操作系统

L04 进程调度基础与IPC

胡燕
大连理工大学 软件学院

投票

最多可选1项

设置

是否已经安装好Linux？是否已经自学如何编辑和编译C代码，并编译执行含fork的代码示例?

A 还没有安装好Linux

B 已经安装好Linux，但尚未学习如何编辑和编译C代码

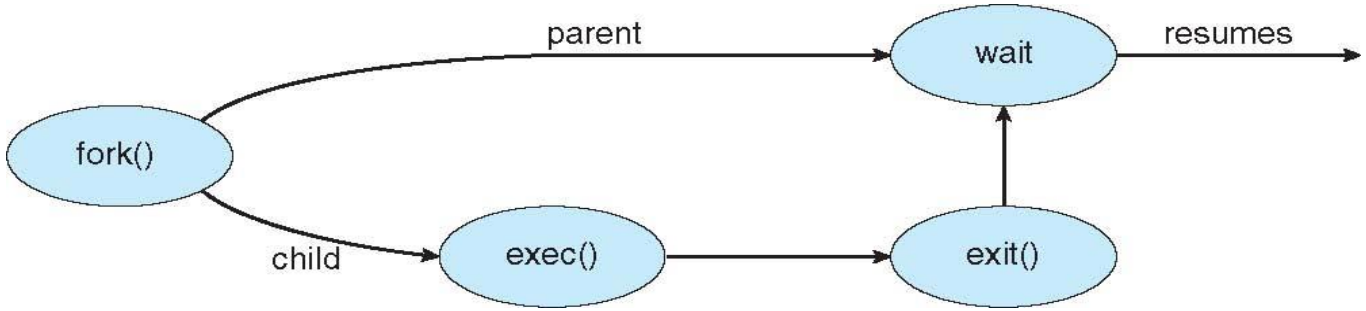
C 已经安装好Linux，尝试编译但还未体验含fork的代码

D 已经安装好Linux，并且已经进行含fork代码的编译执行

提交

Fork示例

```
#include <sys/types.h>
#include <studio.h>
#include <unistd.h>
int main()
{
    pid_t  pid;
     /* fork another process */
     pid = fork();
     if (pid < 0) { /* error occurred */
          fprintf(stderr, "Fork Failed");
          return 1;
     }
     else if (pid == 0) { /* child process */
          execlp("/bin/ls", "ls", NULL);
     }
     else { /* parent process */
          /* parent will wait for the child */
          wait (NULL);
          printf ("Child Complete");
     }
     return 0;
}
```



单选题 1分

设置

进程控制块是描述进程状态和特性的数据结构，一个进程（ ）。

A 可以有多个进程控制块

B 可以和其他进程共用一个进程控制块

C 可以没有进程控制块

D 只能有唯一的进程控制块

提交

单选题 2分

设置

某进程由于需要从硬盘上读入数据而处于阻塞状态。当系统完成了所需的读盘操作后，该进程的状态将（ ）。

A 从就绪变为运行

B 从运行变为就绪

C 从运行变为阻塞

D 从阻塞变为就绪

提交

单选题 1分

设置

程序在运行时需要很多系统资源，如内存、文件、设备等，因此操作系统以程序为单位分配系统资源。

A True

B False

提交

填空题 6分

设置

进程基本的核心状态包括（[填空1]）、（[填空2]）、（[填空3]）

作答

调度

调度

Q1：什么是调度?

调度需要做什么?

你做过调度吗？可否分享你的调度经验?

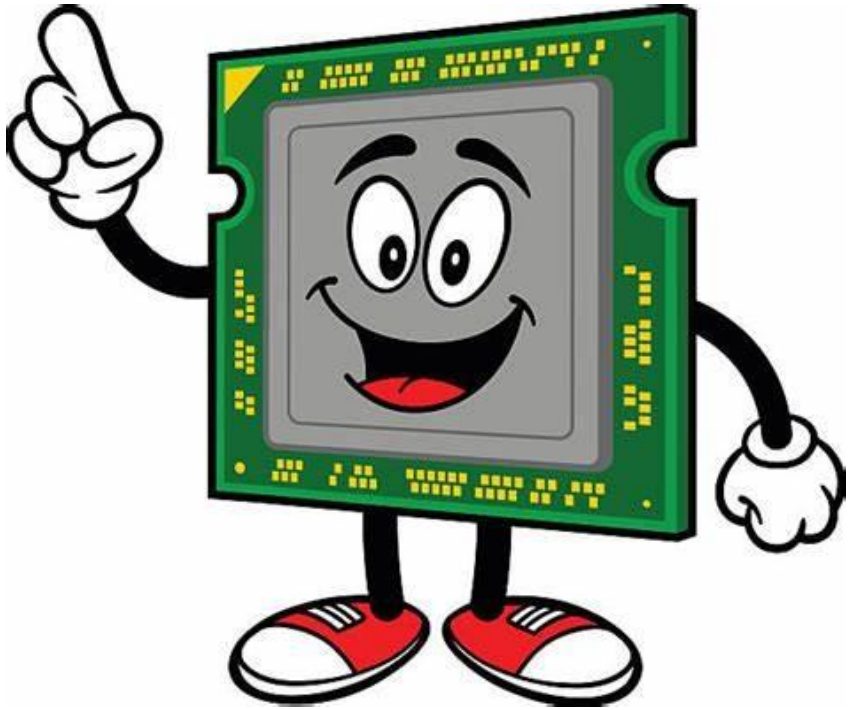
调度

Q2：计算机系统中最宝贵的资源是什么?

调度

Q2：计算机系统中最宝贵的资源是什么?

CPU



调度

Q3：多任务环境下，CPU资源应该如何使用比较合适?

为什么要进行进程调度

- **为什么要进行调度**
  - 多任务竞争使用CPU资源，需要对任务的CPU资源使用请求进行协调
- **调度的基本要求**
  - 主要目标：提高CPU利用率 (Maximize CPU Usage)
  - 在不同进程间快速切换，使得多个进程可以分时共享CPU资源
  - 调度器要在合适的时间点，从就绪的进程中选择下一个在CPU上执行的进程

为什么要进行进程调度

- **为什么要进行调度(必要性探讨)**
  - 如果不进行实施调度：一个进程一旦占用CPU，则一直占用CPU，直到进程终止
  - 有何弊端?

调度的典型目标:

公平、公正，兼顾效率

进程调度基本概念

- **从串行执行形式开始考虑**
  - Sequential Execution

调度问题转换为：当前就绪队列Q=<P1,P2,…,Pn>

以周转时间为调度指标，可以描述调度目标为：
确定P1,…Pn的一个调度顺序，使得按照该顺序先后执行完毕后，整体周转时间最短

在限定进程按照串行执行模式，则调度问题转化为就绪队列的排序问题

进程调度基本概念

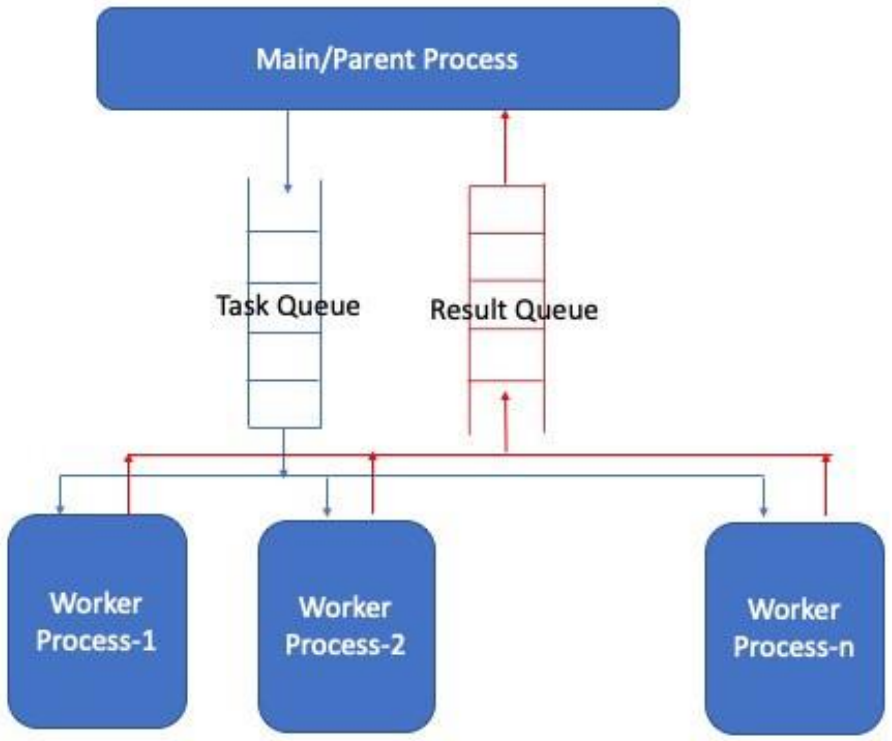
- **串行执行的不足**
  - 效率问题

考虑进程可以被描述为CPU周期和I/O周期

需要挖掘更多的并发性，来提升效率

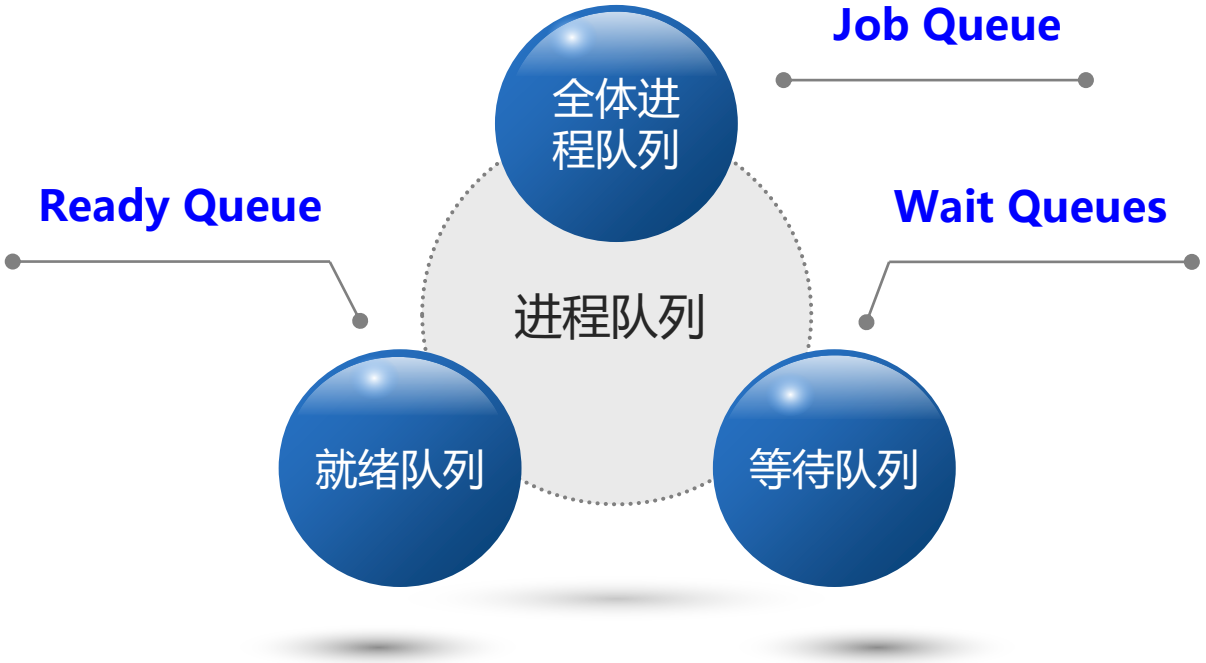
进程调度基本概念

- **为什么要进行调度(必要性探讨)**
  - 如果一个应用从一开始就是以多进程方式设计，则并发/并行会是基本选项，不会考虑串行执行



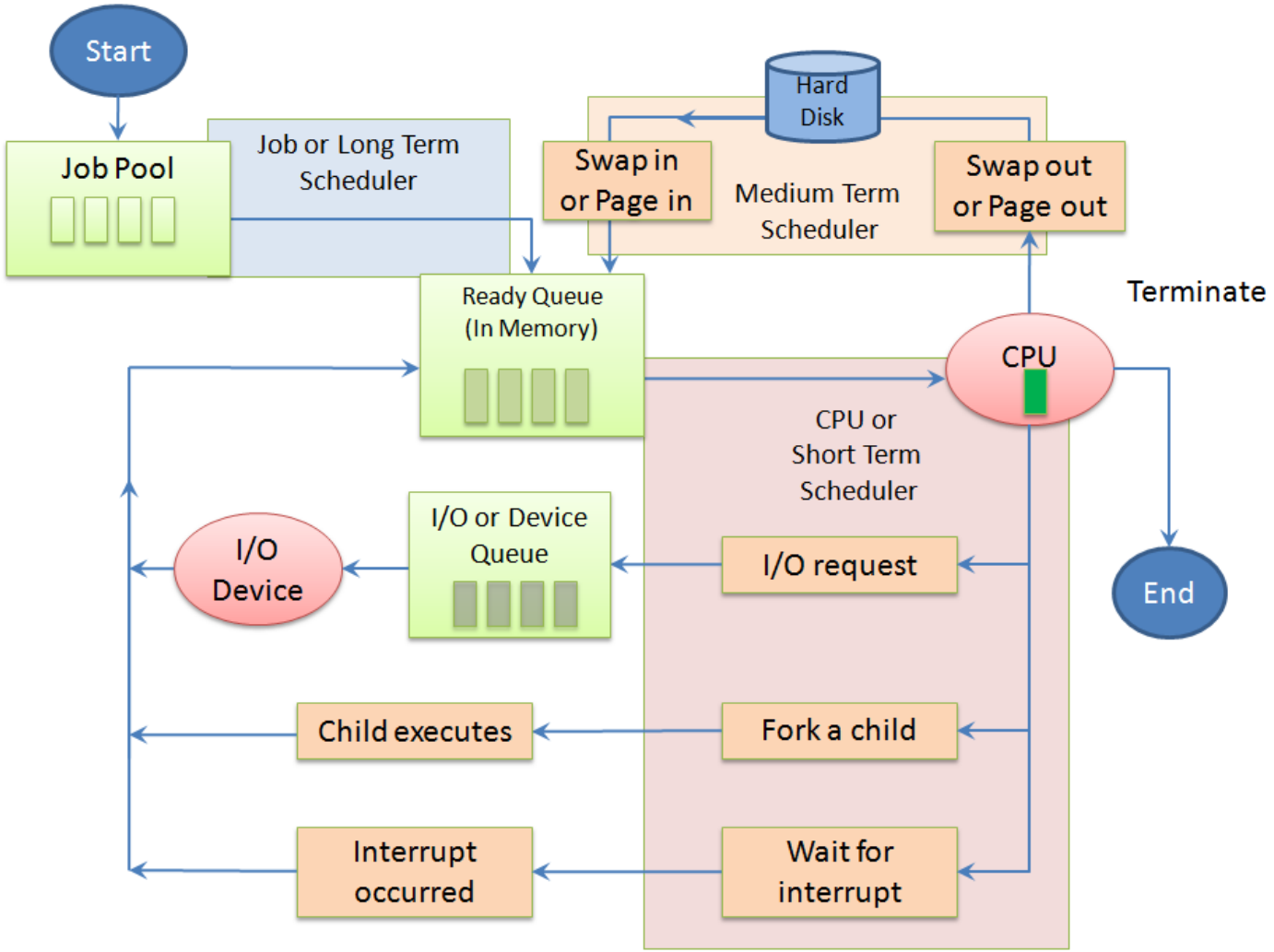
进程队列

- 为了理解调度概念，需要从队列的角度看进程



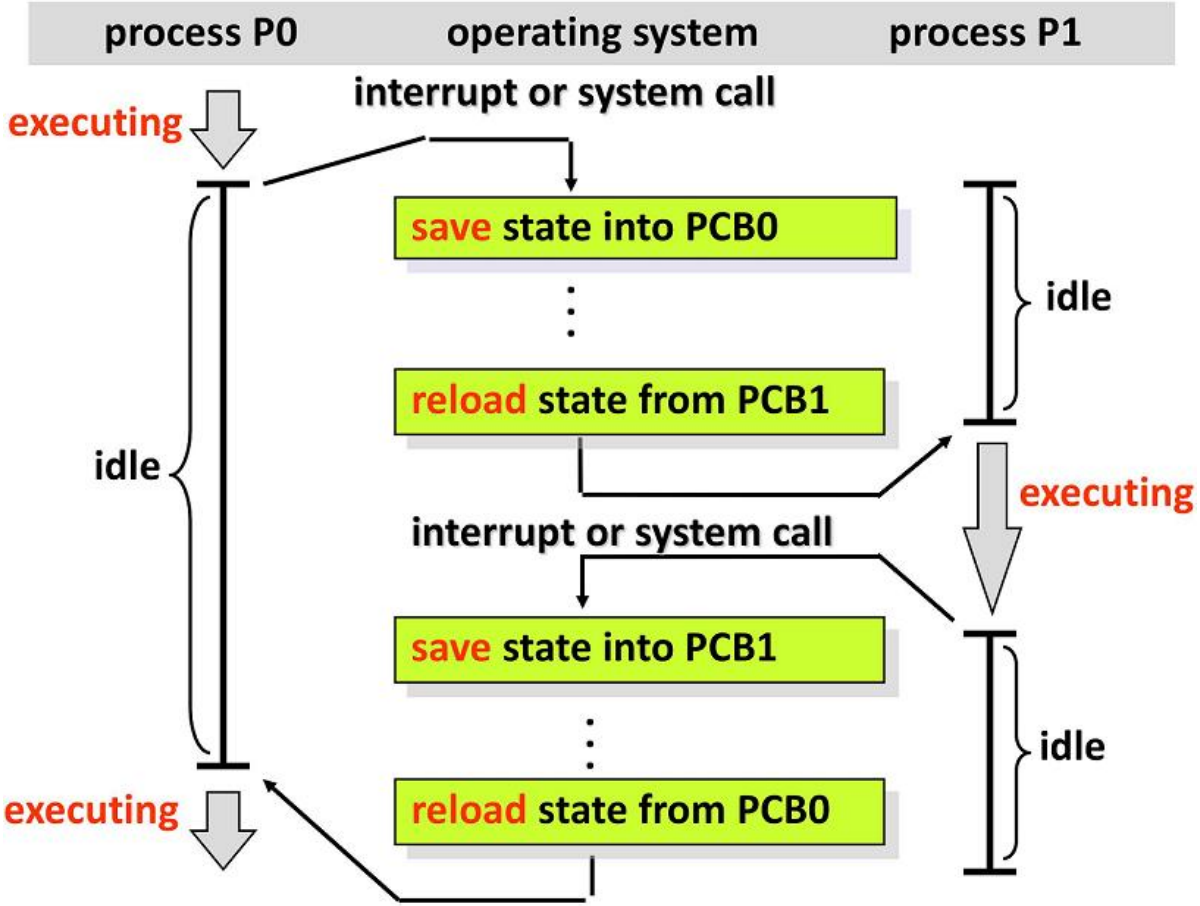
进程队列

- 调度的队列视角：在调度过程中，进程在不同队列之间的迁移图



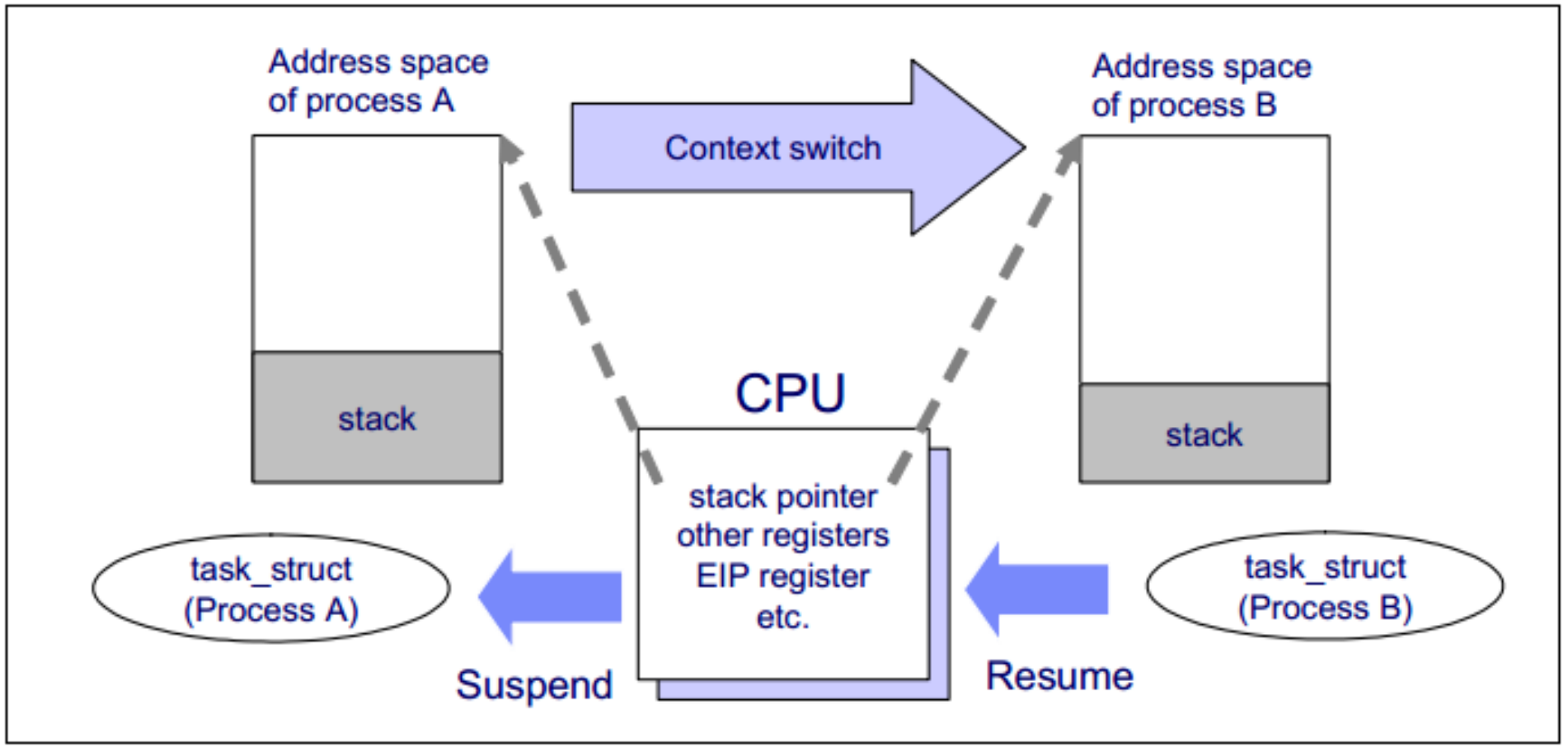
上下文切换

- 当进程获取对CPU控制，以及进程退出对CPU控制时，要保存进程执行的现场（又称**上下文**）



上下文切换

- 上下文切换细节：以Linux为例

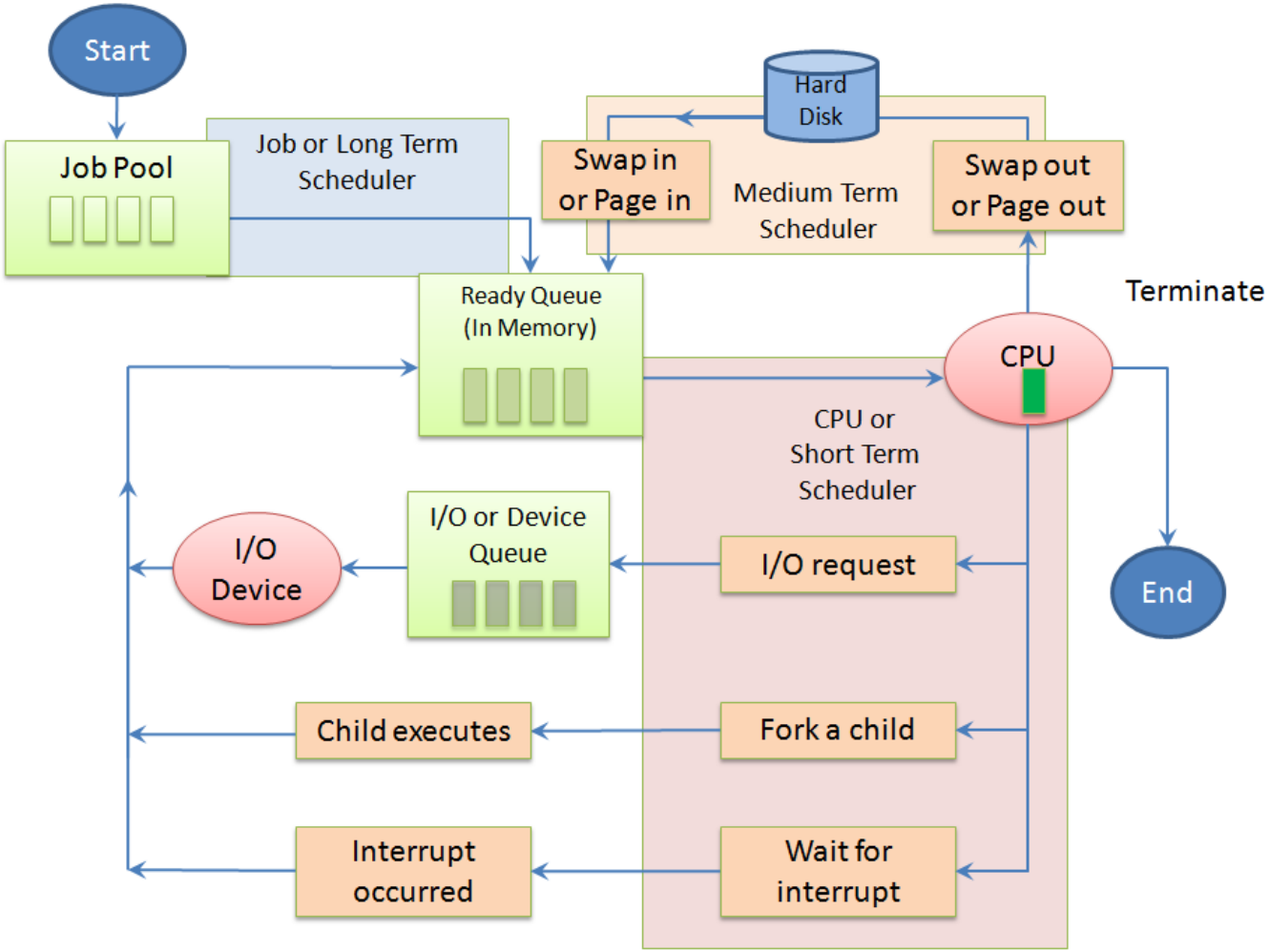


上下文切换

- **进程上下文的细分**
  - ① **用户级上下文：** Code, Data, User Stack&Heap，共享内存区
  - ② **寄存器上下文：** 通用寄存器、程序寄存器(IP)、处理器状态寄存器(如：EFLAGS)、栈指针(例如：ESP)；
  - ③ **系统级上下文：** 进程控制块task_struct、内存管理信息(mm_struct、vm_area_struct、pgd、pte)、内核栈

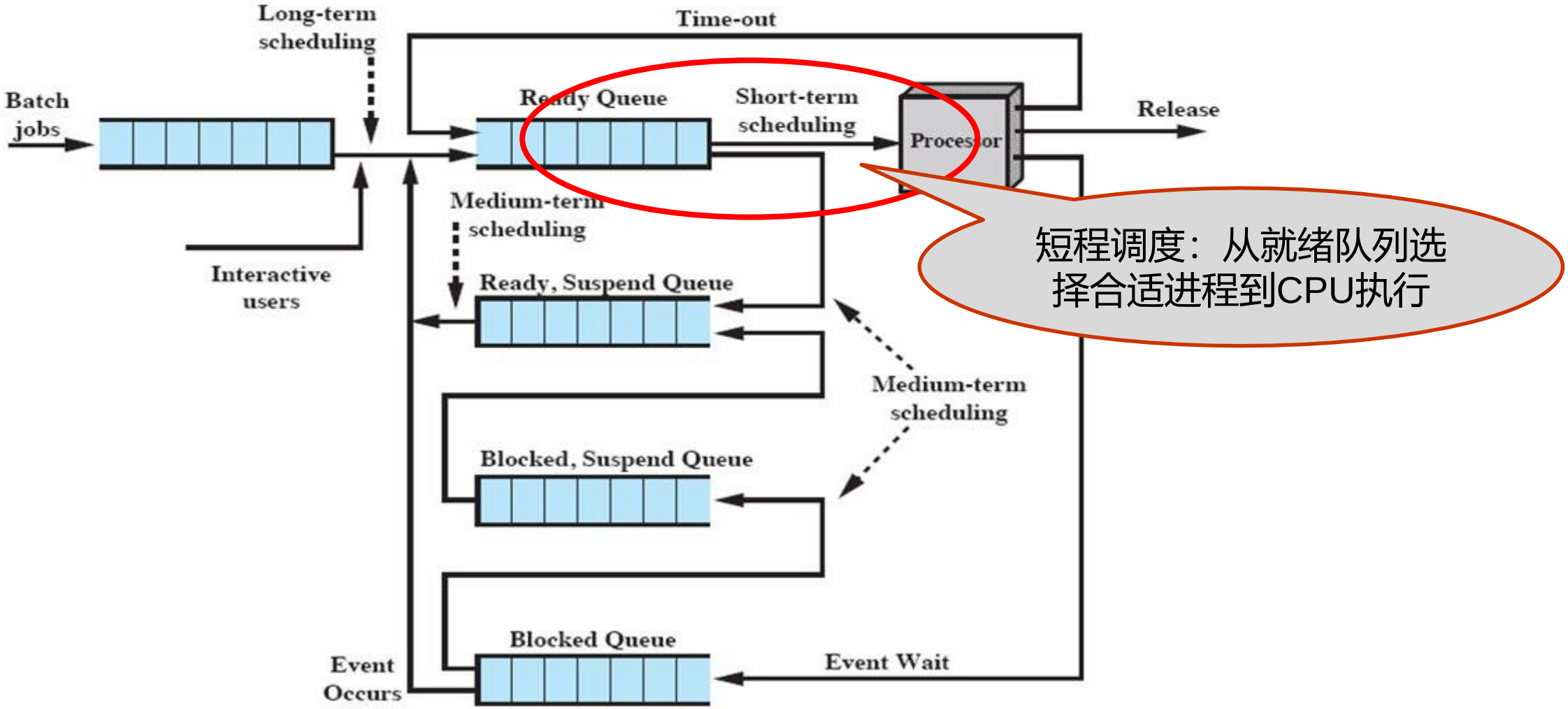
调度的层次

- 短程调度、中程调度、长程调度

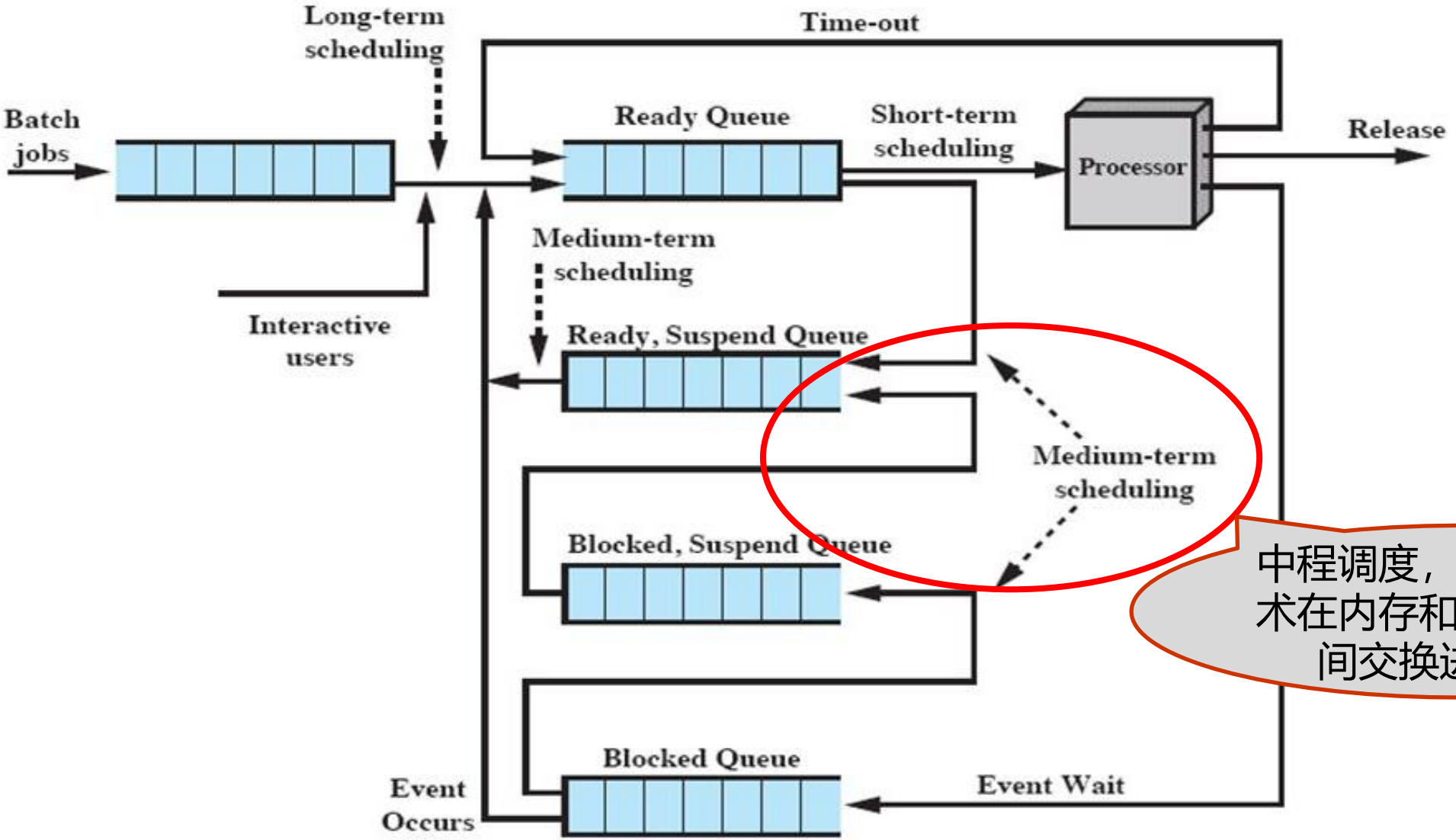


调度的层次

- **短程调度(Short-Term Scheduling)**



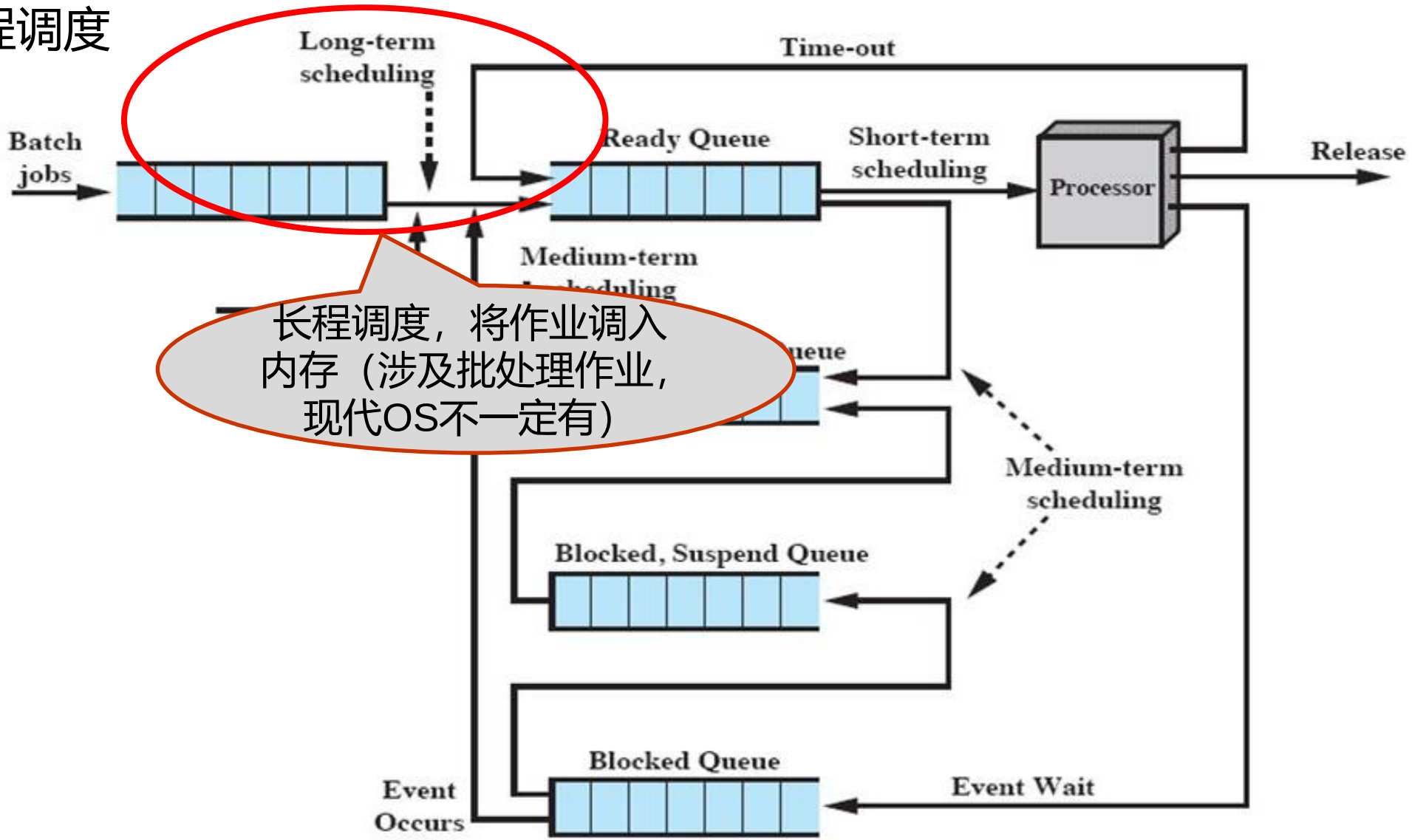
调度的层次



中程调度，通过交换技术在内存和交换空间之间交换进程映像

调度的层次

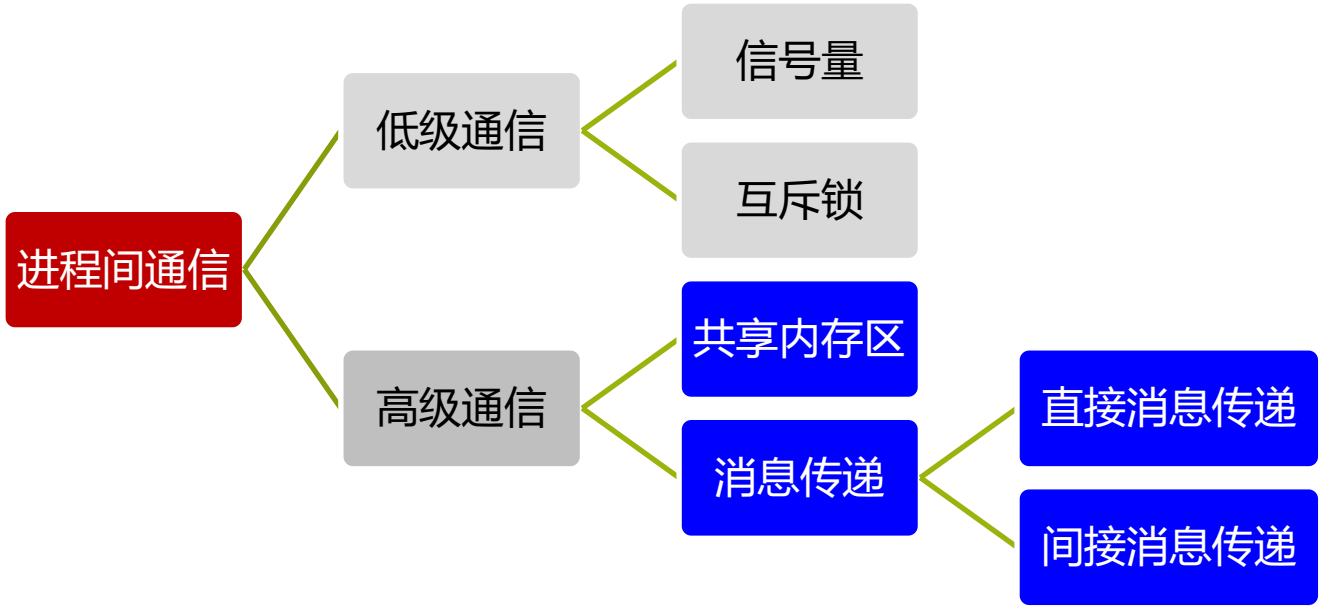
长程调度


Long-term scheduling
Time-out
Batch jobs
Ready Queue
Short-term scheduling
Processor
Release
Medium-term
长程调度，将作业调入内存（涉及批处理作业，现代OS不一定有）
Medium-term scheduling
Blocked, Suspend Queue
Blocked Queue
Event Occurs
Event Wait

进程间通信基本概念

- **为什么需要进程间通信：**
  - **进程之间的关系可能是独立(independent)，也可能是相互协作（cooperating）。**
  - **进程间的协作需要互相传递信息，因此需要专门的通信机制支持**

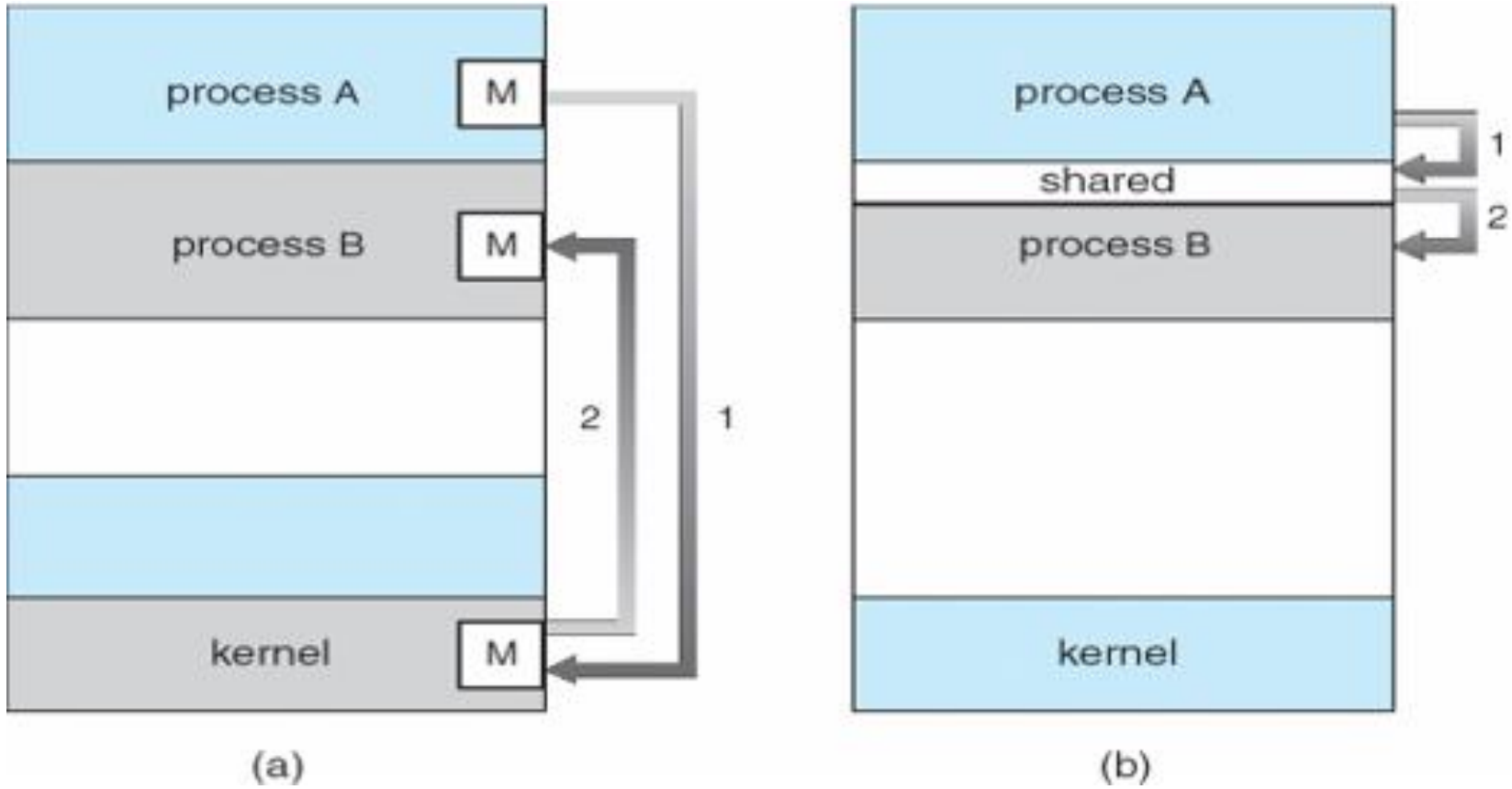
IPC分类



- 低级通信（Low-Level IPC）：用于进程控制信息的传递，传输信息量相对较小
- 高级通信：主要用于进程间信息的交换与共享，传输信息量相对较大

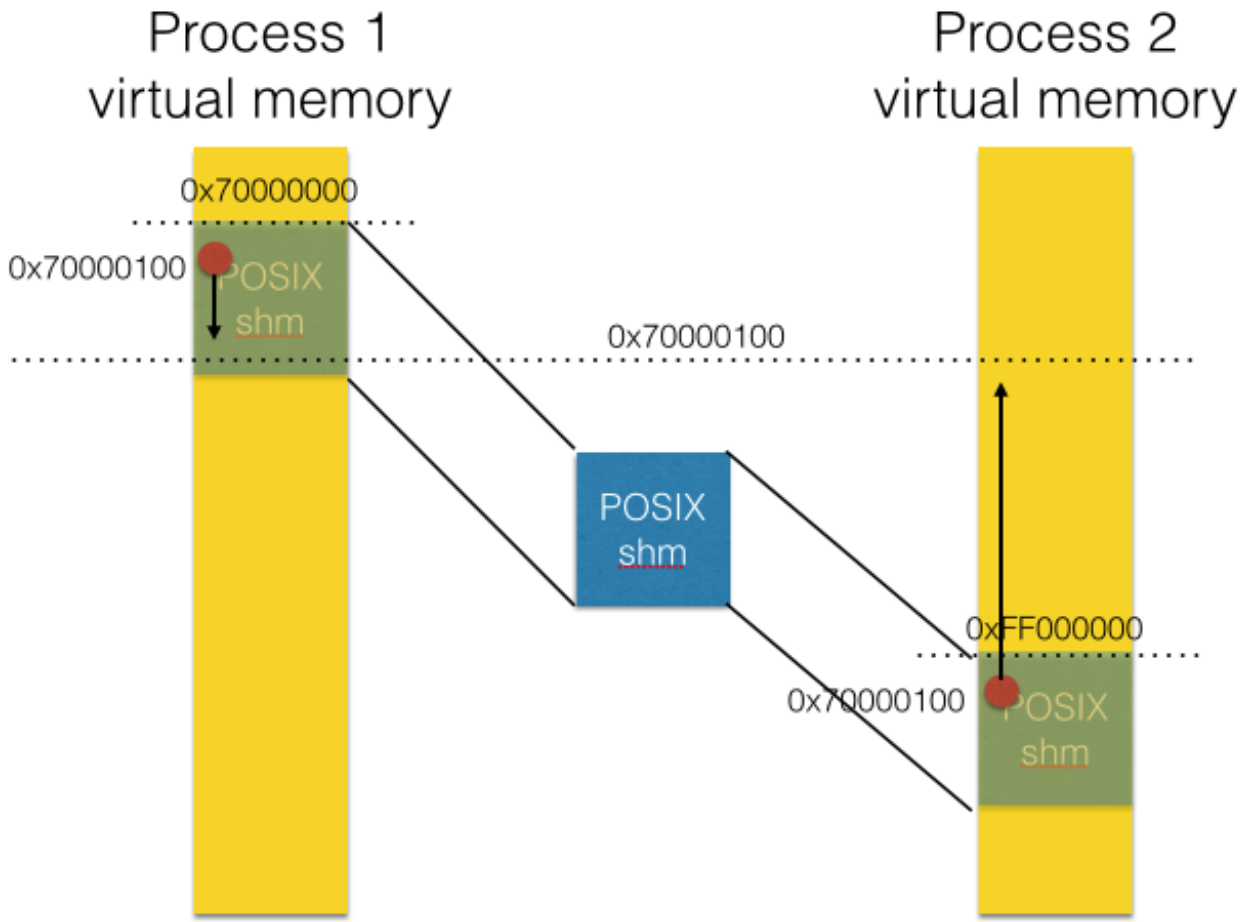
IPC分类

- **消息传递与共享内存原理示意**
  - (a) 消息传递　(b) 共享内存



IPC分类1：共享内存区

Shared Memory示例



POSIX Shared Memory

Process first creates shared memory segment 创建共享内存区

```
int shmget(key_t key, size_t size, int shmflg);
```

Process wanting access to that shared memory must attach to it

```
void * shmat(int shmid, const void *shmaddr, int shmflg);
```

Now the process could write to the shared memory

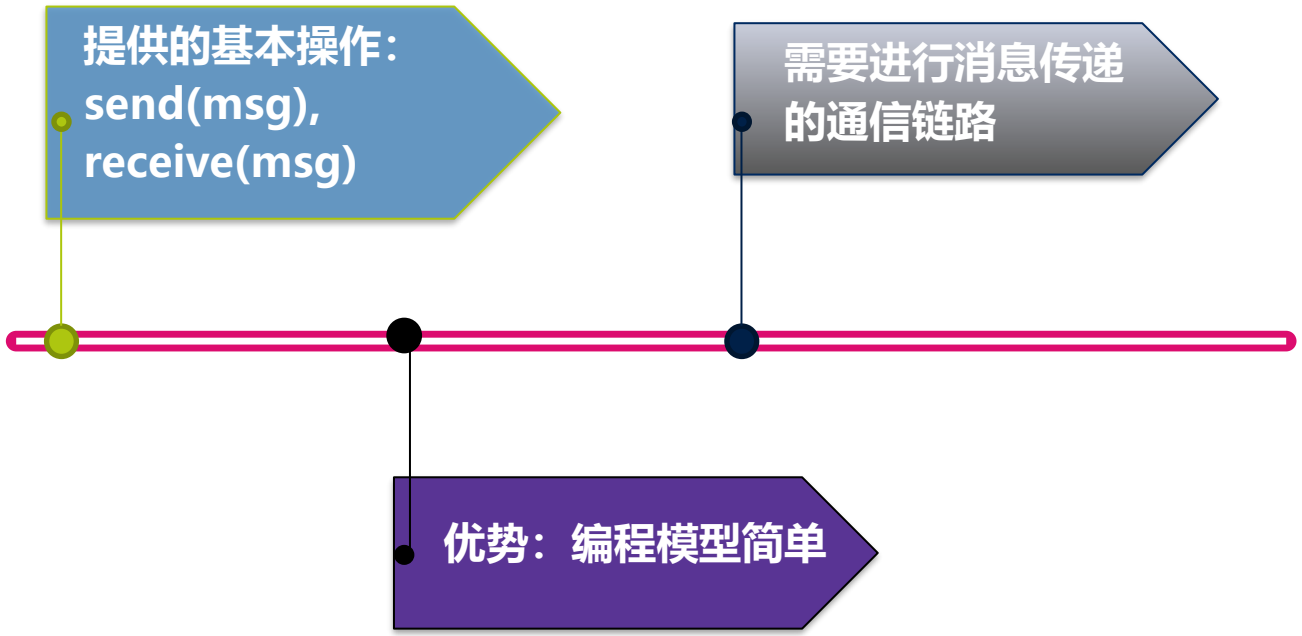
```
sprintf(shared_memory, "Writing to shared memory");
```

When done a process can detach the shared memory from its address space

```
int shmdt(const void *shmaddr);
```

IPC分类2：消息传递

Message Passing



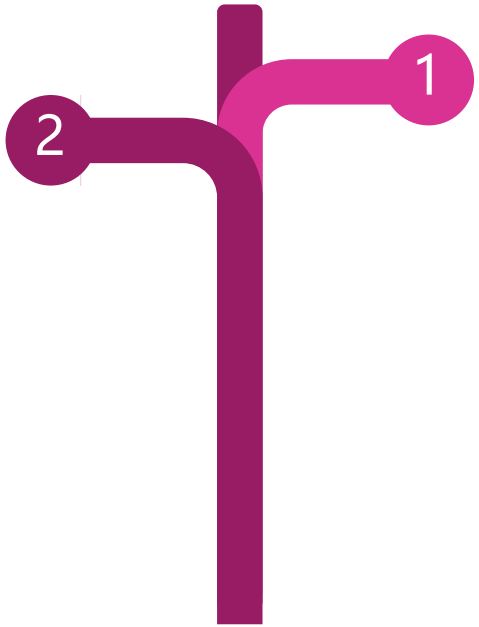
IPC分类2：消息传递

- 消息通信可细分为两类

间接消息通信

(mailbox)

不要求消息发送方和接收方同时在线



直接消息通信

要求消息发送方和接收方同时在线

消息传递实例：Mach Message based communication

- **Mach communication is message based**
  - **Even system calls are messages**
  - **Each task gets two mailboxes at creation- Kernel and Notify**
  - **Only three system calls needed for message transfer**

`msg_send()`, `msg_receive()`, `msg_rpc()`

  - **Mailboxes needed for commuication, created via**

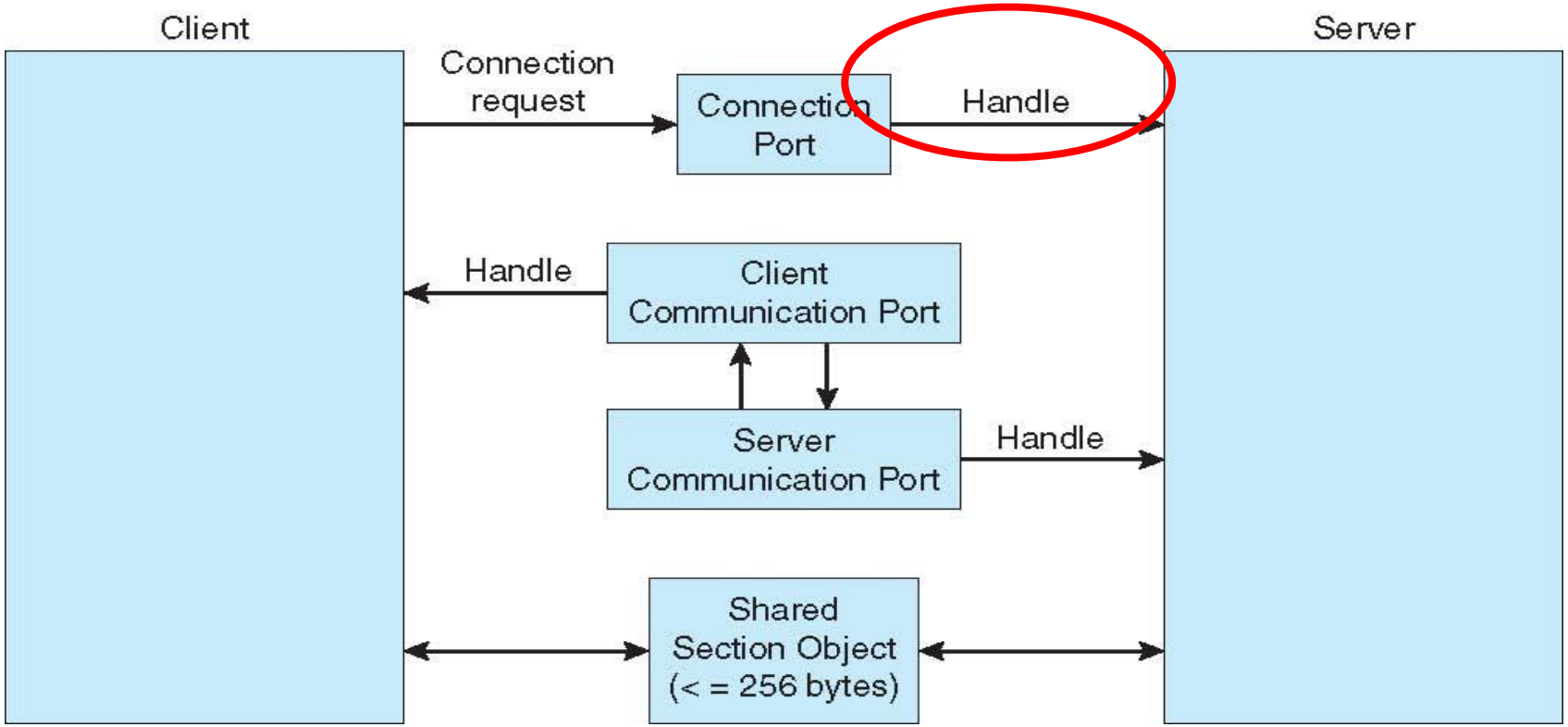
`port_allocate()`

消息传递实例：LPC of Windows XP

- **Message-passing centric via local procedure call (LPC) facility**
  - **Only works between processes on the same system**
  - **Uses ports (like mailboxes) to establish and maintain communication channels**
  - **Communication works as follows:**
    - **The client opens a handle to the subsystem’s connection port object.**
    - **The client sends a connection request.**
    - **The server creates two private communication ports and returns the handle to one of them to the client.**
    - **The client and server use the corresponding port handle to send messages or callbacks and to listen for replies.**

IPC分类2：消息传递

消息传递实例：LPC of Windows XP

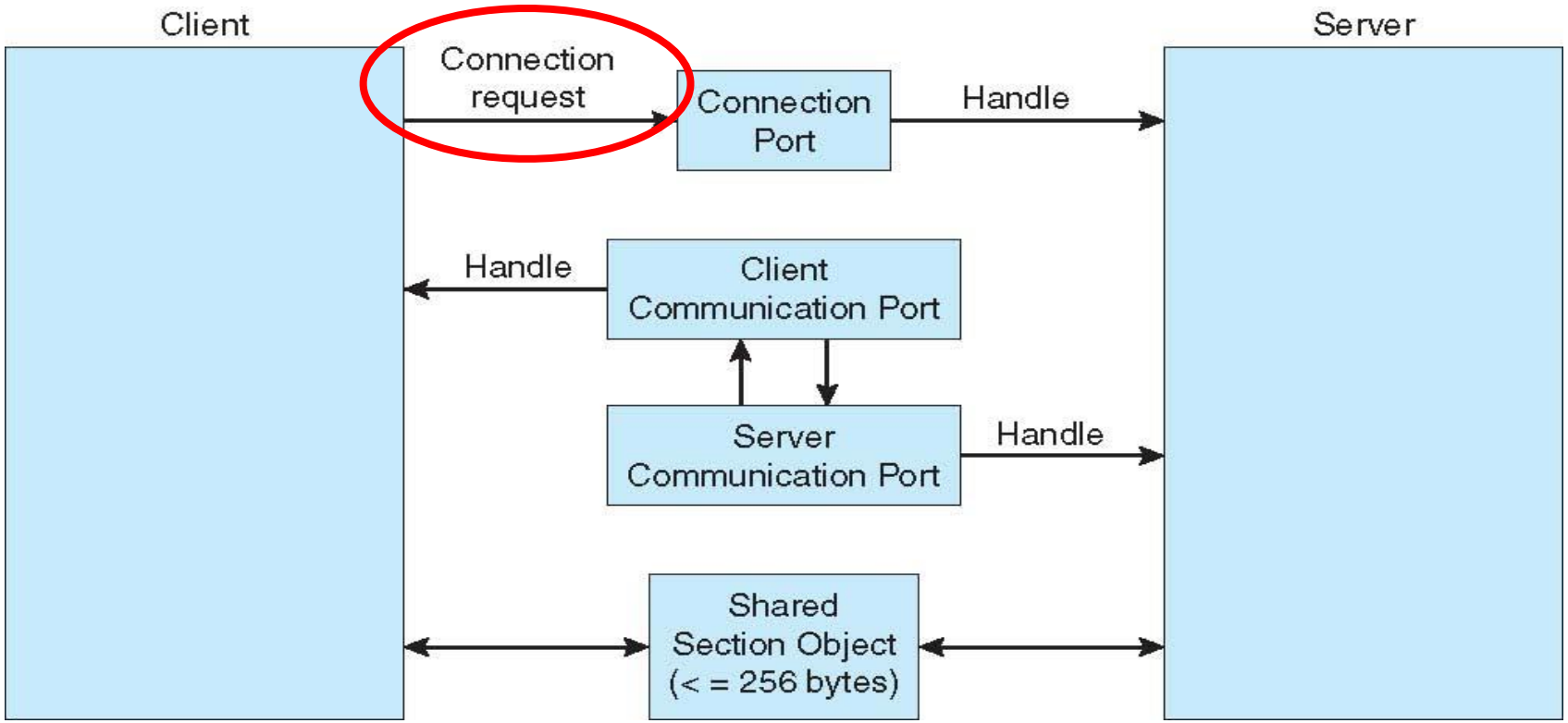


本地过程调用（**LPC**）工作步骤：

(1) The client opens a handle to the subsystem's connection port object.

IPC分类2：消息传递

消息传递实例：LPC of Windows XP

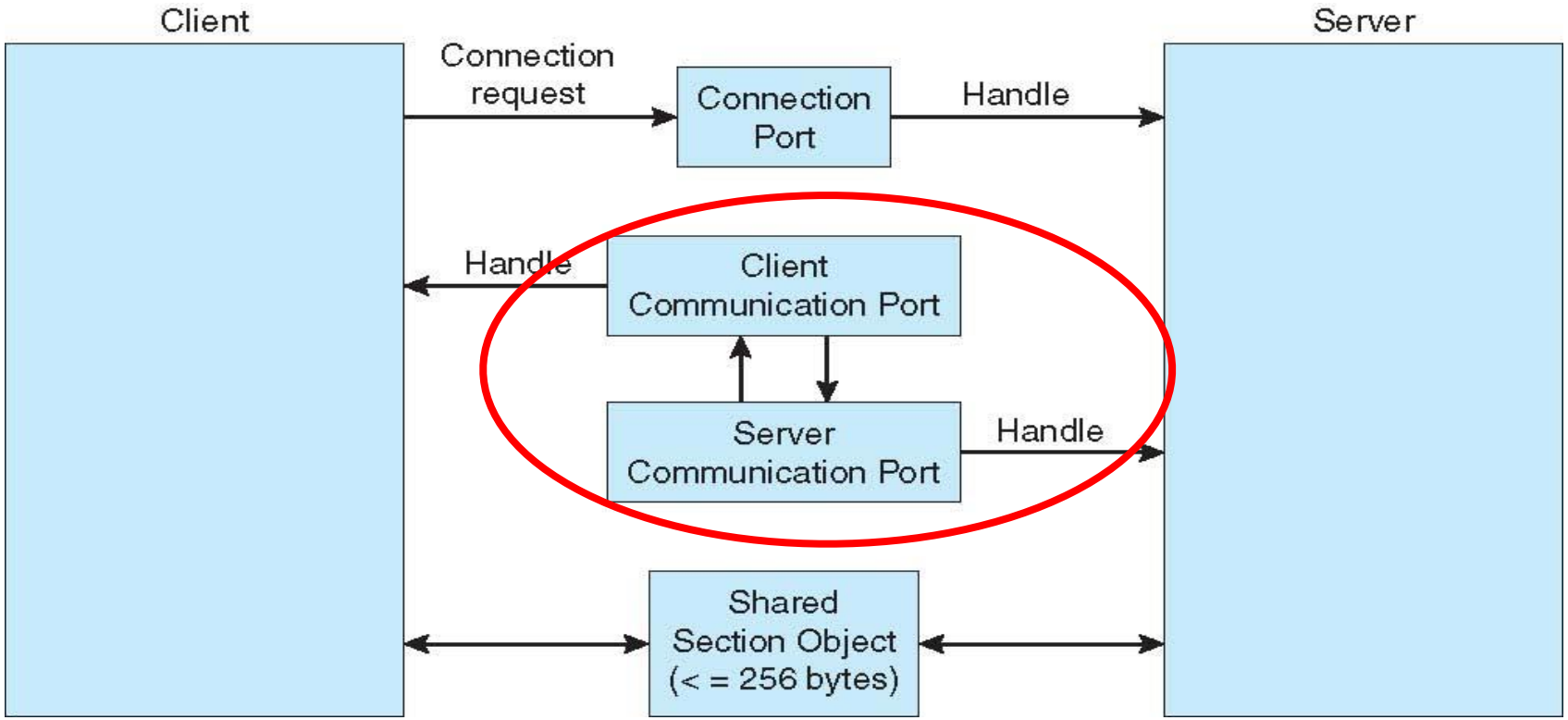


本地过程调用（**LPC**）工作步骤：

(2) The client sends a connection request.

IPC分类2：消息传递

消息传递实例：LPC of Windows XP

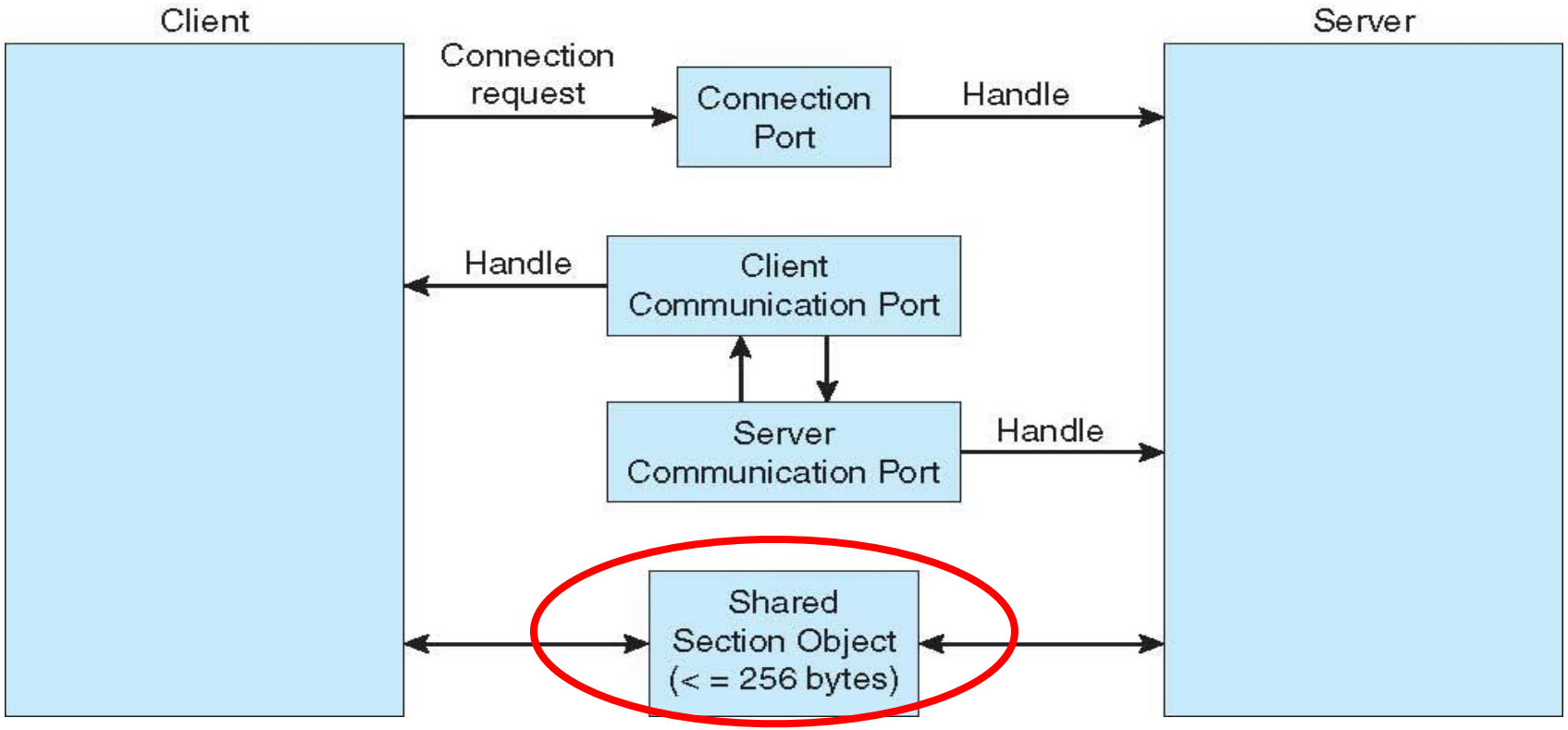


本地过程调用（**LPC**）工作步骤：

(3) The server creates two private communication ports and returns the handle to one of them to the client.

IPC分类2：消息传递

消息传递实例：LPC of Windows XP



本地过程调用（**LPC**）工作步骤：

(4) The client and server use the corresponding port handle to send messages and to listen for replies.

IPC分类2：消息传递

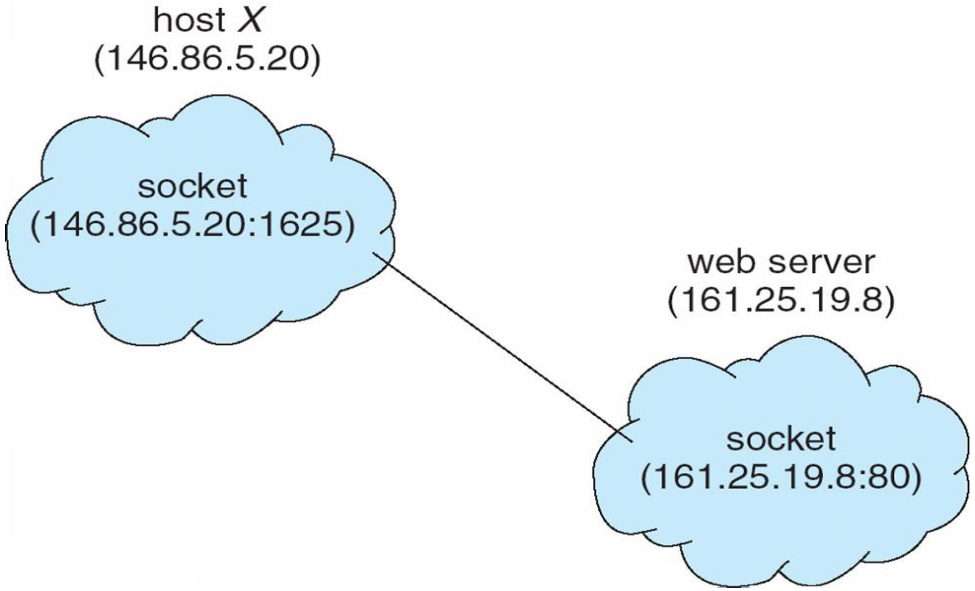
More Cases of IPC:

- **Sockets** 套接字
- **Remote Procedure Calls** 远程过程调用
- **Pipes** 管道
- **Remote Method Invocation (Java)** **Java**的远程方法调用

IPC实例：套接字

Sockets

A socket is defined as an *endpoint for communication*



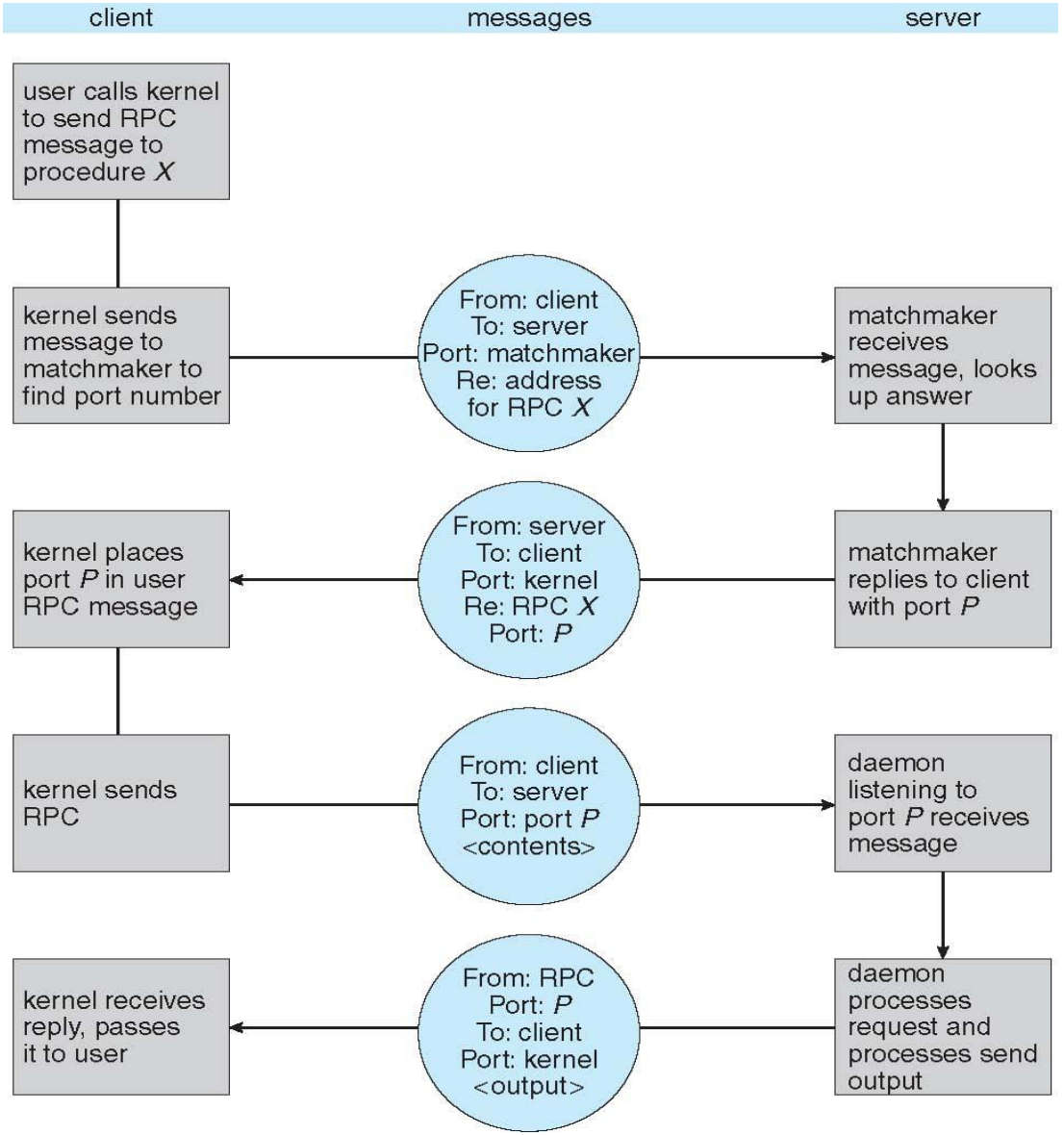
Concatenation of IP address and port

The socket 161.25.19.8:1625 refers to port 1625 on host 161.25.19.8

IPC实例：RPC

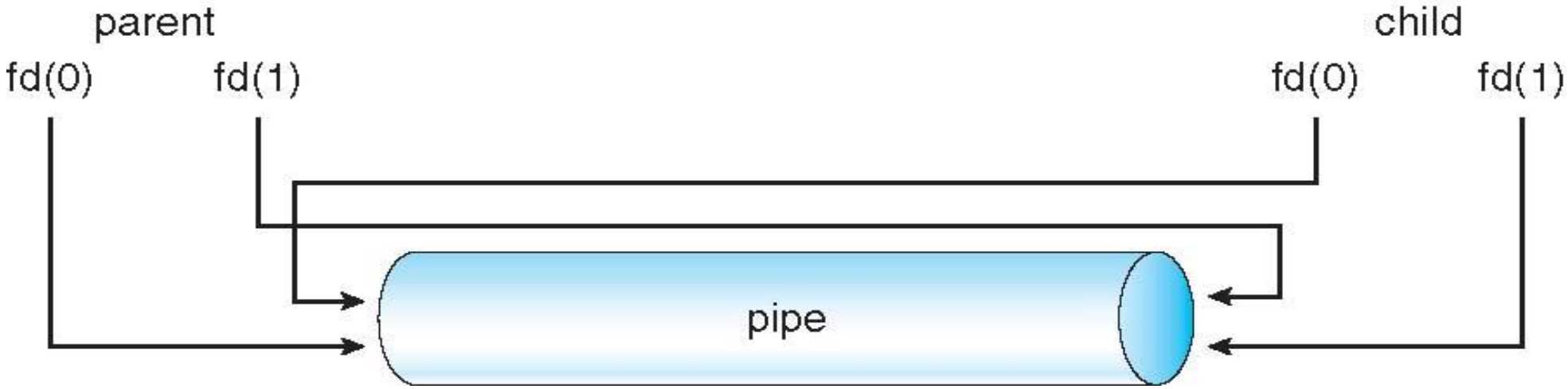
Remote Procedure Call (RPC)

abstracts procedure calls between processes on networked systems



IPC实例：管道

Pipe



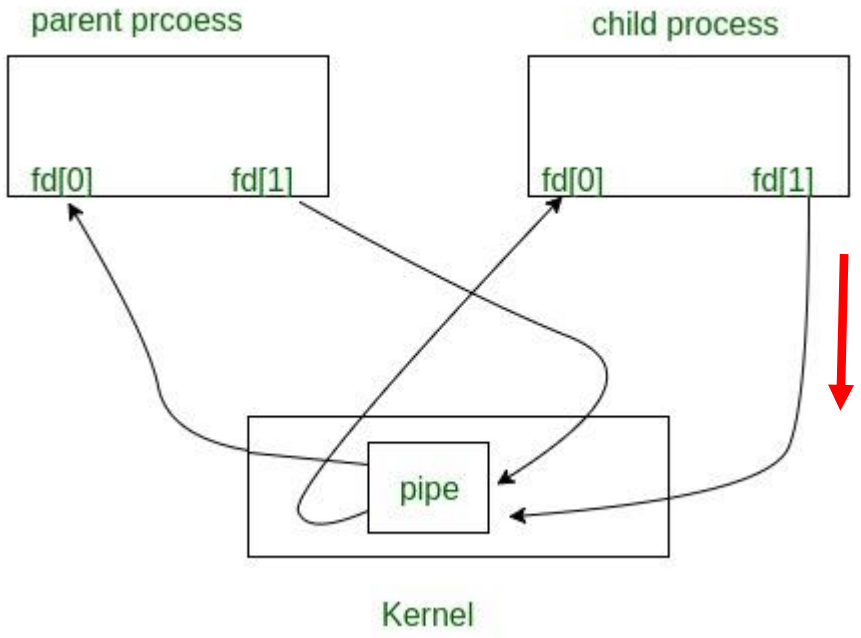
- **Linux IPC示例：管道**

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

main()
{
    int     fd[2];

    pipe(fd);
        .
        .
}
```

```
pid_t   pid = fork();

if(pid == 0)
{
    /* Child process closes up input side of pipe */
    close(fd[0]);
}
else
{
    /* Parent process closes up output side of pipe */
    close(fd[1]);
}
```

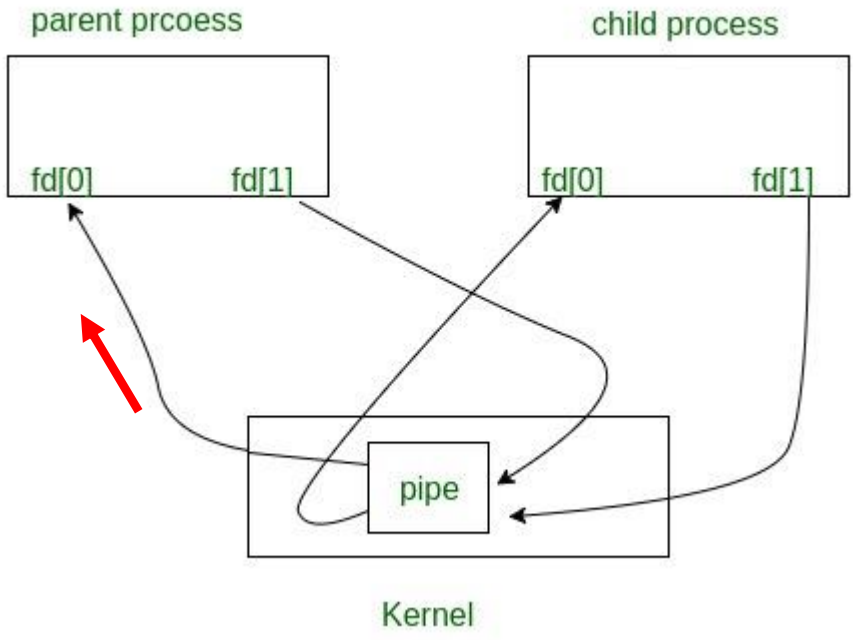
管道

```
if(pid == 0)
{
    /* Child process closes up input side of pipe */
    close(fd[0]);

    /* Send "string" through the output side of pipe */
    write(fd[1],  “string” , (strlen( “string” )+1));
    exit(0);
}
```



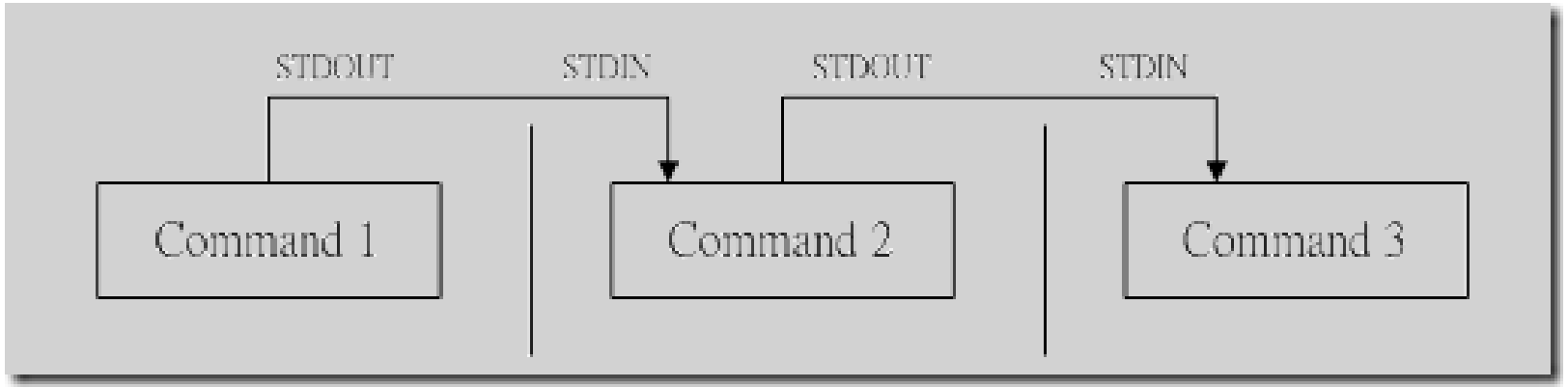
```
int nbytes;
char    readbuffer[80];
```

```
else
{
    /* Parent process closes up output side of pipe */
    close(fd[1]);

    /* Read in a string from the pipe */
    nbytes = read(fd[0], readbuffer, sizeof(readbuffer));
    printf("Received string: %s", readbuffer);
}
```



管道

- **Shell管道**
  - Linux命令行下的管道应用



- **Shell管道**
  - 示例：cat命令和head、tail命令配合显示文件特定部分

cat 命令是concatenate 的缩写，常用来显示文件内容，或者将几个文件连接起来显示

```
cat  filename  |head -n 30

cat  filename  |head -n 30 |tail -n +10
```

- **Shell管道**
  - 示例：cat命令和head、tail命令配合显示文件特定部分

```
$ cat readme.txt | head –n 2
```

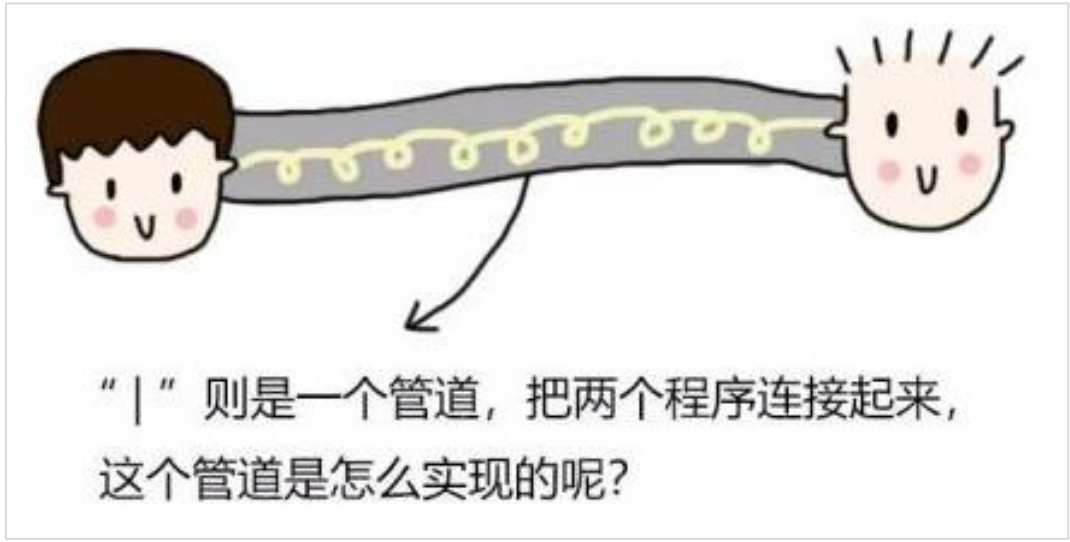
显示文件readme.txt前2行

管道

“cat”是Linux中的一个程序
可以显示文件内容

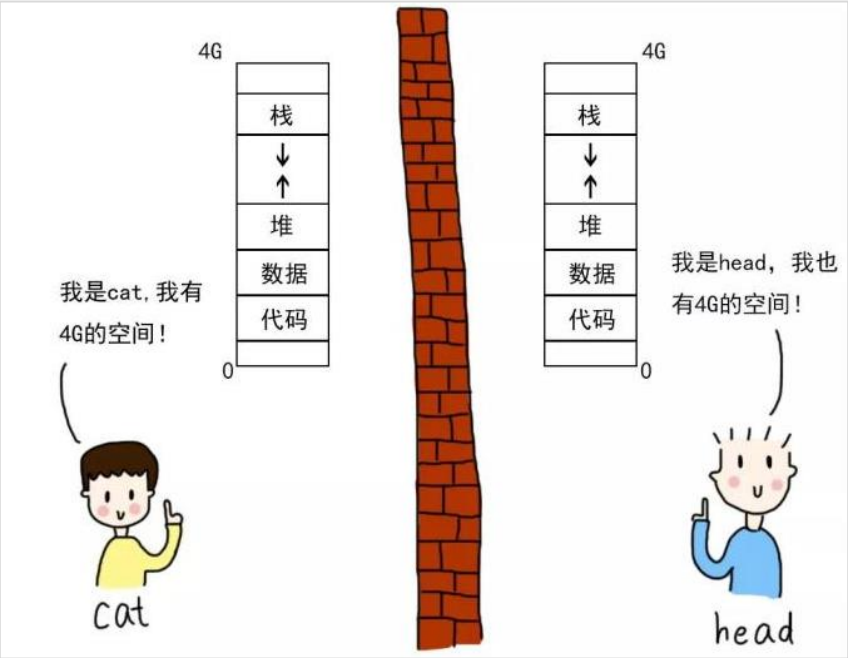
“head”也是Linux中的一个程序
可以输出一个文件的开头部分
例如输出两行

管道



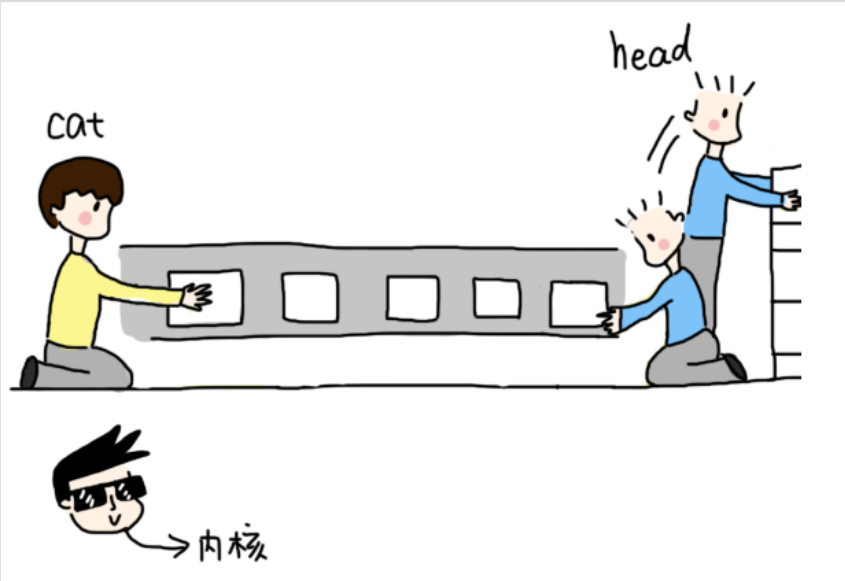
管道

- Linux内，每个进程都有自己的独立地址空间，进程之间如同横亘着一堵墙，他们如何通信?



管道

- 管道通信方案：在内核为需要通信的进程建立一条管道



Summary

小结：

- 进程调度基础概念
- 进程间通信

谢 谢!
Thank you!