计算机与操作系统
第六章 并发程序设计
6.5 进程通信

葛季栋
南京大学软件学院

6.5 进程通信

6.5.1 进程通信

6.5.2 高级进程通信机制

6.5.1 进程通信

(1) 进程直接通信

(2) 进程间接通信

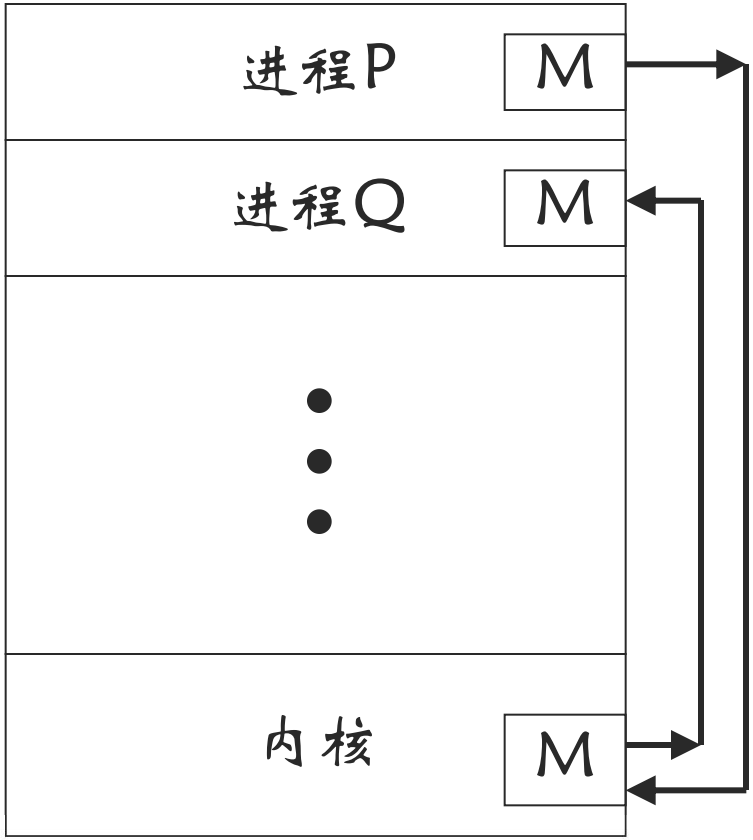
6.5.1 进程通信(消息传递)

- 当进程互相交互时，必须满足两个基本要求：同步和通信
  - 为实施互斥，进程间需要同步
  - 为了协作，进程间需要交换信息
- 消息传递提供了这些功能，最典型的消息传递原语
  - send 发送消息的原语
  - receive 接收消息的原语

(1) 进程直接通信

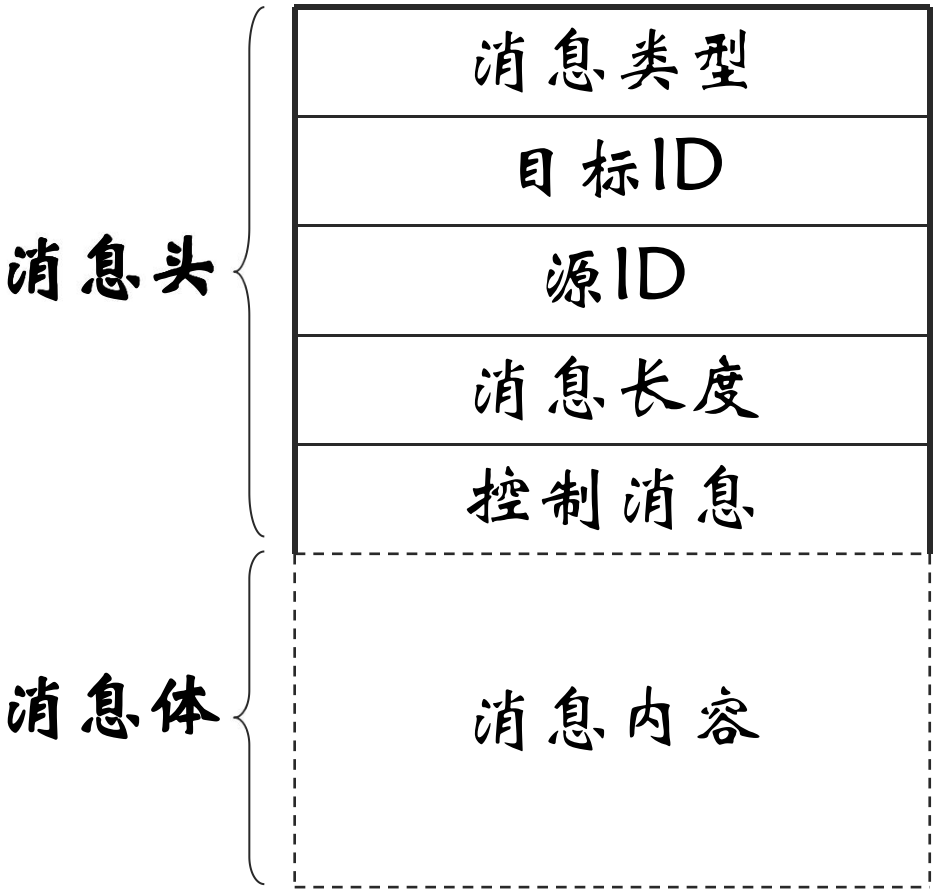
- 对称直接寻址，发送进程和接收进程必须命名对方以便通信，原语send() 和 receive()定义如下：
  - send(P, messsage) 发送消息到进程P
  - receive(Q, message) 接收来自进程Q的消息
- 非对称直接寻址，只要发送者命名接收者，而接收者不需要命名发送者，send()和 receive()定义如下：
  - send(P, messsage) 发送消息到进程P
  - receive(id, message) 接收来自任何进程的消息，变量id置成与其通信的进程名称

(1) 进程直接通信



消息传递：进程P向Q发送消息

消息格式



(2) 进程间接通信

- 消息不是直接从发送者发送到接收者，而是发送到由临时保存这些消息的队列组成的一个共享数据结构，这些队列通常成为信箱(mailbox)
- 一个进程给合适的信箱发送消息，另一进程从信箱中获得消息
- 间接通信的send()和 receive()定义如下：
  - send(A,message)：把一封信件(消息)传送到信箱A
  - receive(A,message)：从信箱A接收一封信件(消息)

(2) 进程间接通信

- 信箱可以分成信箱头和信箱体两部分，信箱头指出信箱容量、信件格式、存放信件位置的指针等；信箱体用来存放信件，信箱体分成若干个区，每个区可容纳一封信
- “发送”和“接收”两条原语的功能为：
- 发送信件：如果指定的信箱未满，则将信件送入信箱中由指针所指示的位置，并释放等待该信箱中信件的等待者；否则，发送信件者被置成等待信箱状态
- 接收信件：如果指定信箱中有信，则取出一封信件，并释放等待信箱的等待者，否则，接收信件者被置成等待信箱中信件的状态

send/receive原语的算法描述

```
type box=record
     size: integer;                  /*信箱大小*/
     count: integer;                 /*现有信件数*/
     letter: array[1..n] of message;      /*信箱中的信件*/
     S1,S2: semaphore;               /*等信箱和等信件信号量*/
end
```

```
procedure send(varB:box,M:message)
  var i:integer;
  begin
   if B.count=B.size  then W(B.s1);
      i:=B.count+1;
      B.letter[i]:=M;
      B.count:=i;
      R(B.S2)
  end;
```

R()和W()是让进程入队和出队的两个过程

```
procedure receive(varB:box, x:message)
  var i:integer;
  begin
   if B.count=0  then W(B.s2);
      B.count:=B.count-1;
      x:=B.letter[1];
      if B.count ≠0 then
     for i=1 to B.count do
         B.letter[i]:=B.letter[i+1];
      R(B.S1);
end;
```

消息传递求解生产者消费者问题

```
creat-mailbox(producer);          //创建信箱
creat-mailbox(consumer);
```

```
void producer_i( ) {  //i=1,… ,n
    message pmsg;
    while(true) {
        pmsg = produce();
        send(consumer, pmsg);
    }
}
```

```
void consumer_j( ) {//j=1,… ,m
    message cmsg;
    while(true) {
     receive (consumer, cmsg);
     consume(csmg);
    }
}
```

```
cobegin
     producer_i( );
     consumer_j( );
 coend
```

有关消息传递问题的若干问题

- 关于信箱容量问题
- 关于多进程与信箱相连的信件接收问题
- 关于信箱的所有权问题
  - 信箱为操作系统所有是指由操作系统统一设置信箱，归系统所有，供相互通信的进程共享，消息缓冲机制就是一个著名的例子
- 关于信件的格式问题和其他有关问题
- 关于通信进程的同步问题

有关消息传递问题的若干问题

- 消息缓冲是在1973年由P.B.Hansan提出的一种进程间高级通信设施，并在RC4000系统中实现
- 消息缓冲通信的基本思想是：由操作系统统一管理一组用于通信的消息缓冲存储区，每一个消息缓冲存储区可存放一个消息(信件)。当一个进程要发送消息时，先在自己的消息发送区里生成待发送的消息，包括：接收进程名、消息长度、消息正文等。然后，向系统申请一个消息缓冲区，把消息从发送区复制到消息缓冲区中，注意在复制过程中系统会将接收进程名换成发送进程名，以便接收者识别。随后该消息缓冲区被挂到接收消息的进程的消息队列上，供接收者在需要时从消息队列中摘下并复制到消息接收区去使用，同时释放消息缓冲区。

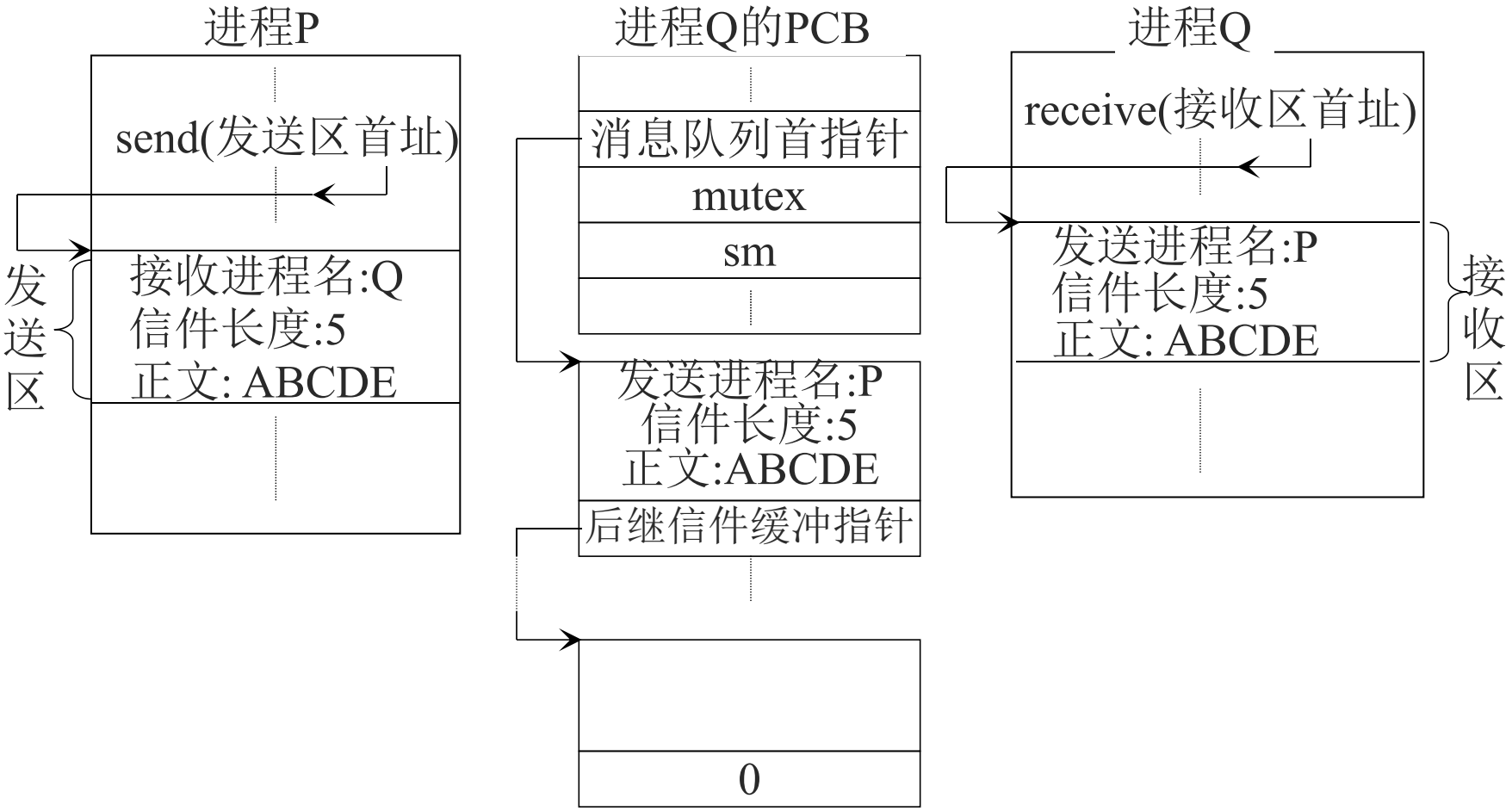
消息缓冲通信

- 消息缓冲通信涉及的数据结构：
  - sender：发送消息的进程名或标识符
  - size：发送的消息长度
  - text：发送的消息正文
  - next-ptr：指向下一个消息缓冲区的指针
- 在进程的PCB中涉及通信的数据结构：
  - mptr：消息队列队首指针
  - mutex：消息队列互斥信号量，初值为1
  - sm：表示接收进程消息队列上消息的个数，初值为0，是控制收发进程同步的信号量

消息缓冲通信

- 发送原语和接收原语的实现如下：
- 发送原语Send：申请一个消息缓冲区，把发送区内容复制到这个缓冲区中；找到接收进程的PCB，执行互斥操作P(mutex)；把缓冲区挂到接收进程消息队列的尾部，执行V(sm)、即消息数加1；执行V(mutex)
- 接收原语Receive：执行P(sm)查看有否信件；执行互斥操作P(mutex)，从消息队列中摘下第一个消息，执行V(mutex)；把消息缓冲区内容复制到接收区，释放消息缓冲区

消息缓冲通信


进程P
send(发送区首址)
发送区
接收进程名:Q
信件长度:5
正文: ABCDE
进程Q的PCB
消息队列首指针
mutex
sm
发送进程名:P
信件长度:5
正文:ABCDE
后继信件缓冲指针
0
进程Q
receive(接收区首址)
发送进程名:P
信件长度:5
正文: ABCDE
接收区

管道和套接字

- 管道(pipeline)是Unix和C的传统通信方式
- 套接字(socket)起源于Unix BSD版本，目前已经被Unix和Windows操作系统广泛采用，并支持TCP/IP协议，即支持本机的进程间通信，也支持网络级的进程间通信
- 管道和套接字都是基于信箱的消息传递方式的一种变体，它们与传统的信箱方式等价，区别在于没有预先设定消息的边界。换言之，如果一个进程发送10条100字节的消息，而另一个进程接收1000个字节，那么接收者将一次获得10条消息

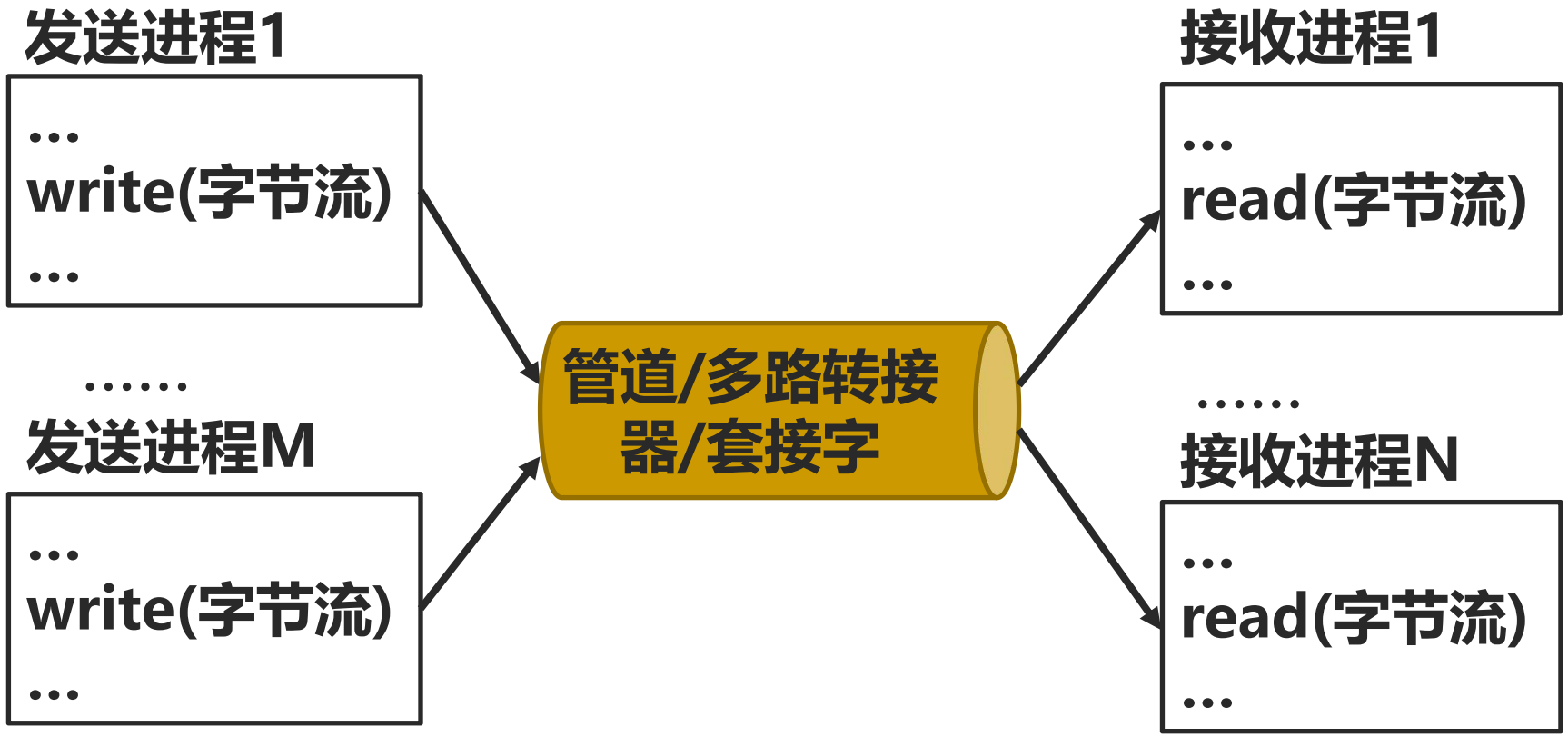
6.5.2 高级进程通信机制

- (1) 基于流的进程通信
- (2) 基于RPC的高级通信规约

(1) 基于流的进程通信

- 多个进程使用一个共享的消息缓冲区（可称为管道、多路转接器、套接字）
- 一些进程往消息缓冲区中写入字符流（send/write）
- 一些进程从消息缓冲区中读出字符流（receive/read）
- 信息交换单位基于字符流，长度任意

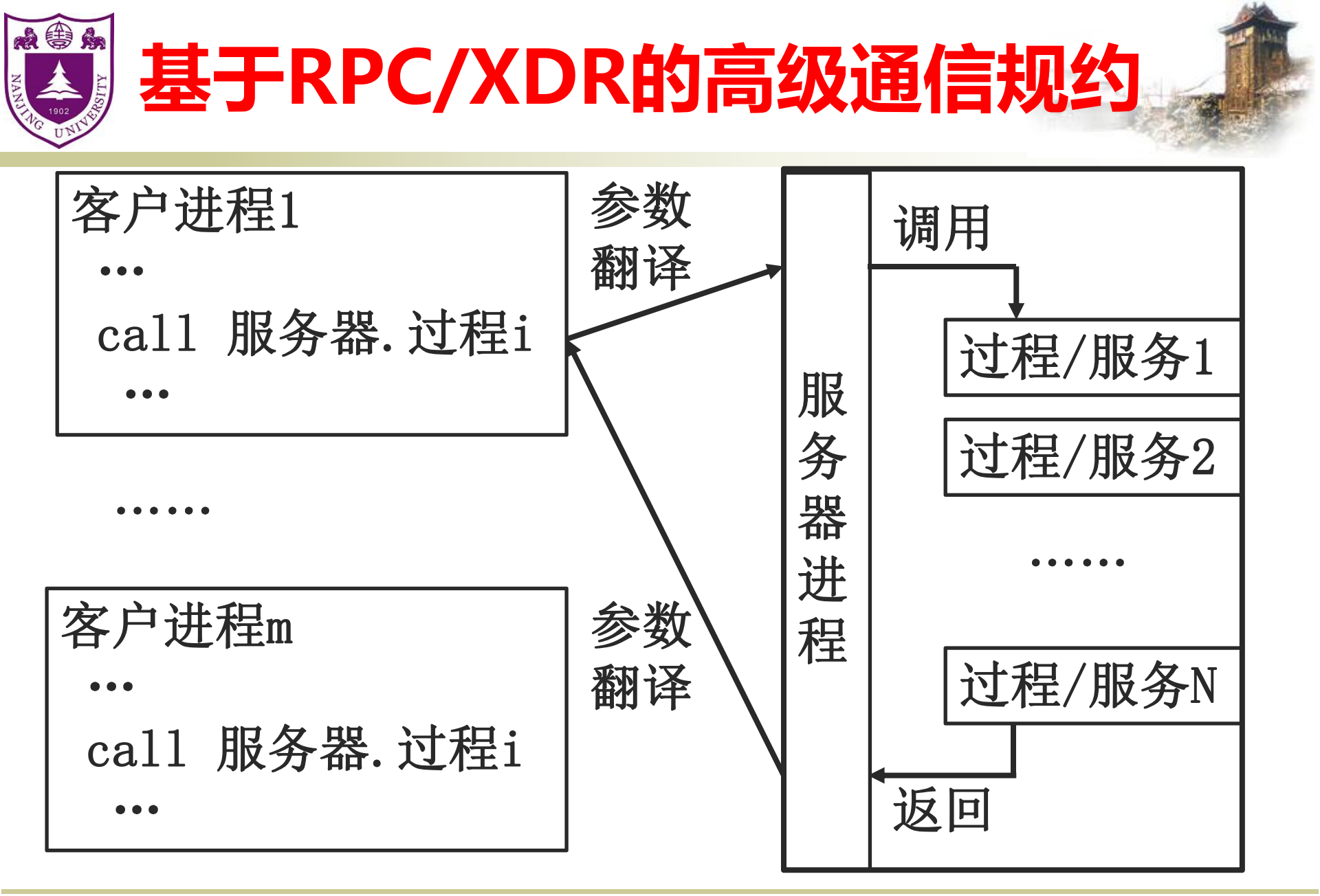
基于字符流的进程通信规约



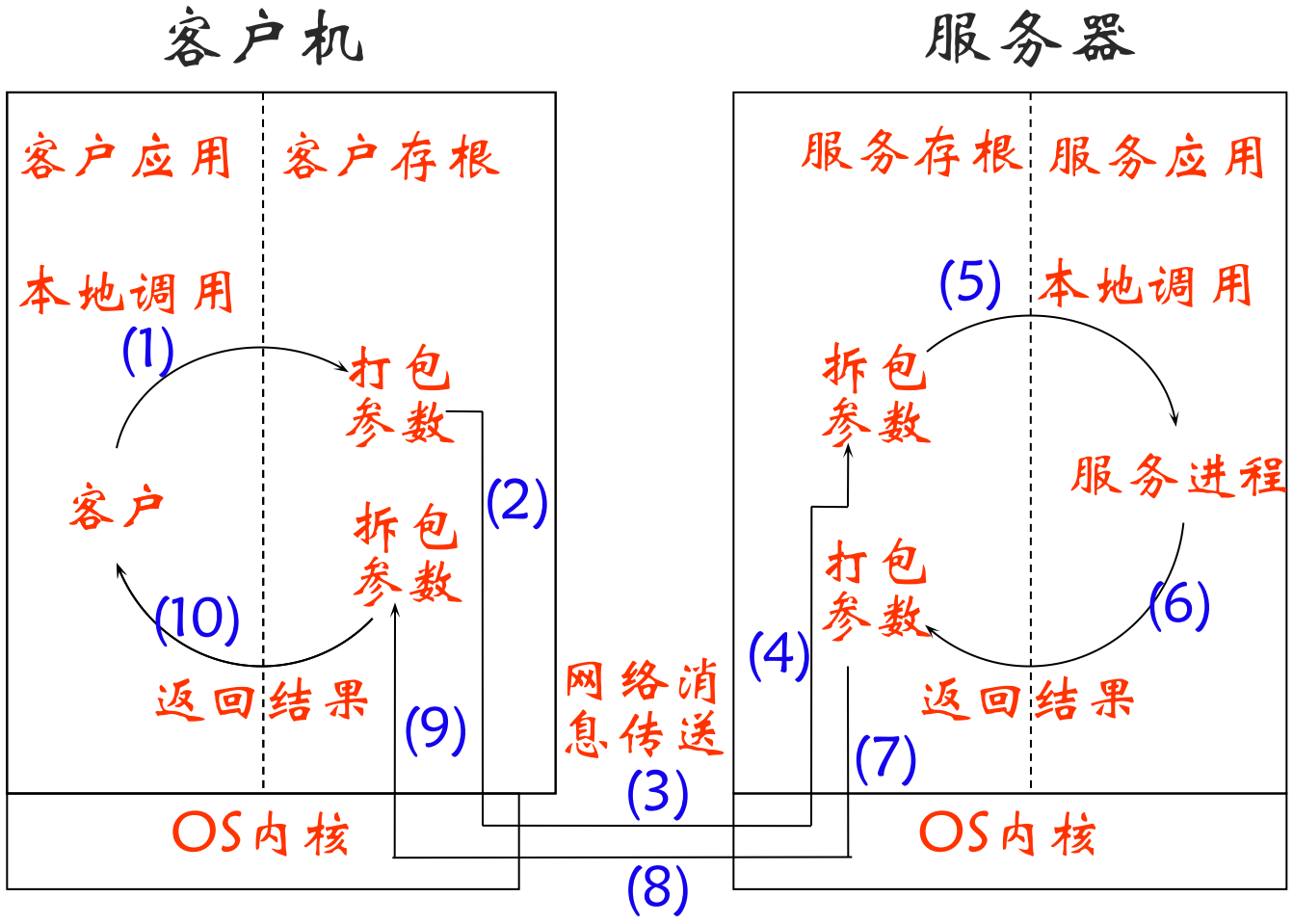
(2) 远程过程调用RPC

- 采用**客户/服务器**计算模式
- **服务器进程**提供一系列**过程/服务**，供客户进程调用
- **客户进程**通过调用服务器进程提供的**过程/服务**获得服务
- 考虑到客户计算机和服务器计算机的硬件异构型，外部数据表示XDR被引入来转换每台计算机的特殊数据格式为标准数据格式

基于RPC/XDR的高级通信规约



远程过程调用
(RPC, Remote Procedure Call)



RPC执行步骤 (1)

(1) 客户进程以普通方式调用客户存根

(2) 客户存根组织RPC消息并执行Send，激活内核程序

(3) 内核把消息通过网络发送到远地内核

(4) 远地内核把消息送到服务器存根

(5) 服务器存根取出消息中参数后调用服务器过程

RPC执行步骤 (2)

(6) 服务器过程执行完后把结果返回至服务器存根

(7) 服务器存根进程将它打包并激活内核程序

(8) 服务器内核把消息通过网络发送至客户机内核

(9) 客户内核把消息交给客户存根

(10) 客户存根从消息中取出结果返回给客户进程

(11) 客户进程获得控制权并得到了过程调用的结果