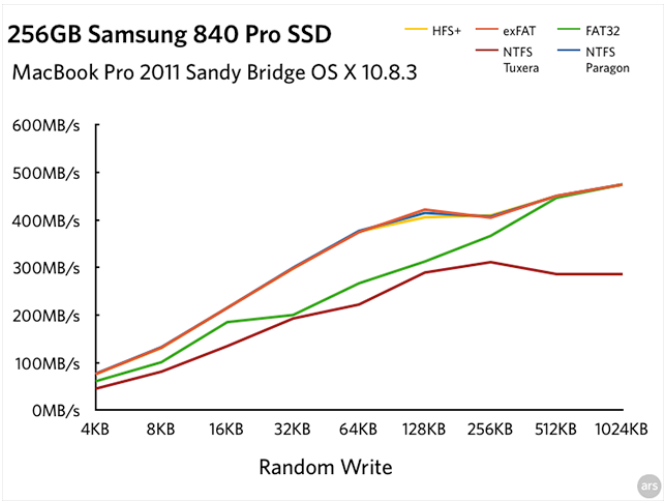
[منبع]

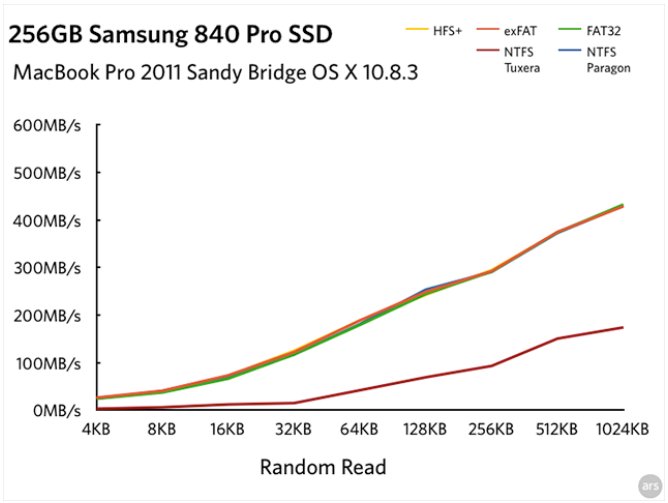




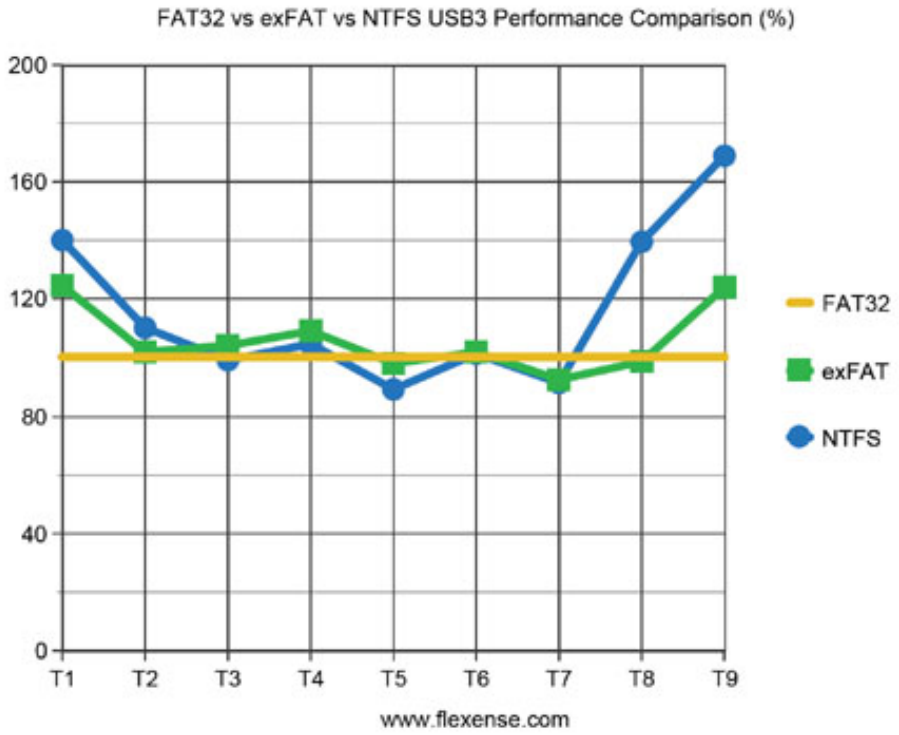








منبع



حال در **منبع اول** به صورت میانگین، FAT عملکرد بهتری دارد اما در **منبع دوم** به صورت میانگین NTFS عملکرد بهتری دارد که نشان می‌دهد در دستگاه‌های مختلف عملکرد متفاوتی می‌تواند داشته باشد اما تحلیل من این است که بخاطر سادگی پیاده‌سازی FAT و این که برای گرفتن یک فایل باید کل link list مربوط به cluster را طی بکند، پس احتمالا در نهایت **سرعتش از پیاده‌سازی‌های پیچیده‌تر مانند NTFS کمتر خواهد بود و زمان متوسط دسترسی طولانی‌تری دارد.**

سوال 13.

از حرکت می‌فهمیم سر سیلندر به سمت چپ حرکت می‌کند. فرض می‌کنیم 199 اخرین سیلندر هست (چون ماکسیمم تعداد سیلندر داده نشده!!!) هر چند که این در ترتیب بررسی به درخواست‌هایی که اورده شده صدمه‌ای وارد نمی‌کند(صرفا باید تا اخرین سیلندر برود و دوباره بازگردد) .

C-Scan : 53 → 37 → 14 →(0)→ 183 → 124 → 122 → 98 → 67 → 65

پس به ترتیب بررسی درخواست‌ها از چپ به راست :

53, 37 , 14, 183 , 124, 122 , 98 , 67 ,65

سوال 14.

ابتدا کل حرکت را بدست می‌آوریم :

SSF : 20 → 18 → 25 → 35 → 39 → 8 → 5 → 3

Total seen cylinders : 39 - 18 + 39 - 3 + 2 = 59

Total time = 59 * 5 ms = 295 ms