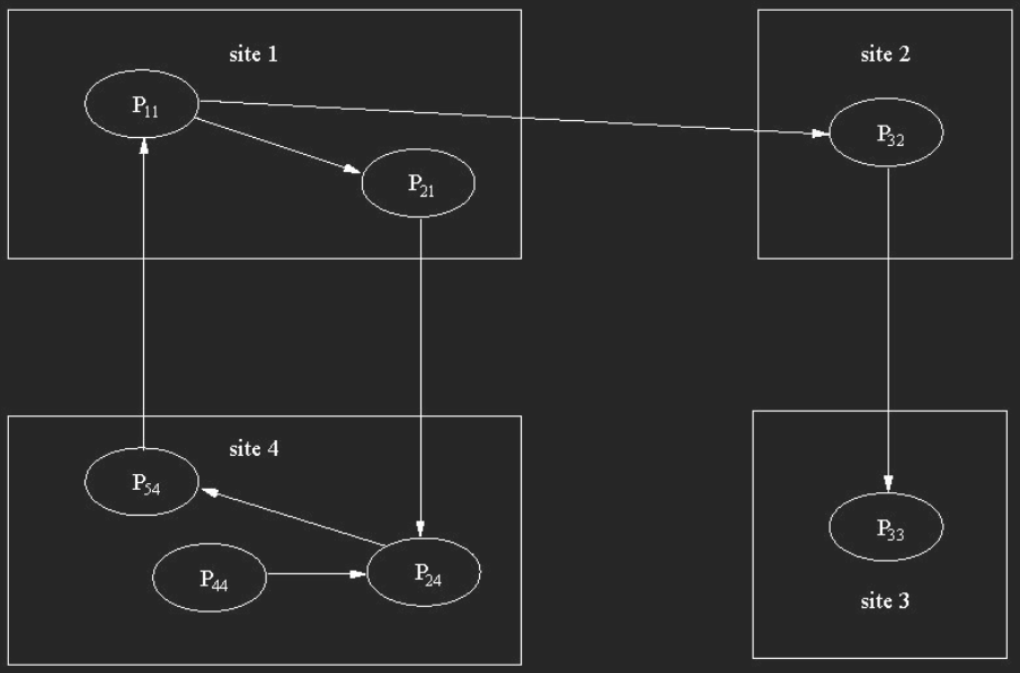


Figure 1: An Example of a WFG

Techniques for Handling Deadlocks

處理死鎖的技巧

- Note: No site has accurate knowledge of the current state of the system
  注意:沒有站點準確了解系統的當前狀態
- Techniques 技巧
  - Ignore it... 忽略它..
  - Deadlock Prevention (collective/ordered requests, preemption)
    死鎖預防(集體/有序請求,搶占)
    - Inefficient, impractical 效率低下,不切實際
  - Deadlock Avoidance 避免鎖死
    如果生成的全局系統狀態是安全的,則將資源授予進
    - A resource is granted to a process if the resulting global system state is safe
    - Requires advance knowledge of processes and their resource requirements
      需要提前了解流程及其資源需求
    - Impractical 不切實際的
  - Deadlock Detection and Recovery 死鎖檢測和恢復
    - Maintenance of local/global WFG and searching of the WFG for the presence of cycles (or knots), local/centralized deadlock detectors
      護本地/全球WFG並蒐索WFG的存在循環(或結),本地/集中式死鎖檢測器
    - Recovery by operator intervention, break wait-for dependencies, termination and rollback
      通過操作員干預恢復,中斷等待依賴,終止和回滾

Deadlock Detection: Correctness Criteria

死鎖檢測：正確性標準

- Safety (No false deadlocks)
  - The algorithm should not report deadlocks which do not exist (called *phantom or false deadlocks).*
- Progress (No undetected deadlocks)
  - The algorithm must detect all existing deadlocks in finite time. In other words, after all wait-for dependencies for a deadlock have formed, the algorithm should not wait for any more events to occur to detect the deadlock.

- 安全（無假死鎖）
  - 算法不應報告不存在的死鎖（稱為幻像或假死鎖）。
- 進度（沒有未檢測到的死鎖）
  - 算該算法必須在有限時間內檢測所有現有的死鎖。換句話說，
    在形成死鎖的所有等待依賴項之後，算法不應等待任何更多事件發生來檢測死鎖。

Global Deadlock Detection

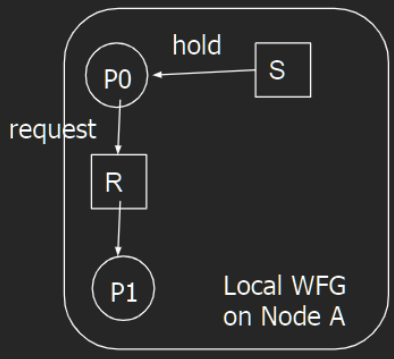
Centralized Algorithm

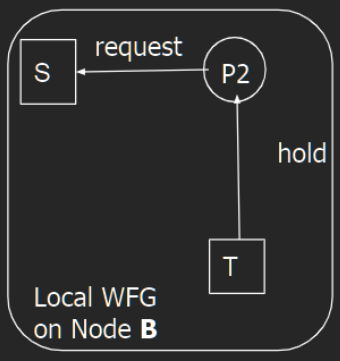
全局死鎖檢測 - 集中式算法

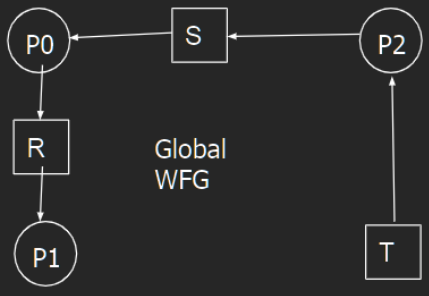
- Choose one site to be the *coordinator* 選擇一個站點作為協調器
- Each site maintains its own WFG 每個站點維護自己的 WFG
- Every site sends its WFG to *coordinator* 每個站點將其 WFG 發送到協調器
- *Coordinator* constructs a global WFG 協調器構建全局 WFG
  - Messages sent to *coordinator* when edges to (resource request/hold/release) WFG are added/deleted 添加/刪除（資源請求/保留/釋放）WFG 的邊緣時發送到協調器的消息
  - Can List be sent periodically? List可以定期發送嗎？
- *Coordinator* checks for cycle in global WFG 協調器檢查全局 WFG 中的循環
  - if yes, possibly deadlock 如果是，可能會鎖死
- *Coordinator* may choose victim 協調器可以選擇受害者

Global WFG - Example

全局 WFG - 範例





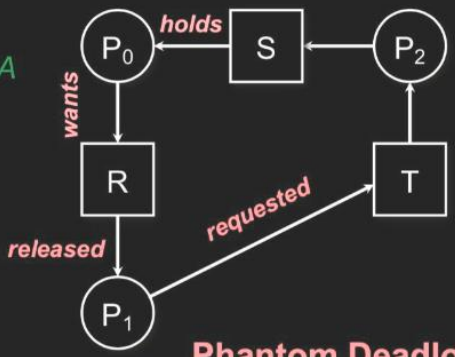


Centralized deadlock detection

Two events occur:
1. Process P_1 releases resource R on system A
2. Process P_1 asks system B for resource T

Two messages are sent to the coordinator:
1 (from A): *release R*
2 (from B): *wait for T*

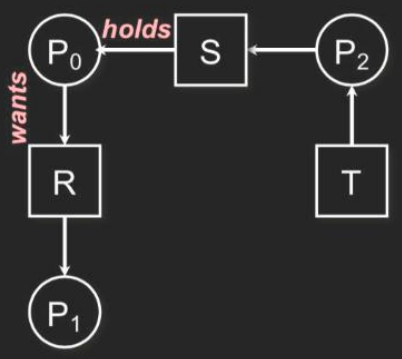
If message 2 arrives first, the coordinator constructs a graph that has a cycle and hence detects a deadlock. This is **phantom deadlock.**



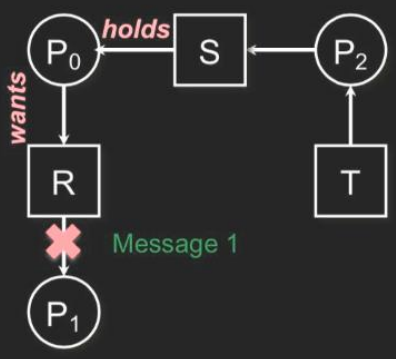
Phantom Deadlock

A *phantom deadlock* is sometimes known as a *false deadlock*

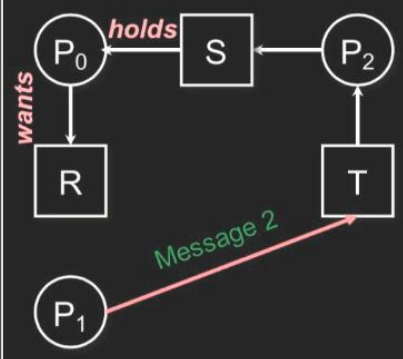
Phantom Deadlock Example



No deadlock



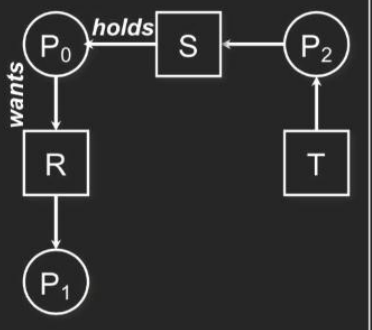
Message 1 from P_1:
release(R)



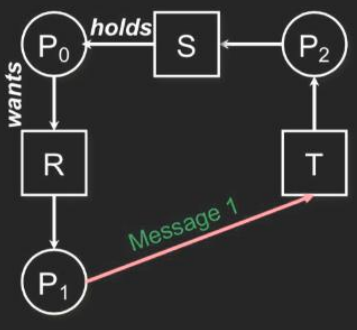
Message 2 from P_1:
wait_for(T)

All good: no deadlock detected!

Phantom Deadlock Example

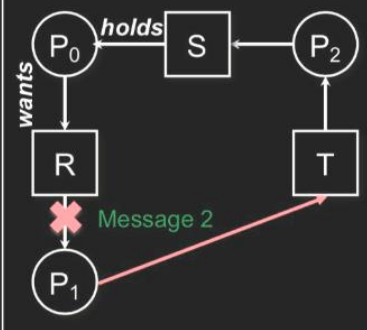


No deadlock



Message 2 from P_1:
wait_for(T)

DEADLOCK detected!
Do Something!



Message 1 from P_1:
release(R)

It really wasn't deadlock since P_1 released R
Too Late!

We detected deadlock because the coordinator received the messages out of order

Avoiding phantom deadlocks

避免幻象鎖死

- Impose globally consistent total ordering on all processes
- Coordinator asks each process if there are release messages

對所有流程施加全局一致的總排序
協調器詢問每個進程是否有發布消息

Deadlock Detection: 死鎖檢測: Models and Types of Algorithms 算法的模型和類型

- Multiple models 多種型號
  - Single-resource, AND, OR , AND-OR, P-out-of-Q model 單資源、AND、OR、AND、OR、P-出的-Q型號
- Classes of Deadlock Detection Algorithms 死鎖檢測算法類
  - Path-pushing 路徑推進
    - distributed deadlocks are detected by maintaining an explicit global WFG (constructed locally and pushed to neighbors) 通過維護一個明確的全局WFG來檢測分佈式死鎖(在本地構建並推送給鄰居)
  - Edge-chasing 邊緣追逐
    - the presence of a cycle in a distributed graph structure is verified by propagating special messages called probes, along the edges of the graph. The formation of cycles can be detected by a site if it receives the matching probe sent by it previously. 通過沿圖的邊緣傳播稱為探針的特殊消息來驗證分佈式圖結構中是否存在環。一個站點如果接收到它之前發送的匹配探針，就可以檢測到循環的形成。
  - Diffusion computation 擴散計算
    - deadlock detection computation is diffused through the WF 死鎖檢測計算通過WF擴散
  - Global state detection 全球狀態檢測
    - Take a snapshot of the system and examining it for the condition of a deadlock. 快照系統並檢查它是否存在死鎖情況。

Distributed Deadlock Detection: Chandy-Mishra-Haas Algorithm

Chandy-Misra-Haas algorithm

分佈式死鎖檢測8
Chandy-Mishra-Haas 算法

Edge Chasing

When requesting a resource, generate a probe message

- Send to all process(es) currently holding the needed resource
- Message contains three process IDs: *{blocked_ID, my_ID, holder_ID}*

1. Process that originated the message (*blocked_ID*)
2. Process sending (or forwarding) the message (*my_ID*)
3. Process to whom the message is being sent (*holder_ID*)

Distributed Deadlock Detection: Chandy-Mishra-Haas Algorithm

分佈式死鎖檢測8
Chandy-Mishra-Haas 算法

- When *probe* message arrives, recipient checks to see if it is waiting for any processes

- If so, update & forward message: *{blocked_ID, my_ID, holder_ID}*
  - Replace ***my_ID*** field by its own process ID
  - Replace ***holder_ID*** field by the ID of the process it is waiting for
  - Send messages to each process on which it is blocked

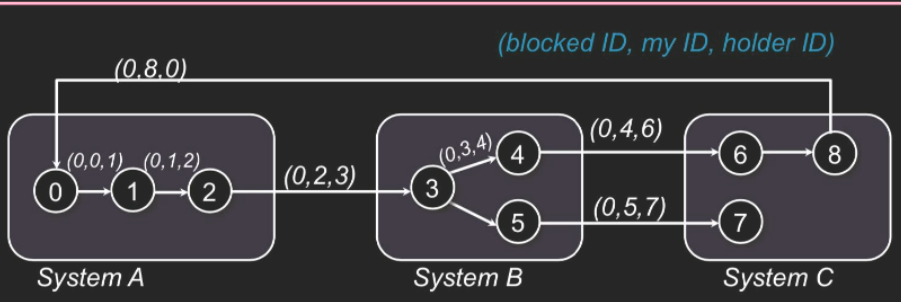
- If a message goes all the way around and comes back to the original sender, a cycle exists
  - *We have deadlock*

© 2014-2018 Paul Krzyzanowski

Distributed Deadlock Detection: Chandy-Mishra-Haas Algorithm

分佈式死鎖檢測：Chandy-Mishra-Haas 算法

Distributed deadlock detection



- Process 0 needs a resource process 1 is holding
- That means process 0 will block on process 1
  - Initial message from P_0 to P_1: *(0,0,1)*
  - P_1 sends *(0, 1, 2)* to P_2 ; P_2 sends *(0, 2, 3)* to P_3
- Message *(0,8,0)* returns back to sender
  - cycle exists: *deadlock*

Distributed deadlock prevention

- Deny circular wait
- Assign a unique timestamp to each transaction
- Ensure that the *Global Wait-For Graph* can only proceed from young to old or from old to young

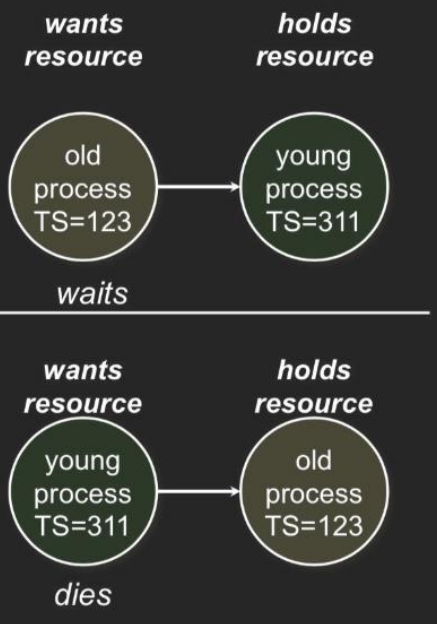
Deadlock prevention

- When a process is about to block waiting for a resource used by another
  - Check to see which has a larger timestamp (which is older)
- Allow the wait only if the waiting process has an older timestamp (is older) then the process waited for
- Following the resource allocation graph, we see that timestamps always have to increase, so cycles are impossible.
- Alternatively: allow processes to wait only if the waiting process has a higher (younger) timestamp than the process waiting for.

Wait-die algorithm

- Old process wants resource held by a younger process
  - old process waits
- Young process wants resource held by older process
  - young process kills itself

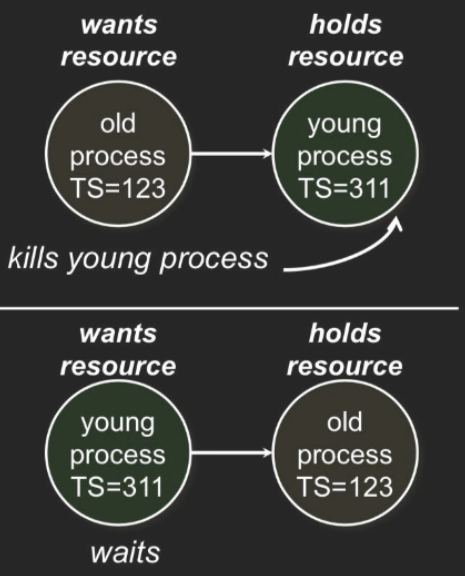
Only permit older processes to wait on resources held by younger processes.



Wound-wait algorithm

- Instead of killing the transaction making the request, kill the resource owner
- Old process wants resource held by a younger process
  - old process kills the younger process
- Young process wants resource held by older process
  - young process waits

Only permit younger processes to wait on resources held by older processes.



Distributed Process and Resource Management 分佈式流程和資源管理

- Need multiple other policies to determine when and where to execute processes in distributed systems – useful for load balancing, reliability
  需要多個其他策略來確定在分佈式系統中何時何地執行進程——對負載平衡、可靠性很有用
  - Load Estimation Policy
    - How to estimate the workload of a node
  - Process Transfer Policy
    - Whether to execute a process locally or remotely
  - Location Policy
    - Which node to run the remote process on
  - Priority Assignment Policy
    - Which processes have more priority (local or remote)
  - Migration Limiting policy
    - Number of times a process can migrate

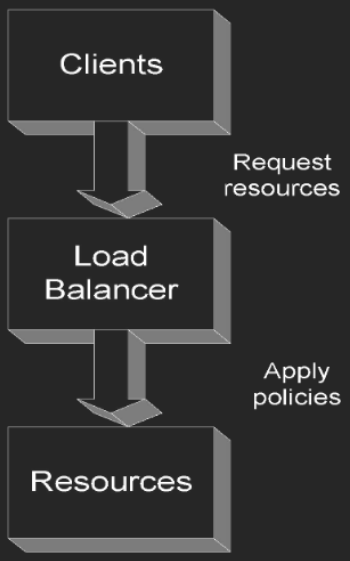
- 負載估算策略
  - 如何估計節點的工作量
- 流程轉移政策
  - 是在本地還是遠程執行進程
- 位置政策
  - 在哪個節點上運行遠程進程
- 優先分配政策
  - 哪些進程具有更高的優先級（本地或遠程）
- 遷移限制策略
  - 進程可以遷移的次數

Load Balancing

負載均衡

- Computer overloaded 電腦超載
  - Decrease load maintain scalability, performance, throughput - transparently 減少負載保持可擴展性、性能、吞吐量、透明
  - Load Balancing 負載均衡
    - Can be thought of as "distributed scheduling"
      - Deals with distribution of processes among processors connected by a network
    - Can also be influenced by "distributed placement"
      - Especially in data intensive applications

可以認為是"分佈式調度"
處理通過網絡連接的處理器之間的進程分佈
也可能受"分佈式佈局"的影響
特別是在數據密集型應用程序中



Load Balancer 負載均衡器
- Manages resources 管理資源
- Policy driven Resource assignment 政策驅動的資源分配

Load Balancing Issues

負載平衡問題

- How
  - To search for lightly loaded machines
- When
  - should load balancing decisions be made
  - to migrate processes or forward requests?
- Which
  - processes should be moved off a computer?
  - processor should be chosen to handle a given process or request
- What should be taken into account when making the above decisions? How should old data be handled
- Should
  - load balancing data be stored and utilized centrally, or in a distributed manner
  - What is the performance/overhead tradeoff incurred by load balancing
  - Prevention of overloading a lightly loaded computer

如何
要搜索輕載機
當
應負載均衡決定進行
遷移過程或轉發請求?
哪個
過程應關閉計算機感動?
處理器應選擇以處理給定過程或請求
在作出上述決定時應考慮哪些因素? 舊數據應該如何處理
應該
負載均衡數據集中存儲和使用，也可以分佈式存儲和使用
負載均衡帶來的性能/開銷權衡是什麼
防止輕載計算機過載

Static vs. dynamic
靜態與動態

- Static load balancing - CPU determined at process creation. •靜態負載平衡 - CPU 在進程創建時確定•
- Dynamic load balancing - processes dynamically migrate to other computers to balance the CPU (or memory) load. •動態負載平衡 - 進程動態遷移到其他計算機以平衡 CPU（或內存）負載•
  - **Parallel machines** - *dynamic balancing schemes seek to minimize total execution time of a single application running in parallel on a multiple nodes* •並行機 - 動態平衡方案尋求最小化在多個節點上並行運行的單個應用程序的總執行時間
  - **Web servers** - *scheduling client requests among multiple nodes in a transparent way to improve response times for interaction*
  - **Multimedia servers** - *resource optimization across streams and servers for QoS; may require admission control*

•Web 服務器 - 以透明的方式在多個節點之間調度客戶端請求，以提高交互的響應時間
•多媒體服務器 - 針對 QoS 跨流和服務器的資源優化；可能需要准入控制

Process Migration
進程遷移

- Process migration mechanism 進程遷移機制
  - Freeze the process on the source node and restart it at the destination node 凍結源節點上的進程並在目的節點重新啟動它
  - Transfer of the process address space 進程地址的傳輸
  - Forwarding messages meant for the migrant process 轉發消息用於移民過程
  - Handling communication between cooperating processes separated as a result of migration 處理因遷移而分離的協作進程之間的通信
  - Handling child processes 處理子進程
- Process Migration Policies: Depend on 進程遷移策略：取決於
  - CPU Load
  - Memory Requirements
  - I/O Activity -- Where is the physical device?
  - Communication - which processes communicate with each other?

•CPU 負載
•內存要求
•I/O Activity - 物理設備在哪裡？
•通信 - 哪些進程相互通信？

Process Migration and Heterogeneous Systems

流程遷移和異構系統

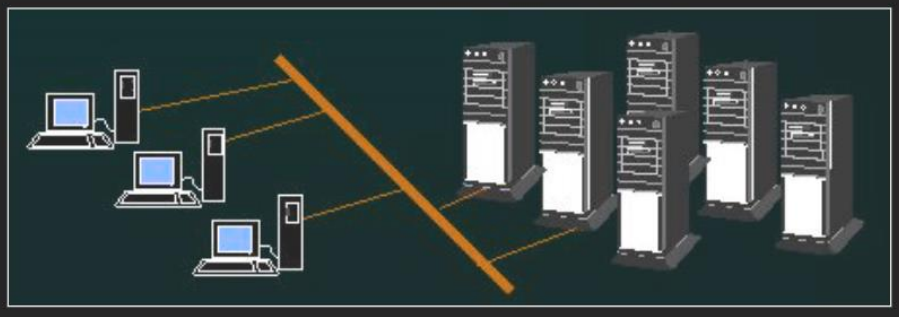
- Converts usage of heterogeneous resources (CPU, memory, IO) into a single, homogeneous cost using a specific cost function.
  使用特定的成本函數將異構資源（CPU、內存、IO）的使用轉換為單一、同質的成本。
- Assigns/migrates a job to the machine on which it incurs the lowest cost.
  將作業分配/遷移到成本最低的機器。
  - Can design online job assignment policies based on multiple factors - economic principles, competitive analysis.
  - Aim to guarantee near-optimal global lower-bound performance.

  可以根據多種因素設計在線工作分配政策——經濟原則、競爭分析。
  旨在保證近乎最佳的全局下限性能。

Mosix: Early Cluster Computing

Mosix：早期集群計算

- Mosix (from Hebrew U): Kernel level enhancement to Linux that provides dynamic load balancing in a network of workstations.
  Mosix（來自 Hebrew U）：Linux 的內核級增強，在工作站網絡中提供動態負載平衡。
- Dozens of PC computers connected by local area network (Fast-Ethernet or Myrinet).
  數十台 PC 計算機通過局域網（快速以太網或 Myrinet）連接。
- Any process can migrate anywhere anytime.
  任何進程都可以隨時隨地遷移。



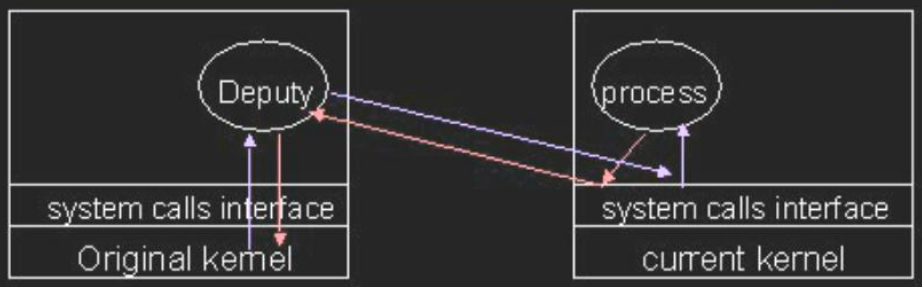
Architectures for Migration

遷移架構

Architecture that fits one system image.
Needs location transparent file system.

(Mosix early versions)

適合一個系統映像的架構。
需要位置透明的文件系統。



適合入口相關係統的架構。
基於當前的Unix更容易實現

Architecture that fits entrance dependant systems.
Easier to implement based on current Unix.

(Mosix later versions)

Mosix: Migration and File Access

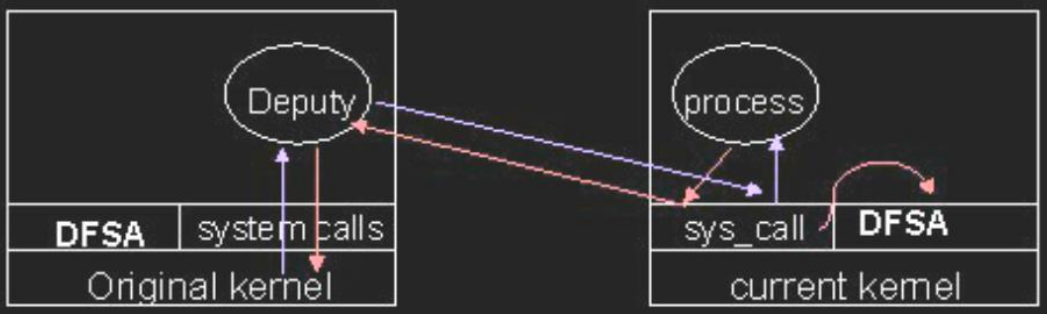
Mosix：遷移和文件訪問

Each file access must go back to deputy…

= = Very Slow for I/O apps.

每個文件的訪問必須回到副...
= =非常慢的I/O的應用程序。
解決方案：允許進程訪問的分佈式文件系統
通過當前內核。

Solution: Allow processes to access a distributed file system through the current kernel.



Distributed File Systems (DFS)

分佈式文件系統(DFS)

A distributed implementation of the classical file system model

經典文件系統模型的分佈式實現

- Requirements
  - Transparency: Access, Location, Mobility, Performance, Scaling
  - Allow concurrent access
  - Allow file replication
  - Tolerate hardware and operating system heterogeneity
  - Security - Access control, User authentication
- Issues
  - File and directory naming – Locating the file
  - Semantics – client/server operations, file sharing
  - Performance
  - Fault tolerance – Deal with remote server failures
  - Implementation considerations - caching, replication, update protocols

- 要求
  - 透明度：訪問、位置、移動性、性能、縮放
  - 允許並發訪問
  - 允許文件複製
  - 容忍硬件和操作系統異構
  - 安全——訪問控制、用戶認證
- 問題
  - 文件和目錄命名—定位文件
  - 語義——客戶端/服務器操作、文件共享
  - 性能
  - 容錯——處理遠程服務器故障
  - 實施注意事項：緩存、複製、更新協議

Issues: File and Directory Naming

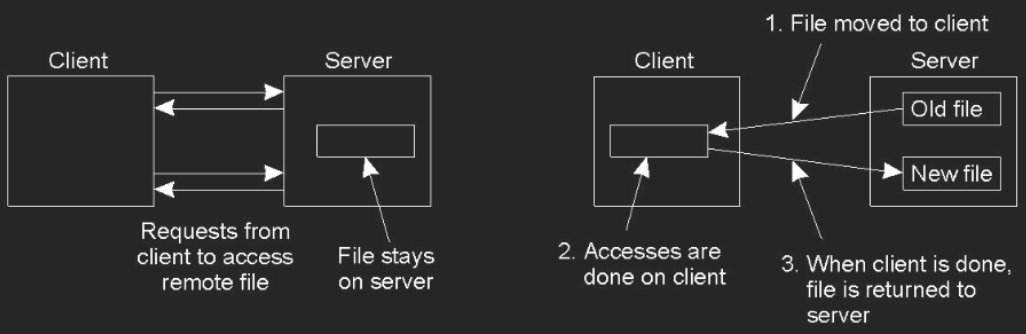
問題：文件和目錄的命名

- Explicit Naming
  - Machine + path /machine/path
    - one namespace but not transparent
- Implicit naming
  - Location transparency
    - file name does not include name of the server where the file is stored
  - Mounting remote filesystems onto the local file hierarchy
    - view of the filesystem may be different at each computer
  - Full naming transparency
    - A single namespace that looks the same on all machines

- 顯式命名
  - 機器+路徑/機器/路徑
  - 一個命名空間但不透明
- 隱式命名
  - 位置透明
  - 文件名不包括存放文件的服務器名稱
  - 將遠程文件系統掛載到本地文件層次結構上
  - 每台計算機的文件系統視圖可能不同
  - 完全命名透明
  - 在所有機器上看起來都相同的單個命名空間

Distributed File Access Alternatives

分佈式文件訪問替代方案



Remote access model

Download/upload model

Semantics – File Sharing

語義 - 文件共享

One-copy (Sequential) Semantics

一份複製（順序）語義

- Updates are written to the single copy and are available immediately

更新寫入單個副本並立即可用

- All clients see contents of file identically as if only one copy of file existed

所有客戶端都看到相同的文件內容，就好像只存在一個文件副本

- Issue : Performance

問題：性能

- Soln: Use Caching: after an update operation, no program can observe a discrepancy between data in cache and stored data
  - Extra state; extra traffic

Soln：使用緩存；更新操作後，沒有程序可以觀察到緩存中的數據和存儲的數據之間的差異

額外狀態：額外流量

Session semantics

會話語義

- more relaxed

更鬆散

- Copy file on open, work on local copy and copy back on close
- Changes initially only visible to file that modified it.

打開時復製文件，在本地工作，複製並在關閉時復制回來

更改最初僅對文件可見修改了它。

Serializable (Transaction semantics)

- Need file locking protocols implemented - share for read, exclusive for write).

可序列化（事務語義）

需要實現文件鎖定協議 - 共享讀取，獨占寫入）。

Semantics - Operational

語義 - 操作

- Support fault tolerant operation
  - At-most-once semantics for file operations
  - At-least-once semantics with a server protocol designed in terms of idempotent file operations
  - Replication (stateless, so that servers can be restarted after failure)

- 支持容錯操作
  - 文件操作的最多一次語義
  - 具有根據冪等文件操作設計的服務器協議的至少一次語義
  - 複製（無狀態，以便服務器可以在故障後重新啟動）

DFS Performance

DFS 性能

- Efficiency Needs
  - Latency of file accesses
  - Scalability (e.g., with increase of number of concurrent users)
- RPC Related Issues
  - Use RPC to forward every file system request (e.g., open, seek, read, write, close, etc.) to the remote server
    - Remote server executes each operation as a local request
    - Remote server responds back with the result
  - Advantage:
    - Server provides a consistent view of the file system to distributed clients.
  - Disadvantage:
    - Poor performance
    - Solution: Caching

- 效率需求
  - 文件訪問延遲
  - 可擴展性（例如，隨著並發用戶數的增加）
- RPC 相關問題
  - 使用 RPC 將每個文件系統請求（例如，打開、查找、讀取、寫入、關閉等）轉發到遠程服務器
  - 遠程服務器將每個操作作為本地請求執行
  - 遠程服務器返回結果
- 優勢：
  - 服務器為分佈式客戶端提供一致的文件系統視圖。
- 缺點：
  - 性能不佳
  - 解決方案：緩存

EXTRA Slide

額外的幻燈片

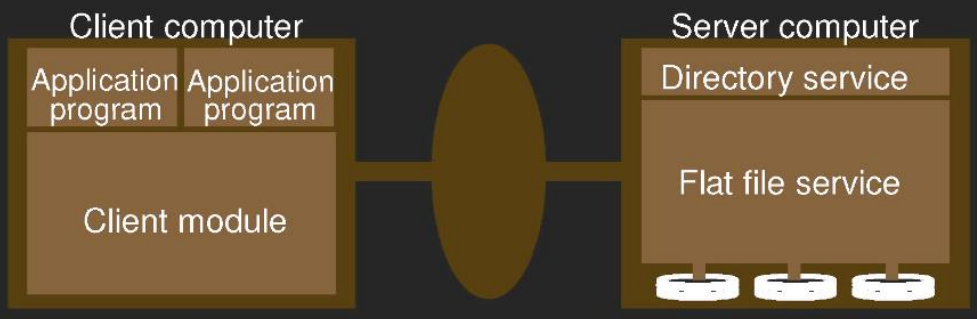
Traditional File system Operations

傳統文件系統操作

- filedes = *open*(name, mode) Opens an existing file with the given name.
- filedes = *creat*(name, mode) Creates a new file with the given name.
- Both operations deliver a file descriptor referencing the open file. The mode is read, write or both. 這兩個操作都提供一個引用打開文件的文件描述符。模式是讀、寫或兩者兼而有之。
- status = *close*(filedes) Closes the open file filedes. close(filedes) 關閉打開的文件 filedes
- count = *read*(filedes, buffer, n) Transfers n bytes from the file referenced by filedes to buffer. 將 filedes 引用的文件中的 n 個字節傳輸到緩衝區
- count = *write*(filedes, buffer, n) Transfers n bytes to the file referenced by filedes from buffer. write(filedes, buffer, n) 將 n 個字節從 buffer 傳輸到 filedes 引用的文件
- Both operations deliver the number of bytes actually transferred and advance the read-write pointer. 這兩個操作都傳遞實際傳輸的字節數並推進讀寫指針。
- pos = *lseek*(filedes, offset, whence) Moves the read-write pointer to offset (relative or absolute, depending on whence). pos lseek(filedes, offset, wherece) 將文學療法移動到 offset（相對或絕對，何時=何時）。
- status = *unlink*(name) Removes the file name from the directory structure. If the file has no other names, it is deleted. status = unlink(name) 從目錄結構中刪除文件名。如果文件沒有其他名稱，則將其刪除。
- status = *link*(name1, name2) Adds a new name (name2) for a file (name1).
- status = *stat*(name, buffer) Gets the file attributes for file name into buffer.

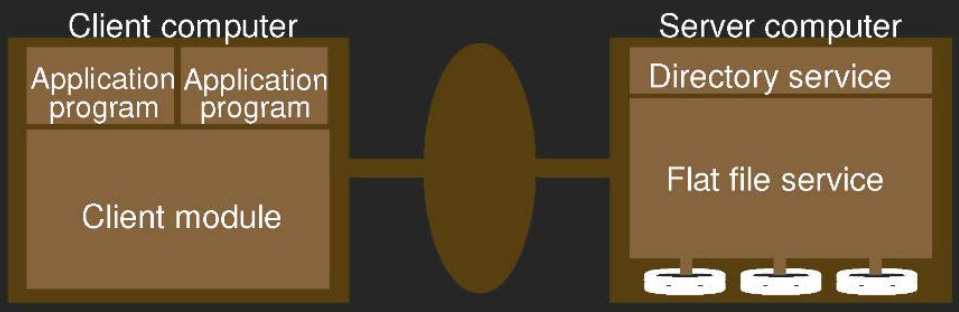
status = link(name1, name2) 為文件 (name1) 添加新名稱 (name2)。
status = stat(name, buffer) 獲取文件名的文件屬性到緩衝區。

Architecture



- Flat File Service
  - Performs file operations
  - Uses "unique file identifiers" (UFIDs) to refer to files
  - Flat file service interface
    - → RPC-based interface for performing file operations
    - → Not normally used by application level programs

Architecture (2)



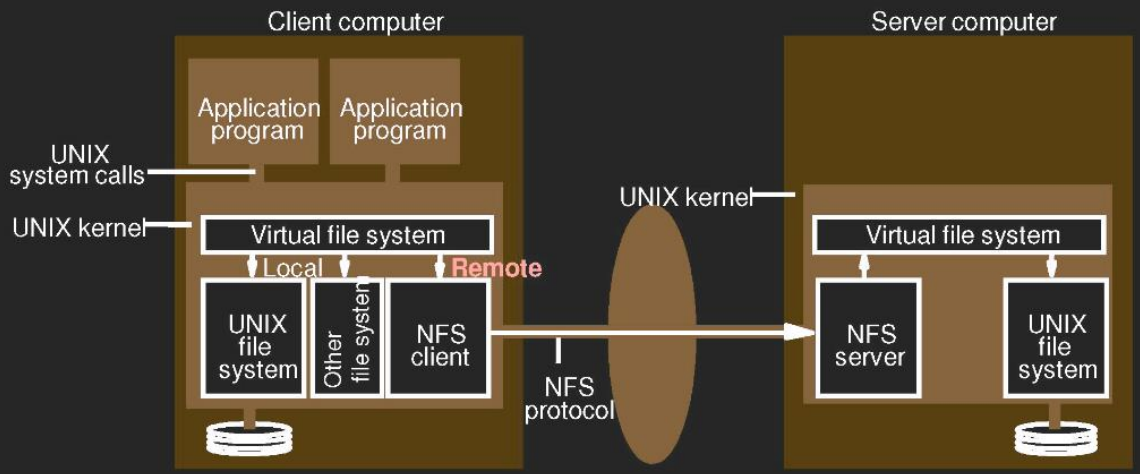
- Directory Service
  - Mapping of UFIDs to "text" file names, and vice versa
- Client Module
  - Provides API for file operations available to application program

Network File Systems : Sun-NFS

- **Supports heterogeneous systems**
  - Architecture
    - Server exports one or more directory trees for access by remote clients
    - Clients access exported directory trees by mounting them to the client local tree
    - Diskless clients mount exported directory to the root directory
  - Protocols
    - Mounting protocol
    - Directory and file access protocol - stateless, no open-close messages, full access path on read/write
  - Semantics - no way to lock files

網絡文件系統：
- Sun-NFS
  - 支持異構系統
    - 架構
      - 服務器導出一棵或多棵目錄樹供遠程客戶端訪問
      - 客戶端通過將導出的目錄樹掛載到客戶端本地樹來訪問它們
      - 無盤客戶端掛載導出目錄到根目錄
    - 協議
      - 掛載協議
      - 目錄和文件訪問協議 - 無狀態、無打開-關閉消息、讀/寫時的完整訪問路徑
    - 語義 - 無法鎖定文件

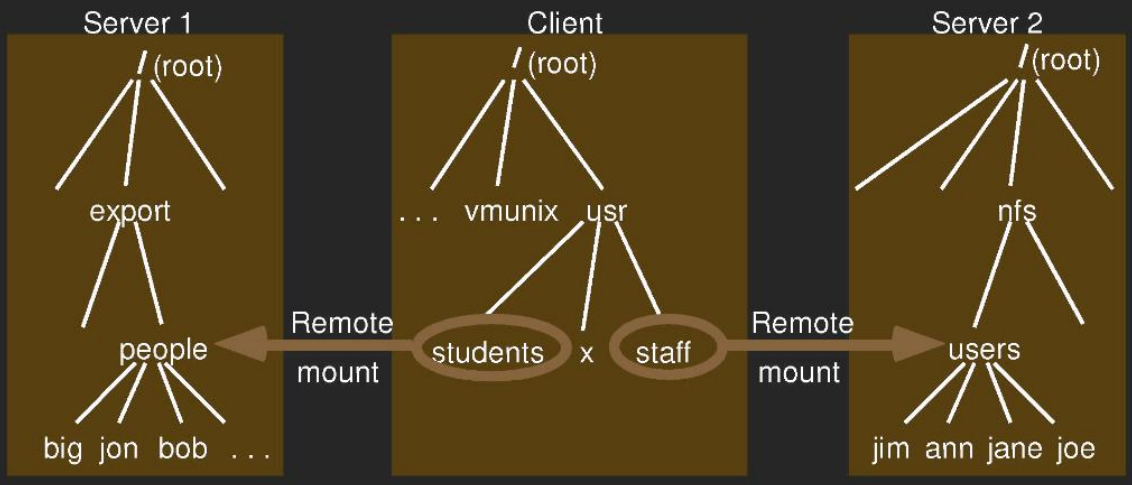
Sun Network File System



Mounting of File Systems

- Making remote file systems available to a local client, specifying remote host name and pathname
- Mount protocol (RPC-based)
  - Returns file handle for directory name given in request
  - Location (IP address and port number) and file handle are passed to Virtual File System and NFS client
- Hard-mounted (mostly used in practice)
  - User-level process suspended until operation completed
  - Application may not terminate gracefully in failure situations
- Soft-mounted
  - Error message returned by NFS client module to user-level process after small number of retries

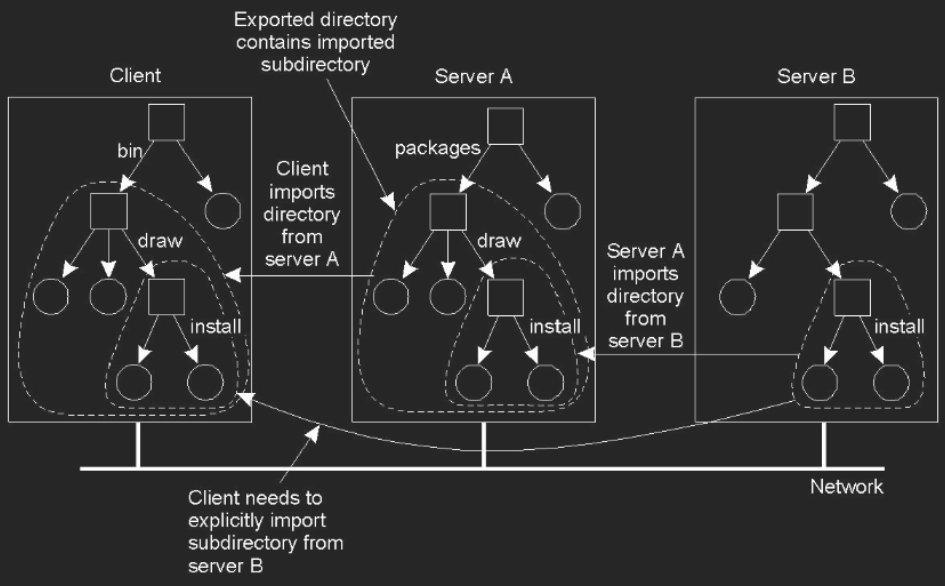
Mounting Example



The file system mounted at */usr/students* in the client is actually the sub-tree located at */export/people* in Server 1; the file system mounted at */usr/staff* in the client is actually the sub-tree located at */nfs/users* in Server 2.

Naming (2)

- Mounting nested directories from multiple servers in NFS.



Example 2 : Andrew File System

Example 2 : Andrew File System

- Supports information sharing on a large scale
  - 10,000+ clients
- Most files are small
  - reads more common than writes; typically one user at a time; access locality expected.
- Uses a session semantics
  - Entire file is copied to the local machine (Venus) from the server (Vice) when open. If file is changed, it is copied to server when closed.
    - Works because in practice, most files are changed by one person
- AFS File Validation (older versions)
  - On open: Venus accesses Vice to see if its copy of the file is still valid. Causes a substantial delay even if the copy is valid.
  - Vice is stateless

支持大規模信息共享

10,000+ 客戶

大多數文件都很小

讀比寫更常見；通常一次一個用戶；預期訪問地點。

使用會話語義

整個文件在打開時從服務器（Vice）複製到本地機器（Venus）。如果文件被更改，則在關閉時將其複製到服務器。

有效，因為在實踐中，大多數文件由一個人更改

AFS 文件驗證（舊版本）

打開時：Venus 訪問 Vice 以查看其文件副本是否仍然有效。即使副本有效，也會導致大量延遲。

Vice 是無國籍的

Example 3: The Coda Filesystem

示例 3：Coda 文件系統

- CoDA -- Constant Data Availability CoDA -- 恆定的數據可用性
- Descendant of AFS that is substantially more resilient to server and network failures. AFS 的後代，它對服務器和網絡故障的恢復能力要強得多。
- General Design Principles 一般設計原則
  - know the clients have cycles to burn, cache whenever possible, exploit usage properties, minimize system wide change, trust the fewest possible entries and batch if possible 知道客戶端有周期要刻錄，盡可能緩存，利用使用屬性，最小化系統範圍的變化，盡可能信任盡可能少的條目和批處理
- Directories are replicated in several servers (Vice) 目錄在多台服務器上複製（副）
- Support for mobile users 支持移動用戶
  - When the Venus is disconnected, it uses local versions of files. When Venus reconnects, it reintegrates using optimistic update scheme. 當 Venus 斷開連接時，它使用本地版本的文件。當金星重新連接時，它使用樂觀更新方案重新整合。

Other DFS Challenges

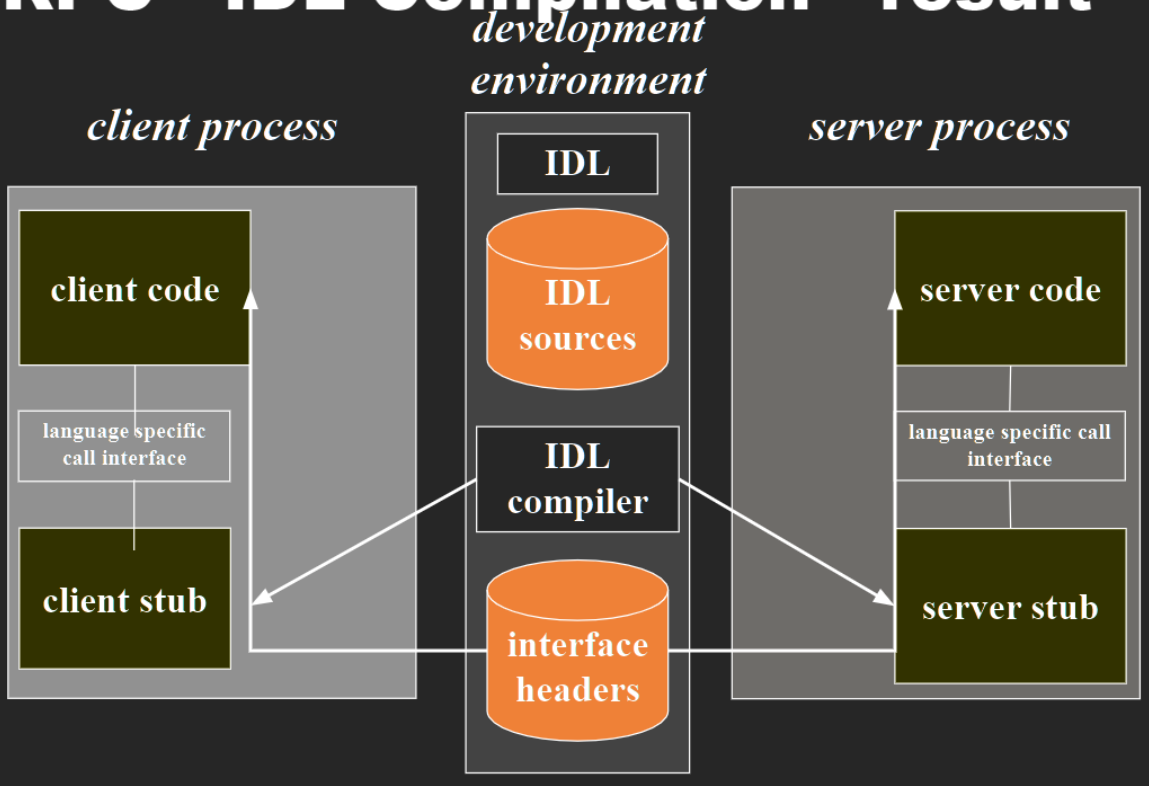
- Naming
  - Important for achieving location transparency
  - Facilitates Object Sharing
  - Mapping is performed using directories. Therefore name service is also known as *Directory Service*
- Security
  - Client-Server model makes security difficult
  - Cryptography based solutions

其他 DFS 挑戰
- 命名
  - 對於實現位置透明很重要
  - 促進對象共享
  - 映射是使用目錄進行的。因此名稱服務也稱為目錄服務
- 安全性
  - 客戶端-服務器模型使安全變得困難
  - 基於密碼學的解決方案

How Stubs are Generated

- Through a compiler
  - e.g. DCE/CORBA IDL – a purely declarative language
  - Defines only types and procedure headers with familiar syntax (usually C)
- It supports
  - Interface definition files (.idl)
  - Attribute configuration files (.acf)
- Uses Familiar programming language data typing
- Extensions for distributed programming are added

RPC - IDL Compilation - result



RPC NG: DCOM & CORBA

- Object models allow services and functionality to be called from distinct processes
- DCOM/COM+(Win2000) and CORBA IIOP extend this to allow calling services and objects on different machines
- More OS features (authentication,resource management,process creation,…) are being moved to distributed objects.

Sample RPC Middleware Products

- JaRPC (NC Laboratories)
  - libraries and development system provides the tools to develop ONC/RPC and extended .rpc Client and Servers in Java
- powerRPC (Netbula)
  - RPC compiler plus a number of library functions. It allows a C/C++ programmer to create powerful ONC RPC compatible client/server and other distributed applications without writing any networking code.
- Oscar Workbench (Premier Software Technologies)
  - An integration tool. OSCAR, the Open Services Catalog and Application Registry is an interface catalog. OSCAR combines tools to blend IT strategies for legacy wrappering with those to exploit new technologies (object oriented, internet).
- NobleNet (Rogue Wave)
  - simplifies the development of business-critical client/server applications, and gives developers all the tools needed to distribute these applications across the enterprise. NobleNet RPC automatically generates client/server network code for all program data structures and application programming interfaces (APIs)— reducing development costs and time to market.
- NXTWare TX (eCube Systems)
  - Allows DCE/RPC-based applications to participate in a service-oriented architecture. Now companies can use J2EE, CORBA (IIOP) and SOAP to securely access data and execute transactions from legacy applications. With this product, organizations can leverage their current investment in existing DCE and RPC applications

Some recent views on RPC

RPC vs. REST

http://steve.vinoski.net/pdf/IEEE-Convenience_Over_Correctness.pdf

New messaging formats
-- Google ProtoBuf
-- Apache Thrift (Facebook)

Protocol Buffer

lóhika

- Designed ~2001 because everything else wasn't that good those days
- Production, proprietary in Google from 2001-2008, open-sourced since 2008
- Battle tested, very stable, well trusted
- Every time you hit a Google page, you're hitting several services and several PB code
- PB is the glue to all Google services
- Official support for four languages: C++, Java, Python, and JavaScript
- Does have a lot of third-party support for other languages (of highly variable quality)
- Current Version - protobuf-2.4.1
- BSD License

protobuf
Protocol Buffers - Google's data interchange format

Apache Thrift

- Designed by an X-Googler in 2007
- Developed internally at Facebook, used extensively there
- An open Apache project, hosted in Apache's Inkubator.
- Aims to be the next-generation PB (e.g. more comprehensive features, more languages)
- IDL syntax is slightly cleaner than PB. If you know one, then you know the other
- Supports: C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, Node.js, Smalltalk, OCaml and Delphi and other languages
- Offers a stack for RPC calls
- Current Version - thrift-0.8.0
- Apache License 2.0

EXTRA Slide

Selected NFS Operations

lookup(dirfh, name) -> fh, attr	Returns file handle and attributes for the file *name* in the directory *dirfh*.
create(dirfh, name, attr) -> newfh, attr	Creates a new file name in directory *dirfh* with attributes *attr* and returns the new file handle and attributes.
remove(dirfh, name) status	Removes file name from directory *dirfh*.
getattr(fh) -> attr	Returns file attributes of file *fh*. (Similar to the UNIX *stat* system call.)
setattr(fh, attr) -> attr	Sets the attributes (mode, user id, group id, size, access time and modify time of a file). Setting the size to 0 truncates the file.
read(fh, offset, count) -> attr, data	Returns up to *count* bytes of data from a file starting at *offset*. Also returns the latest attributes of the file.
write(fh, offset, count, data) -> attr	Writes *count* bytes of data to a file starting at *offset*. Returns the attributes of the file after the write has taken place.
rename(dirfh, name, todirfh, toname) -> status	Changes the name of file *name* in directory *dirfh* to *toname* in directory to *todirfh*
link(newdirfh, newname, dirfh, name) -> status	Creates an entry *newname* in the directory *newdirfh* which refers to file *name* in the directory *dirfh*.

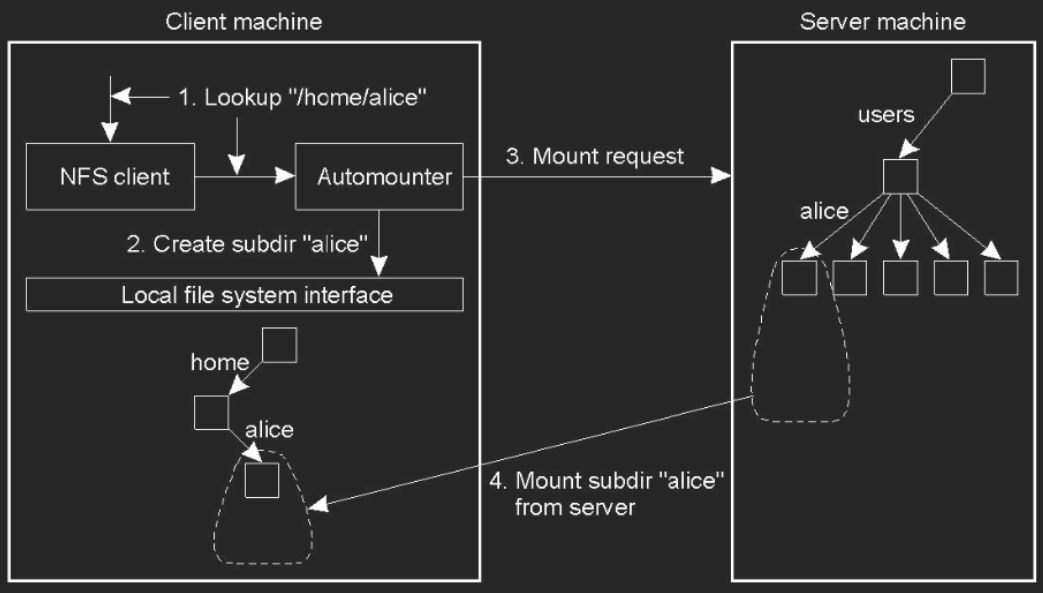
EXTRA Slide

Selected NFS operations (2)

symlink(newdirfh, newname, string) -> status	Creates an entry *newname* in the directory *newdirfh* of type symbolic link with the value *string*. The server does not interpret the *string* but makes a symbolic link file to hold it.
readlink(fh) -> string	Returns the string that is associated with the symbolic link file identified by *fh*.
mkdir(dirfh, name, attr) -> newfh, attr	Creates a new directory *name* with attributes *attr* and returns the new file handle and attributes.
rmdir(dirfh, name) -> status	Removes the empty directory *name* from the parent directory *dirfh*. Fails if the directory is not empty.
readdir(dirfh, cookie, count) -> entries	Returns up to *count* bytes of directory entries from the directory *dirfh*. Each entry contains a file name, a file handle, and an opaque pointer to the next directory entry, called a *cookie*. The *cookie* is used in subsequent *readdir* calls to start reading from the following entry. If the value of *cookie* is 0, reads from the first entry in the directory.
statfs(fh) -> fsstats	Returns file system information (such as block size, number of free blocks and so on) for the file system containing a file *fh*.

EXTRA Slide

Automounting (1)



A simple automounter for NFS.