جلسه ششم

RPC : می خواهد تابعی از کلاینت را روی سرور اجرا کند کاربر نباید متوجه شود که تابع روی کلاینت اجرا می شود یا روی سرور یعنی محلی است یا Remot

نکته: Stub همان M.W است

مشکلات RPC :

1- فراخوانی : در فراخوانی با مقدار مشکلی وجود ندارد، مقدار را می فرستد و تغییری نمی کند(Call by Value) اما در فراخوانی با مرجع با مشکل مواجه می شویم چون آدرس را می فرستد مثلا آدرس 100 ، و این آدرس در کلاینت و سرور یکی نیست به همین دلیل به جای فراخوانی با مرجع از روش Call by copy & Restore استفاده می کنیم.

نکته: قسمت Client Stub مقادیر را در یک message قرار می دهد(مثلا مقادیر یک آرایه را) و به Server می فرستد و سرور اعمالی را روی آن انجام می دهد و در یک message به کلاینت بر می گرداند.

2- ساختار های پیچیده: مثل درخت و گراف، باید نوع ساختمان و Data با هم ارسال شوند تا ساختار با استفاده از pointer های محلی در Sever Stub ساخته شود.

3- نمایش داده ها: باید نمایش واحدی به کار ببریم نمایش به صورت Big endian یا Little endian باشد در این صورت برای کامپیوتر های Sun و Intel که مشکل تفاوت نمایش وجود داشت مشکل از بین خواهد رفت اما برای دو ماشین از یک نوع مثلا Intel برای این که تبدیل های اضافی انجام ندهیم نوع ماشین ها را هم ارسال می کنیم.

ارسال Data Type :

- به صورت صریح
- به صورت ضمنی

روش دیگر به این صورت می باشد که نوع داده ارسال شود.

فراخوانی محلی فقط یک بار اجرا می شود اما فراخوانی را ه دور به تعداد مختلف می تواند اجرا شود.

- صفر بار: چون ممکن است سرور Crash کند و یا داده از بین برود.
- یک بار: در صورتی که همه چیز درست انجام شود.
- یک بار بیشتر: به خاطر تاخیر در شبکه

نکته: در RPC داشتن دقیقا یک اجرا برای فراخوانی راه دور سخت می باشد.

پردازه ها دو نوع هستند.

- ممکن است چندین بار اجرا شوند مثل خواندن تاریخ سیستم
- فقط یک بار اجرا می شود.

کارائی:

کارائی فراخوانی محلی بهتر است. در RPC چون تاخیر زیاد داریم کارائی پایین می آید.

امنیت:

زبان های مورد استفاده:

C، C++، java : Server Stub و Client Stub را پشتیبانی نمی کنند. و بایستی این دو را به وجود آوریم با استفاده از RPC Compiler

نکته: اهداف RPC همان اهداف شبکه است و می خواهد که توزیع شده باشد و همه مثل یک سیستم واحد باشد یعنی بحث مخفی سازی وجود دارد.

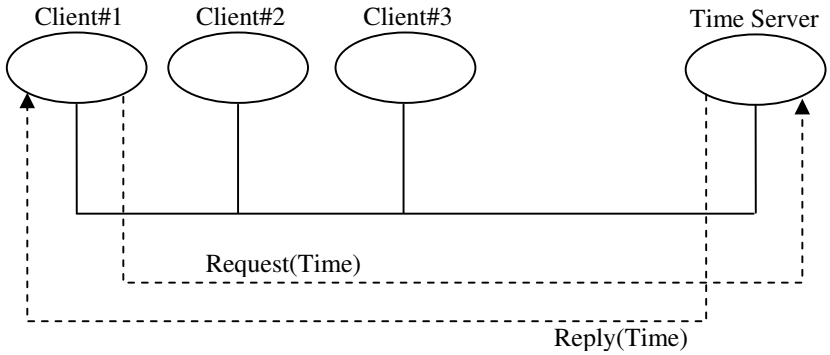
Clock :

باید ساعت تمام کامپیوتر های سیستم یکی باشد برای این کار دو روش وجود دارد.

- ساعت فیزیکی
- ساعت منطقی

ساعت فیزیکی:

ساعت واحد جهانی(Universal Coordinated Time:UTC)



- در روش اول Time Server غیر فعال یا Passive است منتظر در خواست زمان می ماند و در صورت گرفتن درخواست جواب می دهد(الگوریتم Cristian)
- در روش دوم Time Server فعال یا Active است به طوریکه Time Server زمان همه Clientها را گرفته و Over Shot و Under Shot (Clock های پرت) را حذف کرده و یک تابع بر روی این Clock ها به کار برده(به عنوان مثال میانگین) سپس نتیجه این تابع را بر روی تمام Clientها ارسال می کند.

ساعت منطقی:

روش Lamport با نام Happens before(قبل از اتفاق افتادن)

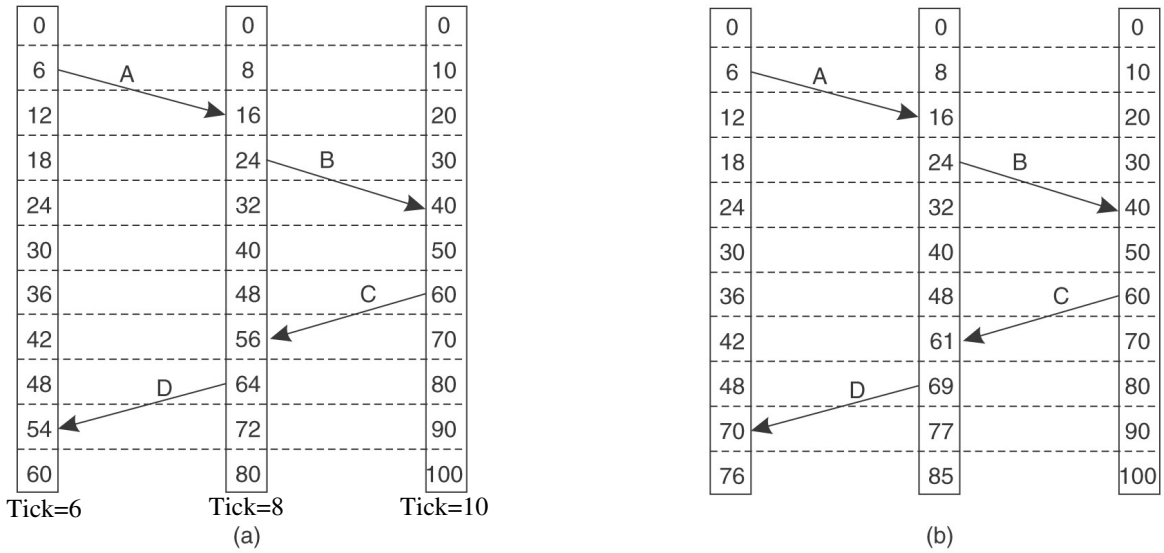
M1 همراه با Package زمان خودش را نیز ارسال می کند و چون Send قبل از Resive انجام می شود بنابراین M2 می فهمد که زمان دارای مشکل است و با توجه به زمان M1 و زمان لازم برای Resive زمان خود را تنظیم می کند.

M1	M2
Send	Resive
t=5	t=2

$\rightarrow t = 6$

فرض می کنیم سه پردازه روی سه ماشین وجود داشته باشد.

زمان ها را بررسی می کنیم باید زمان ها را بررسی می کنییم باید زمان Send زودتر از زمان Resive باشد اگر این طور نبود یعنی T(R)<T(S) در این صورت زمان Send را یک واحد اضافه کرده و در Resive قرار می دهیم و زمان های بعد از آن را با توجه به زمان tick اضافه می کنیم(صفحه بعد این موضوع بررسی شده است).



در شکل اول(شکل a) A و B درست است چون زمان Send زودتر از Resive است اما C و D درست نیست و باید تنظیم شوند که در شکل دوم(شکل b) این تنظیم انجام شده است

اولی: زمان Send + 1

بقیه: زمان بدست آمده + زمان tick

اگر a و b دو پردازه باشند

1- اگر a و b روی یک ماشین باشند و a قبل از b انجام شود قاعده Happen-before صادق است

2- اگر a ، Send باشد و b ، Resive و a و b روی دو ماشین مجزا باشندClock عمل Send باید کوچکتر از Clock عمل Resive باشد به عبارتی C(a)<C(b)

3- اگر a و b دو پردازه مجزا و مستقل باشند و هیچ ارتباطی به هم نداشته باشند و C(a) ≠ C(b) یعنی کاری باهم ندارند و لزومی ندارد که Clock آنها یکسان شود به عبارتی روی ساعت هم تاثیر نمی گذارند

نکته: برای کلیه رخدادهای a و b باید c(a)≠c(b) برقرار باشد.

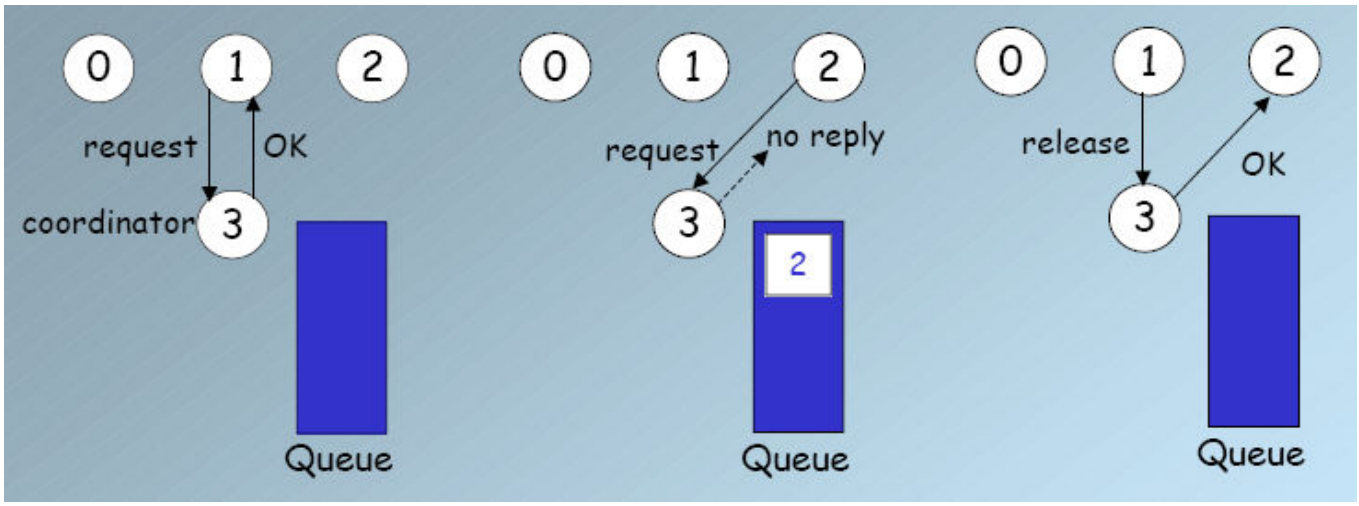
نکته: هدف از ایجاد Clock یکسان یا هماهنگی بین پردازه های وابسته مرتب سازی massage ها می باشد. Massage ها بر اساس مهر زمانی مرتب می شوند.

انحصار متقابل در سیستم های توزیع شده(Mutual Exclusion)

در واقع منظور این است که اگر در یک زمان تعدادی پردازه یک منبع را درخواست کنند به کدام اختصاص داده شود و مدیریت چگونه انجام شود . برای انجام این کار پردازه ای را به نام Cordinator یا هماهنگ کننده خواهیم داشت که به دو شکل زیر چگونگی اختصاص منابع به پردازه های درخواست کننده را انجام می دهد.

روش های بر آورده سازی انحصار متقابل:

1- الگوریتم متمرکز: در این روش هر پردازه های جهت دسترسی به منبع قابل اشتراک درخواستش را به هماهنگ کننده می دهد و هماهنگ کننده در صورتی که پردازه ای در ناحیه بحرانی نباشد اجازه دسترسی می دهد(Ok می دهد) در غیر این صورت پردازه را در لیست پردازه های منتظر آن منبع قرا می دهد. شکل زیر این موضوع را به وضوح نشان می دهد.



(a) : درخواست p1 برای ناحیه بحرانی پاسخ داده می شود.

(b) : در خواست p2 برای همان ناحیه پاسخ داده نمی شود.

(c) : پس از رها کردن ناحیه توسط p1 به p2 پاسخ مثبت داده می شود.

اشکال این روش:

اگر Cordinator خراب شود سیستم خراب می شود به عبارت بهتر درخواست کنندگان به وجود یا عدم وجودcoordinator در صورت عدم پاسخ نمی توانند پی ببرند. مشکل دوم این است که گلوگاه ایجاد می شود(منظور پردازه Cordinait شلوغ شده و تنگنا ایجاد میشود).

مزایای این روش:

- عادلانه عمل میکند (starvation=0) زیرا fifo کار می کند.
- پیاده سازی ساده ای دارد و فقط به سه پیغام (release,request,grant) برای استفاده از یک ناحیه بحرانی نیاز دارد.

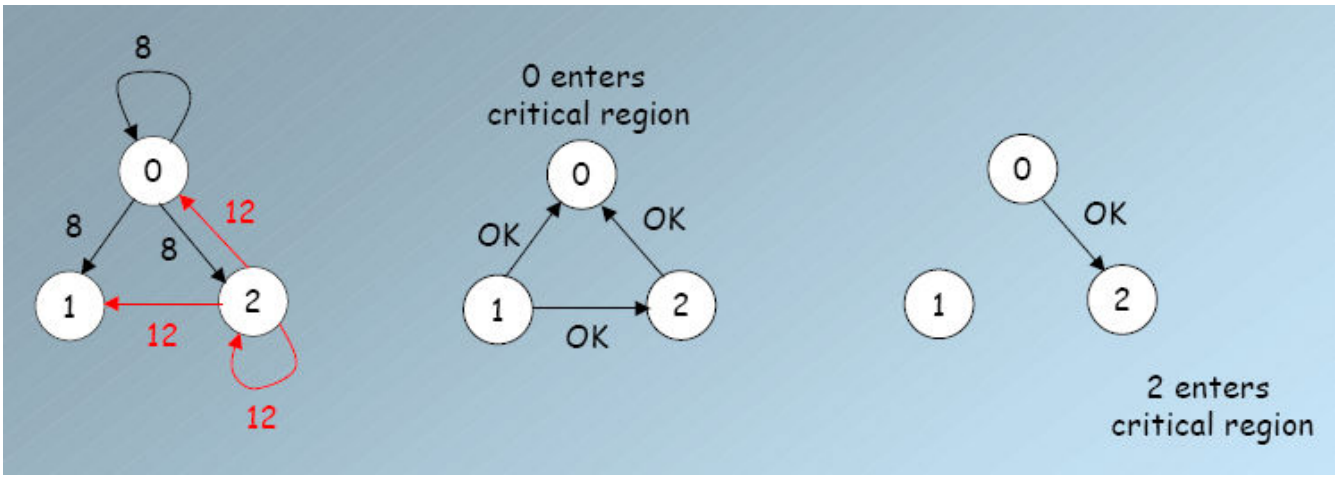
2- الگوریتم توزیعی: در این روش دیگر Cordinator وجود ندارد و هر پردازه ای بخواهد وارد ناحیه بحرانی شود massage درخواستش برای آن وسیله را برای همه broadcast می کند که این massage حاوی اطلاعات زیر است

- اسم ناحیه بحرانی
- شماره پردازه
- زمان فعلی

هرگاه پردازه ای پیغامی از دیگر پردازه ها دریافت می کند، بحد وضعیت خود یکی از کارهای زیر را انجام می دهد:

- داخل ناحیه بحرانی نیست و نمی خواهد وارد شود؛ پاسخ OK پس می فرستد.
- داخل ناحیه بحرانی است ؛ فلذا پاسخی نداده و آن را در صف قرار می دهد و پس از خروج منبع را آزاد می کند.
- داخل ناحیه نیست ولی قصد ورود دارد؛ زمان پیغام ارسالی خود را با زمان پیغام واصله مقایسه می کند و آنکه زودتر است برنده می شود.

شکل زیر این موضوع را به تصویر می کشد.



(a) : دو پردازه در یک لحظه درخواست ورود به یک ناحیه بحرانی را دارند(پردازه شماره 0 و 2).

(b) : با توجه به این که پردازه شماره 0 ، Timestamp کمتری دارد(8) پس برنده شده و وارد ناحیه بحرانی می شود.

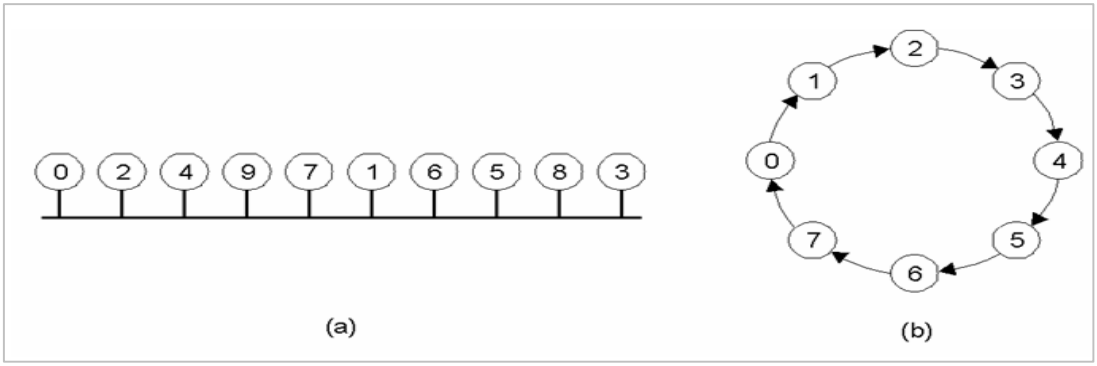
(c) : پردازه شماره 0 پس از خروج از ناحیه بحرانی یک پیغام Ok را به پردازه شماره 2 می فرستد. حال پردازه 2 می تواند وارد ناحیه بحرانی شود.

در رابطه با الگوریتم توزیعی بیشتر بدانید

- هر پروسس درخواست کننده صبر می کند همه Ok بدهند بعد وارد می شود. وقتی هم خارج شد به آنها هم که در صف هستند Ok می دهد و صف را خالی می کند .
- این جا با n پروسس 2(n-1) پیام داریم در حالی که در متمرکز فقط سه پیام داشتیم.
- به جای یک نقطه شکست(که در متمرکز داشتیم) n نقطه شکست داریم(تا به این جا دو تا بدی نسبت به متمرکز داریم) این مشکل را به این طریق می توان حل کرد که گیرنده همیشه پاسخ(رد یا قبول) را به فرستنده بفرستد
- به جای شلوغی یک پروسس همه پروسس ها شلوغ می شوند.
- نیاز به ارتباط گروهی است که ممکن است وجود نداشته باشد
- توقف یک ماشین با انتظار تشخیص داده نمی شود
- حال این الگوریتم با این همه معایب به چه درد می خورد؟ در واقع می خواستیم نشان بدهیم که این الگوریتم هم امکان پذیر است

3- الگوریتم توکن رینگ:

فرض بر این است که پردازه ها از طریق یک Bus مثل Ethernet مرتبط هستند(شکل a)، ولی در نرم افزار در یک حلقه منطقی قرار گرفته انـد که پردازه بعدی خود را می شناسند(شکل b).



در این روش هر پردازه به پردازه دیگر(پردازه بعدی خود) یک massage (Token) می فرستد که مهمترین بخش این توکن IP و شماره پورت است و بدین ترتیب همه پردازه ها تشکیل یک حلقه را می دهند. پردازه ای که الویت بیشتری داشته باشد Token را به دست می گیرد و اگر بخواهد وارد ناحیه بحرانی بشود Token را نگه می دارد تا از ناحیه بحرانی خارج شود آنگاه رها می کند و اگر نخواهد وارد ناحیه بحرانی شود توکن را بـه پـردازه بعدی رد می کند.

نکته: هنگام شروع token به P0 داده می شود.

نکته: توپولوژی شبکه Token_Ring نیست.

اما مشکلات:

- اگر token گم شود باید مجددا ایجاد شود، بعلاوه اینکه تشخیص گم شدن بسیار مشکل است.
- اگر مشکلی برای پردازه ای پیش آید (مثلا crash کند) در چرخاندن token دور حلقه خلل وارد می شود.

> هرپردازه بایستی پیکربندی فعلی حلقه را نگهداری کند تا بداند token را به کی رد کند <.

مقایسه سه الگوریتم قبلی

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

نکته: الگوریتم های توزیعی به خرابی حساس ترند

4- الگوریتم انتخابات:

در این روش یک پردازه شروع کرده و انتخابات را برگزا ر می کند یک فریم انتخابات را بین تمام پردازه ها می چرخاند و هر پردازه ای که نیاز به ناحیه بحرانی داشه باشد آدرسش را در آن فریم یادداشت می کند و اگر نیاز نداشته باشد و یـا خـاموش باشـد ادرس آن وارد نمـی شـود این روند ادامه پیدا می کند تا به آدرس شروع کننده برسد سپس شروع کننده سرگروه را انتخاب کرده و Token را در اختیار آن قـرار مـی دهد.