基础实验篇第5章

多机协作处理

上一节的问题：并行化处理大批量文件中的特殊字符串

随着文件数量增多，可能会超出单台计算机的处理能力

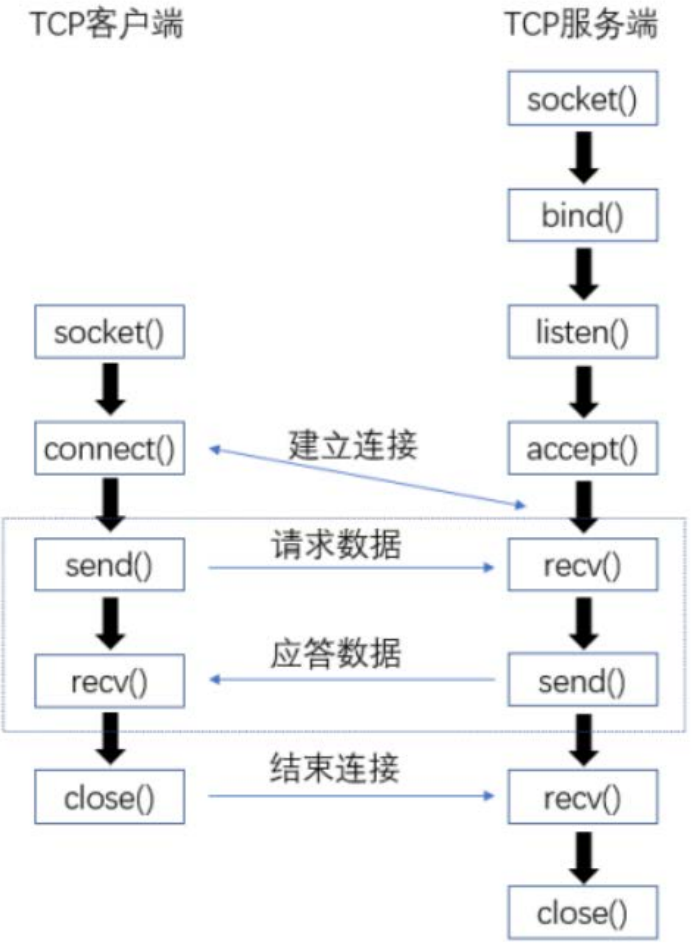
如何更高效地处理这个问题？

- 单台计算机的算力扩展是有限的
- 计算机之间可以通过网络相互协同共同完成任务
- 如何充分利用这个优势提升程序效率？
- 如何解决网络连接不稳定的问题？

- **多机并行处理->网络编程**

网络编程概述

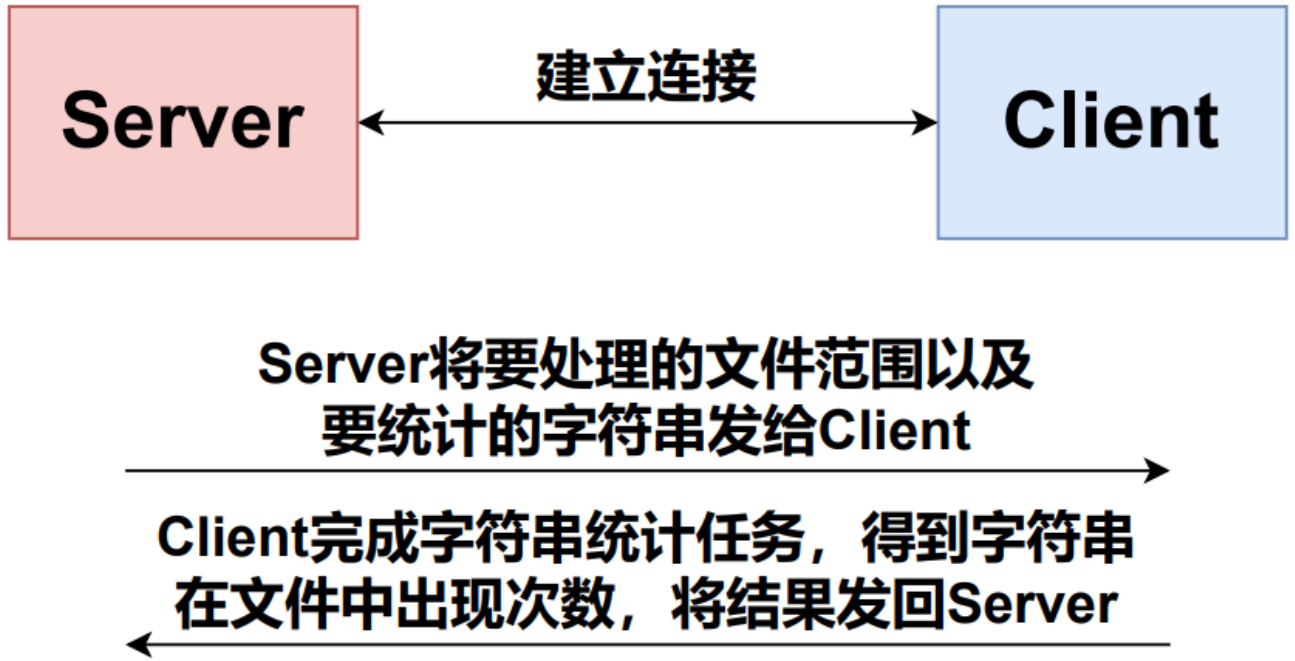
Socket套接字：可以看成是两个网络应用程序进行通信时，各自通信连接中的端点。



使用TCP协议的socket通信流程

基础编程实现

基础编程实现涉及两个主机间的通信
基础任务：统计字符串数量



协议初步设计

由于socket在传输数据时只能传输连续的数据，我们必须设计网络协议，确保计算机网络中的各个设备和系统能够进行有效的通信和协作。

对于此任务，可以规定如下的数据格式：

起始文件编号(int)	结束文件编号(int)	需要统计的字符串（string）

规定格式的意义在于，当服务器端发送时，会将上述内容打包发送到客户端处，而客户端收到信息后，可以按照该协议格式，解析出客户端要处理的文件的起始编号和结束编号，以及要处理的字符串是什么。如果没有统一格式，客户端则无法知道接收到的数据分别对应什么信息。

数据格式的规定只是网络协议中的一部分，在用户自行设计的网络协议中还会包括数据传输方式、协议中设备的地址和标识方式、消息发送和接收的顺序、错误处理和可靠性检测、安全性和加密等内容，需要根据不同的应用场景来完善整个网络协议。

server&client，创建socket套接字

```
int sockfd,connfd; // 服务器套接字和客户端套接字的文件描述符
struct sockaddr_in servaddr,clientaddr; // 存储服务器和客户端地址信息的结构体
socklen_t len = sizeof(clientaddr); // 客户端地址结构体的长度

sockfd = socket(AF_INET, SOCK_STREAM, 0); // 创建套接字
if(sockfd==-1){
    cout << "Socket creation failed...\n";
    return 0;
}

int opt = 1; // 设置套接字选项以重用地址
setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)); // 初始化服务器地址结构体
memset(&servaddr, 0, sizeof(servaddr));
servaddr.sin_family = AF_INET; // IPv4 网络协议的套接字类型
servaddr.sin_addr.s_addr = inet_addr("127.0.0.1"); // 设置服务器 IP 地址
servaddr.sin_port = htons(8080); // 设置服务器端口号
```

server绑定、监听、接受连接

```
// 将套接字绑定到指定地址和端口
if((bind(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)))!=0){
    cout << "Socket bind failed...\n";
    return 0;
}
// 开始监听连接请求，5 表示可以同时处理的最大连接数
if((listen(sockfd, 5))!=0){
    cout << "Listen failed...\n";
    return 0;
}
// 接受客户端的连接请求
connfd = accept(sockfd, (struct sockaddr*)&clientaddr, &len);
if(connfd < 0){
    cout << "Server accept failed...\n";
    return 0;
}
```

client向server发起连接

```
// 连接到服务器
if(connect(sockfd, (struct sockaddr*)&servaddr, sizeof(servaddr)) == -1){
    cout << "Connect failed...\n";
    return 0;
}
```

server向client传输数据，文件范围以及单词

```
// 由于socket只能传输连续数据，将文件范围以及字符串组合成string（或数组）后发送
string message = to_string(a) + " " + to_string(b) + " \"" + c + "\"";
send(connfd, message.c_str(), message.length(), 0);
```

client接收server发来的message

```
char word[1024]; // 创建一个数组来接受server的信息
recv(sockfd, word, sizeof(word), 0);
```

client端根据message信息，完成word_count字符串统计

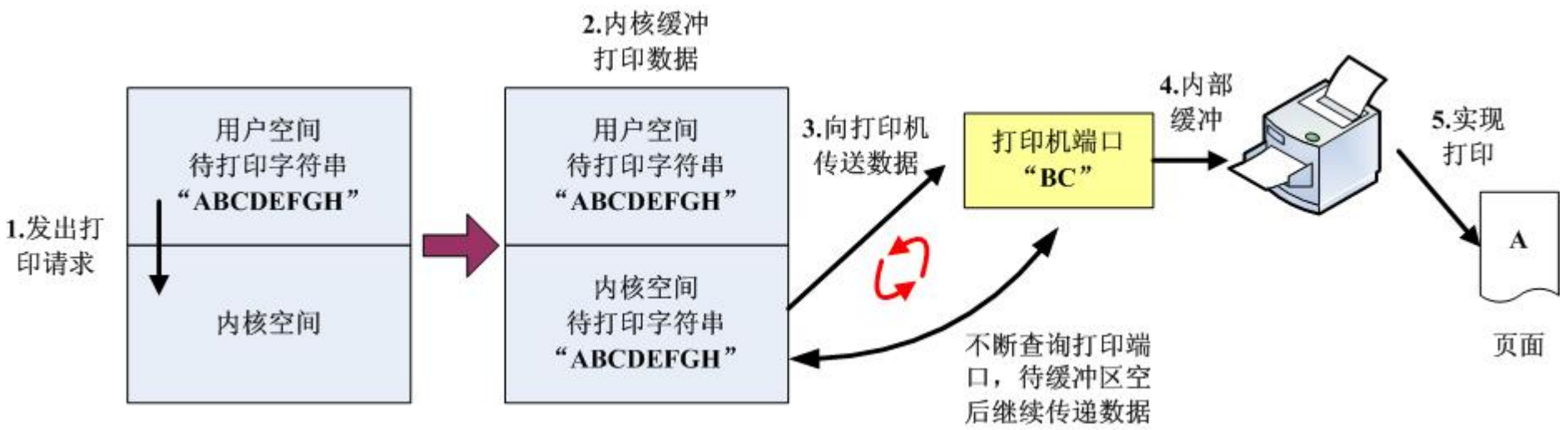
```
string cmd1("./word_count.sh");
cout << cmd1 << endl;
string cmd2 = cmd1 + word;
cout << cmd2 << endl;
string cmd =  string("./word_count.sh ") + " " + word;
cout << cmd << endl;
system(cmd.c_str());.
```

运行结果

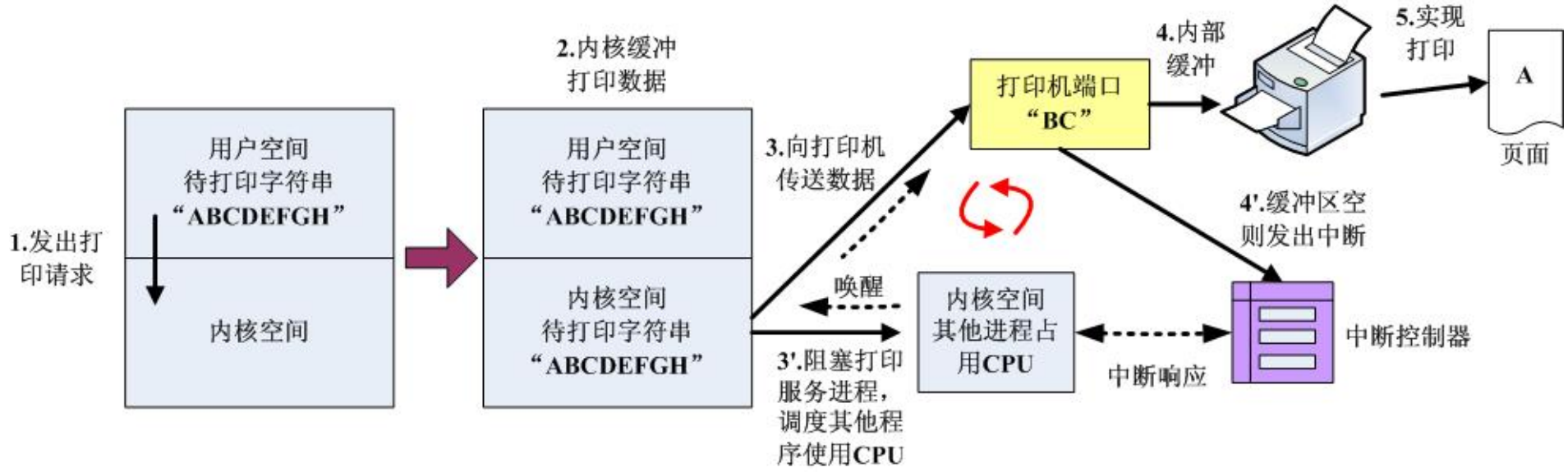
```
shaw@shaw-MS-7D46:~/Desktop/BOOK_Multi_machine/code_basic$ ./server
Enter a:6
Enter b:10
Enter c:system
Message sent to client:6 10 "system"
shaw@shaw-MS-7D46:~/Desktop/BOOK_Multi_machine/code_basic$
shaw@shaw-MS-7D46:~/Desktop/BOOK_Multi_machine/code_basic$ ./client
./word_count.sh
./word_count.sh6 10 "system"
./word_count.sh  6 10 "system"
word 'system' appears 16 times from file No.6 to file No.10
shaw@shaw-MS-7D46:~/Desktop/BOOK_Multi_machine/code_basic$
```

网络通信中的OS原理

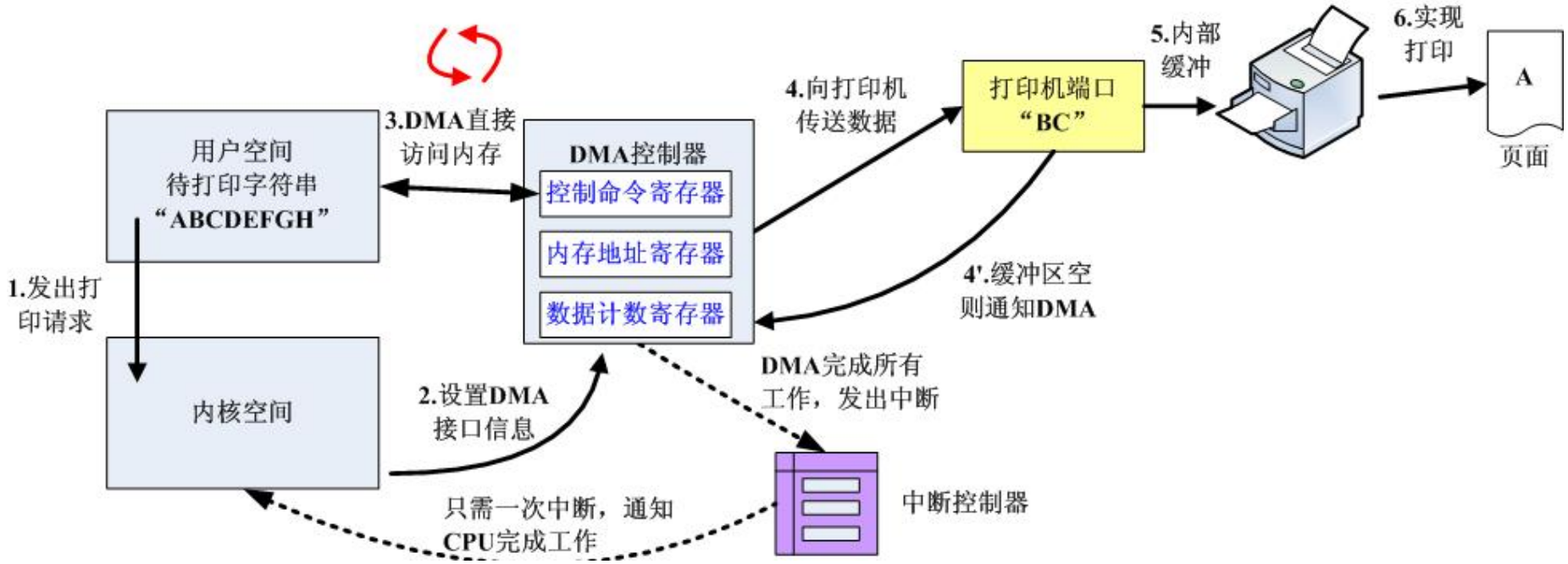
设备与OS的通信模式分析：等待式



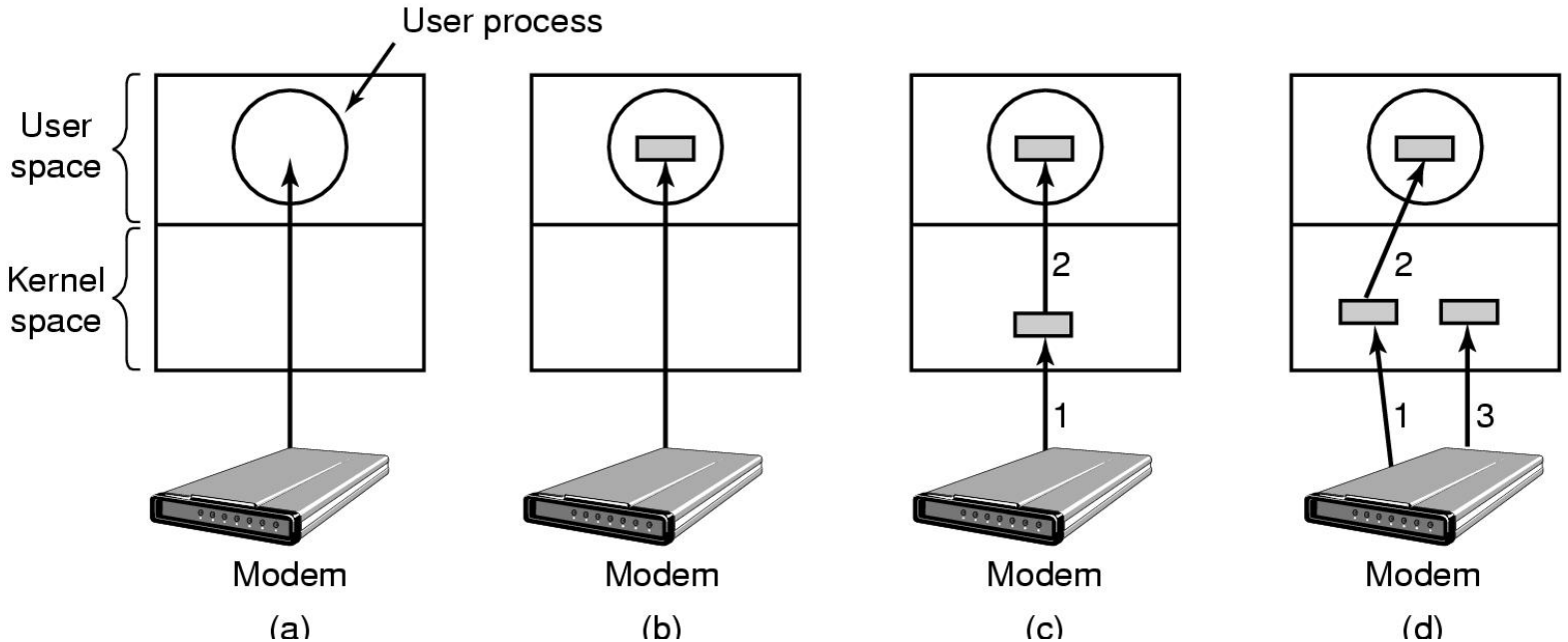
设备与OS的通信模式分析：中断式

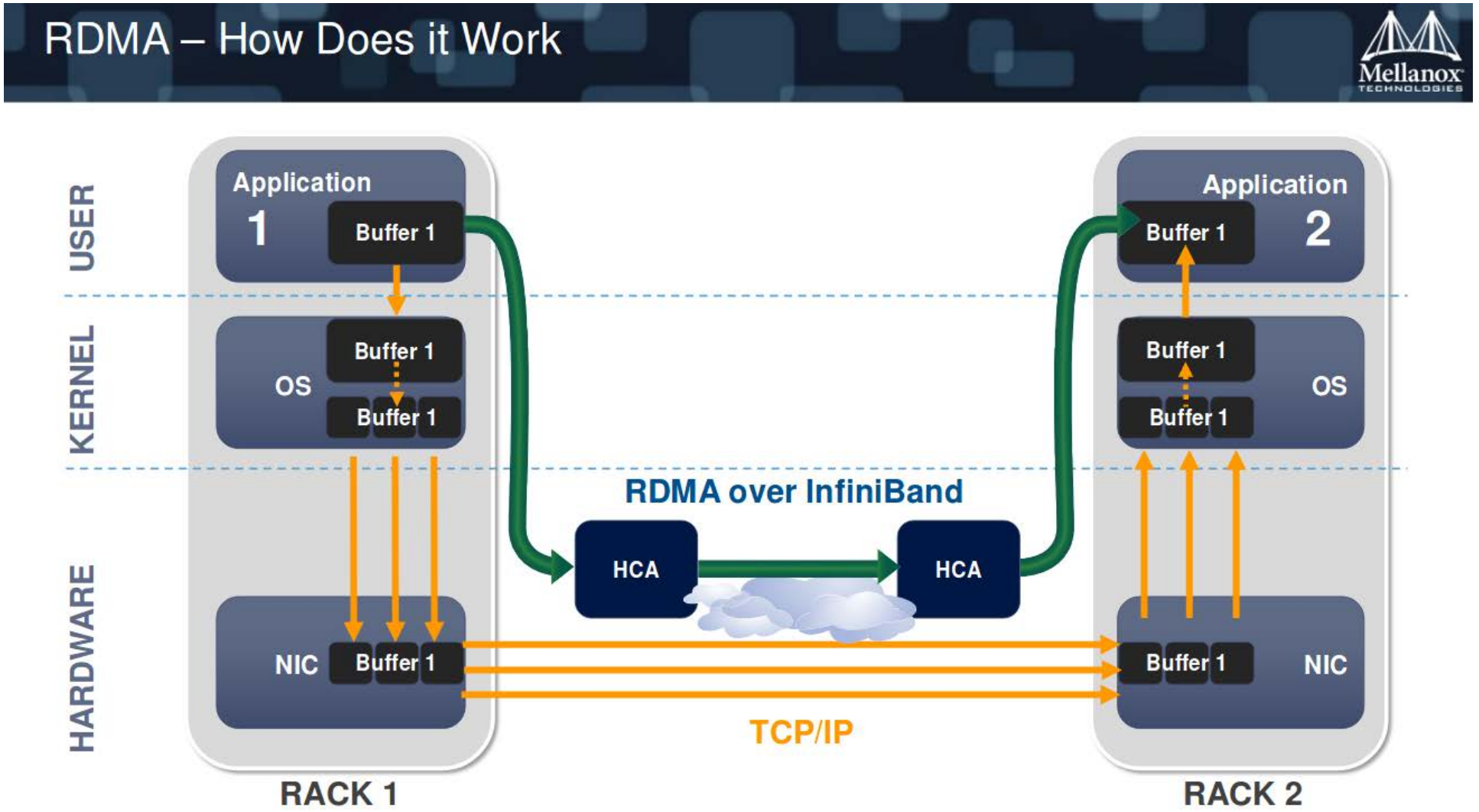


设备与OS的通信模式分析：DMA式



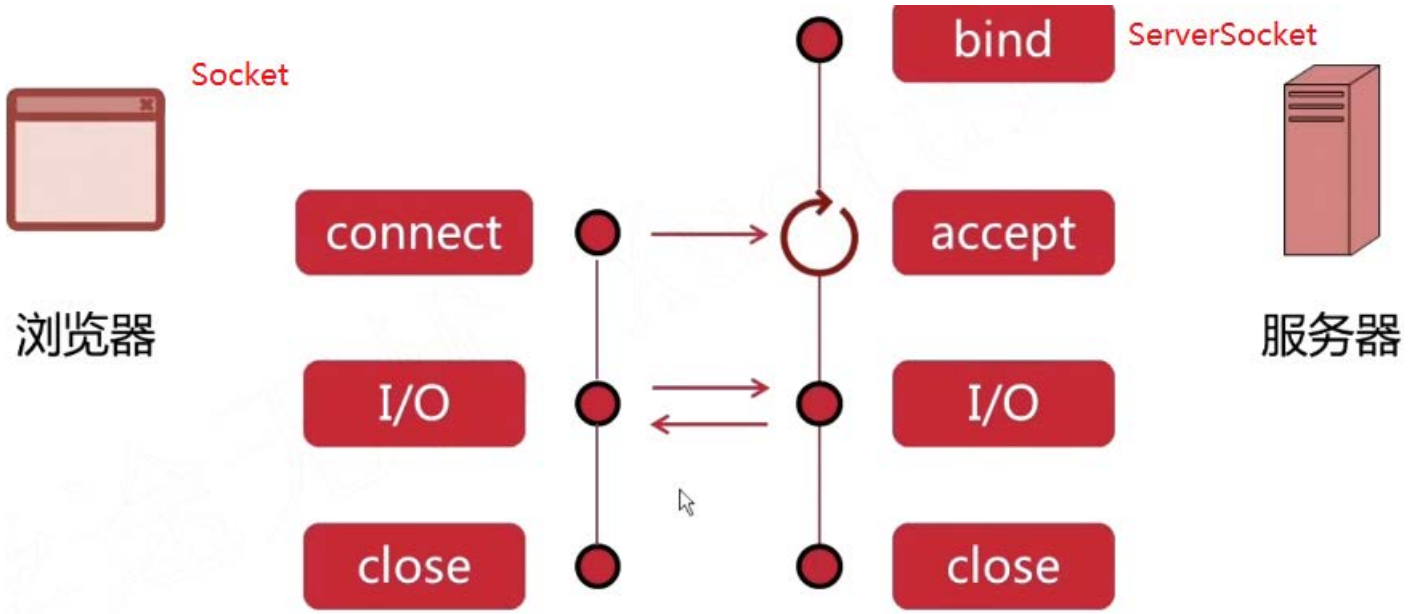
网络与OS的通信原理分析：缓冲的作用

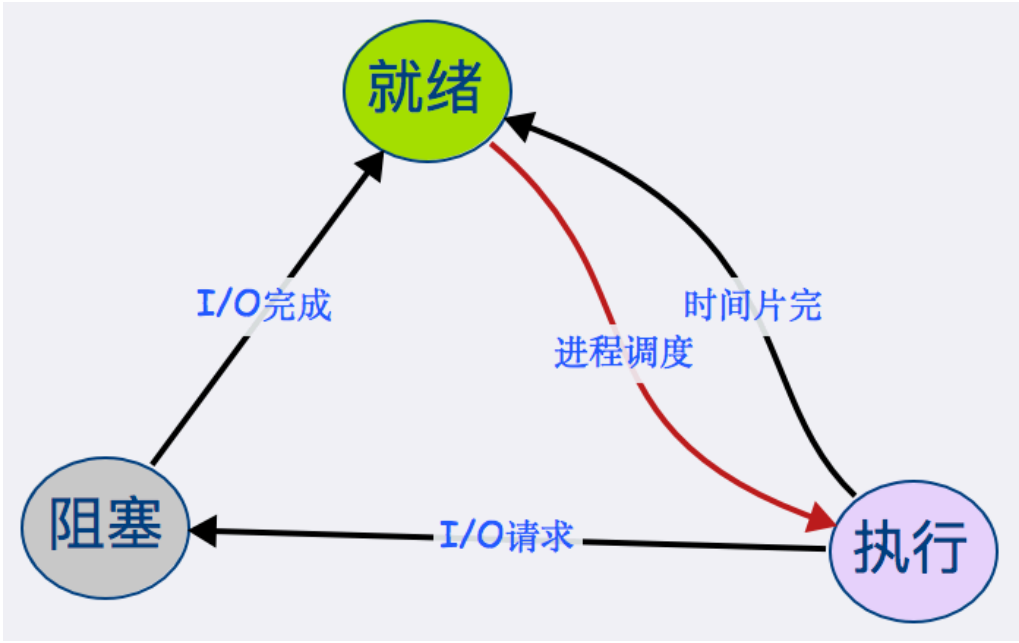




缓冲技术在I/O控制中使用很普遍，但数据传递中太多的缓冲会影响到系统性能.

网络与OS的通信原理分析：accept原理简析





高级任务实现

高级实现涉及多个主机间的通信

Server为多个Client分配任务，将要处理的文件范围以及要统计的字符串发给Client

Client完成字符串统计任务，得到字符串在文件中出现次数，将结果发回Server

Server汇总多个Client统计的结果，得到字符串的出现次数

多客户端实现，设置数组来接收client

```
vector<int> clientSockets; // 初始化客户端数组

while(true){
    // 接收多个客户端发来的连接请求
    connfd = accept(sockfd, (struct sockaddr*)&clientaddr, &len);

    cout << "Client connected. Socket ID: " << connfd << endl;
    // 将接收到的客户端连接存入客户端数组中，便于后续任务分配
    clientSockets.push_back(connfd);
}
```

任务分配：n个文件，平均分配给m个client来处理

```
int filesPerClient = n / m; // 每个客户端处理的文件数量
int remainingFiles = n % m; // 剩余的文件数量
for (int i = 0; i < clientSockets.size(); i++) {
     int startFile = i * filesPerClient + 1; // 起始文件编号
     int endFile = (i + 1) * filesPerClient; // 结束文件编号
     if (i == clientSockets.size() - 1) {
         endFile += remainingFiles; // 最后一个客户端处理剩余的文件
     }
     // 构建消息字符串
     string message = to_string(startFile) + " " + to_string(endFile) + " \"" + word + "\"";
     // 发送消息给客户端
     ssize_t sendBytes = send(clientSockets[i], message.c_str(), message.length(), 0);
     cout << "Message sent to client " << i+1 << ": " << message << endl;
}
```

client端收到server端分配的任务后，分别处理并发回结果

```
string cmd = "./word_count.sh " + string(word); // 构建命令字符串
system(cmd.c_str()); // 执行命令
FILE *fp = popen(cmd.c_str(), "r");  // 执行命令并打开输出文件流

char result[1024]; // 创建发回给服务器的字符串
memset(result, 0, sizeof(result));
fgets(result, sizeof(result), fp); // 读取命令执行结果
pclose(fp); // 关闭文件流

ssize_t sendBytes = send(sockfd, result, strlen(result), 0); // 将结果发送给服务器
```

server端接收client端发回的数据，并汇总出单词出现的总数

```
int totalOccurrences = 0; //设置用于计数
for (int i = 0; i < clientSockets.size(); i++) { //从客户端数组中依次遍历每个客户端
    char buffer[1024]; //创建接收客户端信息的数组
    memset(buffer, 0, sizeof(buffer));
    //接受来自客户端的消息
    ssize_t recvBytes = recv(clientSockets[i], buffer, sizeof(buffer), 0);

    int occurrences = atoi(buffer); //把接收到的统计数组转换为整数
    totalOccurrences += occurrences; //合计字符串出现总次数
}

// 输出统计结果
cout << "Word '" << word << "' appears " << totalOccurrences << " times from file No.1 to
file No." << n << endl;
```

多客户端的问题：在网络传输过程中会出现错误

- 连接错误
- 套接字错误
- 传输错误
- 网路错误

如何处理这个问题?

- **异常处理**

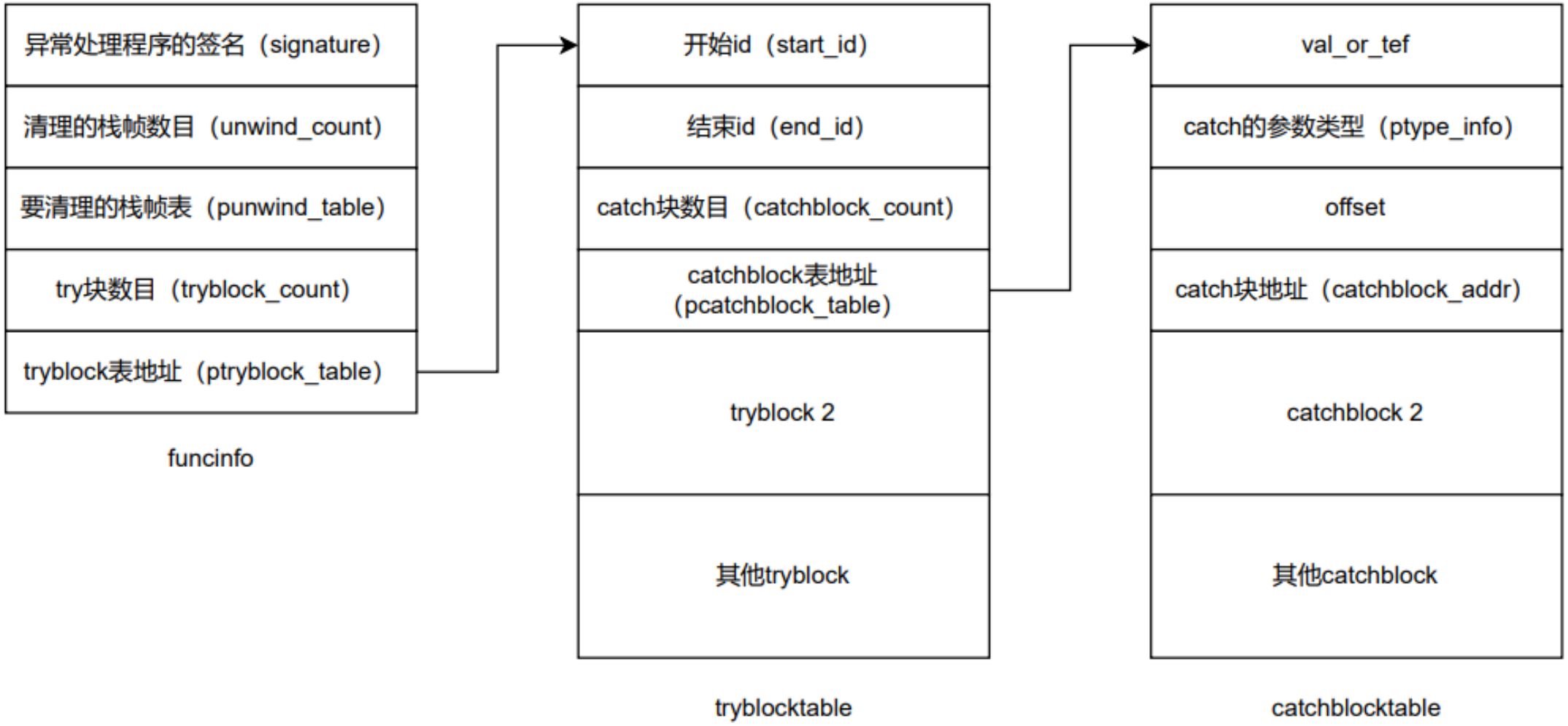
异常处理

编程语言中的异常处理（try-catch）

如：网络编程中的异常处理，如何检测异常并抛出异常？

```
try{
    语句块
    if(出现错误){
        throw 异常类型;
    }
    ...
}
catch(异常类型1){
    异常处理;
}
...
catch(异常类型n){
    异常处理;
}
```

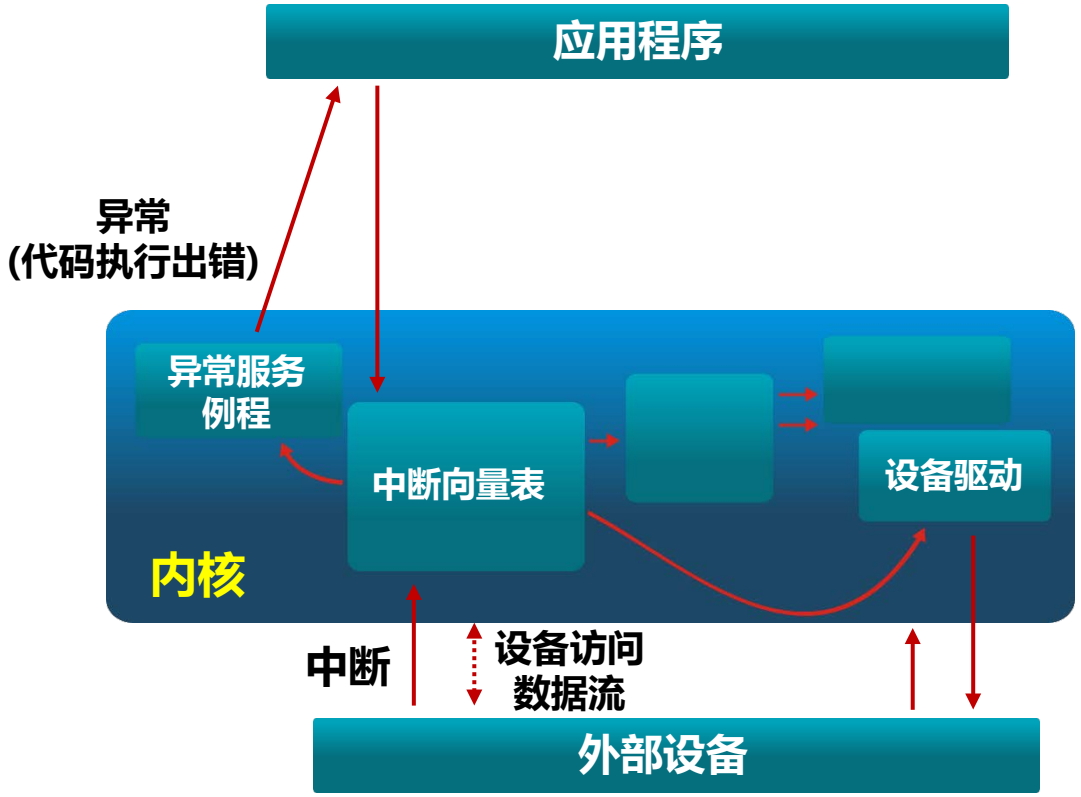
try-catch的原理



增加异常处理示例代码

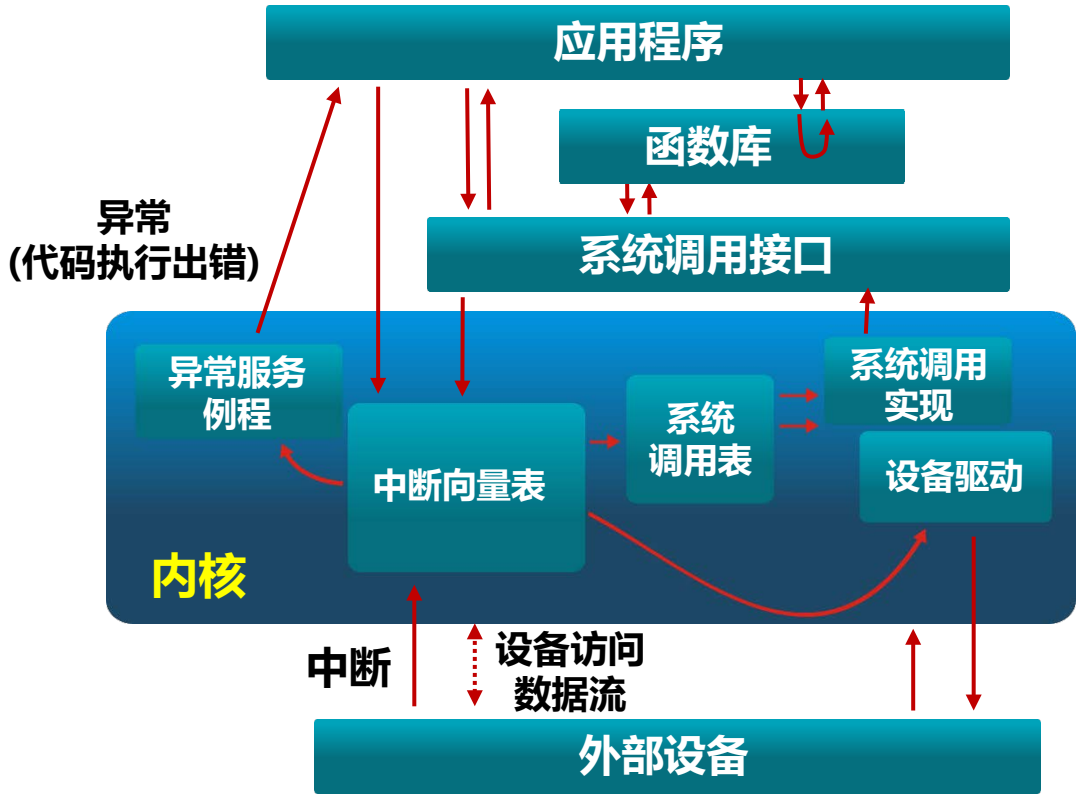
```
for (int i = 0; i < clientSockets.size(); i++) {
    try {
        ssize_t sendBytes = send(clientSockets[i], message.c_str(), message.length(), 0);
        // 在发送信息时，如果返回值为负，则可以抛出发送时产生的错误
        if (sendBytes <= 0) {
            // 抛出运行时异常
            throw runtime_error("Error sending message to client " + to_string(i + 1));
        }
        // 打印异常信息
        cout << "Message sent to client " << i + 1 << ": " << message << endl;
    } catch (const exception& e) { // 捕获到该错误后，会关闭这个socket连接
        cerr << "Error sending message to client " << i + 1 << ": " << e.what() << endl;
        close(clientSockets[i]);
        throw;
    }
}
```

异常在OS中处理的原理



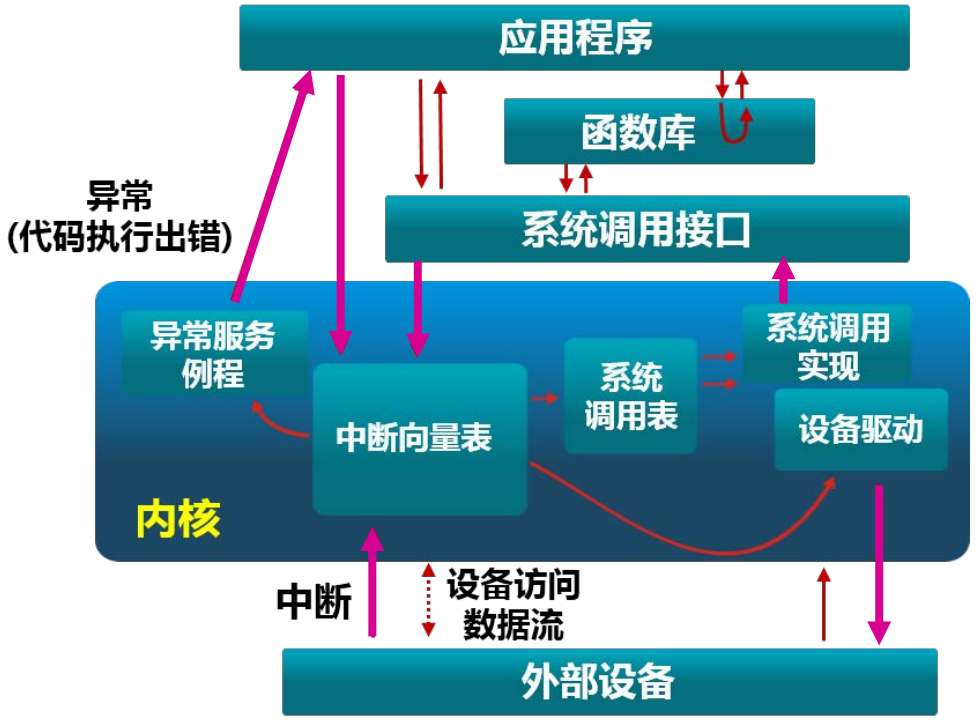
中断和异常是由硬件实现的一种跳转机制，在某些特定事件发生时（如设备按下，或除数为0）可以打断现有的指令流，改变当前权限状态，并跳转到预设的地址

异常、中断和系统调用


应用程序
函数库
异常
(代码执行出错)
系统调用接口
异常服务
例程
中断向量表
系统
调用表
系统调用
实现
设备驱动
内核
中断
设备访问
数据流
外部设备

异常、中断和系统调用

- 源头
  - 中断：外设
  - 异常：应用程序意想不到的行为
  - 系统调用：应用程序请求操作提供服务
- 响应方式
  - 中断：异步
  - 异常：同步
  - 系统调用：异步或同步
- 处理机制
  - 中断：持续，对用户应用程序是透明的
  - 异常：杀死或者重新执行意想不到的应用程序指令
  - 系统调用：等待和持续



延伸阅读：MapReduce

延伸阅读：成熟的分布式系统MapReduce

MapReduce是一种分布式计算框架。MapReduce最初由Google开发，用于处理大规模数据，后来被Apache Hadoop采用并推广。“MapReduce”系统通过编组分布式服务器并运行各种任务，管理系统的各个部分之间的所有通信和数据传输，以及提供冗余和容错来辅助处理任务。

MapReduce的优缺点

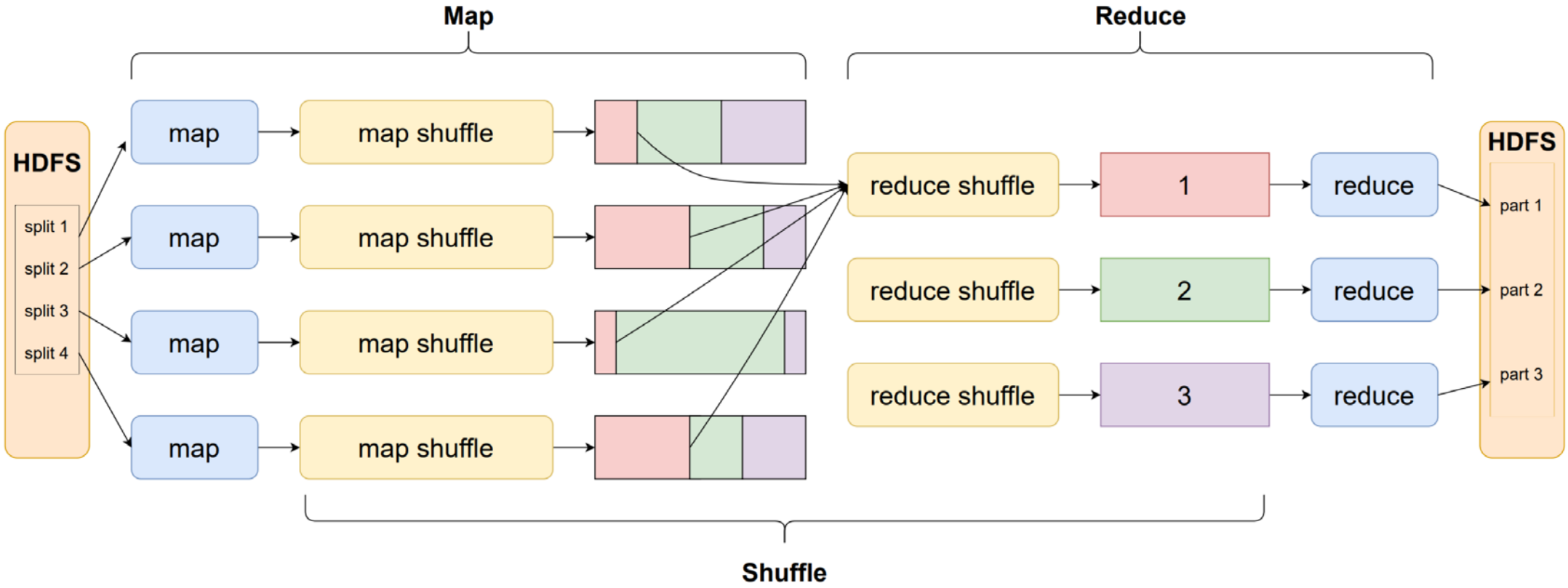
MapReduce的优点：

- 易于编程
- 良好扩展性
- 高容错性
- 适用于海量数据

MapReduce的缺点：

- 不擅长实时计算
- 不擅长流式计算
- 不擅长有向无环图计算

延伸阅读，成熟的分布式系统MapReduce



本章作业

程序设计

利用C语言和网络编程实现对多个文件的扫描，由一台计算机任主机，控制4台计算机协同工作

观察并完成实验报告

1. 观察网络中断或者出错时程序的处理逻辑
2. 观察网络传输过程中系统发生的中断的数量

感谢阅读