

### 第四讲 分布式体系结构基础技术

#### 一、基于消息传递的通信

##### 分布式进程的通信

- 发送原语(send primitive)
  - 格式: send(dest,&mptr)
  - 参数: dest--把消息发送具有dest标识符的进程, mptr - - 表示要发送的消息的地址。
  - 作用: 将指定的消息发送到接收进程
- 接发原语(receive primitive)
  - 格式: receive(addr,&mptr)
  - 参数:addr--等待消息到达的接收进程的地址, mptr - - 指针, 表标消息到后存放何处。
  - 作用: 用地址addr侦听, 所需的消息。
- 阻塞(Blocking)原语
  - 阻塞发送原语(Blocking send primitive)
    - 当一个进程调用一个send原语时, 它指定一个目标进程和一个缓冲区, 调用send后, 发送进程便处于阻塞状态, 直到消息完全发送完毕, send原语的后继指令才能继续执行。
  - 阻塞接收原语 (Blocking receive primitive)
    - 当一个进程调用receive原语时, 并不返回控制, 而是等到把消息实际接收下来并把它放入参数mptr所指示的缓冲区后, 才返回控制, 并执行该原语的后继指令。在这段时间里进程处于阻塞状态。
  - 特点
    - 通信可靠,缓冲区可以重复使用;
    - CPU空等, 不具有并行性。
  - 阻塞原语也称同步原语(synchronous primitive)
- 非阻塞原语(Nonblocking primitive)
  - 非阻塞send原语
    - 如果send原语是非阻塞的, 它在消息实际发送之前, 就立即把控制返回给调用进程, 即发送进程在发送消息时并不进入阻塞状态, 它不等消息发送完成就继续执行其后继指令。
  - 非阻塞原语的特点
    - 提高系统的并行性: 发送进程可在消息实际发送过程中进行连续的并行计算, 而不是让C P U等待。
    - 缓冲区只能使用一次: 发送进程在消息发送完之前, 进程不能修改缓冲区, 在消息传送期间进程改写消息缓冲区可能引起可怕后果, 而何时允许使用缓冲区, 却无法得知。这样缓冲区只能使用一次。
  - 非阻塞receive原语
    - 当进程调用receive原语时, 便告诉内核缓冲区的位置, 内核立即返回控制, 调用者不阻塞状态。
  - 非阻塞原语的两种改进方法
    - 带拷贝的非阻塞send原语 (Nonblocking with copy to kernel)
      - 让内核把消息拷贝到内核的内部缓冲区, 然后允许进程继续运行。
      - 特点:
        - 优点: 调用进程一旦得到控制, 就可以使用缓冲区; 消息是否发送完成, 对发送进程没有影响。
        - 缺点: 每个被发送的消息都要从用户空间拷贝到内核空间 (额外拷贝), 使系统的性能大大降低其后, 网络接口将消息拷贝到硬件传送的缓冲区)
    - 带中断非阻塞send原语 (Noblocking with interrupt)
      - 含义
        - 当消息发送完毕后, 中断发送进程, 通知发送进程此时缓冲区可用。
      - 特点
        - 优点: 不需要消息的拷贝, 从而节省时间; 高效且提高了并行性。
        - 缺点: 由于用户中断, 使编程的难度增加; 程序不能再现, 给程序调试带来困难; 基于中断的程序很难保证正确且出错后几乎不能跟踪。
  - 非阻塞原语也称异步原语(asynchronous primitive)
- 非缓冲原语(unbuffered primitives)
  - 含义
    - 系统提供一个消息缓冲区, 用于存放到来的消息。接收进程调用receive(addr,&mptr)时, 机器用地址addr在网上侦听, 当消息到来后, 内核将它拷贝到缓冲区, 并将唤醒接收进程。
  - 非缓冲通信原语 - 抛弃非期望消息 (discard the unexpected message)
    - 含义

- receive(addr,&mptr)告诉它所在的机器上的内核，用地址addr 侦听，并准备去接收发送给addr的一个消息，当有消息到达后，如果非该地址的消息则抛弃，否则接收进程的内核将它拷贝到mptr所指示的缓冲区，并唤醒该接收进程。
- **特点**
  - 一个地址联系一个进程；
  - 若服务器调用在客户调用send之前，非常有效。
  - 如果send原语在receive之前已经完成，则服务器内核就不知道到达的消息拷贝何处。只好抛弃这个非期望的消息，并采用超时重传机制。
  - 假定有两个或更多客户使用同一个服务器，当服务器从他们之中的一个客户收到一个消息后，在它未完成处理之工作之前，不再用这个地址去侦听发送，在它返回到循环顶部重新调用receive之前，由于要进行相应的处理工作需要花费一些时间而其他客户可能多次尝试发送消息给它，并且有些客户可能放弃。在这种情况下，客户是否继续重发，取决于重发计时器的值以及他们的忍耐程度。
  - **消息丢失的机会大。**
- **非缓冲通信原语 - 消息保留一段时间(keep incoming messages for a little while)**
  - **含义**
    - 让接收内核对传送来的消息保留一段时间，以便使得一个合适的receive原语能很快地接收：**一个非期望的消息到来时，就启动一个计时器，如果计时器在走完之前仍无合适的receive出现，就将该消息丢弃。**
  - **特点**
    - 减少消息丢失的机会；
    - 如何存储和管理过早到达的消息存在问题。
- **缓冲通信原语 - 信箱(mailbox)**
  - 一个希望接收消息的进程要告诉内核为其创建一个信箱，并在网络消息包中指定一个可供查询的地址，然后，包含这个地址的所有到达的消息都被放进信箱。现在调用receive就可以从信箱中取出一个消息，如果没有消息就阻塞（假定用阻塞原语）。
  - **特点**
    - 解决了消息丢失和客户放弃消息所引起的一系列问题；
    - 信箱是有限的，有可能装满而造成消息的丢失。
- **不可靠(Unreliable)原语**
  - **不可靠send原语**指：系统不保证消息能被对方正确接收，要想实现可靠的通信，由用户自己完成。
- **可靠(Reliable)原语**
  - **REQ-ACK-REP-ACK**
    - 客户向服务器发送一个请求(REQ)，服务器对这一请求由内核向客户机内核返回一个确认（ACK），当客户机内核接收到这一确认后，释放这一用户进程。
    - **说明**
      - 确认(ACK) 只在内核间传递，客户和服务器进程都看不到这样的过程；
      - 服务器到客户的应答也存在REP-ACK的过程。
  - **REQ-REP-ACK**
    - 客户在发送一个消息后被阻塞，服务器内核并不返回一个确认(ACK)，而是利用应答(REP)本身作为一个确认(ACK)，客户如果等待时间太长，其内核可重发一个请求，以防请求消息丢失。客户在收到应答(REP)后，向服务器发送一个确认(ACK)，服务器在发送应答后处于阻塞状态，直到确认(ACK from client's kernel)，服务器也可以采用超时重传机制。
  - **REQ-(ACK-)REP-ACK**
    - **是一种折衷的方案：** 当一个请求到达服务器内核时，就启动一个定时器，如果服务器送回应答足够快(计时器时间未到)则应答(REP)充当一个确认(ACK)；如果定时器时间到就送一个单独的确认(ACK)。于是大多数情况下，仅需要两条消息：请求和应答，在复杂情况下，需要另一个确认(Acknowledgement)消息。