

جلسه دهم

Communication یا ارتباطات در یک سیستم توزیع شده

ابتدا مفهوم کلی **Communication** و روش های مطرح در آن مورد بررسی قرار خواهد گرفت. سپس هر یک از این روش ها را به تفصیل بررسی کرده و جزئیات، مزایا و معایب آن بررسی خواهد شد. در دیدگاه کلی یکی از متداولترین روش های ارتباط در یک سیستم توزیع شده، استفاده از ارسال پیغام می باشد. اما ایرادی که بر این سیستم می توان وارد کرد، این است که در یک سیستم مبتنی بر ارسال پیغام ایجاد سیستم های بزرگ بر پایه شبکه های **on Secure** کار چندان ساده ای نیست و در واقع به دلیل امن نبودن شبکه امکان به دست آوردن امکانات اولیه در سیستم موجود نیست و چه بسا پیغام هایی که بدین سان در این شبکه ارسال می شوند گم شوند و ارتباط را نامیسر سازد.

به طور کلی در یک سیستم مبتنی بر لایه چهار مدل ارتباطی مورد بررسی قرار خواهد گرفت.

۱- مدل **RPC** یا **Remote Procedure Call** است. این مدل مناسب برای سیستم های کلاینت سرور یا مشتری کارگزاری می باشد.

۲- **RMI** یا **Remote Method Invocation** است. یعنی روش مبتنی بر فراخوانی متدهای راه دور.

در مباحث بعدی، تفاوت بین **Procedure Call** و **Method Invocation** مورد بررسی قرار می گیرد. که در واقع این روش مبتنی بر **Object** های توزیع شده است و در قالب **ROI** یا **Invocation Remote object** قرار خواهد گرفت.

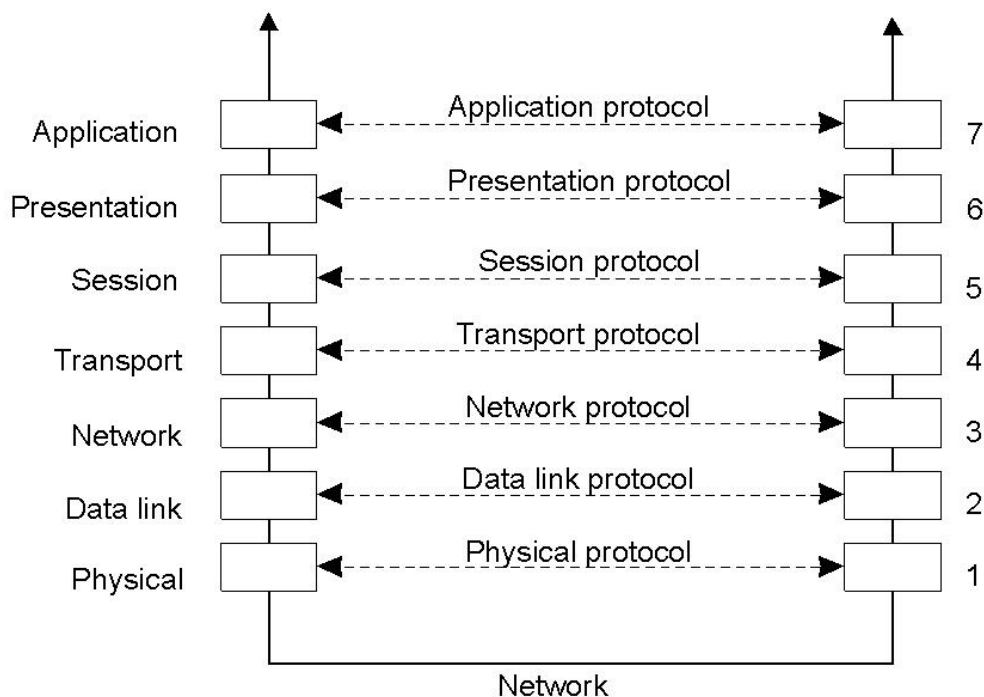
مدل بعدی مدل **MOM** یا **Message Oriented Middle Ware** به معنای یک میان افزار مبتنی بر پیغام است. کاربرد آن در پیغام های سطح بالا و مدلینگ صفحه ی سطح بالاست.

روش دیگر روش **Stream** ها یا جویبارهای داده ای است که جریانی از پیغام های پیوسته با توجه به محدودیت های زمانی موجود در سیستم ها در این تکنیک ایجاد خواهد شد.

قبل از بررسی هر از این مدل‌ها، پروتکل‌های لایه‌ای موجود در سیستم‌های توزیع شده و شبکه‌های ارتباطی به صورت مختصر عنوان خواهد شد.

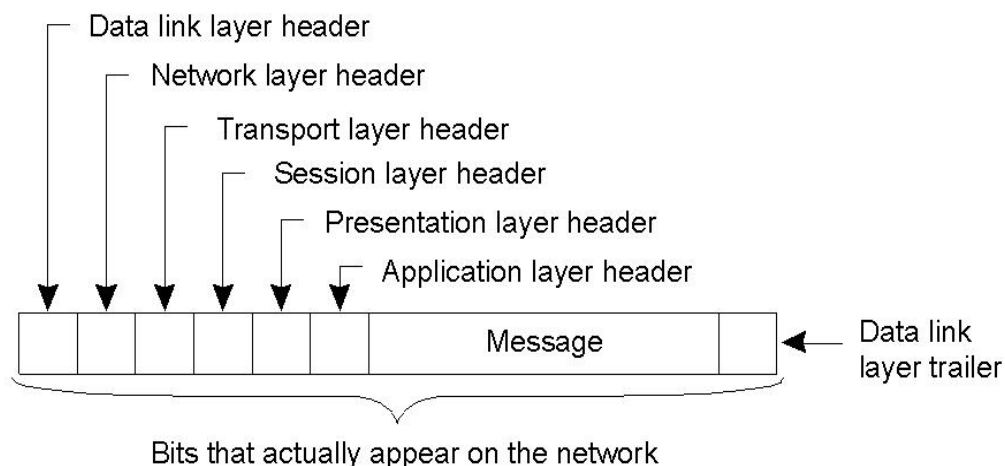
پروتکل‌های لایه‌ای

با فرض اینکه هر یک از مدل‌های مطرح شده را به کار ببریم بطور حتم نیاز به ارسال پیغام وجود دارد و آنچه که در این مدل‌ها مورد توجه قرار می‌گیرد ساختارهای مختلف پیغام است. برای ارسال پیغام از یک ماشین به ماشین دیگر باید توافقات مختلفی بین ماشین‌ها انجام شود، این توافقات از پائین‌ترین سطح ارتباطی که لایه‌ی فیزیکی نام دارد و مبتنی بر انتقال بیت‌ها می‌باشد شروع می‌شود و در بالاترین سطح ارتباطی که در واقع نحوه‌ی پردازش اطلاعات است به پایان می‌رسد. بنابراین در تمامی این لایه‌ها باید توافقاتی انجام شود این توافقات همان پروتکل‌هایی است که مدنظر در یک سیستم ارتباطی قرار می‌گیرند. این پروتکل‌ها بر حسب مدل‌های مختلفی دسته‌بندی می‌شوند و در هر مدل پروتکل‌های خاصی قرار می‌گیرد. یکی از معروف‌ترین مدل‌ها، مدل مرجع OSI است که سطوح مختلفی در ارتباطات را مورد بررسی قرار می‌دهد و در این مدل مشخص می‌شود که در هر لایه چه کاری انجام خواهد شد. استفاده از این مدل باعث خواهد شد که سیستم‌های باز با استفاده از قوانین استاندارد ایجاد شوند و بتوانند با یکدیگر کار کنند.



پروتکل های لایه ای

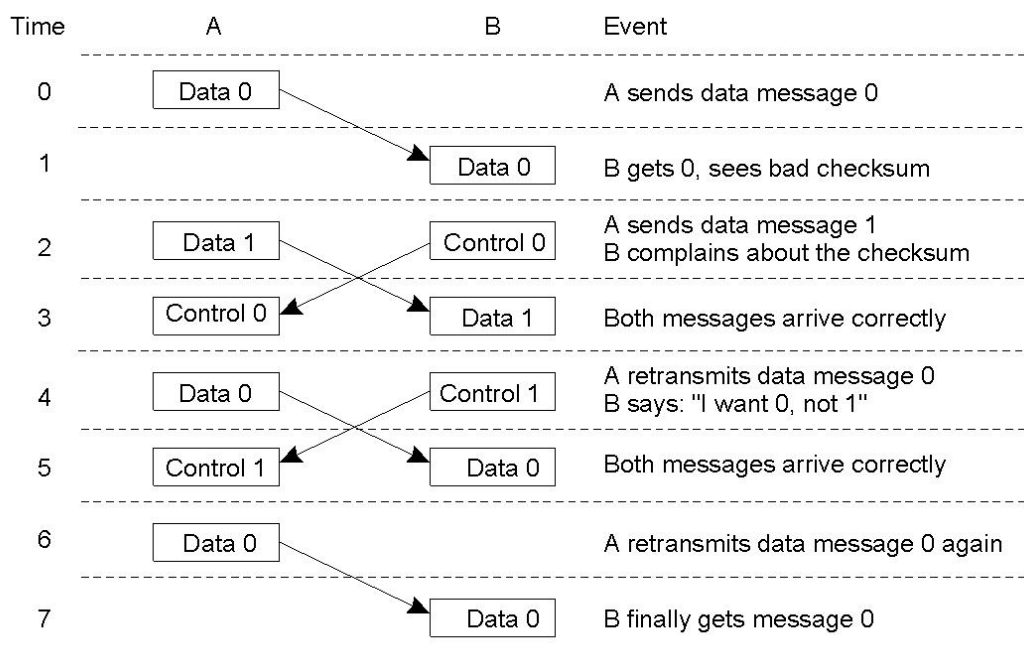
لایه های مختلف مدل OSI در شکل مشاهده می شود، با توجه به شکل، هفت لایه ی مختلف در مدل OSI وجود دارد، که بالاترین لایه، لایه ی Application و پائین ترین لایه، لایه ی فیزیکی است. برای هر کدام از این لایه ها در مدل OSI تعدادی پروتکل تعریف می شود. با توجه به شکل، بعد از لایه Application ، لایه ی Presentation (لایه نمایش)، لایه ی Session (لایه دوره کار) ، لایه ی Transport (لایه ی انتقال)، لایه ی Network (شبکه) و Data link (لایه داده) است. در این مدل برای اینکه یک پیغام از یک ماشین به ماشین دیگر ارسال شود باید از لایه های مختلفی عبور کند.



با توجه به شکل، یک پیغام دارای Headerهای مختلفی است که هر یک از این Headerها اطلاعات مربوط به یک لایه را نشان می دهند. به عنوان مثال، در این پیغام که یک پیغام نوعی است. یعنی ساختار نوعی یک پیغام منتشر شده در شبکه ای که بر اساس مدل مرجع OSI طراحی شده است را نشان می دهد، در قسمت های مختلف این پیغام Headerهای مختلفی قرار گرفته است. حال به طور مختصر هر یک از این لایه ها مورد بررسی قرار می گیرد. در مدل OSI پائین ترین لایه، لایه ی فیزیکی است که وظیفه ی لایه فیزیکی انتقال صفر و یک ها خواهد بود و وظیفه ی دیگر آن استاندارد سازی مفاهیم الکتریکی، مکانیکی و سیگنالینگ است. لایه ی بالاتر آن لایه ی Data link است که در واقع وظیفه ی اصلی آن بررسی صحت الگوی بیتی (Bit Pattern) است و چنانچه در واقع خطایی در الگوی بیتی یا بیت های ارسال شده کشف کند درخواست انتقال مجدد را خواهد کرد.

در شکل زیر نمایی که در لایه ی data link اتفاق می افتد مورد بررسی قرار می گیرد.

دو ماشین A,B در زمان صفر قصد ارسال اطلاعات را دارند با توجه به شکل، Data 0 از سمت A به سمت B ارسال می شود این Data 0 در قالب یک پیغام ارسال می شود.

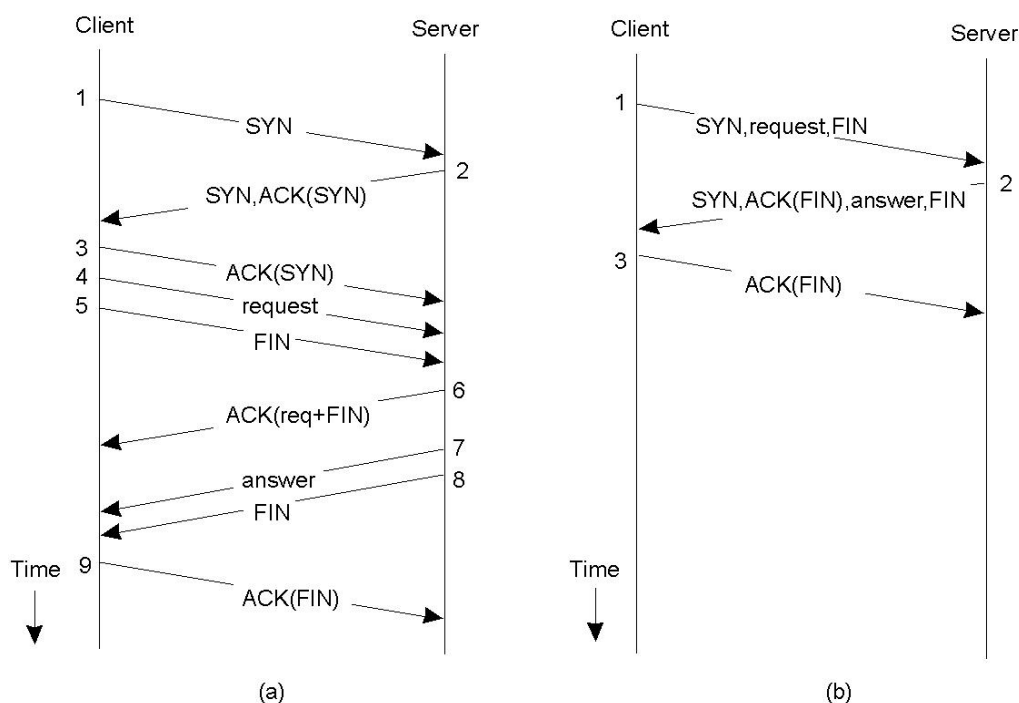


در زمان ۱ ماشین B این Data را دریافت می کند و در لایه ی Data link درستی bit های آن را بررسی می کند و متوجه می شود که خطایی در این Packet وجود دارد. بنابراین در زمان ۲ یک bit کنترلی به سمت ماشین A می فرستد و مشخص می کند که ایرادی در Data اولیه وجود دارد. هر زمان ماشین A داده ی دیگری برای B می فرستد در زمان ۳ کنترل ۰ و داده ی ۱ به هر دو ماشین به سلامت می رسند. در زمان ۴، A مجدداً داده ی ۰ را ارسال می کند اما B یک داده ی کنترلی ارسال می کند به این معنی که می گوید من داده ی صفر را می خواستم اما شما داده ی ۱ را فرستاده اید در واقع شکایت از ارسال نامناسب داده است. در زمان ۵ کنترل ۱ به ماشین A می رسد و داده ی صفر مجدداً به ماشین B می رسد، اما چون کنترل یک حاوی اطلاعاتی مبنی بر Retransmits داده صفر است. در زمان ۶، ماشین A مجدداً داده ی صفر را ارسال می کند و در زمان ۷ در نهایت داده ی صفر به سلامت به مقصد می رسد. همانطور که در این Level مورد توجه قرار گرفت، با توجه به پروتکل موجود در Data link layer در واقع داده ی صفر، ۳ بار منتقل (Transmit) شد و انتقال اول با خطا همراه بود. Transmit دوم گرچه صحت داشت به دلیل وجود پروتکل Data link layer مورد پذیرش قرار نگرفت و در Transmit سوم داده ی مورد نظر به ماشین مورد نظر منتقل شد.

لایه ی شبکه در پروتکل OSI وظیفه ی مسیریابی (Routing) را بر عهده دارد که معمولاً استفاده از پروتکل IP یا Internet Protocol بسیار مورد استفاده می باشد. لایه ی بعدی لایه ی Transport یا لایه ی انتقال است که وظیفه ی اصلی آن این است که شبکه ی زیرین این لایه را به گونه ای تغییر دهد که یک برنامه نویس (application developer) یا کسی که کار کاربردی انجام می دهد بتواند از آن استفاده کند. در واقع در این راستا پیغام ها را به تکه های کوچک برای انتقال می شکند و به هر تکه یک شماره ی ترتیبی (sequence number) اختصاص می دهد و سعی می کند که پیغام ها را بدون خطا و در یک ترتیب مشابه ارسال کند. در این لایه دو پروتکل TCP (Transmission Control Protocol) و UDP (Universal Datagram Protocol) مورد استفاده قرار می گیرد.

Client- Server TCP

در این بخش نحوه عملکرد پروتکل TCP در یک سیستم کلاینت سرور مورد بررسی قرار خواهد گرفت.



با توجه به شکل، برای ارتباط بین کلاینت و سرور، ابتدا از سوی کلاینت یک سیگنال هماهنگی به سوی سرور ارسال می شود. سرور پیغام تأیید دریافت هماهنگی (synchronization) را به همراه یک سیگنال synchronization دیگر و یا سیگنال همگام سازی به سوی کلاینت ارسال می کند.

کلاینت، تأیید دریافت synchronization را در گام سوم به سرور می فرستد. این مراحل تنها به معنای آماده سازی بستر ارتباطی بین کلاینت و سرور است و از این پس این دو می توانند با یکدیگر ارتباط داشته باشند. در مرحله چهارم به بعد درخواست های کلاینت می تواند به سوی سرور ارسال شود. بنابراین با توجه به آنچه در شکل نشان داده شده است، یک درخواست از کلاینت به سمت سرور ارسال می شود و بعد از آن در زمان ۵ در واقع سیگنال مبنی بر پایان درخواست از کلاینت به سمت سرور خواهد رفت. در زمان ۶ سرور Acnolagh درخواست و Finalize شدن درخواست را به سوی کلاینت می فرستد و درخواست را پردازش کرده و پاسخ را در زمان ۷ به همراه سیگنال پایان پاسخ به سوی کلاینت خواهد فرستاد و کلاینت تأیید دریافت سیگنال پاسخ را ارسال می کند.

همانطور که در این پروتکل می بینید به ازای هر درخواست حتماً سمت مقابل سیگنال تأیید، دریافت آن سیگنال را ارسال خواهد کرد. ایرادی که بر این روش وارد خواهد شد، این است که تعداد سیگنال synchronization بین سمت کلاینت و سرور زیاد است. در مدل دیگری از TCP که در شکل بعد نشان داده شده است و به عنوان Transactional TCP شناخته می شود، سعی شده است که تعداد این سیگنال ها که در زمان های مختلف و به صورت مستقل ارسال می شوند کاهش پیدا کند. بنابراین سیگنال های synchronization و Request و Final همزمان در یک بسته از سمت کلاینت به سرور فرستاده می شود و سرور سیگنال synchronization، تأیید دریافت فایل، پاسخ و سیگنال مربوط به فایل شدن پاسخ را در یک بسته به سمت کلاینت می فرستد و در واقع در زمان ۳ تنها تأیید دریافت سیگنال فایل از سمت کلاینت به سمت سرور ارسال خواهد شد.

پروتکل های سطح بالاتر (Higher Level Protocol)

لایه های بالاتر از لایه مربوط به Transport layer در واقع سطوح دسترسی بالاتری را ایجاد می کند. این لایه ها مانند، Session، Presentation، Application هستند. در لایه Session می توان گفت

این لایه یک گونه بهبود یافته از **transport** یا لایه انتقال است. دیالوگ های کنترلی و امکانات هماهنگ سازی در این لایه وجود دارد و در واقع این امکان وجود دارد تا به کمک این لایه مشخص شود کدام قسمت از شبکه در حال برقراری ارتباطات است و در واقع امکان اضافه شدن **Check point** هایی در پیغام ها برای بکار بردن این **Check point** ها در زمان خرابی نیز وجود دارد. در لایه **Presentation** معانی بیت ها معلوم می شود و در واقع رکوردهایی ایجاد می شود که شامل فیلدها هستند. در واقع تفسیر رشته بیتی در لایه **Presentation** به نوعی انجام می شود و لایه آخر که لایه **Application** است شامل تمامی برنامه های کاربردی و پروتکل ها است، که در سایر لایه ها قرار نمی گیرد.

نقطه ضعف هایی در مدل **OSI** وجود دارد. از مهمترین آنها این است که در این مدل نمی توان بین **Application** و **Application specific protocol** ها یعنی پروتکل هایی که به نوعی توسط **Application** ها مشخص می شوند فرق قرار داد. از جمله این **Application specific protocol** ها می توان به پروتکل **FTP** که در واقع به نوعی جایگاه آن در لایه **Application** است اشاره کرد و شاید سایر پروتکل های عام منظوره (**general purpose**) نیز در این گروه قرار گیرند. برای رفع این مشکل از پروتکل های میان افزار (**Middleware**) استفاده می شود. یک پروتکل **Middleware** در واقع **Application** است که در لایه **Application** زندگی می کند و شامل پروتکل های عام منظوره، مانند پروتکل های تعیین هویت (**Authentication**)، پروتکل های نهایی سازی **Transaction** ها (**Commit**)، پروتکل های قفل گذاری و محافظت از منابع می باشد.

هر یک از این پروتکل ها خود جای بحث بسیار دارد و در این مرحله تنها با کلیات آنها آشنا شدید.

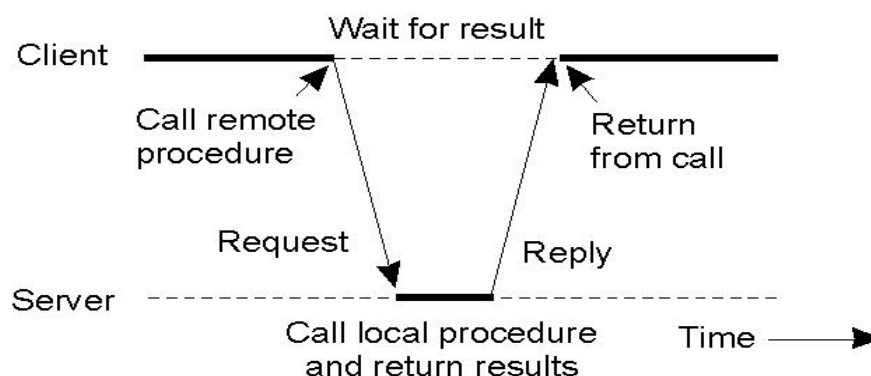
پروتکل های ارتباطی

بعد از عنوان مطالب مربوط به پروتکل های ارتباطی به بحث در مورد مدل های ارتباطی خواهیم پرداخت. با توجه به مطالب پیشین، یکی از این مدل ها **Remote Procedure Call** یا فراخوانی یک روبه از راه دور است. در این مدل اجازه می دهیم که یک برنامه، **Procedure** ها یا برنامه هایی را که بر روی ماشین دیگری در سطح شبکه قرار گرفته اند فراخوانی کند و فراخواننده تا زمانیکه پاسخ آن آماده شود معلق خواهد شد. در واقع در این سیستم، ماشینی زیر برنامه ای را بر روی ماشین دیگر اجرا می کند. در این متد

هیچ ارسال پیغامی در سطح **programmer** به هیچ وجه قابل مشاهده نیست، این به این معنی نیست که ارسال پیغام در **Remote Procedure Call** وجود ندارد. در واقع یک لایه دسترسی بالاتر در اینجا قرار گرفته است و برنامه نویس ارسال پیغام ها را نمی بیند. مشکلاتی که در این روش وجود دارد این است که ماشین های مختلف ممکن است فضای آدرس دهی مختلفی داشته باشند و این در اجرای کدهای مختلف تأثیرگذار خواهد بود، و در نتیجه مدیریت آنها پیچیدگی هایی را به **RPC** اضافه می کند. نکته دیگر این است که اگر ماشینها مشابه نباشند، ممکن است از پارامترها و نتایج تفسیر یکسانی نشود و این هم پیچیدگی هایی را در سیستم اضافه می کند. به عنوان مثال اگر **RPC** از روی یک ماشین **PC** بر روی یک ماشین **main frame** انجام شود از آنجایی که ساختار و معماری آنها لزوماً یکسان نیستند تفسیر فضای آدرس ممکن است یکسان نباشد. نکته دیگر، خطاهای رخ داده در ماشین هاست. اگر ماشینی که **RPC** بر روی آن انجام شده است به هر دلیل دچار خطا شود، ممکن است باعث معلق ماندن بینهایت در ماشین فرا خواننده شود.

اجزای RPC

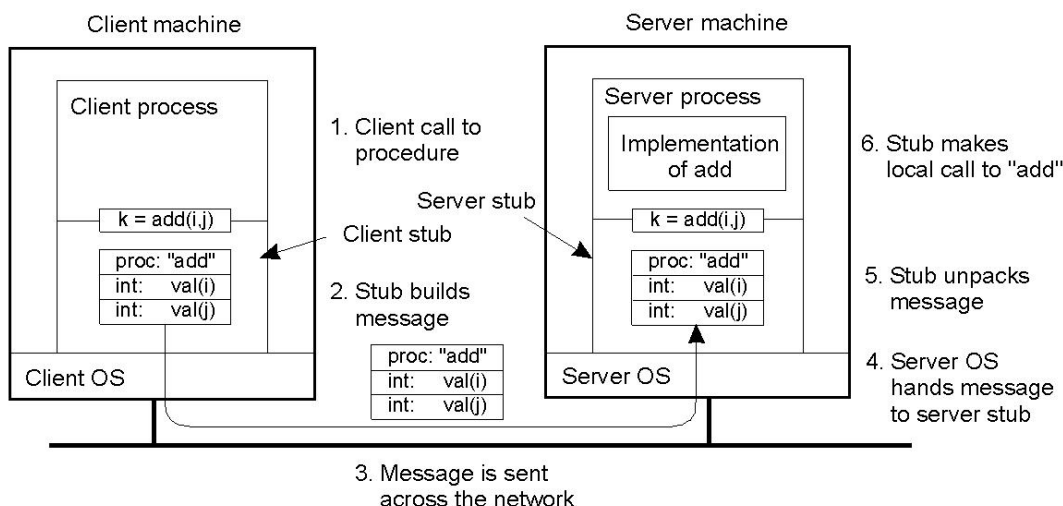
به طور کلی در روند **RPC** دو جزء اصلی وجود دارد: یک جزء را به عنوان بر چسب به سمت مشتری (**client stub**) و جزء دیگر را به عنوان بر چسب به سطح سرور (**server stop**) خواهد بود. **Client Stub** وظایفی بر عهده دارد: از جمله آنکه مانند فراخوانی های ماشین های سنتی، ترتیب اعمالی که باید در فراخوانی انجام شود را بکار می برند و مانند سیستم های سنتی از سیستم عامل محلی درخواست می کند و فراخوانی را با آن انجام می دهد. اما برخلاف ماشین های سنتی از سیستم عامل نمی خواهد که داده ها را دریافت کند، بلکه پارامترها را در قالب یک پیغام بسته بندی می کند و از سیستم عامل می خواهد که آن را به سمت سرور ارسال کند. در سمت سرور، **server stub** روندی برعکس را طی می کند و درخواست ها را به فراخوان های محلی تبدیل می کند و پس از اینکه فراخوانی به پایان رسید پاسخ ها را در قالب یک پیغام به سمت کلاینت ارسال می کند.



در شکل روند کلی یک RPC نشان داده شده است. همانطور نشان داده شده است کلاینت در نقطه ای یک Remote Procedure Call انجام می دهد و درخواست را به سمت سرور می فرستد و خود منتظر می ماند و سرور یک فراخوانی محلی انجام می دهد و درخواست ها را به سمت مشتری می فرستد. اما نکته مهم در سیستم RPC بحث انتقال پارامترهای یک رویه است.

انتقال پارامتر RPC (RPC Parameter Passing)

برای انتقال پارامترها دو تکنیک وجود دارد، یک تکنیک را ارسال به کمک مقدار (Call by value) می دانیم که یکی از تکنیک های متداول و پرکاربرد است. یکی از مفاهیم مهم در تکنیک Call by value یا ارسال با کمک مقدار تکنیک یا مفهوم Parameter Marshaling است. در واقع در Parameter Marshaling پارامترها در داخل Packetها بسته بندی شده و به سمت سرور ارسال می شوند. در شکل روال یک RPC که تکنیک Parameter passing آن Call by value است را مشاهده می نمایید.



با توجه به شکل، در مرحله اول در سمت ماشین کلاینت، کلاینت فراخوانی را انجام می دهد و این فراخوانی به client stub می رسد. Client stub در گام دوم پیغام مورد نظر را می سازد. در این پیغام، عوامل مورد نیاز و ضروری Procedure مورد نیاز و پارامترهای آن است. به عنوان مثال در اینجا Procedure add، Procedure و پارامترهای آن I و G است. دو مقدار در Packet مربوطه قرار می گیرد، این Packet در قالب یک پیغام از شبکه عبور کرده به سمت سرور می رود. سیستم عامل سرور در گام چهارم پیغام را دریافت کرده و با توجه به Header آن تشخیص می دهد که باید آن را به server stub بفرستد. Server stub در گام پنجم پیغام را Unpack می کند و در واقع می بیند که این پیغام نیازمند اجرای Procedure add با دو پارامتر مشخص در آن است، که در واقع این پارامترها در Package یا در Message، Marshal شده اند. در گام ششم stub یک فراخوانی محلی به Procedure add که پیاده سازی آن در سطح سرور صورت گرفته است، انجام خواهد داد و مجدداً پاسخ را برای ماشین کلاینت ارسال خواهد کرد. مسیر برگشت پاسخ در این شکل نشان داده نشده است. گرچه ارسال پارامتر با استفاده از مقدار، روند خوب و ساده ای است که به سادگی می توان از آن استفاده کرد، اما مشکلاتی نیز دارد.

انتقال پارامتر RPC (RPC Parameter Passing)

از جمله مشکلات روش Call by value می توان به این نکته اشاره کرد که در این روش ممکن است وجود کدینگ های مختلف بر روی ماشین های مختلف مشکلاتی ایجاد کند. به عنوان مثال ممکن است

ماشینی از کدینگ ASCII استفاده کند، یعنی در واقع جدول کدینگ ASCII را به کار برد و در نتیجه در این حالت تفسیر یک عدد با توجه به جدول کد ASCII خواهد بود در حالی که ماشین سرور از کدینگ دیگری مانند EBCDIC استفاده کند، که در اینجا مقادیر عددی تفسیرشان بر روی دو ماشین متفاوت خواهد شد. مشکل دیگر در بحث نمایش داده‌ها می‌باشد که با توجه به مفهوم Little endian و Big endian ممکن است در ماشین‌های مختلف رخ دهد. جهت بررسی بیشتر این مسأله به مثال زیر توجه کنید. فرض کنید یک داده ۸ بایتی از روی یک ماشین پیتموم که مبتنی بر معماری intel است به یک ماشین Spark منتقل شود. شکل a ساختار این داده را نشان می‌دهد. اعداد ریز نوشته شده در هر سلول نشان دهنده شماره بایت است. وقتی که این داده به سمت ماشین دیگری می‌رود و از آنجایی که تکنیک بکار برده شده در ماشین Spark مشابه تکنیک بکار برده شده در ماشین intel نیست، یعنی ترتیب بایت‌ها تغییر می‌کند، بنابراین در بایت شماره صفر مقدار ۵ قرار می‌گیرد. آنچه که در اینجا مهم است این است که ترتیب ارائه در حافظه تغییر کرده است. یعنی اگر به شکل a نگاه کنید بایت شماره ۳ در گوشه سمت چپ بالا قرار گرفته است، اما در شکل b بایت شماره ۵ در گوشه سمت چپ بالا قرار گرفته است. لذا اگر قرار باشد تفسیری که در ماشین a انجام می‌شد بر روی byte sequence در ماشین b انجام شود، احتمال دارد که داده دیگری برداشت شود. بنابراین باید در اینجا عملیات اضافه‌ای انجام شود که تفسیر به درستی صورت گیرد، این عملیات اضافه در واقع با توجه به نحوه نمایش داده‌ها در دو ماشین خواهد بود و شکل c نمایش حاصل از پردازش را که می‌تواند در ماشین Spark استفاده شود، نشان می‌دهد.

3	2	1	0
0	0	0	5
7	6	5	4
L	L	I	J

(a)

0	1	2	3
5	0	0	0
4	5	6	7
J	I	L	L

(b)

0	1	2	3
0	0	0	5
4	5	6	7
L	L	I	J

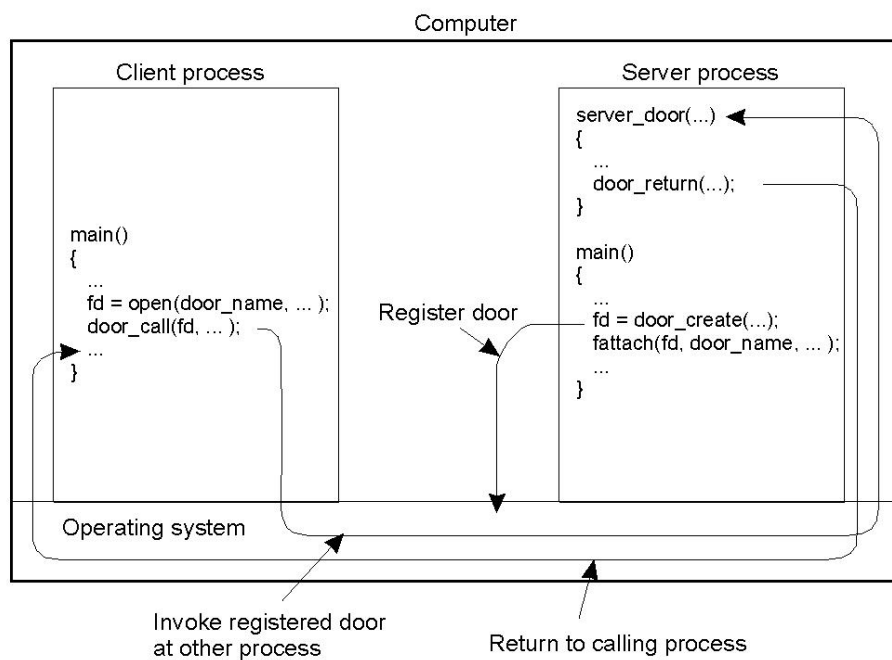
(c)

ارسال پارامترهای ارجاعی (Passing Reference Parameter)

در ارسال پیغام در RPC از مفهوم دیگری نیز می‌توان استفاده کرد که آن را ارسال پارامترها با Reference یا پارامترهای ارجاعی می‌نامند. در این روش به جای آنکه مقدار پارامتر در داخل یک پیغام به سمت ماشین سرور فرستاده شود ارجاع آن یا آدرس آن فرستاده خواهد شد و بنابراین در داخل هر پیغام آدرس پارامترها وجود خواهد داشت. بدیهی است که ایراداتی به این روش وارد است. از جمله اینکه استفاده از آن برای ساختارهای داده پیچیده چندان ساده نیست و در عین حال برای دسترسی به فضای آدرس یک ماشین دیگر باید سطح دسترسی های لازم را ایجاد کرد. بنابراین به طور کلی می‌توان عنوان کرد در پروتکل مربوط به RPC مواردی مانند فرمت پیغام، نحوه نمایش داده ساختارها یا ساختمان‌های داده‌های ساده و همچنین نحوه برقراری ارتباط داده‌ای بین دو ماشین باید تعیین شود، که می‌تواند Connection oriented یا Connection less باشد که باید در پروتکل های مربوط به RPC تعیین شده و مورد استفاده قرار گیرد و چه بسا که در پروتکل های مختلف تکنیک های مختلفی به کار گرفته شوند.

RPC توسعه یافته (Extended RPC)

با توجه به ایرادات و یا حالت‌های خاصی که در RPC وجود دارد، RPC های توسعه یافته یا Extended RPC نیز وجود دارد. از جمله مفاهیم RPC های توسعه یافته می‌توان به مفهوم Door یا درها اشاره کرد. این مفهوم این گونه بیان می‌شود که وقتی مشتری و کارگزار بر روی یک ماشین قرار دارند آنگاه در واقع ارسال پیغام زیاد عقلانی نیست. بنابراین در این حالت به جای آنکه از تکنیک RPC با مفاهیم قبلی استفاده شود از تکنیک Door استفاده می‌شود. پس به طور کلی می‌توان گفت که Door مفهومی است هم ارز RPC برای فرایندهایی که بر روی یک ماشین قرار دارند. در واقع از دیدگاه پیاده سازی می‌توان گفت که Door یک اسم Generic است که در فضای آدرس سرور ثبت می‌شود و توسط فرآیندی که بر روی همان سرور قرار دارد فراخوانی می‌شود؛ بدیهی است که این مفهوم باید در سطح سیستم عامل محلی Support شود و فرآیند سرور باید یک Door را قبل از استفاده از آن در فضای نام‌ها رجیستر کند تا امکان استفاده از آن فراهم گردد. در شکل زیر مفهوم استفاده از Doorها نشان داده شده است.



با توجه به شکل، در اینجا یک ماشین وجود دارد، اما یک فرآیند سرور و یک فرآیند کارگزار هر دو بر روی یک ماشین قرار گرفته است. فرآیند سرور در ابتدا یک Door را ایجاد می کند و این باعث می شود که Door مورد نظر در فضای آدرس سیستم عامل ثبت گردد و سپس فرآیند کلاینت می تواند با دستورات ساده ای مانند Open این Door را باز کرده و امکان فراخوانی زیر برنامه هایی که در سمت سرور قرار دارند را مهیا کند.