دانشگاه آزاد اسلامی واحد بافت

جزوه درس سیستم های توزیع شده

Distributed systems

تهیه کننده :

کاظم فریدی

دانشجوی کارشناسی ارشد مهندسی معماری سیستمهای کامپیوتری

با سلام

جزوه تهیه شده از ترمهای قبل تدریس شده توسط استاد گرامی جناب آقای دکتر کروپی به دانشجویان کارشناسی ارشد نرم افزار دانشگاه آزاد اسلامی واحد بافت می باشد که توسط جناب آقای اسدی و سرکار خانم مرادیان در اختیار بنده قرار داده شده است تا بتوانم آن را برروی سایت گذاشته و در دسترس کلیه دانشجویان قرار دهم .پیشاپیش از زحمات این عزیزان تقدیرو تشکر به عمل می آید .

فصل اول Distributed System

فصل دوم Communication in Distributed System

فصل سوم Synchronization in Distributed System

فصل چهارم Prosesses and Processors in Distributed System

فصل ششم Distributed Shared Memory

cpu های قدرتمند و شبکه‌های LAN ، باعث شد بتوانند که بر جمهوری سیستم‌ها را کنار هم قرار دهند و به سیستم‌های توزیع شده تبدیل کنند.

cpu های قدرتمند / شبکه‌های LAN ⟵ سیستم‌های توزیع شده

Distributed system :

مجموعه‌ای از کامپیوترهای مستقل هستند که از دید کاربر به نظر یک کامپیوتر می‌آیند. دو فاکتور در عنوان نهفته است:

۱. سخت افزار: از نظر سخت افزاری کاربر نباید بفهمد که cpu های متعدد داریم. و پایانه بینهایت [illegible] برنامه‌های کاربردی [illegible] cpu های دیگر کاملاً پنهان می‌شود

۲. نرم افزار: از نظر نرم افزاری کاربر نباید بفهمد که نرم افزار کجا دارد اجرا می‌شود.

Centralized system :

سیستم‌های متمرکز مبنی بر تک cpu هستند مثل mainframe که یک سیستم متمرکز است.

اهداف سیستم‌های توزیع شده در مقایسه با سیستم‌های متمرکز:

1. اقتصادی Economics:

مزیت: سیستم‌های توزیع شده نسبت به متمرکز اقتصادی‌تر هستند یعنی با خرید چندین CPU می‌توانند آن را بهترش دهند و عملکرد بهتری در مقابل پولی که می‌دهید می‌گیرید.

2. سرعت Speed:

اگر یک سیستم با شرایط روبرو داشته باشیم:

10,000 CPU => 50 MIPS سیستم توزیع شده

=> 500,000 MIPS

یعنی برای اجرای یک دستور 0.02 ns زمان لازم است

$10 \div 500 = 0.02$ ns

که روی یک سیستم امکان پذیر نیست.

3. توزیع اصلی یا اساسی Inherent Distribution:

نرم افزارهایی هستند که ذاتاً توسعه پذیرند.

مثل: رزرو بلیت هواپیما، در کل ایران همه باید بفهمند که یک صندلی اشغال شده یا نه؟

مثل: بانک‌ها، در کل ایران باید بفهمند مانده حساب شما چقدر است.

4. اعتمادپذیری : Reliability

سیستم های توزیع شده نسبت به سیستم های متمرکز ذاتاً اعتمادپذیری بیشتری دارند. اگر یک CPU خراب شود بقیه CPU ها به کارشان ادامه می دهند.

5. رشد تدریجی : Inceremental Growth

رشد تدریجی در سیستم های توزیع شده نسبت به سیستم های متمرکز راحت تر است. چون می توان با اضافه کردن چند CPU به آن توانش را بیشتر کرد.

ولی در سیستم های متمرکز اگر نیاز به افزایش داشته باشیم باید یک Mainframe دیگر بخریم.

- مزایای سیستم های توزیع شده روی ~~Lape~~ ها : (در مقایسه با ماشین های شخصی)

1. Data sharing : روی سیستم های توزیع شده می توانیم داده ها را به اشتراک بگذاریم. (به اشتراک گذاشتن داده)

2. ~~Hardware~~ sharing : در سیستم های توزیع شده می توانیم سخت افزار را به اشتراک بگذاریم. (device)

3. Communication : برقراری ارتباط با یکدیگر، در سیستم های توزیع شده می توانیم ایمیل (message) یا فکس و هر چیز دیگر را بدون کاغذ به دیگری بفرستیم.

4. Flexibility : انعطاف پذیری، در سیستم های توزیع شده می توان یک کار (یا برنامه) را روی چندین CPU به صورت موازی انجام داد تا کار سریع تر پیش رود و انجام گیرد.

- معایب سیستم های توزیع شده :

1. نرم افزار : تعداد نرم افزارهایی که به وسیله سیستم های توزیع شده کار می کنند بسیار کم هستند.
2. شبکه : سیستم های توزیع شده در محیط شبکه کار می کنند. با افزایش کاربران، سیستم دیگر نمی تواند جواب خوبی به کاربران بدهد.
3. امنیت : اگر امنیت را درست تعریف نکنیم از بیرون می توانند وارد شوند و هرکاری که دلشان خواست انجام دهند.

- تقسیم بندی فلین :

آقای Flynn ساختارهای کامپیوتر را به دو گروه تقسیم کرد:

نحوه جریان دستورات :

1. Instraction stream

Flynn

نحوه جریان داده :

2. Data stream

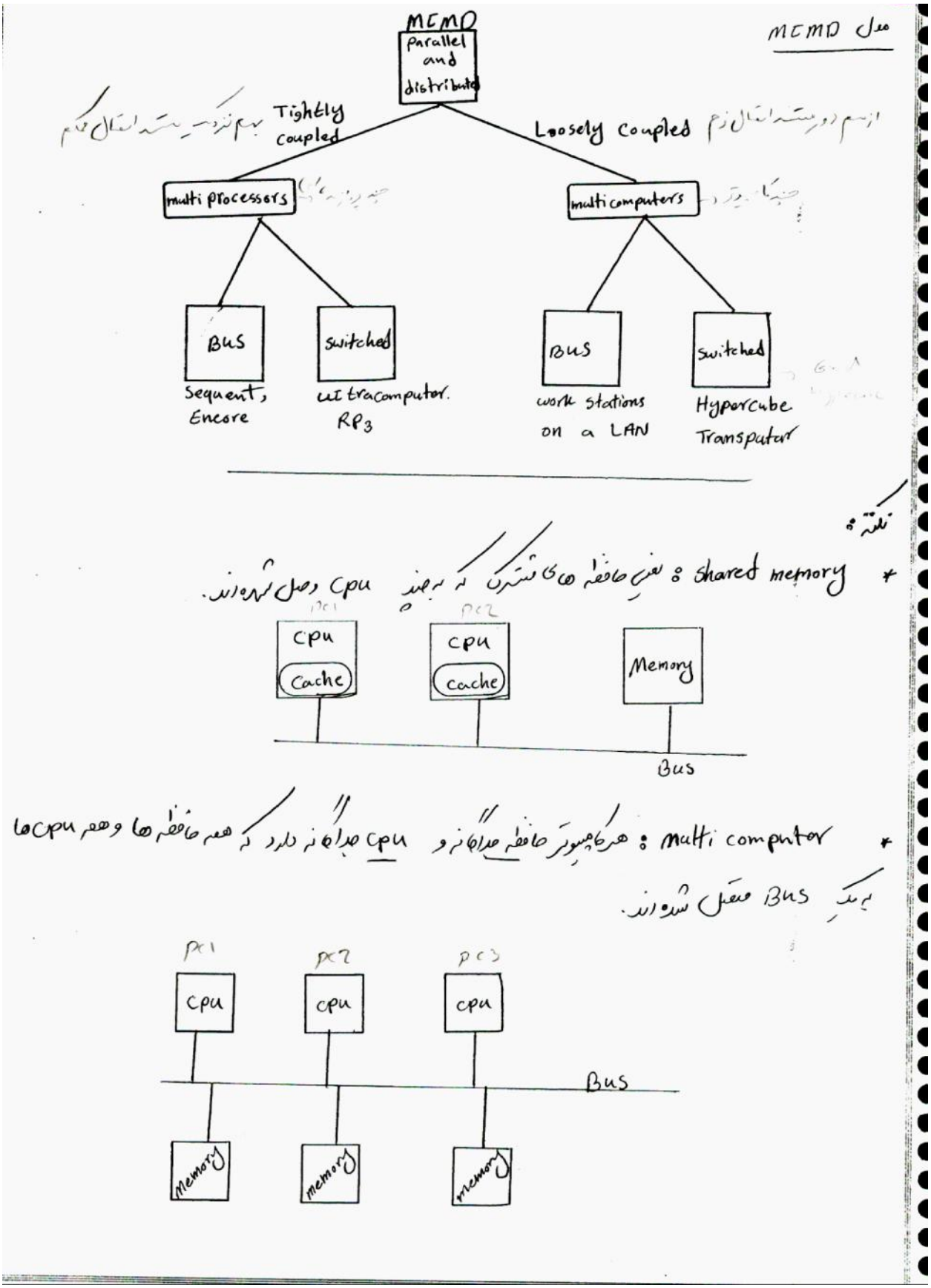
1. SISD (Single Instraction single Data) : مثل کامپیوترهای شخصی

2. SIMD : Single Instraction mullti Data : ضرب دو ماتریس با شطرنج کامپیوتر

3. MISD : دستورهای متعددی که روی یک داده کار می کنند - این چنین کامپیوتری نداریم.

4. MIMD : داده های متعدد روی دستورات متعدد - مثل سیستم های توزیع شده

شکل صفحه ی بعد


MIMD
parallel and distributed
Tightly coupled
Loosely Coupled
multi processors
multicomputers
Bus
Switched
Bus
Switched
Sequent, Encore
ultracomputor. RP3
work stations on a LAN
Hypercube Transputer
MIMD
Shared memory
cpu
cache
cpu
cache
Memory
Bus
multi computor
cpu
cpu
cpu
Bus
Memory
Memory
Memory

Tightly coupled : هم نزدیک هستند – اتصال محکم

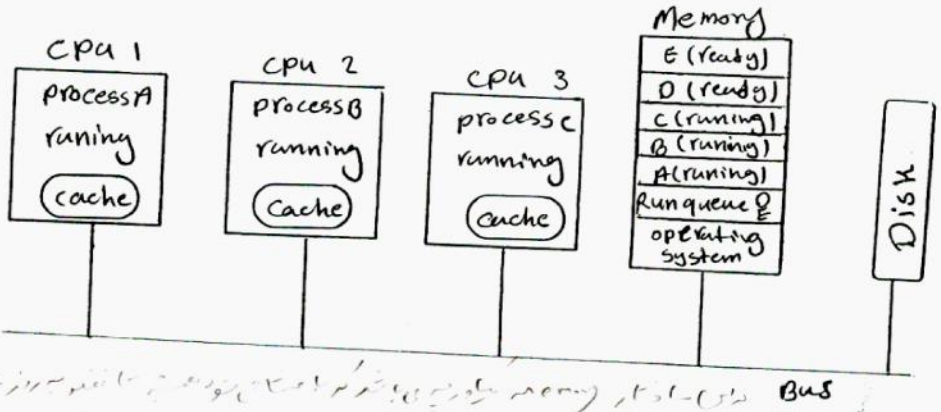
Loosely coupled : از هم دورهستند – اتصال نرم

نرم افزار و سخت افزار

تعداد CPU نزدیک هم ← Data Rate بالا و Delay پایین است.

تعداد CPU دور از هم ← Data Rate پایین و Delay بالا است.

- Bus-Based multiprocessor :



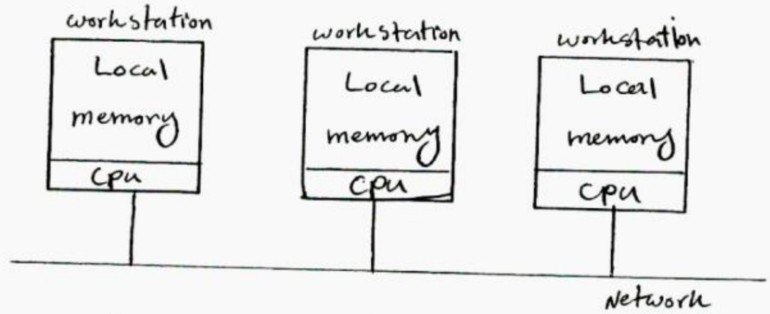
مزیت: اگر یک CPU عدد 16 را از حافظه بخواند و آدرس این خانه از حافظه را عوض کند و مقدار 100 را برآن بدهد. اگر CPU دیگری آدرس عدد 16 را خواست باید از آدرس 100 استفاده کند.

برای 4 تا 5 CPU خوب کار می کند.

عیب:

با بالا رفتن تعداد CPU ها ترافیک Bus زیاد می شود. ترافیک در اثر درخواست های CPU ها روی Bus می گذارند زیاد می شود که باعث می شود سرعت وکارایی پایین بیاید.

Bus - Based multicomputers : باس بیس مالتی کامپیوترز



هر کامپیوتر حافظه و cpu جداگانه دارد. اگر Bus قطع شود تمام cpu ها از کار می افتند.

کش وانواع آن :

- coherent (کهرنت ، هم پیوسته) : با نداشتن کش ترافیک شبکه کند می شود. درخواست اول به کش می رود اگر کش توانست به درخواستها پاسخ دهد دیگر درخواستها به معدری یا حافظه از طریق Bus نمی رود.

- in coherent (غیر کهرنت) : چون داده های cpu تغییر می کند حافظه کش کارایی خود را از دست می دهد.

راه حل

1. write through cache : از کش هایی استفاده کنیم که هر تغییری در cpu را روی معدری می نویسند.

2. snoopy cache کش های فضول : هر تغییری که در Bus ایجاد می شود آن را روی خودشان اعمال می کنند اگر به آن مربوط باشد. به Bus گوش می دهد.

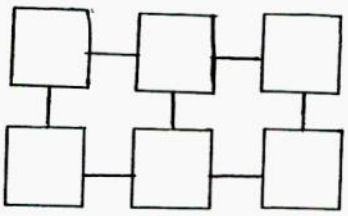
Cache hit : اطلاعات در کش باشند.

Cache miss : اطلاعات در کش نباشند. ⟶ Cache miss = 1 - Cache hit

نکته: Cache hit به 90% می رسد و می توانیم 32 تا 64 عدد cpu را روی یک Bus قرار بدهیم. با داشتن یک کش بین 64 تا 1MB سرعت بالا می رود. در این صورت Cache hit = %90

- switch multi computers : هر کامپیوتر حافظه و cpu جداگانه دارد.

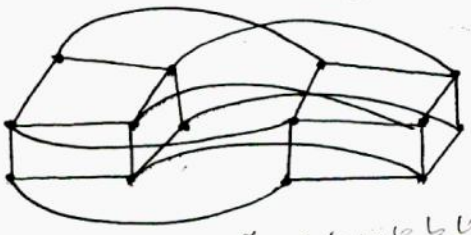
1. Grid : هر کامپیوتر به وسیله کابل مخصوص به فرد به یک کامپیوتر دیگر وصل شده است.



2. Hypercube : سه بعدی است و هر کامپیوتر به 3 کامپیوتر دیگر وصل است.

هر قطعه یک کامپیوتر است. برای 4 بعدی کامپیوترها به صورت متناظر به هم متصل می‌شوند.

دو تا 4 بعدی یا چهار تا 3 بعدی که کنار هم یک 5 بعدی را تشکیل می‌دهند.



- switch/multi processor :
  1. cps
  2. omega-switch

1. (CPS) cross point switch :

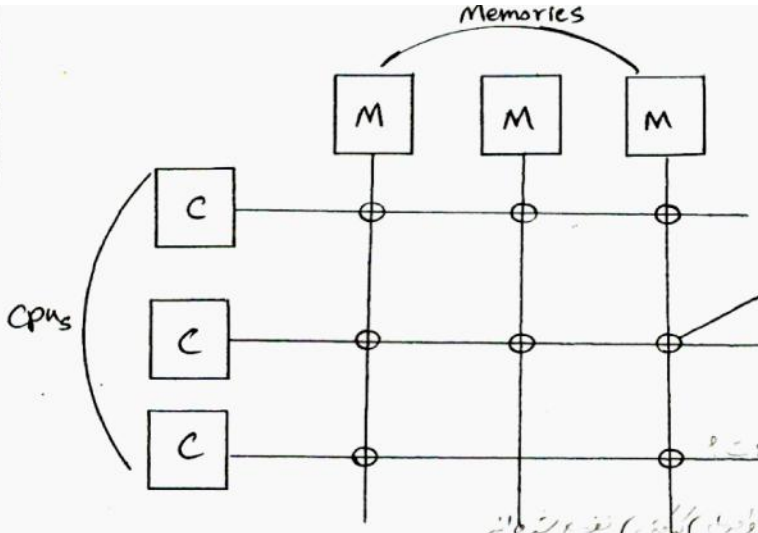
هر CPS یک اتصال دارد که وصل و قطع می‌شود. هر cpu می‌تواند به هر صورتی ارتباط برقرار کند.

n cpu = n^2 switch ⟸ cpu عدد n به تعداد n^2 عدد سوئیچ نیاز داریم

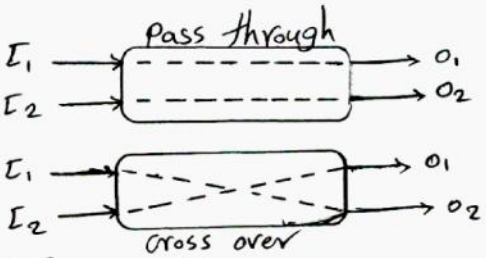
معایب: ① ② هزینه زیادی را روی هم اعمال می‌کند.

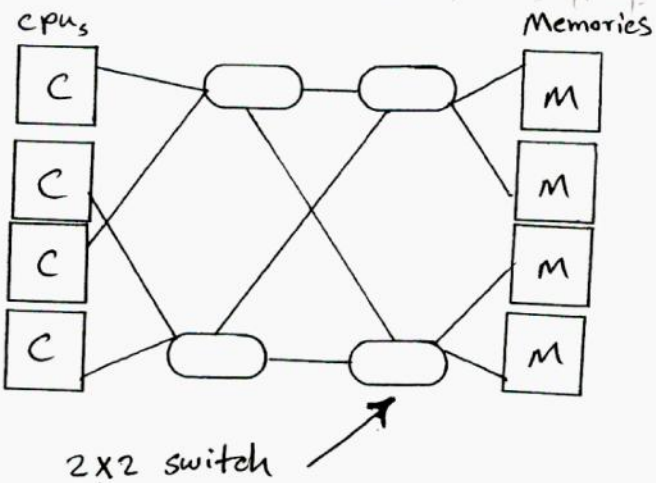
③ یک حافظه است که بلاک بندی شده است.

شکل : Crosspoint



2. Omega switch :





تعداد مراحل : $\log_2^n$ تعداد سوئیچ‌های هر مرحله : $n/2$ تعداد کل سوئیچ‌ها : $n/2 \log_2^n$

Example: مراحل omega برای 4 عدد cpu :

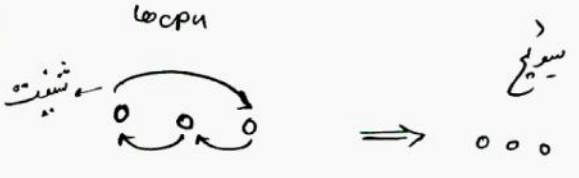
1. تعداد مراحل ؟ $= \log_2^n = \log_2^4 = 2$ تعداد مرحله در امگا

2. تعداد سوئیچ ها ؟ $= n/2 = 4/2 = 2$ تعداد سوئیچ ها در امگا

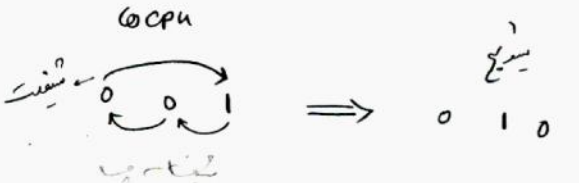
3. تعداد کل سوئیچ ها ؟ $= n/2 \log_2^n = 2 \times 2 = 4$ تعداد کل سوئیچ ها در امگا

سوئیچ کردن :

در هر سوئیچ، بیت به بیت مقایسه میشود :



1. اگر بیت ها برابر بودند Pass Through



2. اگر بیت ها متفاوت بودند cross over

Example :

اگر $n = 1024$ باشد تعداد مرحله ها $\log_2^{1024} = 10$ میشود یعنی از مبدأ تا مقصد باید از 10 سوئیچ عبور کنیم. اگر سرعت کامپیوتر 100 MIPs باشد هر دستور در 10ns اجرا میشود.

باید 20 عدد سوئیچ را به 10ns طی کند. هر سوئیچ در 5 ns باید دستور را در خود عبور دهد.

تعداد سوئیچ هر مرحله $= \frac{1024}{2} = 512$

تعداد کل سوئیچ ها $= 512 \times 10 = 5120$

$20 \times 5 = 100$ ← مدت دستور بعد به حافظه و برگردد

← تعداد سوئیچ ها برای رفت و برگشت $10 + 10 = 20$

- مفاهیم نرم افزاری Software concepts :

سیستم عامل های شبکه یا Network oprating sys در گروه Lossly coup software یعنی
نرم افزار با اتصال نرم هستند که با Lossly coup hardware پیاده سازی می شوند . سیستم عامل شبکه
در شبکه های LAN چون کامپیوترها از هم جدا هستند به صورت Lossly coup hardware هستند . و چون
نرم افزارهایی که روی آنها اجرا می شوند ممکن است متفاوت باشند Lossly coupled software اند .

Remote login machine از راه دور که Lossly coupled hardware است .
که در آنها نرم افزارها از هم مستقل و کامپیوترها با فاصله از هم قرار گرفته اند . به هر دو نوع هستند .

{ Lossly coupled software
Lossly coupled hardware

multiprocessor time is sharing
tightly coupled software
tightly coupled hardware

نکته :
یک client تمام دایرکتوری ها را روی سرور خودش آورده و تمام درایورهای مجازی شبکه را mount (فعال) می کند .

- True Distributed system : سیستم های توزیع شده واقعی

Lossly Tightly coupled (Hardware - software)
باز و گسترده — نزدیک و محکم

هدف این سیستم این است که وانمود کند که این سیستم تک pc یا کامپیوتر و سیستم است و کاربر چه از بعد نرم افزاری
و چه از بعد سخت افزاری نباید بفهمد که با سیستم چند پردازنده ای کار می کند .
که این هدف سیستم های توزیع شده است . یعنی سیستم یکپارچه است .

- سیستم‌های multi processor

هم نرم‌افزار و هم سخت‌افزار متمرکز (نزدیک و محکم) Tightly coupled است.

چون حافظه مشترک داریم پس سیستم نرم‌افزاری باید یکپارچه باشد (باتوجه به شکل صفحه قبل جزوه)

اگر به دیسک I/O wait .1 یا Time run out .2 بخواهد A را انجام دهد پروسه در حال انتظار را اجرا کند.

در ابتدا باید تا آن زمان تمام اطلاعات پروسه A را در جدول پروسه ذخیره کند تا بتواند A را برگرداند

همان شرایط سابقه هم هست یعنی دو cpu به صورت همزمان برای اجرای این پروسه‌ها به رقابت

برخیزند در حافظه ← باید حافظه در حالت انحصاری درآید تا شرایط سابقه پیش نیاید.

اگر یک cpu پروسه D را اجرا کند در ابتدا رغبت پایین است چون تمام اطلاعات پروسه A را از دست

می‌شود. باید چندین Page fault بدهد تا کش از اطلاعات D پر شود.

پروسه به همان cpu ای فرستاده می‌شود که قبلاً روی آن cpu اجرا می‌شد. چون امید داریم داده‌های آن

در کش باقی مانده باشد.

هر پروسه‌ای که زمان اجرای زیادی می‌خواهد بطور میانگین بیشتر cpu ها می‌رود.

اگر پروسه‌ای داشته باشیم که در حال انتظار باشد آن را سوئیچ می‌کنیم مگر اینکه

اجرا می‌شود اگر I/O آن کمتر از زمان سوئیچ است، پس بهتر است سوئیچ نشود.

: Fig1-12

multiprocessor OS	Distributed OS	Network OS شبکه	Item
بله	بله	نه	1. آیا به نظر یک سیستم تک پردازنده به نظر می آید؟
بله	بله	نه	2. آیا همه باید یک سیستم عامل را اجرا کنند؟
1	N	N	3. در این سیستم چند نسخه سیستم عامل نصب شده داریم؟
اشتراک حافظه	پیام	اشتراک فایل	4. ارتباط چگونه برقرار می شود؟
نه	بله	بله	5. آیا همه اینها از قرارداد پروتکل شبکه پیروی می کنند؟
بله	نه	نه	6. آیا صفی از برنامه‌های در حال اجرا در سیستم داریم؟
بله	بله	معمولاً نه	7. آیا برای اشتراک فایلها قوانین خاصی وجود دارد؟

*- فاکتورهای طراحی : Design Issues

1. Transparency شفافیت
2. Flexibility انعطاف پذیری
3. Reliability قابلیت اعتماد پذیری
4. Performance کارایی
5. Scalability توسعه یا گسترش پذیری

*1. شفافیت : یعنی کاری کنیم که کاربر هیچ وقت متوجه نشود دارد با یک سیستم چند پردازنده کار می کند از تعداد CPUها یا ...
یا برنامه‌ای که می نویسیم در کدام ماشین اجرا می شود

(A) Location Transparency : شفافیت مکان

کاربر نباید که کامپیوترها یا منابع که دسترسی داشته باشد و جای آنها باخبر باشد. کاربر از مکان منابع باخبر نباشد

مثلاً: اگر دستور پرینت دهد نباید آدرس چاپگر را بدهد پرینتر روی یک سیستم دیگر چاپ می شود.

(B) Migration Transparency : شفافیت مهاجرت

باید بتوانیم منابع را جابجا کنیم بدون اینکه بخواهیم اسم آنها را عوض کنیم. بدون نیاز به عوض کردن اسم.
یعنی بدون اینکه کاربر متوجه شود منابع را از جایی به جای دیگر انتقال دهیم

(C) Replication Transparency : شفافیت در تکثیر

یعنی سیستم باید بتواند بدون اینکه کاربر بفهمد از فایلها کپی بگیرد (جهت افزایش سرعت)
کاربر نباید بفهمد که از روی یکی از فایلها چند کپی گرفته می شود. برای افزایش اعتماد پذیری از کپیها استفاده
...

D) Concurrency Transparency : شفافیت همروندی

کاربر نباید بفهمد که همزمان کاربران دیگری هم دارند از منابع مشترک استفاده می‌کنند.

یعنی به طور همزمان [illegible]

E) Parallelism Transparency : شفافیت موازی سازی

یعنی کاربر نباید بفهمد کارها دارند موازی با هم پیش می‌روند یعنی روی چند کامپیوتر اجرا می‌شوند.

یعنی کارهای [illegible]

2. انعطاف‌پذیری : Flexibility

انواع سیستم از نظر ساختاری

A) monolithic : کاربر کمترین میزان دسترسی با سیستم را دارد. همه وظایف برعهده سیستم عامل است

[illegible]

کاربر کمترین مسئولیت و سیستم عامل بیشترین مسئولیت را برعهده دارد. (شکل 1)

[illegible]

B) micro kernel :

فقط دستورات خاصی که می‌خواهیم سیستم عامل داشته باشد را روی سیستم می‌نشانیم. سیستم عامل کوچکتر است.

چون اکثر کارها برعهده کاربر است و min کارها برعهده سیستم عامل است. (شکل 2)

[illegible]

[illegible] A و B [illegible]

C) exo kernel :

[illegible]

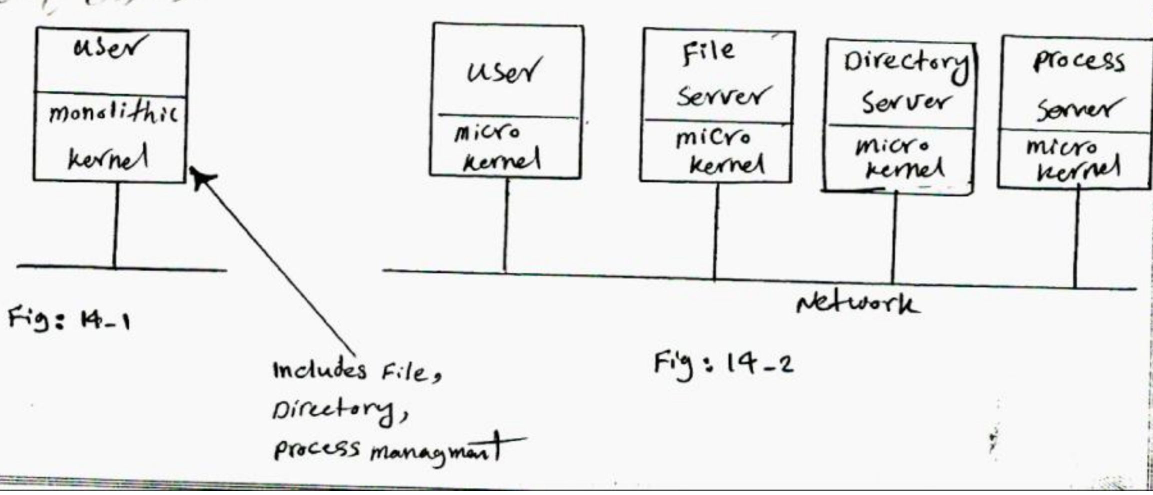


Fig: 14-1

Fig: 14-2

3- قابلیت اعتمادپذیری:

سیستم‌هایی که توزیع شده باشند اعتماد بیشتری به آنها است نسبت به سیستم های دیگر اگر یک سیستم خراب شود سیستم های دیگر به کار خود ادامه می دهند.

اگر سیستم تک cpu باشد با از کار افتادن cpu کل سیستم از کار می افتد.

cpu	احتمال از کار افتادن	احتمال در کار بودن
1	٪5	٪95
4	$٪5^4 = ٪0.0006$	٪99.94

$P = ٪100 - ٪0.0006$

احتمال ضعیف پایین است وقتی تعداد cpu ها زیاد باشد کل cpu ها از کار بیفتند.

Reliability با Avalibility در ارتباط هستند و با هم کار می کنند. یعنی سیستم قابل اعتماد است که در دسترس هم باشد.

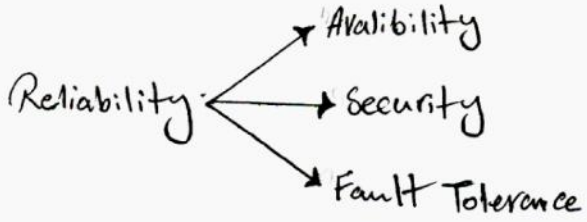
Avalibility زمانی بالاتر می رود که تعداد منابع را ایجاد کند. اگر یکی از کار افتاد آن یکی کار را انجام دهد.

Reliability با Security هم کار می کند. سیستم باید امن باشد نباید بگذاریم هرکس از سیستم استفاده کند.

مثلاً: رمز گذاشتن و ...

Reliability با FT (Fault Tolerance) تحمل پذیری خطا هم در ارتباط است.

سیستم باید در برابر خطا تحمل پذیر باشد.



4. کارایی: ...

سیستمی که نتواند درست و سریع کار کند ارزشی ندارد.

Throughput زیاد باشد. یعنی تعداد واحد زمانی که سیستم دستورات را اجرا می‌کند و بهره‌وری و اندازه باید زیاد شود

توان عملیاتی: تعداد عملیات سیستم که در زمان انجام می‌شود باید بیشتر باشد

Responce Time کم باشد یعنی زمانی که درخواستی می‌دهیم تا زمانی که CPU به درخواست پاسخ می‌دهد باید کم شود

زمان پاسخگویی: زمان لازم برای ... باید کم باشد

5. توسعه‌پذیری: ...

سیستم باید به راحتی قابل گسترش باشد.

یادداشت

2

Communication in Distributed Systems

- Bus – Based multi processor

در سیستم های با حافظه شراکتی ارتباط خیلی راحت از طریق عمل کردن فایل ها یا سمافور، Sph و ... برقرار می شود.
پروتکل تعاملی است که پروسه ها براساس آن ارتباط برقرار می کنند

- پروتکل OSI : دارای مدل سطحهای سازی شده است تا ارتباط برقرار شود ... Atm ,I ...
Open system Interconnection Reference Model

این مدل ارتباطات بین کامپیوتری را به هفت لایه یا سطح تقسیم می کند. و هریک از آنها براساس استانداردهای موجود در سطوح زیرین بنا می شوند.

قراردادهایی که در این لایه ها تعریف می شوند. هر پروتکل وظایفی دارد که به پروتکل های بالاتر از خود می دهد.

پروتکل ها باید با هم تفاهم داشته باشند. مثلاً: اینکه چند بیت بفرستند - چه خطایی رخ داده و چه کاهی برای رفع خطا ...

- ارتباط در مدل OSI : ...
اگر سیستم عامل ها باهم فرق کنند می توانیم آنها را به هم وصل کنیم.

① Connection oriented :

قبل از اینکه چیزی بین دو مدل A و B رد و بدل شود باید ارتباطی بین آنها برقرار شود بعد از ارسال این ارتباط حذف یا قطع می شود. ...
...

2. Connection Less oriented : ...

نیاز به اتصال ندارد و از پروسه A به B اطلاعات را ارسال می کند. می فرستد دیگر دریافت می کند

به امید اینکه کسی دریافت کند. ...
...


Fig 2-1

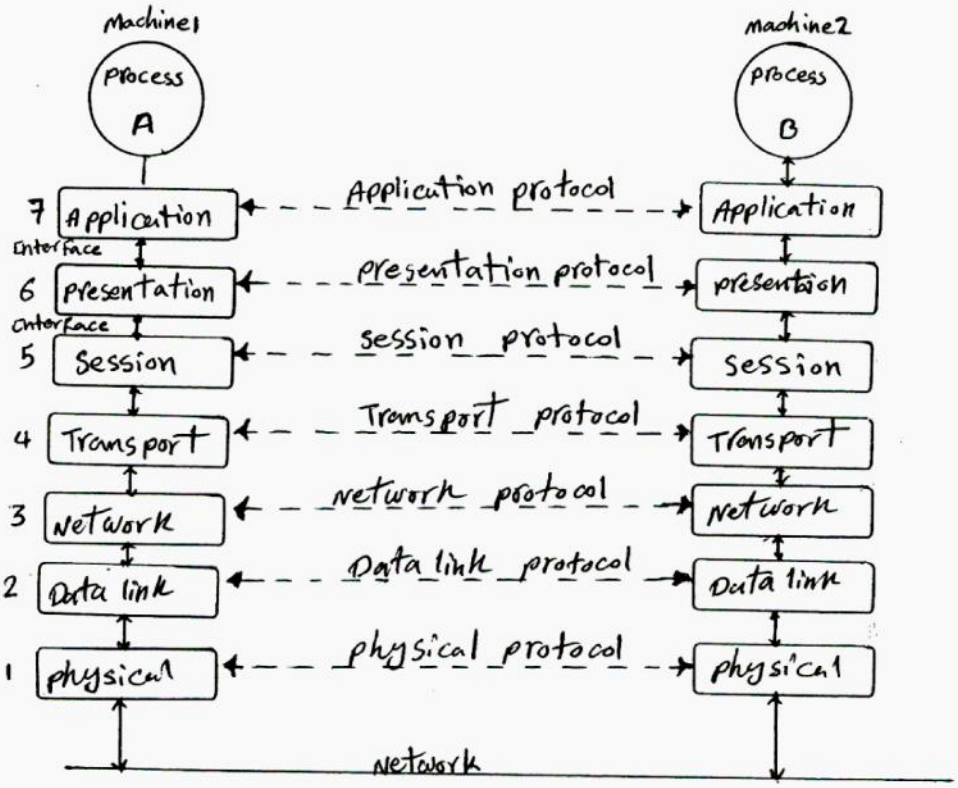


Fig 2-9

لایه فیزیکی : The physical Layer

این لایه اولین و پایین ترین لایه می باشد. این لایه برای انتقال تکتکی از بیت های نامنظم

از یک کامپیوتر به کامپیوتر دیگر تعریف شده است. در این لایه مشخص می شود که هر بیت اطلاعات

تا چه سانتی در محیط شبکه می تواند منتقل شود. رمزگذاری و همزمان سازی اطلاعات از وظایف

این لایه است. در حقیقت لایه فیزیکی ارتباط الکترونیکی و بانوری با کابل شبکه دارد.

در این لایه، لایه Hardware گفته می شود.

- The Data link layer : لایه پیوند داده

وظیفه اساسی این لایه شناسایی و خطایابی است. این لایه اطلاعات را از لایه شبکه گرفته و به لایه فیزیکی تحویل داده می‌شود. کامپیوتر گیرنده این اطلاعات را از لایه فیزیکی دریافت کرده و به صورت فریم داده (Data Frame) تبدیل می‌کند. آدرس فیزیکی شبکه یا Mac Address در این لایه قرار دارد. مهمترین وظیفه آن سالم رسیدن پاکت‌ها به مقصد در این لایه روش checksum برای چک کردن است. چک می‌کند که آیا فریم‌ها درست به مقصد رسیده‌اند.

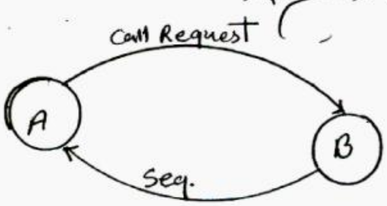
- The network layer : لایه شبکه

۱. هزینه
۲. مسافت
۳. hop count
بحث مهم در این لایه هستند.

مسیردهی یک پیام از مقصد به مبدأ و یا بلعکس وظیفه اصلی این لایه است.

بهترین مسیر بین مبدأ و مقصد را پیدا می‌کند. کوتاه‌ترین مسیر بهترین مسیر نیست. از طریق الگوریتم‌ها بهترین مسیر را پیدا می‌کند. متداول‌ترین پروتکل‌های لایه شبکه شامل:

۱. X25 : مورد علاقه شرکت‌های مخابراتی است.
بین مبدأ و مقصد یک مسیر را انتخاب می‌کند که پاکت‌ها فرستاده شود حال مقصد می‌تواند این پاکت‌ها را تأیید یا رد کند اگر تأیید کرد یک شماره به مبدأ می‌فرستد که برای ارسال دیگر پاکت‌ها از این شماره استفاده کند.



هنگامی که اتصال قطع شود شماره هم پاک یا قطع می‌شود.

از نوع Connection ~~oriented~~ می‌باشد.

2. IP :
پاکت‌ها را تقسیم می‌کند و آنها را از مسیرهای مختلف (یعنی هر مسیر دلخواهی که بخواهد) ارسال می‌کند.
در مقصد پاکت‌ها را به هم متصل می‌کند. از نوع Connection less ~~oriented~~ می‌باشد.

- لایه انتقال : The Transport layer

یکی از وظایف مهم این لایه این است که اطلاعات را با قابلیت اعتماد بالا از مبدأ به مقصد برساند.

کنترل جریان داده ها و یا تحمل بر خطاهای بوجود آمده در محیط شبکه توسط این لایه مدیریت و پیاده سازی می شود.

در این لایه صحت اطلاعات چک می شود. از وظایف بسیار مهم این لایه این است که در صورت سالم دریافت کردن اطلاعات یک پیام Acknowledge برای کامپیوتر فرستنده ارسال می کند.

- لایه نشست : The session layer

۱. هماهنگی ارسال و دریافت بین گیرنده و فرستنده را کنترل می کند که کدام در نوبت و کدام ارسال کند

۲. و عمل Synchoranization بین دو کاربر انجام می شود

در این لایه سرویس های امنیتی برای حفاظت از نرم افزارهای موجود در شبکه تعریف می شود. درخواست های کاربران توسط کانال های ارتباطی در این لایه نگهداری می شود. و در صورتی که یکی از کانال های ارتباطی به درستی کامل برقرار نشده باشد مجدداً همان کانال برای ارسال به کاربر انتخاب می شود.

- لایه نمایش : The presentation layer

این لایه فقط با مفاهیم صفر و یک سروکار دارد. که به آن نمایش می گویند

این لایه وظیفه تبدیل پروتکل ها، فشرده سازی اطلاعات برای کاهش حجم اطلاعات و مشخص کردن قالب تبادل اطلاعات بین دو کامپیوتر می باشد. اصطلاحاً به این لایه، لایه مترجم شبکه نیز گفته می شود. قسمتی از نرم افزارهای اطلاعاتی در این لایه مدیریت می شود.

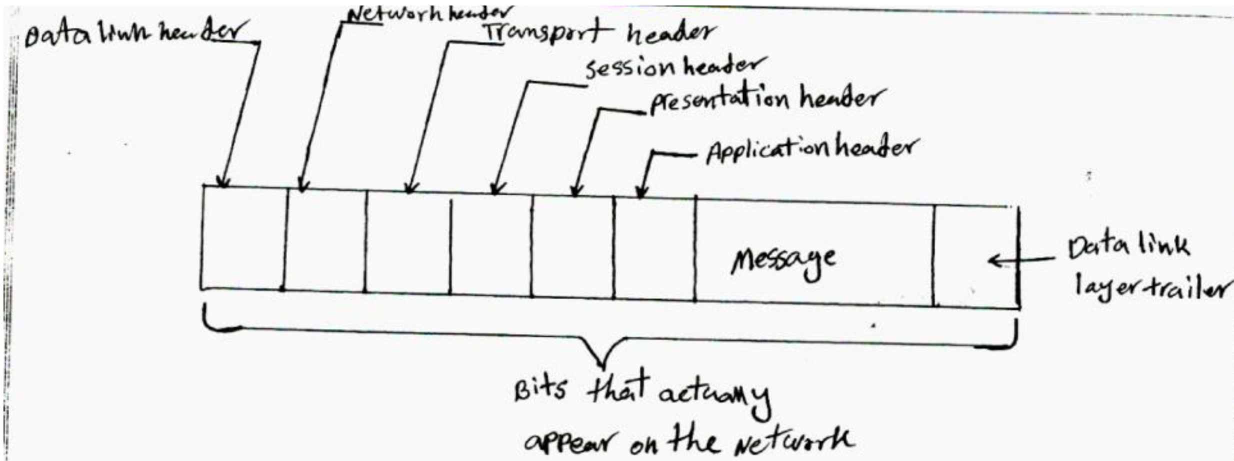
- لایه کاربردی : The Application Layer

این لایه بالاترین لایه در مدل OSI می باشد که وظیفه مدیریت و کنترل نرم افزارها را برعهده دارد.

بطور کلی این لایه وظیفه کنترل سرویس هایی که مستقیماً با نرم افزارهای کاربردی کار می کنند را دارد.

چون این خطاهای نرم افزاری برای برنامه های کاربردی را برعهده دارد.

. TCP/IP
. e-mail
. Remote login



در پایین یک هدر با عنوان دارد و با استفاده از آن هر پروتکل می داند که پاکت باید به کجا رود.

- ATM (Asynchronous Transfer mode):

شبکه های ATM نرخ متفاوتی دارد و نیاز با استفاده از این روال تغییرات و حجم آن در تغییرات

در شبکه دو نکته حائز اهمیت است. 1. داده data 2. صدا Voice

مثل پیام رسان ها و تلفن ... می خواهیم

- دلیل به وجود آمدن ATM:

شبکه های مخابراتی دچار مشکل شدند و گفتند صدا متفاوت است و نیاز به پهنای باند کمی دارد ولی داده که می خواهد از طریق خطوط تلفن برود متفاوت نیست و نیاز به پهنای باند زیادی دارد.

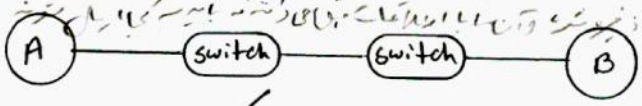
2 الگوی زیر وجود داشت:

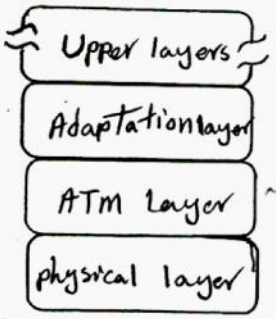
1. Packet switch: پاکت ها از مبدأ به مقصد از مسیرهای مختلفی عبور می کنند. پاکت را با پاکت ...

2. Circuit switch: ابتدا یک مدار از مبدأ به مقصد ایجاد می شود سپس پاکت ها از این مسیر مدار عبور می کنند وقتی ارتباط تمام شد ارتباط قطع می شود.

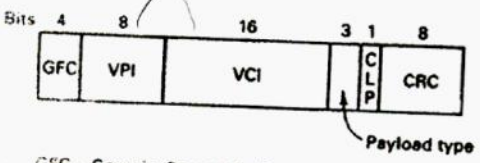
این دو روش، روش بجا و قابل قبولی نبودند به همین دلیل ATM ها به وجود آمدند.

حالا می خواهیم هم داده و هم صدا را از خطوط تلفن ...





The ATM reference model.



GFC = Generic flow control
VPI = Virtual path identifier
VCI = Virtual channel identifier
CLP = Cell loss priority
CRC = Cyclic redundancy checksum

4 بیت اول مربوط به کنترل کردن جریان است که به

هیچ وجه از آن استفاده نمی‌شود در حال حاضر از آن استفاده

می‌شود

GFC ⟶ 4
VPI ⟶ 8 ⟹ 24 bit
VCI ⟶ 16

تا اطلاعات مربوط به سوئیچ‌ها که باید از آن عبور کند در این 24 بیت ذخیره شده‌اند.

سلول وقتی به سوئیچ می‌رسد باید بداند از کدام خروجی سوئیچ عبور کند که در هر سوئیچ این اطلاعات را می‌خواند.

به این 3 بیت Payload type می‌گویند که مشخص می‌کند این سلول یک سلول کنترل کننده یا حاوی Data است.

CLP: یعنی اینکه اولویت یا اهمیت این سلول چقدر است. اگر با اهمیت است حذف نمی‌شود. اگر بی‌اهمیت است براساس اولویت حذف می‌شود. سلول‌هایی که سوئیچ‌ها خواهند آن را حذف کنند براساس CLP حذف می‌کنند

CRC: چک می‌کند آیا سلول به درستی به مقصد رسیده است یا خیر.
این همان checksum است

Payload type: چک می‌کند که ببیند سلول داده حمل می‌کند یا اطلاعات کنترل را.

عرض

نکته: 40 بیت یا 5 بایت هدر

48kb + 5kb = 53kb
data ↙ Header ↙

پس هر سلول 53kb است.

حد بزرگتر برای داده و کوچکتر برای هدر

: ATM switching

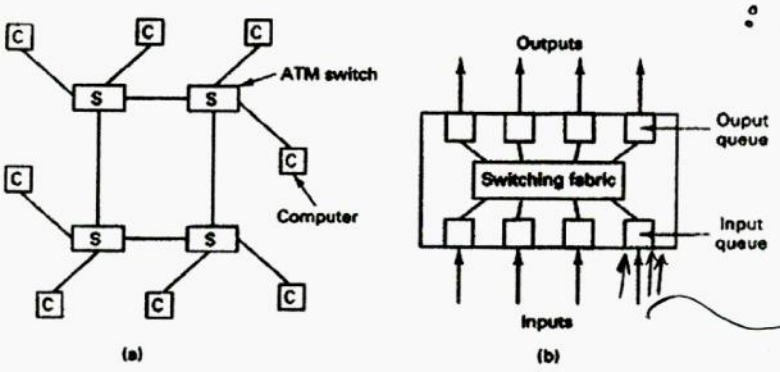
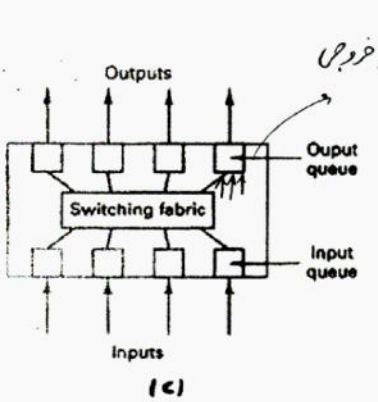


Fig. 2-6. (a) An ATM switching network. (b) Inside one switch.



در شکل (a) 4 تا سوئیچ و 8 کامپیوتر مشاهده می‌کنید.

در شکل (b) یک سوئیچ 4 پورت مشاهده می‌کنید هر پورت هم برای ورودی و هم برای خروجی استفاده می‌شود. در این شکل ترافیک blocking روی ورودی است پس اسم آنرا head ofline switching

در اینجا باید با الگوریتم‌ها ترافیک را در صف، صف بندی کرد.

در شکل (c) همانطور که مشاهده می‌کنید ترافیک روی خروجی است که سوئیچ‌ها ازدحام بر این اساس ساخته شده‌اند یعنی صف ترافیک را روی خروجی قرار می‌دهند.

Example: مسئله‌ای با سرعت 64Kb و یک فایل 1MB می‌خواهیم زمان ارسال این فایل را بدست آوریم.

$$\frac{1Mb \times 1000 = 1000Kb}{64\ kb} = 15.6\ sec$$

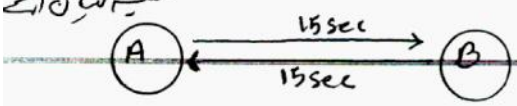
این فایل در 15.6sec ارسال می‌شود.

15 sec برای ارسال اطلاعات

15 sec برای رسیدن تأیید به اطلاعات ⟸ 30 sec

$15.6 + 30 = 45.6$ sec

سرعت ارسال و دریافت همیشه یکی است



Example: شبکه ATM 155 mbps با فایل 1 mb

$$\frac{1\,mb \times 1000 = 1000\,kb}{155 \times 100} = \frac{1}{155} = /006 = 6\ mses$$

6 میلی ثانیه طول فایل می‌رود

$30 + /006 = 30.0006$

تأخیر در فایل بالاتر از این فایل است.

Example: شبکه 622 mbps با فایل 1 mb

$$\frac{1\,mb \times 1000 = 1000\,kb}{622} = 1.6\ sec + 30 = 31.6\ sec$$

→ همیشه یکسان است سرعت ارسال ویدیو

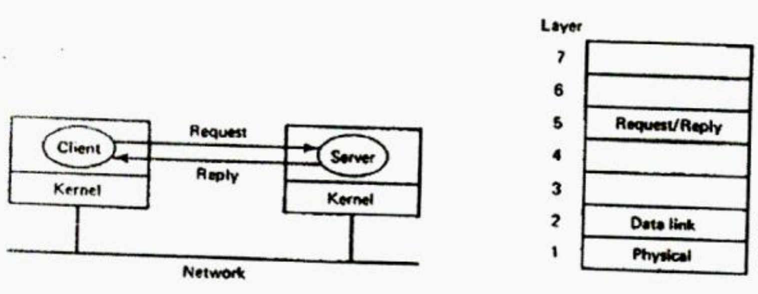
نکته: در شبکه‌های ATM مشکل اصلی تأخیر اطلاعات برای ارسال ویدیو است که اغلب خط خالی است و کاری انجام نمی‌دهد. مثلاً: [illegible]

از 100٪ فقط 5٪ زمان مفید است. 95٪ به هدر می‌رود (زمانی که ارسال می‌شود)

همچنین یکی از مشکلات ATM است [illegible] تلف می‌شود [illegible]

در حالت کلی مدل OSI به دلیل مدیریت دشوار و هدرهای ایجاد شده و مدل ATM به دلیل هزینه‌های سرسام‌آور استفاده نمی‌شوند و از مدل ↓ استفاده می‌شود.

The client_server model



(a)　　　(b)

} → وظیفه ارسال ویدیو دارند

- مزایای client-server model :

1. ساده بودن : simplicity : بر روی شبکه کسانی که ... request ... reply ...

2. کارایی : efficiency : ...

+ پروتکل های client-server :

1. Send (dest , mptr); ارسال کننده (هم پیام و هم مقصد مشخص است)

2. receive (add , mptr) دریافت کننده (پیام مشخص شده را ...)

+ نحوه آدرس دهی Addresing :

Send (dest , mptr);

آدرس دهی فرآیند با یکی از موارد زیر رو برو باشد : ② یکی از پروسس های روی سرور ① server

Exampls:

Send (240 , & mptr);

send (240 , 04 , & mptr);

Send (04 , 240 , & mptr);

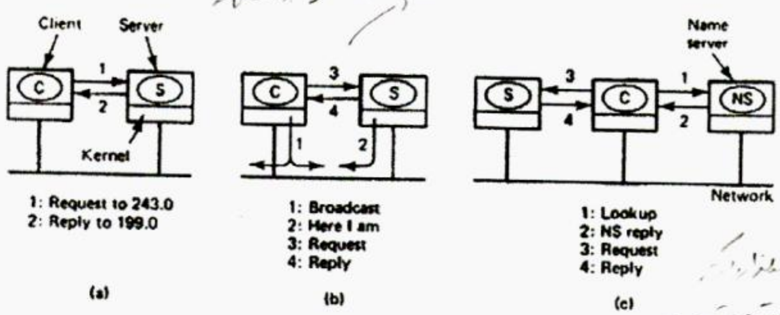


Fig. 2-10. (a) Machine.process addressing. (b) Process addressing with broadcasting. (c) Address lookup via a name server.

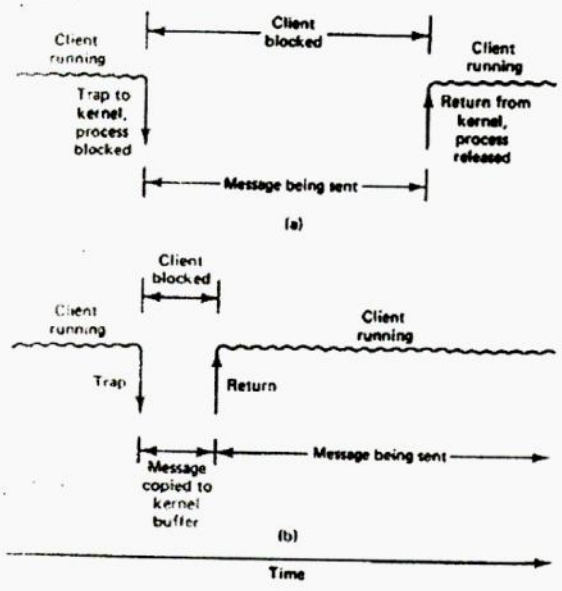


Fig. 2-11. (a) A blocking send primitive. (b) A nonblocking send primitive.

Send
Receive
: Blocking -

استفاده درستی از CPU نیست چون مطابق شکل روبرو
تا زمانی که پیام از سمت client به مقصد نرسیده باشد
باید client در حالت block قرار بگیرد.

: Non blocking -

مادامی که پیام در حال ارسال است و پیامی از سمت
مقصد به client مبنی بر تایید دریافت پیام ارسال
می شود و در حال ارسال پیام است. شکل (b)

روشها یا مشکلات Non-blocking به شرح زیر می باشد:

1- client چطور در سمت (b) می فهمد که پیام به مقصد رسیده است یا نه؟

2- زمانی که client در حال ارسال پیام است و آزاد می شود آیا می تواند از Buffer استفاده نماید؟

راه حل

ابتدا پیام از پروسه A به Kernel می رود
و در آنجا ذخیره می شوند پس از Kernel به
Kernel سرور ارسال می شود و از آنجا به پروسه B و سرور می رسد.

مشکل این روش بافر پر می شود از پیام هایی که باید به Kernel ارسال شد.

2) Interrupt:

این روش نیاز به برنامه نویسی به روش Interrupt دارد
و از آنجا که برنامه نویسی با Interrupt ها دشوار است
پس مشکل است.

: Buffering and Non Buffering -

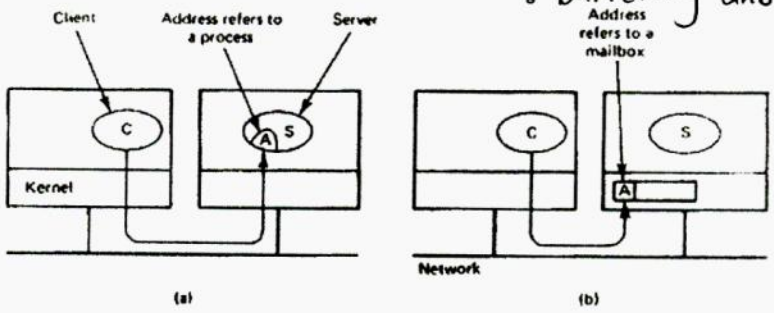


Fig. 2-12. (a) Unbuffered message passing. (b) Buffered message passing.

در شکل (a) عمل بافرینگ روی سرور انجام می‌شود و بطور ضمنی به سرور فرستاده می‌شود.

در شکل (b) عمل بافرینگ روی سرور انجام نمی‌شود و تمام پیام‌ها باید بافر شوند که این مشکل است.

: Reliable and unreliable -

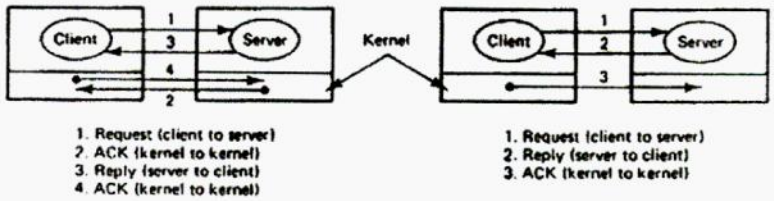


Fig. 2-13. (a) Individually acknowledged messages. (b) Reply being used as the acknowledgement of the request. Note that the ACKs are handled entirely within the kernels.

Reliable : باتوجه به شکل (a) تعداد درخواست‌ها زیاد است ولی اطمینان می‌یابیم که حتماً پیام به مقصد رسیده است. پاسخ تأیید از سرور به کلاینت با ACK ← ACK Nowledgement

unreliable : باتوجه به شکل (b) تعداد پیام‌ها کمتر شده و اعتماد پذیری ما بیشتر کرده‌ایم ما جواب سرور به کلاینت را در قالب جواب تأیید و پاسخ دادن استفاده کرده‌ایم.

Item	Option 1	Option 2	Option 3
Addressing	Machine number	Sparse process addresses	ASCII names looked up via server
Blocking	Blocking primitives	Nonblocking with copy to kernel	Nonblocking with interrupt
Buffering	Unbuffered, discarding unexpected messages	Unbuffered, temporarily keeping unexpected messages	Mailboxes
Reliability	Unreliable	Request-Ack-Reply Ack	Request-Reply-Ack

Fig. 2-14. Four design issues for the communication primitives and some of the principal choices available.

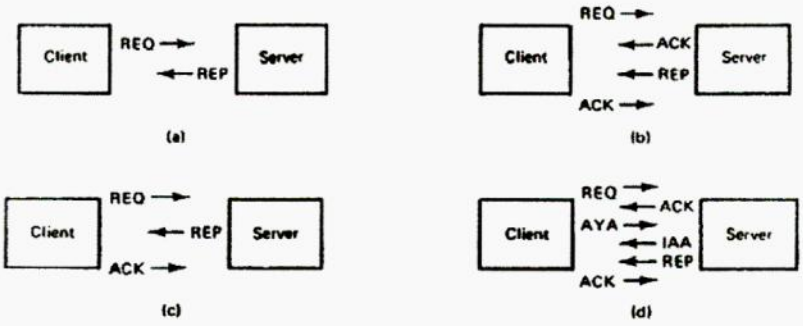


Fig. 2-16. Some examples of packet exchanges for client-server communication.

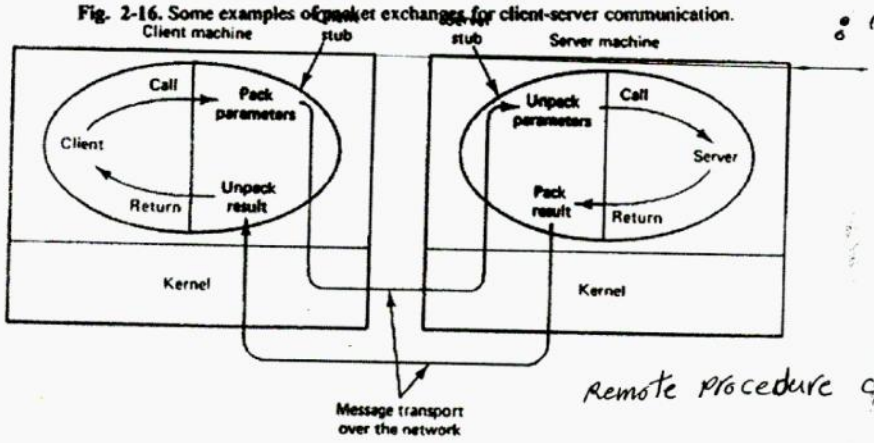


Fig. 2-18. Calls and messages in an RPC. Each ellipse represents a single process, with the shaded portion being the stub.

RPC Semantics in the Presence of Failures

1. The client is unable to locate the server.
2. The request message from the client to the server is lost.
3. The reply message from the server to the client is lost.
4. The server crashes after receiving a request.
5. The client crashes after sending a request.

① کلاینت نتواند سرور را پیدا کند.
② درخواست از سمت کلاینت به سرور گم شود.
③ جواب از سمت سرور به کلاینت گم شود.
④ سرور crashe کرده باشد
⑤ کلاینت crashe کرده باشد.

1. کلاینت نمی‌تواند سرور را پیدا کند:
در این حالت باید یک پیام فرستاد که سرور پیدا نشده است. مثلاً 5- یا ... اگر پیدا نکرد سرور است.
دوم رسم خطاهایی که باید توسط برنامه پیدا شود exception می‌باشد

2. درخواست از سمت کلاینت به سرور گم شده باشد:
بهترین راه این است که مجدداً کلاینت درخواست خود را دوباره ارسال کند البته با Time out ... شخص کرده است بعد از اتمام Timeout درخواست را دوباره ارسال کند. ... Timeout

3. جواب از سمت سرور به کلاینت گم شده باشد:
آگر به این صورت رخ داد کلاینت دوباره درخواست می‌کند البته باید درخواست به صورت idempotent یا بدون ... آزار باشد که در این موارد مشکل ندارد. در کل مبارزه با این حالت کار ساده‌ای نیست. چون ... مشکل درخواست‌هایی که آزار دهنده هستند را به صورت اینکه به هر پیام یک عدد یا شماره اختصاص دهیم که مشخص کننده باشد که این پیام دوباره فرستاده شده است. ...

4. سرور crashe کرده باشد:

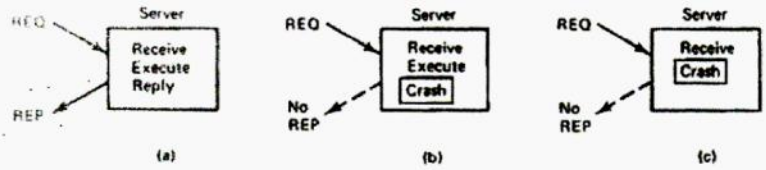


Fig. 2-24. (a) Normal case. (b) Crash after execution. (c) Crash before execution.

- زمانی که در سرور محاسبه صدرت گرفته باشد و بعد crashe کرده باشد می‌دانیم جوابی وجود دارد ولی برگشت نداره است. at most once semantic حداکثر یکبار حداقل هیچ بار

در بهترین حالت از یک جواب تا چندین جواب دارد 1 - many
ممکن است کلاینت زمانی چندین درخواست تکراری بفرستد

در حالت سوم که در بخش دریافت از سرور سیستم crashe میدانیم در حالت محاسبه یا جواب نرسیده است.

← اگر client درخواست نکرده باشد ← صفر جواب دارد بهترین حالت

← اگر client درخواست کرده باشد ← at least once semantic

صفر تا چند جواب دارد ← 0 — many

5. کلاینت crashe کرده باشد:

کلاینت درخواست خود را به سرور می‌فرستد، سرور پس از محاسبه آن را به کلاینت می‌خواهد بفرستد به صورتی که کلاینت crashe کرده است. در این حالت محاسبه صاحبی ندارد.

به محاسباتی که صاحب ندارند orphan یا یتیم می‌گویند این نوع محاسبات هم cpu و هم منابع را درگیر می‌کنند.

روش های مقابله با orphan ها:

(A) Extermination: نابودی یا براندازی یا از بین بردن ساختن کار

سرور همه‌ی محاسباتی که صاحب ندارند یا یتیم هستند را از بین می‌برد.

(B) Reincarnation: تولد یابی یا تن یابی جدید یا تجسد

هنگامی که کلاینت دوباره زنده شود دوباره با پیام Iam alive زنده بودن خودش را می‌فرستد. سرور در این موقع چک می‌کند که آیا محاسباتی که متعلق به این کلاینت هستند را به آن

Kazem.faridi@iaubaft.ac.ir

می‌فرستد. البته کلاینت مادامی که زنده بودن خود را به سرور می‌فرستد همراه با زمان می‌فرستد تا سرور بداند این کلاینت از این زمان به بعد زنده هست. اگر دوباره این کلاینت پیام بفرستد سرور می‌فهمد که این درخواست با قبلی یکی هست و محاسبات را کلاً حذف می‌کند. چون دوباره پیام می‌فرستد [illegible]

(C) Gentle Reincarnation : تجسد واسته

[illegible]

در این حالت برعکس حالت قبل وقتی کلاینت دوباره بعد از زنده شدن درخواست داد سرور محاسباتی که مربوط به این کلاینت هستند و محاسبه شده اند را برای کلاینت می‌فرستد و آنهایی منظور محاساتی که در حال محاسبه شدن هستند و هنوز کامل نیستند را از بین می‌برد.

(D) Expiration : زمان انقضاء یا اتمام

یک زمان در هر زمان انقضاء یا اتمام شدن است برای سرور در نظر می‌گیریم اگر در این زمان کلاینت به آن درخواست داد محاسبات را برای آن می‌فرستد اگر درخواستی از کلاینت در این زمان به سرور نرسید سرور می‌فهمد که کلاینت crashe کرده و در نهایت تمام محاسبات را حذف می‌کند. درخواست‌ها از کلاینت به سرور می‌رود و از تاریخ انقضا آن در زمان
جدایی از سرور کلاینت برگشت ok می‌باشد [illegible]
محاسبات [illegible]

Kazem.faridi@iaubaft.ac.ir

3

Synchronization in Distributed Systems

هماهنگی بین پروسه‌ها

- هماهنگی ساعت‌ها clock synchronization

- الگوریتم‌های توزیع شده چه خصوصیاتی دارند؟

1. اطلاعات مربوط، بین ماشین‌های مختلف پراکنده است.
2. پروسه‌ها بر اساس اطلاعات محلی تصمیم‌گیری می‌کنند.
3. نباید اطلاعات در یک جا باشند.
4. چیزی به عنوان ساعت کل (عمومی) نداریم.

نکته: UniX در ابتدا برای اجرای یک برنامه اول فایل object آن را اجرا می‌کند. اگر آن را پیدا نکرد کد اصلی آن برنامه را پیدا کرده آن را کامپایل کرده سپس object file آن برنامه را اجرا می‌کند.

مثلاً: test.com test.o

- ساعت منطقی و ساعت فیزیکی

ساعت منطقی:

شخصی به اسم لمپورت گفت:

1. اگر دو پروسه هیچ نیازی به هم ندارند نیازی نیست ساعت‌هایشان با هم کار کند.
2. اگر دو پروسه به هم نیاز دارند از قانون happens-Before استفاده می‌کنیم.

← یعنی اگر a و b به گونه‌ای هستند که a باید قبل از b رخ دهد پس باید ساعت a قبل از ساعت b باشد. $C(a) < C(b)$ (ساعت a، ساعت b)

a happens — Before b

نکته: پروسه‌هایی که هم زمان رخ می‌دهند و ارتباط به هم ندارند هیچ مشکل ایجاد نمی‌کنند.

نکته: اگر a یک پیام است که از یک ماشین به ماشین دیگری رود نمی‌تواند در همان زمان یا در زمان کوتاه‌تر برسد. باید زمان بیشتری طول بکشد. a نیز باید از زمان b کمتر باشد.

$a \longrightarrow b \longrightarrow c$

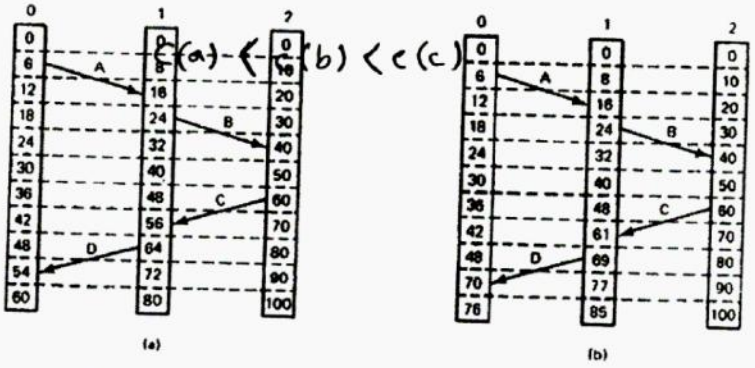


Fig. 3-2. (a) Three processes, each with its own clock. The clocks run at different rates. (b) Lamport's algorithm corrects the clocks.

در شکل (a) ماشین صفر یک پیام در زمان 6 فرستاده و در زمان 16 توسط ماشین 1 دریافت شده

ماشین 1 یک پیام در زمان 24 فرستاده و در زمان 40 توسط ماشین 2 دریافت شده

ماشین صفر $6 \longrightarrow 16$ ✓ ماشین یک $24 \longrightarrow 40$ ✓

اما ماشین 2 در زمان 60 پیام فرستاده که در زمان 56 دریافت کرده توسط ماشین یک که غیر قابل قبول است. در شکل (b) مشکل ارسال پیام از ماشین 2 به 0 و 1 به صفر درست شده است.

ماشین 2 $60 \longrightarrow 56$ ✗ ماشین 1 $64 \longrightarrow 54$ ✗

← زمان ارسال پیام باید کمتر از زمان دریافت باشد. در موارد بالا که برعکس این موضوع است آنها را بزرگتر از زمان ارسال پیام تغییر می‌دهیم. چه کمتر و چه مساوی زمان ارسال باشد باید بزرگتر

ماشین 2 $60 \longrightarrow 61$ ✓ ماشین 1 $69 \longrightarrow 70$ ✓

نکته: یک پیام نمی‌تواند قبل از زمانی که ارسال شده به مقصد برسد.

در حالت کلی داریم که:

1. اگر a قبل از b رخ دهد پس باید زمان a قبل از زمان b باشد. $C(a) < C(b)$

2. اگر a و b دو پیام هستند باید زمان a کمتر از زمان b باشد (در سیستم زمان فرستادن باید کمتر از زمان دریافت باشد). $C(a) < C(b)$

3. اگر پروسه‌های ما هیچ ارتباطی با هم ندارند پس منطقی است که زمان آنها هیچ ارتباطی به هم ندارند.
$C(a) \neq C(b)$

- ساعت فیزیکی:

باید سیستم‌های موجود را کنار هم قرار داد تا ساعت‌های cpu ها را کنترل کنند.

1. الگوریتم کریستین *: Cristian Algoritm

یک پروسه هر زمان که بخواهد می‌تواند درخواست زمان را از سیستم بکند.

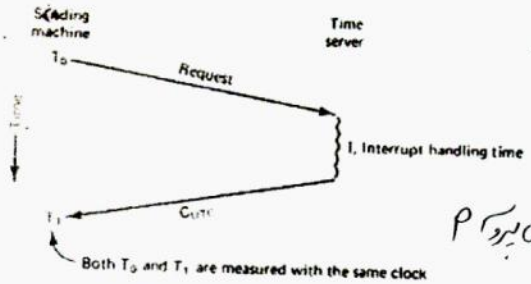


Fig. 3-6. Getting the current time from a time server.

پروسه مطابق شکل یک پیام به Time Server می‌فرستد.

و T_s جواب می‌دهد که همان T است.

زمان برای پروسه P $= T + T_{trans}$ Time Server $= T_s$

T_{trance}: پیام از T_s به P برگشته است.

T_{round}: زمان که رفته و برگشته است.

$T_{trance} = T_{round}/2$ ← تقسیم بر 2 برای رفت و برگشت

نحوه محاسبه زمان: T_s در T زمان

$T + min$ ⟵ بهترین حالت

$T + [t_{trance} - min]$ ⟵ بدترین حالت

$T \pm [T_{trance}/2 - min]$ ... اگر min کمترین زمان باشد که پیام یک مسیر را طی نموده است.

2. Berkelay Algorithm :

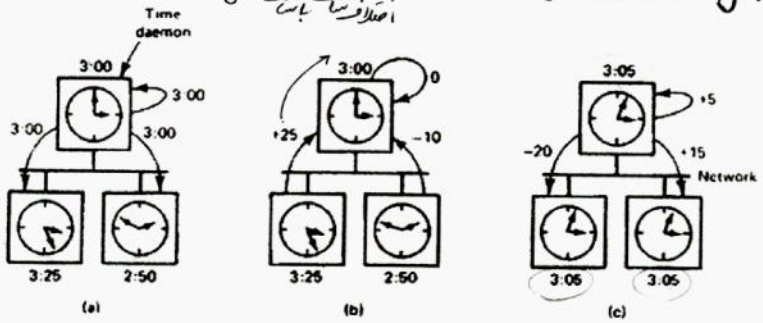


Fig. 3-7. (a) The time daemon asks all the other machines for their clock values. (b) The machines answer. (c) The time daemon tells everyone how to adjust their clock.

$$\text{میانگین} = \frac{0 + 25 - 10}{3} = +5$$

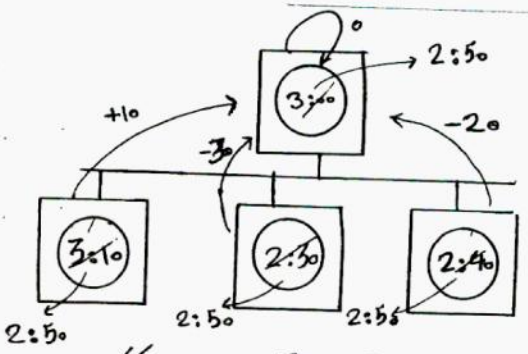
A $-25 + 5 = -20$

B $+10 + 5 = 15$

C $0 + 5 = 5$

میانگین را با مخالف علامت‌های اختلاف ساعت‌ها جمع می‌کنیم.

Example :



$$\text{میانگین} = \frac{0 - 30 - 20 + 10}{4} = -10$$

نکته: اگر حاصل میانگین عدد اعشاری مثل 6.25 و یا هر عدد دیگری درآمد آن را گرد می‌کنیم برای مثال می‌شود 6

A $0 - 10 = -10$

B $-10 - 10 = -20$

C $+30 - 10 = +20$

D $+20 - 10 = +10$

میانگین را با کمترین اختلاف ساعت‌ها جمع می‌کنیم و حاصل را از ساعت‌های موجود کم می‌کنیم.

: MUTUAL EXCLUSION انواع روش‌های منطقه بحرانی

A Token Ring Algorithm (۳ Distributed Algorithm (۲ A Centralized Algorithm -۱

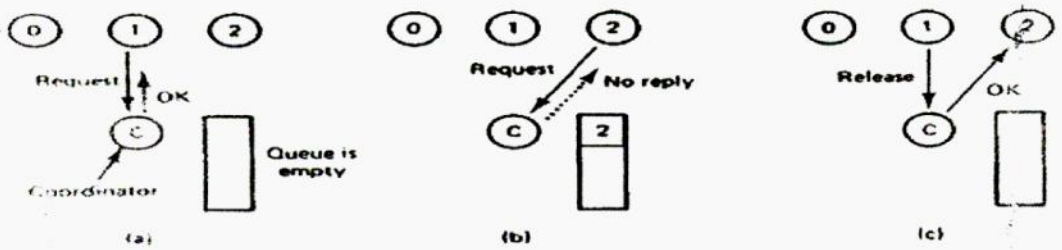
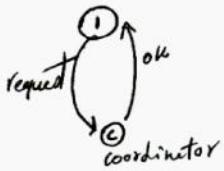


Fig. 3-8. (a) Process 1 asks the coordinator for permission to enter a critical region. Permission is granted. (b) Process 2 then asks permission to enter the same critical region. The coordinator does not reply. (c) When process 1 exits the critical region, it tells the coordinator, which then replies to 2.

یک پروسه برای ورود به منطقه بحرانی استفاده کند.

۱. بهترین حالت: پروسه ۱ درخواست Request را به Coordinator می‌فرستد.



2. بدترین حالت:

پروسه یک و دو با هم درخواست می‌فرستند. چون پروسه ۱ وارد منطقه بحرانی شده و دیگری اجازه نمی‌دهد که پروسه 2 وارد شود. پس آن را در صف می‌گذارد و دوباره پروسه ۱ را در صف نگه می‌دارد.

مزایا: ۱. در هیچ زمان ۲ پروسه وارد منطقه بحرانی نمی‌شوند.

2. عدالت برقرار است پروسه‌ها به ترتیب وارد صف می‌شوند

3. هیچ پروسه‌ای بی‌نهایت زمان در صف نمی‌ماند. گرسنگی رخ نمی‌دهد.

بدترین حالت برای Crash کردن Coordinator

معایب: ۱. اگر C خراب شود کل سیستم از کار می‌افتد و سیستم Crashe می‌کند.

2. ممکن است پیام‌ها گم شوند.

3. عدم جواب دادن Coordinator برای این است که Crashe کرده یا در صف قرار دارد.

: MUTUAL EXCLUSION

A Distributed Algorithm ـ۲

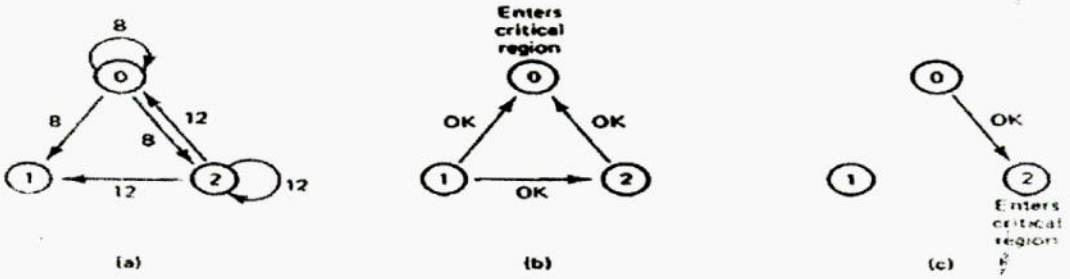


Fig. 3-9. (a) Two processes want to enter the same critical region at the same moment. (b) Process 0 has the lowest timestamp, so it wins. (c) When process 0 is done, it sends an *OK* also, so 2 can now enter the critical region.

: MUTUAL EXCLUSION

۳- A Token Ring Algorithm

هر پروسه‌ای تقاضا پروسه‌ای که حلقه‌ی آن باید اجرا شود راه داند

در بدترین حالت یک پروسه یک دور کامل منتظر می‌ماند تا نوبت به آن برسد

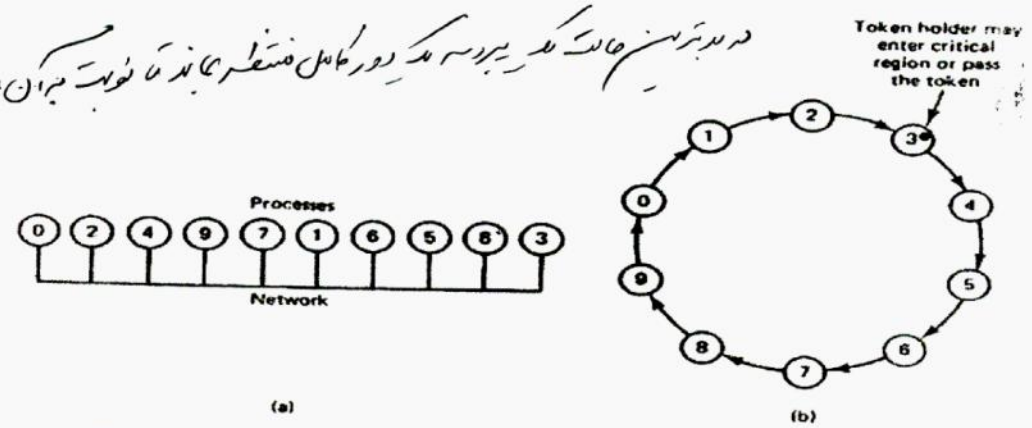


Fig. 3-10. (a) An unordered group of processes on a network. (b) A logical ring constructed in software.

پروسه‌ها باشت سرهم کرده حلقه قرار گرفته‌اند. فقط یک Token وجود دارد. پروسه باید Token داشته باشد تا بتواند وارد منطقه بحرانی شود و، وقتی از منطقه بحرانی خارج شد Token را آزاد کند تا بتواند از یک Token چندبار استفاده کند. فقط یک بار پروسه می‌تواند با استفاده از آن وارد ناحیه بحرانی شود

در این حالت محلی برای گرسنگی نداریم - عدالت هم رعایت می‌شود.

مرحله ۱ → غیر از خودش به $n-1$ پیام می‌فرستد

مرحله ۲ → بقیه $n-1$ تا پیام می‌فرستند که تو وارد نشو

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Distributed	$2(n-1)$	$2(n-1)$	Crash of any process
Token ring	1 to ∞	0 to $n-1$	Lost token, process crash

Fig. 3-11. A comparison of three mutual exclusion algorithms.

- مقایسه ۳ الگوریتم از لحاظ ۳

۱. تعداد پیام‌هایی که لازم است برای ورود و خروج از منطقه بحرانی

۲. تعداد پیام‌هایی که قبل از ورود به منطقه بحرانی

۳. مشکلات

- الگوریتم‌های انتخابی Election Algorithms :

نکته : تمام الگوریتم‌ها پروسه‌ای را انتخاب می‌کند که بالاترین شماره را دارد .

الگوریتم‌ها در نحوه انتخاب پروسه‌ها تفاوت دارند ولی ساختار انتخاب یکی است .

۱- الگوریتم زورگو The Bully Algorithm :

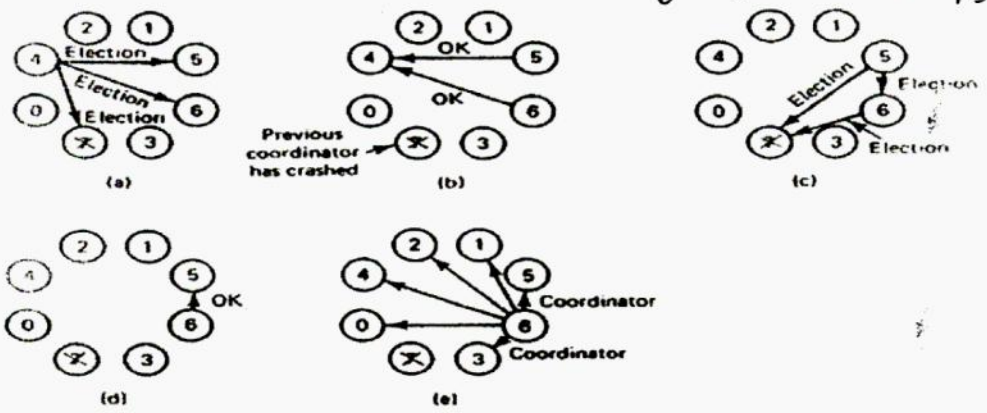


Fig. 3-12. The bully election algorithm. (a) Process 4 holds an election. (b) Processes 5 and 6 respond, telling 4 to stop. (c) Now 5 and 6 each hold an election. (d) Process 6 tells 5 to stop. (e) Process 6 wins and tells everyone.

پروسه‌ای که بالاترین شماره را دارد به عنوان C انتخاب می‌شود .

پروسه 4 فرض می‌کند که پروسه 7 از کار افتاده است . به پروسه‌های 5 و 6 پیام می‌فرستد . پروسه 5 بدلیل اینکه از پروسه 4 بزرگتر است انتخاب می‌شود و بعد به پروسه 6 پیام می‌فرستد پس پروسه 6 که از همه‌ی پروسه‌ها بزرگتر است انتخاب می‌شود و بعد به همه پروسه‌ها پیام می‌دهد که به عنوان Coordinator انتخاب شده است

پروسه	ارسال پیام	دریافت پیام
0	n-1	n-2
1	n-2	n-3
⋮	⋮	⋮
6	1	0

مثلاً : پروسه 4 به پروسه‌های بزرگتر از خودش پیام می‌فرستد

یعنی 5 و 6 و 7 ولی 2 پروسه جواب می‌دهند $(n-2)$

$8-2=6$ ← در بهترین حالت که 6 ، Coordinator شده و به همه پیام می‌دهد که C شده ، پروسه 7 هم که از کار افتاده

$$\text{مجموع ارسال} = \frac{n(n-1)}{2}$$

پیچیدگی $O(n^2)$

2. الگوریتم حلقوی : A Ring Algorithm

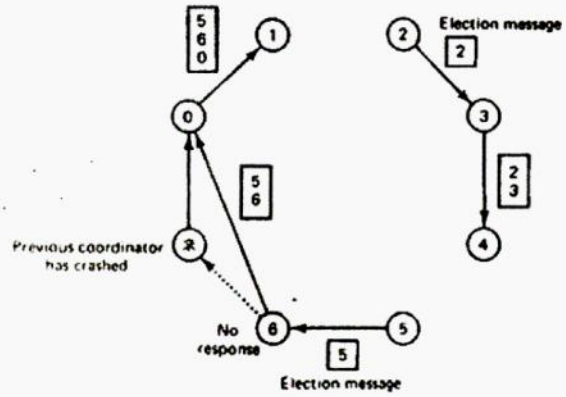


Fig. 3-13. Election algorithm using a ring.

وقتی Coordinator از کار می‌افتد. پس انتخابات شروع می‌شود.

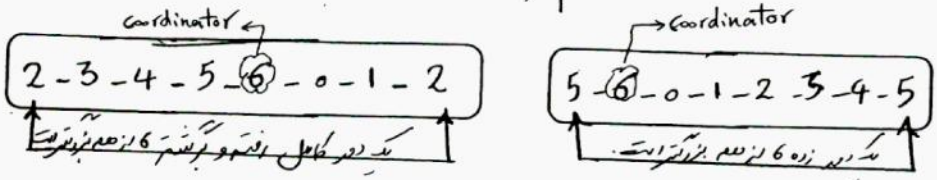
پروسس 2 انتخاب را شروع کرده

پروسس 2 می‌فهمد که پروسس 6 از همه بزرگتر است.

پروسس 2 در ابتدا به پروسس 3 پیام می‌فرستد.

پس پروسس 3 به پروسس 4 و بعد پروسس 4 به پروسس 5

و بعد پروسس 5 به پروسس 6 پیام می‌فرستد و بازهم پروسس 6 به عنوان Coordinator انتخاب می‌شود.

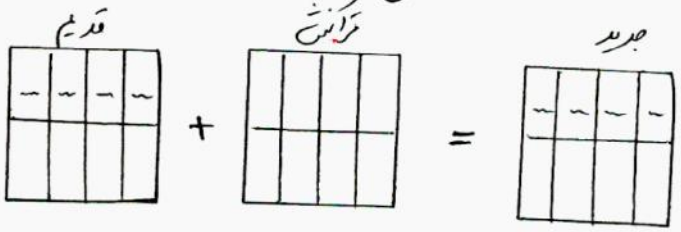


بزرگترین مقدار به عنوان Coordinator انتخاب می‌شود.

- تراکنش های اتمیک : Transaction Atomic

تراکنش که یا باید همه آن یکجا انجام شود یا اصلا انجام نشود در واقع به صورت یکپارچه انجام شود.

مثل آب : Example



ساختار جدید و قدیم مثل هم هستند.

- 5 دستور که در تراکنش هست:

1. Begin : شروع تراکنش را نشان می‌دهد.
2. End : پایان تراکنش را نشان می‌دهد.
3. Abort : تراکنش را ناقص باطل می‌کند.
4. Read : برای خواندن تراکنش است.
5. Write : برای نوشتن تراکنش است.

- هر تراکنش باید ۴ خاصیت داشته باشد: ACID *

1. Atomicity (همه یا هیچ): یک تراکنش یا باید بطور کامل انجام شود یا اصلاً اجرا نشود.
2. Consistency (یکپارچگی): همه باید قوانین را رعایت کنند.
3. Isolated (انزوا): به صورت سریال انجام شوند و پروسه ها روی هم تأثیر نداشته باشند.
4. Durable (ماندنی): اگر نتایج یک پروسه نهایی شود آنها دائمی می شوند هیچ راه برگشتی وجود ندارد.

- نحوه پیاده سازی تراکنش ها: ۱- فضای خصوصی ۲- write aheadlog ۳- two-phase commit protocol ۴- concurrency control و optimistic concurrency

(1) Private work space (فضای خصوصی):

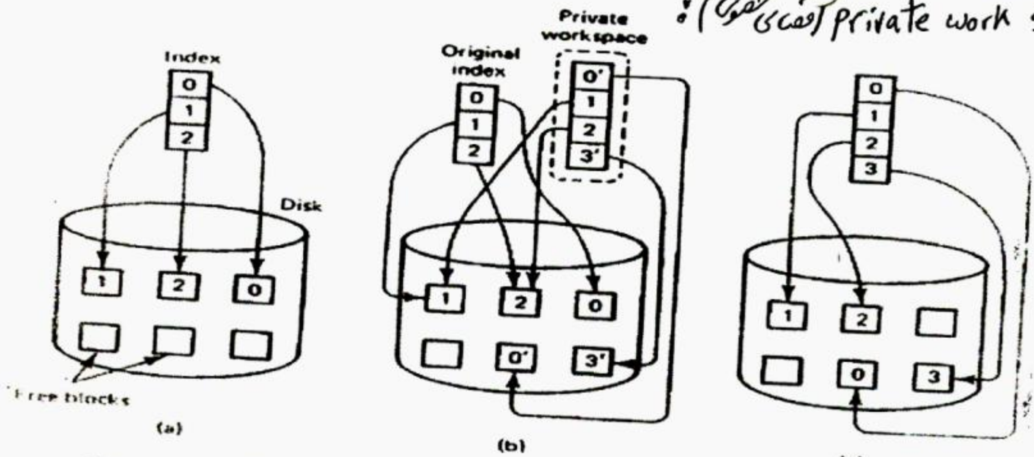


Fig. 3-18. (a) The file index and disk blocks for a three-block file. (b) The situation after a transaction has modified block 0 and appended block 3. (c) After committing.

می خواهیم از فایل a یک کپی بگیریم سه فایل داریم که در یک فضای دیگر در تراکنش ها با آن کار کنند (اگر کارش با فایل a تمام شد و آن را Commit کرد تمام تغییرات به فایل اصلی منتقل می شود. اگر Abort کرد نتایج (تغییرات) حذف می شوند و فایل بسته می شود.

برای Read می توان از همان فایل اصلی استفاده کرد. به کپی کردن فایل نیازی نیست.

برای write کردن باید فایل را کپی کنیم و تغییرات را انجام دهیم.

: writeahead log (2)

فایل log تمام اتفاقاتی که در سیستم می‌افتد را ثبت می‌کند.

```
BEGIN_TRANSACTION
  x = x + 1;
  y = y + 2;
  x = y * y;
END_TRANSACTION
```

(a)	(b)	(c)	(d)
	Log: x = 0/1	Log: x = 0/1, y = 0/2	Log: x = 0/1, y = 0/2, x = 1/4

Fig. 3-19. (a) A transaction. (b)–(d) The log before each statement is executed.

هر خط این فایل شامل اطلاعات قدیم و اطلاعات جدید است در واقع هر دو اطلاعات هستند. تراکنش در هر زمان با تابع کردن بر این فایل می‌تواند بفهمد که در چه حالتی است.

: Two-phase commit protocol (3)

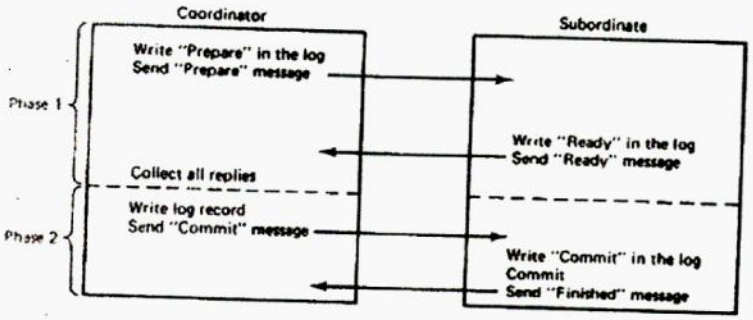


Fig. 3-20. The two-phase commit protocol when it succeeds.

در phase اول تراکنش با تمام تراکنش‌هایی که می‌خواهد در ارتباط باشد یک پیام می‌فرستد که Prepare کند.

در phase دوم تراکنش‌ها پیغام می‌دهند که Commit کن.

: کنترل همزمانی Concurrency control (4)

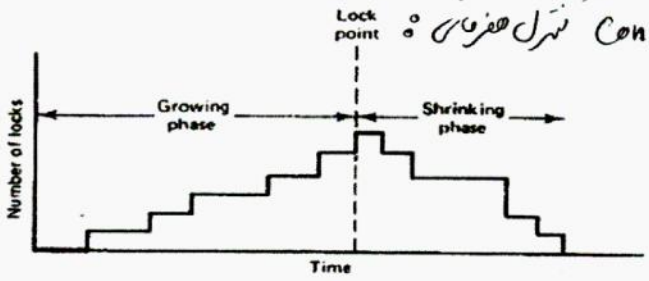


Fig. 3-21. Two-phase locking.

می‌توانند یک رکورد یا فایل یا DB را قفل کنند. می‌توانند وقتی با فایل کار دارند آن را قفل کنند و سپس commit کنند.

قفل کردن رکوردها هزینه بر است اما اگر با یک رکورد کار داشته باشیم منطقی نیست که کل فایل را قفل کنیم.

بعد از اینکه کار با فایل یا رکورد تمام شد باید قفل را برداریم تا هیچ مشکلی به وجود نیاید.

نکته: ← برای read باید قفل shared بگذاریم. S

← برای write باید قفل Exclusive بگذاریم. X

⑤ optimistic concurrency

با حالت خوش بینانه به مسئله نگاه می‌کند. نیازی به قفل نیست.

برای هر عمل read و write یک مهر زمانی می‌گذاریم.

به هر تراکنش به صورت تراکنش با مهر زمانی نگاه می‌کنیم.

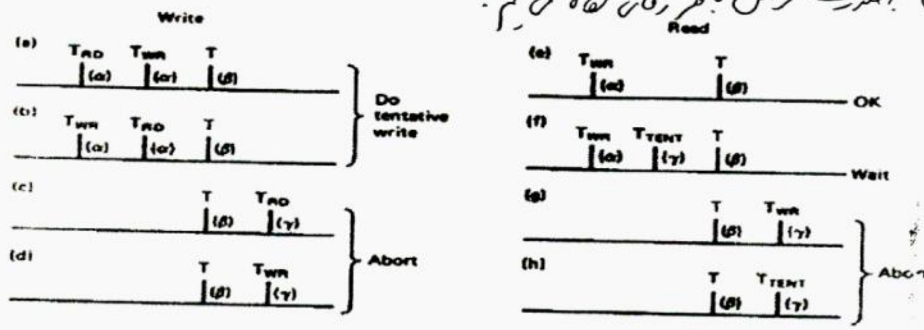


Fig. 3-22. Concurrency control using timestamps.

- بن‌بست در سیستم‌های توزیع شده:

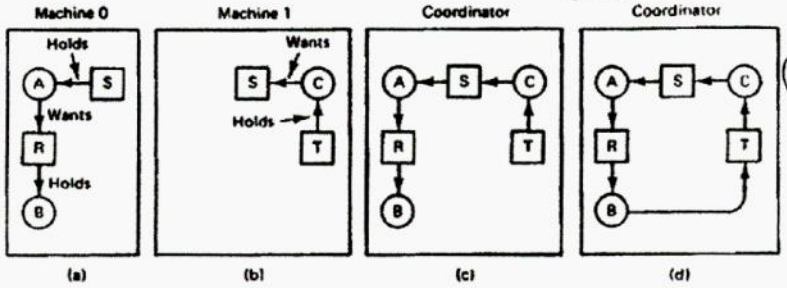
1. بن‌بست ارتباطی Communication Dead lock
2. بن‌بست منابع Resource Dead lock

* 4 استراتژی برای بن‌بست *

1. الگوریتم شترمرغ (نادیده گرفتن همه چیز، فواصل زمانی کم هستند)
2. شناسایی و رفع بن‌بست
3. ~~اجتناب~~ جلوگیری از بن‌بست
4. ~~حذف یا پیش گرفتن یکی از شرایط رخ دادن بن‌بست~~ 4) اجتناب از بن‌بست (امکان پذیر نیست)

A-2 شناسایی و کشف کردن و رفع آن (بن بست) Detection

① متمرکز کردن : centeralize



فقط با Coordinator کار می کند

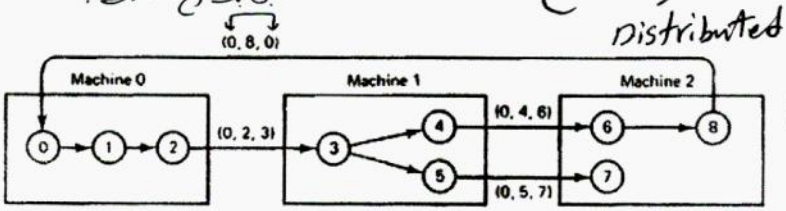
Fig. 3-23. (a) Initial resource graph for machine 0. (b) Initial resource graph for machine 1. (c) The coordinator's view of the world. (d) The situation after the delayed message.

تمام نودها در Coordinator وجود دارند. این Coordinator همیشه باید update باشد.

اگر در قسمت (d) پروسه 13 بخواهد منبع T را آزاد کند حلقه یا بن بست به وجود می آید.

باید پیام ها در یک زمان نباشند تا تداخل به وجود نیاید که بن بست نشود. (False Deadlock)

② شناسایی بن بست با الگوریتم های توزیع شده : Distributed



فقط با پیام کار می کند

Fig. 3-24. The Chandy-Misra-Haas distributed deadlock detection algorithm.

پیام ⇒ [بلاک , ارسال کننده , دریافت کننده]

1. کدام پروسه بلاک شده است.
2. کدام پروسه پیام را ارسال کرده است.
3. برای کدام پروسه ارسال شده است.

[0,0,1] → [0,1,2] → .... [0,8,0] بن بست

[2,2,3] → .... [2,1,2] بن بست

algorithm is called **wait-die**.

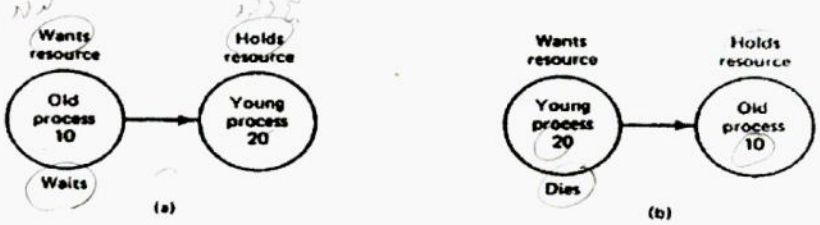


Fig. 3-25. The wait-die deadlock prevention algorithm.

Fig. 3-26. The wound-wait deadlock prevention algorithm.

4

Processes and Processors in Distributed Systems

- پردازنده‌ها در سیستم‌های توزیع شده

(1) ایستگاه کاری : workstation

هر کس پشت کامپیوتر خودش است. برخی از کامپیوترها Busy هستند و برخی idle (مشغول و بیکار)

و به دو گروه : 1. دیسکی disk و 2. dishless یعنی با دیسک و بدون دیسک تقسیم می‌شوند.

(A) بدون دیسک یا : disk less

مزیت:

1. هزینه پایین است. چون نیازی به دیسک درایوها نداریم.
2. امنیت بالا است. چون کسی نمی‌تواند دیسک روی کامپیوتر بگذارد.
3. آلودگی صوتی ندارد چون نیازی به فن ندارد.
4. واقعیت از بابت نرم افزار و نرم افزار همتراست.
5. Bachup تهیه ساده‌تر است.
6. تعمیر و نگهداری کمتر است.

عیب: 1. اگر File server از کار بیفتد دسترسی به وجود نمی‌آید.

(B) با دیسک : disk ← شکل ثبت صفحه

به 4 روش می‌باشد:

1. اگر client می‌خواهد با یک فایل کار کند هیچ کس قرار نیست با آن کار کند.
2. فایل‌های exe. را به دلیل ماهیت‌شان می‌توانیم روی تمام ایستگاه به دلیل اینکه تغییر نمی‌کند نصب کنیم سرعت بالای رود
3. روی ایستگاه‌ها کش بگذاریم و اطلاعات را روی کش بگذاریم. caching به صورت محلی است.
4. یعنی ایستگاه کاری یک ایستگاه کامل است. یعنی همه چیز داشته باشد.

③ استخر پردازنده‌ها * processors pool

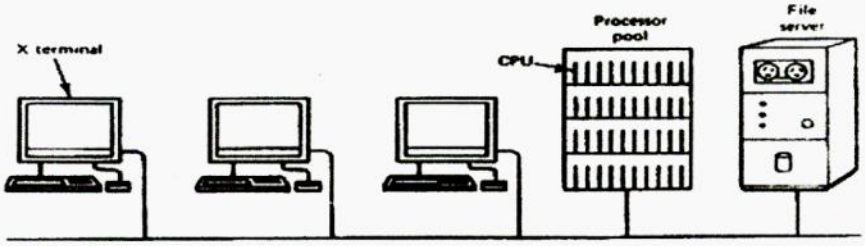


Fig. 4-13. A system based on the processor pool model.

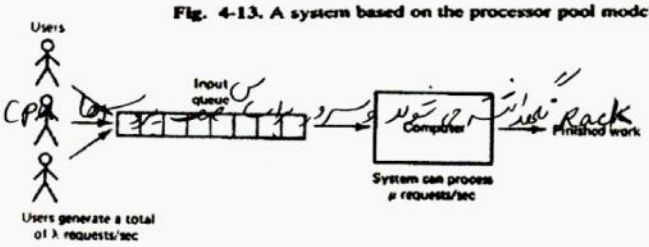


Fig. 4-14. A basic queueing system.

λ request/sec

λ ← تعداد درخواست‌ها در ثانیه برای استفاده از CPU

μ ← تعداد پردازنده‌ای که سرور می‌تواند پاسخگو باشد.

اگر $\lambda > \mu$ باشد یعنی درخواست‌ها از CPU‌ها بیشتر است که در این صورت صف به وجود می‌آید.

و درخواست‌هایی که در این درخواست‌ها منتظر شدند CPU آزاد شود و کارشان را انجام دهند.

$$\text{میانگین زمان پاسخگویی} = \frac{1}{\mu - \lambda}$$

اگر μ را به عنوان تعداد CPU‌ها و 40 را به عنوان تعداد درخواست‌ها در نظر بگیریم آنگاه:

$$\text{میانگین زمان پاسخگویی} = \frac{1}{60 - 40} = 1/20$$

اگر $\lambda > \mu$ مشکل پیش نخواهد آمد.

اگر $\frac{1}{60}$ را به عنوان تعداد CPU ها و صفر را به عنوان تعداد درخواستها در نظر بگیریم آنگاه:

زمان اجرا کمتر از $\frac{1}{60}$ برای هر برنامه می شود. میانگین $= \frac{1}{\mu - \lambda} = \frac{1}{60 - 0} = \frac{1}{60}$

نکته: در این روش نسبت به روش قبل عدالت برقرار است.

مثال: 2 عدد CPU داریم.

$1\ CPU_1 \xrightarrow{\text{سرعت}} 1000\ MIPS$

$10\ CPU_2 \longrightarrow 100\ MIPS$ ✓

کارها با سرعت متوسط سرویس می گیرند.

منطقی تر است اگر یکی از CPU ها از کار نیفتد تنبل وجود دارند.

Design Issues for Processors Allocation Algorithm

الگوریتم های تخصیص پردازشگر

1. Deterministic بر علیه Heuristic
2. Centeralize بر علیه Distributed
3. optimal بر علیه Sub optimal
4. Local بر علیه Global
5. Sender بر علیه Receiver

1. Deterministic برعلیه Heuristic

← Desterministic : از قبل همه چیز را جمع بزنیم cpu را می‌دانیم و با یک دانش کامل تصمیم می‌گیریم.

← Heuristic : هیچ چیز را از قبل نمی‌دانیم.

2. Centeralize برعلیه Distributed

← centeralize : فقط یک cpu داریم که همه برای استفاده از آن در صف می‌روند.

← Distributed : چند cpu داریم که می‌توانیم با آن کارها را بصورت موازی با هم انجام دهیم.

3. optimal برعلیه sub optimal

← optimal : الگوریتم‌های این گروه بهترین حالت را پیدا می‌کنند. پیچیدگی سخت‌تر – زمان بیشتر

← sub optimal : الگوریتم‌های این حالت بهترین حالت را پیدا نمی‌کنند. پیچیدگی کمتر – زمان کمتر

4. Local برعلیه Global

← Local : cpu محلی خودش را به پردازش می‌دهد. روی cpu خودش کار را انجام می‌دهد.

← Global : یکی cpu را به پردازش می‌دهد. cpu بیکار را پیدا می‌کند.

5. Sender برعلیه receiver:

← Sender : client خودش درخواست می‌دهد که دیگر توانایی انجام کار ندارم دیگر کار نفرستید.

← receiver : client خودش درخواست انجام کار می‌دهد.

6

Distributed Shared Memory

- کپی کردن داده Data Replication :

از یک فایل کپی بگیریم و روی سرورهای مختلف آن را کپی کنیم.

مزیت های کپی کردن :

(1) Reliability قابل اعتمادی :

(A) Data corruption : وقتی قسمتی از یک فایل از بین برود.

(B) Data Distraction : وقتی یک فایل به طور کامل از بین برود.

(2) Performance کارایی :

وقتی یک پروسس فایل F را از یک سرور می خواند نزدیک ترین فایل ها به جایی که می خوانند خوانده شوند.

نزدیکترین فایل ها به جایی که دارند صدا زده می شوند. ← باعث می شود سرعت بالا رود. و عملکرد بهتر شود.

عیب های کپی کردن :

(1) Scaling مقیاس پذیری : هدف این است که عملکرد را بهترین وقتی یک فایل را به صورت متعدد روی سرورهای مختلف کپه کنیم، مقیاس را بزرگ کرده ایم.

(2) Band width پهنای باند : برای کام updateها مثلاً اگر یک فایل تغییر کند باید روی همه سرورهایی که این فایل کپی شده است تغییر کند که به این صورت به پهنای باند زیادی نیاز داریم.

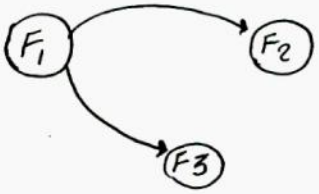
مثال : پروسس P در N Time/second به یک فایل جهت r و w (write, read) همین فایل در M(Time/second

درحال update شدن است. r

اگر N > M باشد وقتی دفعه ای کمتر است از update یعنی تا خیلی از updateها را نمی بیند.

فقط قبل از کپی فایل کپی شود. چون از پهنای باند هیچ استفاده ای نمی شود و updateها به هیچ دردی نمی خورند.

3. consistency انجام با همگامی یا همزمانی



اگر یک فایل تغییر کرد باید کلیه کپی های آن تغییر کند.

اگر یک پروسس مانند F_1 فایل خود را تغییر داد پس فایل های 2 و 3 که کپی های فایل 1 هستند نیز باید تغییر کند.

پس اگر یکپارچگی نباشد اطلاعات نادرست را همه می خوانند که فایده ای ندارد.

- مدل یکپارچه سازی:

قراردادی است بین پروسس و بین Data store که می گوید پروسس ها قول می دهند که یکسری قوانین را رعایت کنند و در مقابل رعایت کردن این قوانین Data store درست کار کند تغییرها update ها به روز خواهند بود.

- انواع مدلهای consistency جهت یکپارچه کردن:

1. Data centric داده محور: یعنی اینکه یک پروسس عمل خواندن و نوشتن را از فایل ها چگونه انجام دهد

2. client centric کاربرمحور:

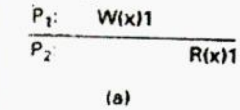
+ Data centric شامل موارد زیر می باشد:

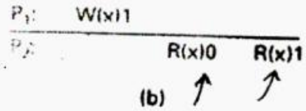
1. strict 2. sequential 3. causal 4. FIFO 5. weak 6. Release 7. Entry

* + client centric شامل موارد زیر می باشد:

1. Monotonic Reads 2. monotonic writes

3. Read your writes 4. writes follow Read





1. strict consistency: یکپارچه سازی سخت گیرانه:

هر read که روی متغیری با اسم x صورت بگیرد باید نتیجه آخرین w را برگرداند.

w(x)a یعنی متغیر x ← w (نوشته شده) و مقدار آن a است.

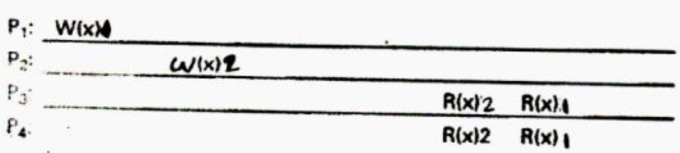
r(x)b یعنی متغیر x ← r (خوانده شده) و مقدار آن b است.

شکل (a) ⇐ پروسس P_1 ← w کرده x را و مقدار آن را یک داده.
پروسس P_2 ← r کرده x را و مقدار آن b را گرفته است.

شکل (b) ⇐ پروسس P_1 ← x را w کرده و مقدار یک را به آن داده.
پروسس P_2 ← x را r کرده و یک مقدار دیگر را خوانده است. پس طبق قانون strict باید اگر متغیری عوض شد باید همه جا عوض شود این قانون را نقض کرده.

2. sequential consistency: یکپارچه سازی پی در پی یا ترتیبی

وقتی پروسس‌ها دارند به صورت همزمان اجرا شوند هر تداخل w و r باید توسط تمام پردازه‌های دیگر هم به همان صورت دیده شود. یعنی در x اول a بعد b ⇐ تداخل بین w و r



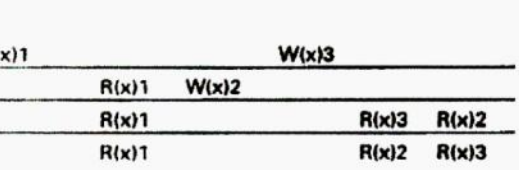
نکته: در اینجا زمان هیچ نقشی ندارد.

هر کدام روی data store خودشان کار می‌کنند داده‌ها جدا هستند.

نکته: توسط تمام پروسس‌ها تداخل بین w و r باید مثل هم باشد.

اگر جای یک و دو عوض شود اشتباه است. زمان مهم نیست. ترتیب r و w مهم است.

Kazem.faridi@iaubaft.ac.ir



writeهایی که بطور بالقوه با هم واسته اند باید توسط همه پروسس ها به همان صورت دیده شوند.

writeهایی که با هم واسته نیستند یعنی هیچ ارتباطی ندارند می توانند توسط پروسس های مختلف به هر ترتیبی دیده شوند.

یادآوری: بین P_1 و P_2 وابستگی وجود دارد.
بین P_3 و P_4 وابستگی وجود ندارد.

یعنی ۱ و ۲ وابسته اند ولی بین 2 و 3 نیست که هیچ مشکلی هم پیش نمی آید.

نکته: اگر یک خواندن بعد از یک نوشتن بیاید آنها با هم وابسته اند.

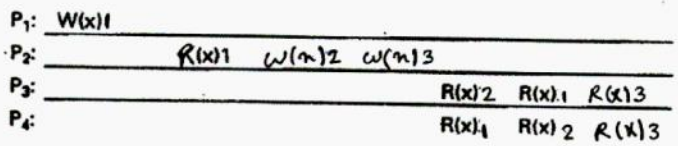
4- FIFO consistency:

writeهایی که توسط یک پروسس انجام شوند باید توسط پروسس های دیگر هم به همان صورت دیده شوند.

ولی writeهایی که توسط پروسس های مختلف صورت پذیرد می تواند به صورت مختلف هم دیده شود.

شکل ← ترتیب بین 2 و 3 در پروسس 3 و 4 رعایت شده است.

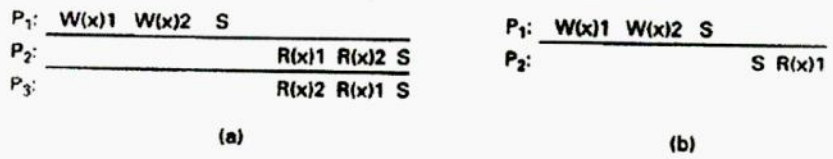
چون ۱ در یک پروسس دیگر لحاظ شده است نمی تواند در هر کجا قرار گیرد.



5. Weak consistency: یکپارچه سازی ضعیف:

1. دسترسی به متغیر محافظت شده باید به صورت ترتیبی باشد.
2. هیچ گونه عملیاتی روی محافظ شده اجازه ندارد انجام شود مگر اینکه تمام نوشتن های قبلی همه جا بطور کامل update شوند.
3. هیچ گونه read و write روی یک محافظ شده اجازه ندارد انجام شود مگر اینکه تمام عملیات قبلی روی محافظ شده انجام شده باشد.

عیب: وقتی یک S (محافظ شده) را انجام می دهیم نمی دانیم مربوط به read است یا write. شخص نیست کدام عمل را می خواهد کند



در شکل (a) ⇐ آرایه P_2 بعد از S خواهد عملیاتی را انجام دهد باید R(x)2 باشد چون $W(x)_2$ P_1 آخر است.

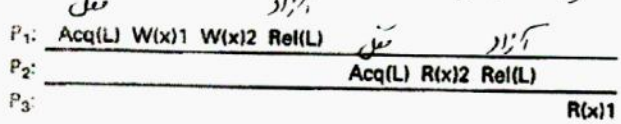
در شکل (b) ⇐ قانون weak را نقض کند ولی از نظر یکپارچه سازی است.

6. Release consistency: یکپارچه سازی آزاد کردن

بین r و w فرق قائل نمی شود: ۳ قانون داریم:

1. قبل از اینکه هر گونه r یا w روی داده صورت گیرد تمام درخواست های قبلی به طور کامل انجام شده باشند یعنی تا نمی توانید متغیری را بگیرید و روی آن کار کنید مگر اینکه عمل درخواست قبلی آن تمام شود.
2. قبل از اینکه یک آزادسازی انجام شود تمام r و w های قبلی باید روی آن متغیر بطور کامل اجرا شوند.
3. دسترسی به متغیر S (محافظ شده) باید بصورت FIFO باشد.

نکته: P_2 باید صبر کند تا P_1 عملیات r یا w را آزاد کند، در اینجا زمان مهم نیست.



عیب: شخص نمی کند کدام متغیر را قفل کرده است.

7. Entry consistency: یکپارچگی ورودی

P1: Acq(Lx) W(x)1 Acq(Ly) W(y)2 Rel(Lx) Rel(Ly)
P2: Acq(Lx) R(x)1 R(y)nill
P3: Acq(Ly) R(y)2

R(y)nill ← چون قفل نکردیم ممکن است هر جوابی بدهد.

3 قانون داریم:

1. دسترسی به متغیر S قابل پیاده سازی نیست تا زمانی که تمام update ها روی آن متغیر ها گرفته باشند.

2. دسترسی انحصاری به متغیر S توسط یک پروسس زمانی امکان پذیر است که هیچ پروسس دیگری متغیر S را در اختیار نداشته باشد.

3. زمانی که دسترسی انحصاری توسط S انجام شده باشد پروسس هایی که در حالت غیرانحصاری هستند می توانند به متغیر S دسترسی پیدا کنند.

client consistency کاربرمحور

1. Monotonic Reads:

اگر پروسس ای مقدار X را بخواند دفعه ی بعدی می باید مقدار X جدیدتری را برگرداند.

تضمین می کند اگر در زمان T متغیری را بخواند در زمان بعدی متغیر قبلی را برنمی گرداند.

شکل اول: در مکان L1 عمل ws روی X1

در مکان L2 عمل ws روی X2

تا زمانی که L2 - X1 منتقل نشوند نمی توانند X2 را بخوانند.

مکان L1	WS(X1)	R(X1)
مکان L2	WS(X1,X2)	R(X2)

در شکل روبرو اشتباه است چون در L2 تغییر X نوشته نشده

L1	WS(X1)		R(X1)
L2	WS(X2)	R(X2)	WS(X1, X2

مثال: اگر در اهواز رمز ایمیل خود را عوض کردیم هنگامی که در تهران می خواهیم رمز ایمیل خود را عوض کنیم باید رمز قبلی را دا...

2. Monotonic write :

عمل w که روی یک متغیر توسط یک پردازه باید حتماً قبل از هر عمل w دیگری روی همان متغیر توسط همان پردازه کامل شود.

هر w که می خواهد انجام شود باید w قبلی انجام شوند.

L_1 $w(x_1)$

L_2 $w(x_1)$ $w(x_2)$

3. Read your writes :

نتیجه عملیات w روی یک داده توسط یک پردازه باید همیشه توسط همان پردازه روی همان داده و به همان ترتیب خوانده شود.

L_1 $w(x_1)$

L_2 $ws(x_1, x_2)$ $R(x_2)$ ✓

X

L_1 $w(x_1)$

L_2 $ws(x_2)$ $R(x_2)$

4. writes follow Read (w هایی که به دنبال R می آیند)

عملیات w که توسط یک پردازه روی یک داده که به دنبال یک Read توسط همان پردازه روی همان داده آمده است باید روی آفرین نسخه X اعمال شود.

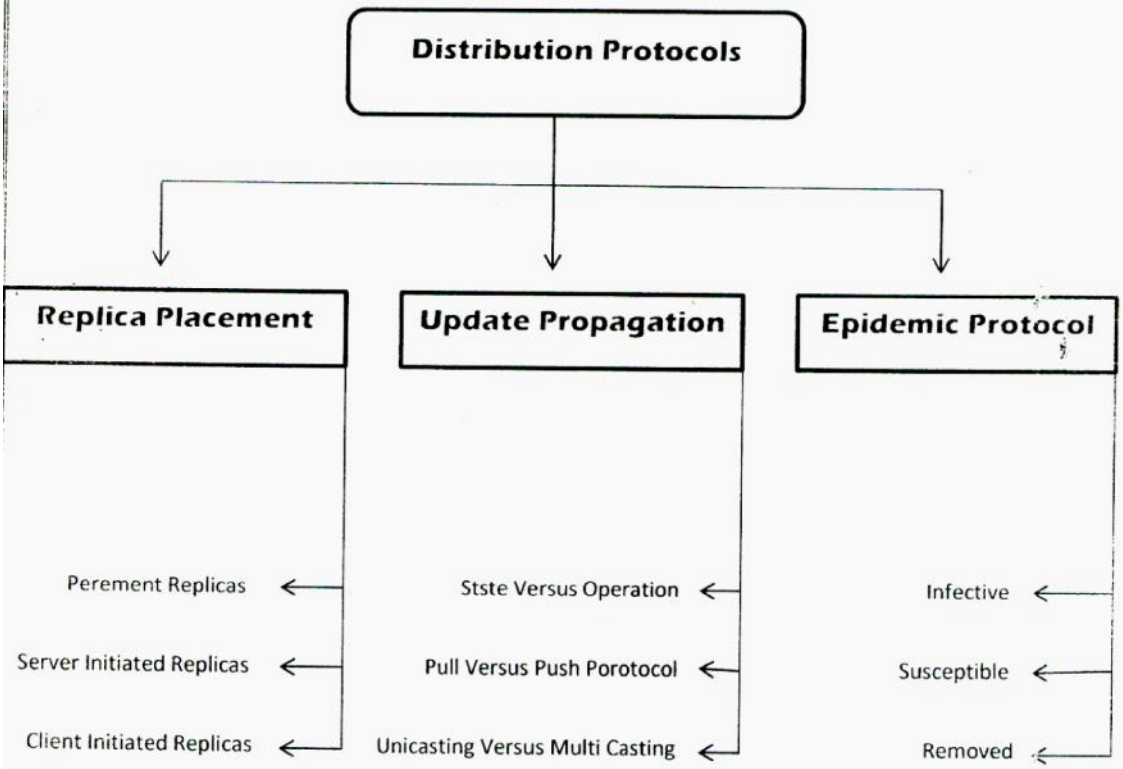
L_1 $ws(x_1)$ $R(x_1)$

L_2 $ws(x_1, x_2)$ $w(x_2)$ ✓ → چون x_1 به پایین هم تشکیل شده است

L_1 $ws(x_1)$ $R(x_1)$

L_2 $ws(x_2)$ $w(x_2)$ X → x_1 آفرین نسخه w در L_1 است که باید به L_2 منتقل شود

نمای کلی پروتکل های توزیع



- Distributionprotocols پروتکل های توزیع

(1) Replica placement : update propagation (2) Epidemic protocol (3)

قرار دادن این کپی فایل ها که و کجا این کپی ها باید قرار داده شوند ؟

بکاو چه زمان کپی ها باید قرار داده شوند ؟

به 3 صورت انجام می گیرد :

(A) Permanent Replicas : permanent

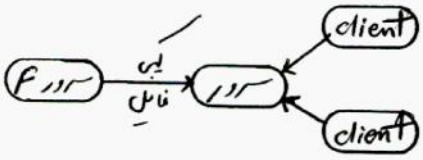
کپی فایل ها روی سرورها به صورت دائمی نگه داشته می شوند .

(B) Server Initiated Replicas : سرور محور

کپی فایل ها را روی سرور نزدیکتر به client هایی که دارند فایل را درخواست می کنند نصب می کند .

اگر درخواست ها به سرور F زیاد شود سرور F کپی فایل را به نزدیکترین سرور به client ها ارسال می کند .

و اگر درخواست ها کم شوند دیگر کپی فایل روی آن سرور را حذف می کند .



(C) client Initiated Replicas : کاربر محور

client تصمیم می گیرد چه زمان فایل ها کپی شوند .

یک کش برای کلاینت تعریف می شود و داده ها روی آن کش قرار می گیرند . اگر داده در کش بود که داده را از آن می خواند و اگر در کش نبود در نزدیکترین فایل کپی شده دنبال داده می گردد .

(2) update propagation :

- روش‌های انتشار update :

(A) state versus operation :

A-1 فقط یک پیغام به Data center بفرستیم که اطلاعاتش دیگر معتبر نیست .
حجم پیغام کم و پهنای باند کوچک است و هیچ ترافیک هم ایجاد نمی‌شود .
اطلاعات update شده را می‌فرستد بعد سرور آن‌ها را می‌خواند.
در read/update آن read کم باشد خوب است یعنی update زیاد است .

A-2 هر بار که داده‌ها تغییر کردند یک کپی برای همه بفرستیم (یک update عوض نمی‌برد)
آن read زیادتر باشد بصرفه است چون آن update زیادتر باشد همه update data را نمی‌خواند و این خوب نیست

A-3 فقط طی یک پیغام، update هایی که باید توسط آن سیم اجرا شود اعلام می‌شود .

(B) pull versus push protocol :

B-1 Push Based approach :
خود سرور در فواصل زمانی مشخص اطلاعات را به بقیه می‌فرستد.
این روش سربار محدود است . محدودیت ارسال update ها است . آن read زیاد باشد بصرفه است.

B-2 Pull Based approach :
سرورها درخواست update می‌کنند .
محدودیت دریافت update ها است . در این روش آن update ها زیاد باشد بصرفه است .

C) unicasting versus multi casting

C-1 unicasting :

سرور ها n پیام به n سرور، update های خودش را یک به یک ارسال می کند.

C-2 multi casting :

سرور یک پیام می فرستد و انتظار دارد سرورها خودشان را update کنند.

3) Epidemic protocol : الگوریتم اپیدمیک

سعی می کند کاری کند که update ها به اکثر سرورها برسد

(A) infective : به سروری می گویند که به update ها آلوده شده اند و آن را دارند.

(B) susceptible : به سروری می گویند که به update ها آلوده نشده اند و آماده آلوده شدن هستند.

(C) removed : به سروری می گویند که به ویروس آلوده شده اند و تمایلی به آلوده کردن دیگر سرورها ندارند.

توضیح الگوریتم اپیدمیک

1) سروری که فایل update شده را دارد سروری که فایل update شده را ندارد به صورت تصادفی پیدا می کند و فایل update را برای آن می فرستد. push

چون هدف از تصادفی است احتمال دارد مستقیم آلوده نشود.

(2) سرورهایی که فایل update ندارند تقاضای دریافت فایل update را از سرورهایی که فایل update را دارند می کنند. Pull

هرکس که می داند آلوده نیست و فایل update را ندارد درخواست می دهد.

- احتمال update نشدن فایل ها کم است.

(3) به صورت Hybrid است (هم pull و هم push)

روش مناسبی نیست - هر دو سرور تقاضای بفرستند برای update

قرارداد هایی یکپارچه سازی پروتکل
Consiste protocol

1. Remote write protocol : نوشتن از راه دور

هم برای w و هم برای r به سرور مراجعه می کند. تعداد پیام ها زیاد است.

ولی برای read هر client یک کپی از فایل دارد یعنی r ها به صورت محلی صورت می گیرند و w ها از سرور درخواست می

آن سرور از کار بیفتد Back up داریم.

2. Local write protocol :

به صورت محلی است. تعداد پیام ها کم است. تمرکز این روش روی write ها است

client جهت برای w و جهت برای r درخواست را به سرور می فرستد پس درخواست به سرور اصلی فرستاده می شود

نکته: client ابتدا درخواست را به سرور خودش می فرستد تا update ها را بگیرد از سرور اصلی r فقط یک پیام رفت و برگشت دارد.

دکتر کروپی

نمونه سوالات گذشته سیستم های توزیع شده

۱- اهداف توزیع شده در مقابل متمرکز ؟

۲- مزایا و معایب توزیع شده ؟

۳- Bus – Based multi Computers ؟

۴- سیستم عامل های شبکه و شبکه های LAN و Remote Login Machine از چه نوعی هستند ؟

۵- Bus – Based Processors ؟

۶- فاکتورهای طراحی در سیستم های توزیع شده ؟

۷- پروتکل OSI و لایه های آن ؟

۸- شبکه های ATM ؟

۹- اجزای هر سلول را نام ببرید ؟ هر سلول چند بایت است ؟

۱۰- خصوصیات الگوریتم های توزیع شده ؟

۱۱- ساعت منطقی و ساعت فیزیکی را تعریف کنید ؟

۱۲- انواع روش های منطقه بحرانی . عیب آنها ؟

۱۳- الگوریتم های انتخاب Coordinator ؟

۱۴- نحوه پیاده سازی تراکنش ها ؟

۱۵- استراتژی های بن بست را توضیح دهید ؟

۱۶- Data Replication را توضیح و معایب و مزایا را نام ببرید ؟

۱۷- انواع Consistency را نام ببرید و توضیح دهید ؟

۱۸- پروتکل های توزیع شده را نام ببرید ؟ Distributed Protocol

۱۹- قراردادهای یکپارچه سازی ؟

۲۰- System Models را نام ببرید ؟

۲۱- الگوریتم های تخصیص پردازشگر ؟

۲۲- انواع (Switch Multi (Processors & Computers ؟

۲۳- ATM Switch را توضیح دهید ؟ سوئیچ ۴ پورت صف قبل از ورودی