سیستم‌عامل پیشرفته

محمد داورپناه جزی

گروه مهندسی کامپیوتر و فناوری اطلاعات

دانشگاه صنعتی فولاد

ترم دوم ۹۴-۹۳

فصل دوم: ارتباطات در سیستم‌های توزیع شده (ادامه)

- پیاده‌سازی مدل Client-Server
  - خلاصه حالات در جدول شکل ۲-۱۴ ص ۶۵ ۸۱ ترکیب که همه آنها به دردبخور هستند.
  - هر شبکه یک Packet Size مشخصی (حداکثر چند هزار بیت) دارد و پیام‌های بزرگتر باید شکسته شوند.
  - با توجه به امکان گم شدن یا ناقص شدن پاکت‌ها یا رسیدن بدون ترتیب آنها شماره‌گذاری می‌شوند یعنی در هر پاکت علاوه بر شماره پیام یک شماره پاکت هم وجود دارد.
  - برای تأیید می‌توان هر پاکت را ack کرد که تعداد Packet زیاد می‌شود ولی Recovery ساده است.
  - یا می‌توان کل پیام را ack کرد که تعدا Packetها کم می‌شود ولی با یک پاکت خراب کل پیام باید تکرار شود.
  - انتخاب بسته به ضریب اطمینان شبکه دارد.
  - موضوع جالب دیگر پروتکل ارتباطی است در شکل ۲-۱۵ ص ۶۶ یک نمونه ارائه شده است. شکل ۲-۱۶ چند نمونه پروتکل
  - برای حالت بدون بافر سیستم می‌تواند با درخواست Server پروسس‌ها را ثبت نام کند تا پیغام‌های رسیده قبل از Receive را با TA برگرداند نه با AU

فصل دوم: ارتباطات در سیستم‌های توزیع شده (ادامه)

4. Remote Prcedure Call – احضار روال از راه دور

- I/O به عنوان بحث مهم در سیستم‌های توزیع شده و ماندن عده‌ای به غلط در حل آن
- احضار برنامه‌ای روی ماشین B توسط برنامه‌ای روی ماشین A (پس از احضار برنامه روی A معلق می‌شود تا خاتمه کار)
- پارامترها می‌توانند ردوبدل شوند. هیچ I/O ای از دید برنامه‌نویس موجود نیست.
- مسئله نظیر وجود دو فضای آدرس متفاوت، مبادله پارامترها بین دو ماشین متفاوت، توقف ماشین‌ها مطرح است.
- با وجود اینها RPC زمینه‌ساز خیلی از سیستم‌های عامل توزیع شده است.

- **عملیات ابتدایی RPC**

- توجه به یک احضار معمولی شکل ۲-۱۷ ص ۶۹، دو نوع انتقال پارامتر (Value، Reference و Copy/Restor)
- اینکه چه نوع ارسال پارامتر داشته باشیم به زبان بستگی دارد (C) و گاهی هم انتخابی است (Pascal)و گاهی انواع (Ada)
- هدف از RPC این است که آنرا از دید کاربر درست شبیه Call عادی انجام دهیم یعنی جزئیات مخفی باشد.

فصل دوم: ارتباطات در سیستم‌های توزیع شده (ادامه)

- مثال احضار Read ، افزودن روتین Read توسط Linker، گذاشتن پارامترها در Reg های مربوطه انجام System Call
- پس Read یک واسط بین کاربر و سیستم عامل است که از طریق Kernel انجام می‌پذیرد احضار عادی نیست.
- جزئیات Read از کاربر مخفی است و مثل یک Call عادی به کار گرفته می‌شود.
- نحوه کار RPC هم مشابه Read است.
- اگر یک RPC Read داشته باشیم برنامه کاربر به شکل عادی (شکل ۲-۱۷) Client Stub را احضار می‌کند.
- Cilent Stub پارامترها را در قالب یک پیام در می‌آورد و از Kerel می‌خواهد که آنرا بفرستد به مقصد
- Cilent Stub بعد از احضار Send و ارسال پیام Receive را احضار کرده و بلوکه می‌شود تا جواب بیاید.
- شکل ۲-۱۸ ص ۷۱ Server Stub هر بیضی یک پروسس است و Stub زیر روالی است که احضار می‌شود.
- در Serverای که باید پیغام را بگیرد Server Stub در Loop اصلی خود Receive را احضار کرده و منتظر است

فصل دوم: ارتباطات در سیستم‌های توزیع شده (ادامه)

- Server با دریافت پیام آنرا به Server Stub می فرستد تا آنرا باز کرده پارامترها را جدا کند.
- Server Stub به طور معمول (ش ۱۷-۲) روتین موجود در Server را احضار می کند.
- این روتین پس از انجام عمل، نتیجه را در پارامترها قرار می‌دهد و به Stub برمیگرداند
- Server Stub پارامترها را در قالب پیام بسته‌بندی کرده و از طریق Send به Client می‌فرستد. با احضار Receiver منتظر پیام بعدی می‌شود.
- Kernel مربوط به Client پیغام را می گیرد و می‌فهمد به کدام پروسس بدهد (آنرا به Process Stub می‌دهد) ولی Client چیزی از این نمی‌داند.
- Client Stub پیغام را باز می کند و نتایج را به برنامه احضار کننده می‌فرستد و این برنامه فکر می کند که احضار عادی انجام داده بود.
- پس آنچه برای Client جذاب است انجام احضار عادی به جای Send و Receiver است
- جزئیات مراحل در ص ۷۲ ولی Client و Server از آنها بی‌خبرند.

فصل دوم: ارتباطات در سیستم‌های توزیع شده (ادامه)

- مبادله پارامترها
  - گرچه مبادلهٔ پارامترها با استفاده از Stubها به ظاهر ساده است ولی نکاتی در عمل دارد .

(Parameter Marshalling)

  - جزئیات یک احضار در شکل ۱۹-۲ ص ۷۳ آمده است.
  - در صورتی که دو ماشین Clinet و Server یکسان باشند این روند درست کار می کند.
  - اگر دو کامپیوتر متفاوت داشته باشیم در بستن و باز کردن پیام‌ها اشکال پیش می‌آید.
  - مثال مبادله بین Intel 486 که Little Endian است و SPARK که Big Endian است شکل 2-20 ص75
  - راه حل ساده است باید یک قرارداد بین Client و Server در مورد نوع‌های اولیهٔ داده گذاشته شود .شکل 2-21 ص75
  - راه اول تعریف یک استاندارد انتقال مثلاً ASCII + ones comp و Litt Endian و الزام به رعایت در مبدأ و مقصد
  - بسیار خوب با تنها عیب که ماشین‌های مشابه ممکن است دو تبدیل بیخودی انجام دهند.
  - راه دوم ارسال اطلاعات مربوط به نوع‌ها همراه پیام با این شرط که هر دو بتوانند تبدیلات انجام دهند.

فصل دوم: ارتباطات در سیستم‌های توزیع شده (ادامه)

- روال‌های Stub از کجا می‌آیند؟ با داشتن اطلاعات Server کامپایلر می‌تواند دستورات لازم را اتوماتیک تولید کند. (بدون خط)
- یک روال بسته‌بندی پیغام و یک روال باز کردن پیغام با توجه به نوع‌های داده و نوع ماشین، تولید می‌شود.
- نحوهٔ ارسال Pointerها ؟ راه اول منع آن به طور کامل و ارسال همه پارامترها به صورت مقدار یا C/R
- این راه حل قبول نیست
- راه دوم اینکه Client Stub محتویات را کپی کند بفرستد، Server Stub روی آن کار کند برگرداند و Cilent Stub دوباره محتویات پیام را در محل اصل کپی کند (شبیه‌سازی C/R)
- دوباره کپی کردن وقت گیر است ولی چاره‌ای نیست
- با دانستن Input، Output یا هر دو (نوع پارامتر) می‌توان کپی‌ها غیر لازم را انجام نداد.
- برای اینکه در تعریف RPC باید نوع پارامترها و حداکثر طول آنها گفته شود.
- برای ساختمان داده‌های پیچیده (درخت‌ها و گراف‌های دینامیک) این روش عملی نیست
- راه حل پیشنهادی ارسال Pointer و سپس انجام عملیات روی اطلاعات در قالب مبادله پیام است که گرچه کارآیی خوبی ندارد ولی از هیچ بهتر است.

فصل : ارتباطات در سیستم‌های توزیع شده (ادامه)

- چگونه Client موفق می‌شود Server را پیدا کند (پیدا کردن Client را Server,)
  - راه حل ساده گذاشتن اطلاعات داخل برنامه Client به صورت Hardwiered که اصلاً انعطاف ندارد. (نیاز به ترجمه دوباره همه برنامه‌ها در صورت کوچکترین تغییر)
  - راه حل بهتر Dynamic binding یا وابسته کردن به طور پویا
  - اول نیاز به تعریف فرمان برای Server داریم ش ۲۲-۲ ص ۷۸ برای Server ش ۹-۲ ص ۵۵
  - یک Stateless server است یعنی نیازی به دانستن وضعیت قبلی (Open بودن فایل‌ها مثلاً) ندارد.
  - Stub generator در کامپایلر ازاین تعاریف فرمان برای تولید Stubها در زمان کامپایل استفاده می‌کند و نتیجه برای Link شدن در کد باینری در زمان Link در یک Library قرار می‌گیرد. (برای Server ، Client)
  - با شروع کار Server دستور Initialize ش ۹-۲ باعث ارسال یک پیغام به برنامه Binder برای ثبت نام (register) کردن Server می‌شود یعنی من هستم! (به این کار export کردن server گویند)
  - برای ثبت نام نیاز به اسم، handle, id, version و مجوزهای دسترسی می‌باشد.

فصل : ارتباطات در سیستم‌های توزیع شده (ادامه)

- Handle وسیله شناسایی فیزیکی است مثل شماره IP یا SPI یا ....
- حذف نام هم در زمان توقف Server انجام می‌شود. خلاصه در ش ۲۳-۲ ص ۷۹ رابط Binder
- حال وقتی یک RPC انجام می‌شود مثلاً یک Read توسط Client
- Client Stub عدم اتصال به Server مورد نیاز را متوجه می‌شود.
- پیامی به Binder می‌فرستد برای Import کردن Version خاصی از واسط Server مربوطه
- اگر چنین واسطی از هیچ Servery تا حالا export نشده با شماره version ها تطبیق ندارد Fail می کند.
- اگر نه، Handle و شماره شناسایی را برمی گرداند تا توسط Client در جوف پیام گذاشته شود.
- بعد از ارسال پیام، Server ها آن را چک کرده فقط Server مورد نظر پیام را برمی‌دارد با در نظر گرفتن Version
- انعطاف‌پذیری زیادی در این روش وجود دارد.
- داشتن چند Server ارائه دهنده خدمات مشابه
- امکان تقسیم بار کاری به طور اتوماتیک روی Serverها
- Poll کردن Serverها و حذف نام آنها که خوابیده‌اند به طور اتوماتیک
- رعایت کردن مجوزهای دسترسی به Serverهای خاص

فصل : ارتباطات در سیستم‌های توزیع شده (ادامه)

- اشکالاتی هم دارد از جمله هزینه سر بار برای عملیات بالا و کند بودن در سیستم‌های بزرگ
- در سیستم‌های توزیعی وسیع می‌توان چند Binder داشت که با هر تغییر کلیه آنها باید آگاه شوند که خود یک بار زیادی است.

- **عملکرد RPC هنگام بروز شکست (Failure)**
  - با توجه به آنچه ذکر شد در صورت درست کار کردن هر دو ماشین عملکرد مورد نظر توسط RPC تامین می‌شود.
  - حال اگر اتفاقی افتاد چه می‌شود؟
  - پنج نوع شکست:
  - عدم امکان یافتن Server توسط Client (نیافتن Client، Server را)
  - گم شدن پیغام ارسالی از Client به Server
  - گم شدن پیغام ارسالی از Server به Client
  - سقوط Server پس از وصول پیام
  - سقوط Client پس از ارسال پیام
  - عدم یافتن Server به دلیل down بودن یا عدم تطبیق version هاست. Server جدید، Client قدیمی راه حل:؟

فصل : ارتباطات در سیستم‌های توزیع شده (ادامه)

- مشابه ش ۹-۲ بر گردان ۱- توسط توابع در زمان خطا
- در Unix متغیر error حاوی کد نوع خطاست که یکی هم می‌تواند "Can’t locate server" باشد.
- اگر برنامه‌ای مثل SUM باشد ۱- می‌تواند یک جواب واقعی باشد (8-)+7)
- راه حل دیگر چیزی شبیه ON ERROR است که در بعضی زبان‌ها هست و با شرط Transparency مغایرت دارد در بعضی زبان‌ها هم نیست.
- گم شدن پیام درخواست، استفاده از Timeout و تکرار پیام
- اگر پیغام وقعاً گم شده، با تکرار آن مسئله حل می‌شود.
- اگر با چند بار تکرار حل نشد باز می‌شود Can’t locate server
- گم شدن پیام پاسخ، یک راه همان Timeout و تکرار پیام درخواست است.
- بعضی درخواست‌ها تکرارشان بدون اشکال است مثل خواندن یک بلوک (Idempotent)
- بعضی نیستند مثل انتقال پول بین دو حساب، زیر ممکن است کار انجام شود ولی پاسخ گم شود.
- یک راه دادن شماره ردیف به پیغام‌هاست تا Server مواظب باشد همیشه آخرین پیام هر Client چه شماره‌ای بوده
- راه دوم گذاشتن یک Flag و 1 کردن آن برای پیغام‌های تکراری

فصل : ارتباطات در سیستم‌های توزیع شده (ادامه)

- توقف Server: مشکل این است که توقف قبل از انجام کار بوده و یا بعد از آن ش ۲۴-۲ ص ۸۲
- Client فقط می‌داند جواب نیامده (Timeout)
- سه طرز فکر
- ۱) At least once semantics: تکرار درخواست تا حصول نتیجه (reboot شدن یا rebind شدن مجدد) تضمین اجرای حداقل یک بار شاید هم بیشتر! در کارهای non idempolent اشکال دارد.
- ۲) At most once semantics: توقف کار بلافاصله و گزارش خطا، کار یا یکبار انجام شده یا اصلاً انجام نشده
- بدون تضمین: Client هیچ تعهدی ندارد کار می‌تواند بین صفر تا n بار انجام شود (پیاده‌سازی ساده)
- هیچکدام exacly once را که می‌خواهیم نمی‌دهد و این قابل وصول نیست.
- به طور خلاصه موضوع توقف Server تفاوت بین single process و distributed را مشخص می‌کند دیتا و به نوعی recovery داریم.
- توقف Client: بعد از درخواست و قبل از وصول جواب توقف کند یک پروسس یتیم وجود دارد.
- مسائل: هدر دادن CPU و سایر منابع، پس از reboot و درخواست مجدد، جواب قبلی یا جدید تمیز داده نمی‌شود.

فصل : ارتباطات در سیستم‌های توزیع شده (ادامه)

- راه حل ۱) Client Stub قبل از ارسال پیام یک Log درست کند در مورد درخواست روی دیسک
- بعد از reboot می‌توان در Log گشت و یتیم‌ها را از بین برد ولی کار ساده نیست
- یتیم‌ها خود می‌توانند یتیم تولید کرده باشند و جستجویشان مشکل است.
- شبکه ممکن است چند بخشی شده باشد و جستجو امکان نداشته باشد.
- پس روش مفیدی نیست (قلع و قمع extranunation)
- راه حل ۲) (reincarnation) زمان به تعدادی فواصل تقسیم می‌شود، بعد از reboot هر Client، یک پیغام روی شبکه broadcast می کند که شروع فاصله زمانی جدیدش اعلام می‌شود.
- با شروع فاصله جدید تمام درخواست‌های راه دور آن kill می‌شوند.
- راه حل ۳) (Gentle reincarnation) شبیه بالا ولی وقتی شروع فاصله است برای هر RPC اعلام کننده آن چک می‌شود اگر خوابیده (crash) آنگاه پروسس kill می‌شود.
- راه حل ۴) (Expiration) تخصیص برش زمانی T به هر RPC و نیاز به درخواست مجدد T در صورت تمام نشدن. اگر بعد از crash server به اندازه T صبر کند همه یتیم‌ها حذف شده‌اند ولی مشکل اندازه T است. با توجه به شرایط متعاقب RPC هیچ کدام راه مناسبی نیست

فصل : ارتباطات در سیستم‌های توزیع شده (ادامه)

- تازه kill کردن فرزندان هم مسئله‌ای است و قفل‌های ایجاد شده روی منابع یا درخواست‌های منتظر در صفت‌ها که از طرف این یتیم‌ها انجام شده.

- موضوعات مربوط به پیاده‌سازی
  - موفقیت یک سیستم توزیع شده بستگی به عملکرد آن دارد که عملکرد هم بستگی به سرعت ارتباط دارد.
- پروتکل RPC
  - در تئوری هر پروتکلی که بیت‌ها را منتقل کند کافی است ولی در عمل مسائلی وجود دارد.
  - آیا خط connection criented باشد یاconnectionless
  - در نوع .C.O ارتباط ایجاد می‌شود و ارتباط را راحت می‌کند و پیام هم گم نمی‌شود و ack هم نمی‌خواهد.
  - این مسائل در نرم‌افزارهای سطح پایین‌تر حل می‌شود.
  - اینها در WAN خوب است ولی در LAN نیازی نیست.
  - وقتی کل سیستم در یک ساختمان است از .C.L استفاده می‌شود زیرا عملکرد بهتری دارد و نرم‌افزار اضافی نمی‌خواهد.

فصل : ارتباطات در سیستم‌های توزیع شده (ادامه)

- موضوع بعدی استاندارد بودن یا خاص بودن پروتکل است.
- چون پروتکل استانداردی وجود ندارد از پروتکل خاص برای RPC استفاده می‌شود و اختلاف نظر هم هست.
- عده‌ای از IP یا UDP که روی IP است استفاده می کنند.
- دلیل: طرح و پیاده‌سازی شده، همه انواع UNIX آنرا قبول دارد، اکثر شبکه‌های موجود آنرا می‌پذیرند.
- ولی IP برای اینکار طرح نشده بلکه برای سوار کردن TCP روی آن طرح شده.
- در Packet LANهای کوچک همراه ۱۳ تا Reader ارسال و دریافت می‌شود که فقط سه تا (فرستنده، گیرنده، طول) مهمند.
- راه دیگر ایجاد پروتکل خاص RPC است که مسائل طراحی، پیاده‌سازی و اجرای آن زیاد است و همینطور قبولش.
- موضوع دیگر برای جلوگیری از Over head، بزرگ کردن اندازه Packet است (8k در Sun و 1536 بایت Ethernet).
- بالا سری لازم برای شکستن پیام‌ها

فصل دوم: ارتباطات در سیستم‌های توزیع شده (ادامه)

- **Acknowledgements** (اعلام وصول)
- aclc برای هر پاکت شکل ۲-۲۵ ص ۸۷ یا یک aclc برای کل پیغام مزایا و معایب هر کدام.
- پروتکل **Stop twat** یا **Selective Repent Blast** که سومی مشکل‌تر است در پیاده‌سازی (چک کردن کدام ..... شده در خواست مجدد).
- در **LAN** معمولاً مشکلی نیست ولی در **WAN** هست از نظر گم شدن پاکت‌ها.
- Chipها قدرت دریفات پی در پی پاکت‌ها رلا ندارند وقتی پرشوند از کار می‌افتند آنگاه پاکت‌ها گم می‌شوند و به این **Overran Error** گویند که خیلی بیشتر از گم شدن بدلیل **Noise** اتفاق می‌افتد.
- این خطا در شکل ۲-۲۵ ص ۸۷ وجود دارد ولی کم سرعتش می گذارند.

فصل دوم: ارتباطات در سیستم‌های توزیع شده (ادامه)

- یک راه قبول مسئله و انتظار مشغول است (چند صدمتکروثانیه) بین پاکت‌ها تا رسیدن آن است یا انجام کار دیگر.
- اگر بافر وجود دراد بعد از هر n پاکت مدتی منتظر شود یا اینکه aclc بگیرد.
- بهتر است پروتکل قابل تنظیم با امکانات شبکه باشد (انعطاف‌پذیری نسبت به IP و UDP).
- در شکل ۱۶-۲ ص ۶۷ گم شود Server جواب را نه می‌دارد ولی کار از نظر Client تمام شده است.
- در پروتکل جدید باید Server اگر تا یک Δt نتوانست aclc بگیرد Timeout شود.

فصل دوم: ارتباطات در سیستم‌های توزیع شده (ادامه)

- مسیر بحرانی
- به مراحل انجام RPC می گویند شکل ۲۶-۲ ص ۸۹ (شرح کامل مراحل داده شود).
- اخذ بافر به دو صورت است: یکی ساختن یک بافر بطول ثابت و دیگری انتخاب یک بافر از بین یک Pool.
- سئوال مهم: بیشترین زمان در چه بخش از مسیر بحرانی است.
- نتایج یک تحقیق روی ماشین خاص DEC با ۵ پروسسور VAX، Pool از بافرهای UDP، حافظ مشترک بین Kernal و User و روتین‌های نوشته شده به اسمبلی در شکل ۲۷-۲ ص ۹۱ آمده.
- این فقط یک مثال است با امکانات فوق: مالتی پروسسور، مجموعه بافرهای UDP، حافظ مشترک بین Kernal و User برنامه‌نویسی RPC به اسمبلی با دست.

فصل دوم: ارتباطات در سیستم‌های توزیع شده (ادامه)

- کپی کردن
- بدلیل وجود حافظه مشترک دو مثال بالا کپی کردن وجود نداشت هولی در سایر موارد بین یک تا ۸ بار کپی داریم.
- با 500nsec برای ۳۲ بیت برای ۸ کپی برای هر کلمه ۴ میکروثانیه داریم که می‌شود 1Mbyte/sec بدون توجه به سرعت شبکه، در عمل رسیدن به یکدهم این هم خوب است.
- یک روش سخت‌افزاری حذف کپی روش Scatter-Gather است که می‌تواند پاکت رسیده را به چند بخش تقسیم کند و هر قسمت را به یک بافر حافظه بفرستد و به همین ترتیب هم جمع کند Headerها جدا در Kernel و بدنه پیام را جدا در .Stub
- Reuse کردن Header ها در Kernel.
- حذف کپی در مبدأ راحت‌تر است تا در مقصد زیرا حداقل کار فهمیدن این است که پیام مال کدام Server است پس باید کپی شود به Server.
- در صفحه‌بندی اگر پاکت یک صفحه باشد می‌توان ثباتهای مربوط را طوری Set کرد که Kernel و Stub به یکجا اشاره کنند.
- مسائلی دارد که ممکن است صرف نکند.

فصل دوم: ارتباطات در سیستم‌های توزیع شده (ادامه)

- مدیریت Timer
- همیشه امکان گم‌شدن پیام هست برای همین روش Timeout بکار می‌رود. (توضیح)
- خود نگهداری و مدیریت همه Timerها وقت‌گیر است شکل ۲-۲۸a ص ۹۵ لیست مرتب شده تایمرها.
- اکثر تامیرها Expire نشده و از لیست حذف می‌شوند یعنی تلاش بیهوده زیادی بوده است.
- مدت Timer out کوتاه = تعداد زیادی تلاش مجدد.
- مدت Timer out بلند = انتظار زیاد برای پیامهای گم‌شده.
- بجای لیست پیوندی می‌توان در جدول پروسس‌ها یک فیلد فرمان داشت شکل ۲-۲۸ b ص ۹۵ و وقتی تایمر نداریم آنرا صفر کرد. چک کردن این جدول سریعتر از حذف و اضافه در لیست پیوندی است. (الگوریتم Sweep)