3.5 进程通信
（IPC：Inter Process Communication）

- 信号通信机制
- 管道通信机制
- 消息通信机制

3.5.0 进程通信

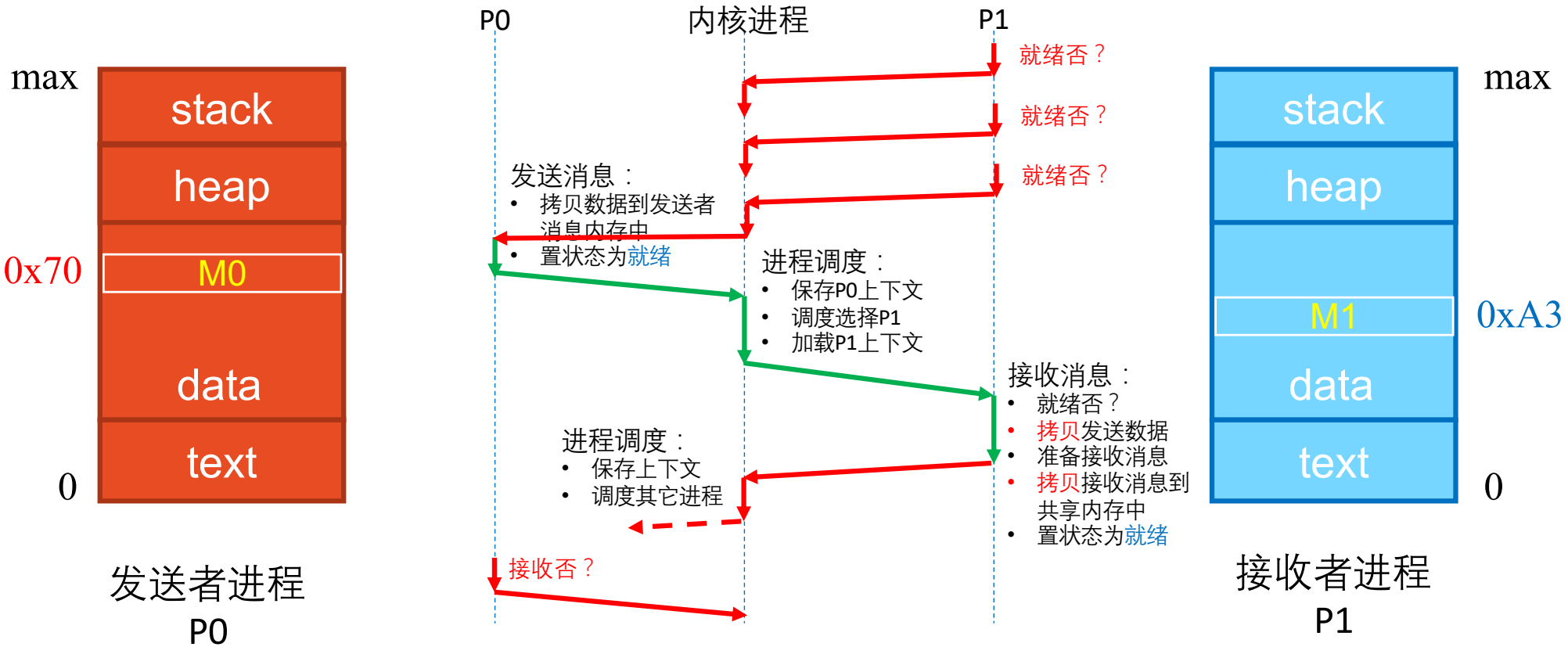
- 进程通信：进程之间的信息交换
- 低级通信：交换信息量少，通信控制复杂，对用户不透明
  - 信号通信机制（3.5.1）
  - 信号量及其原语操作

进程同步也是一种进程通信，用于在进程间传递控制信号

- 高级通信：用户直接利用OS提供的通信命令传输大量数据，通信控制细节由OS完成，对用户透明
  - 管道提供的共享文件通信机制（3.5.2）
  - 共享存储区通信机制（3.5.3）
  - 信箱和发信/收信原语的消息传递通信机制（3.5.4）

高级通信专门用于在进程间传递**大批**数据

基于共享内存的通信过程

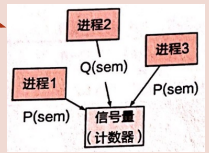
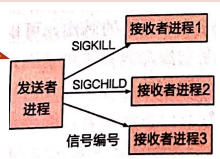
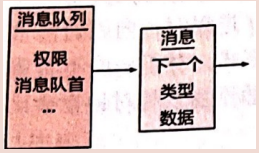
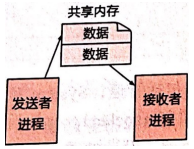
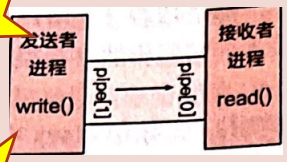
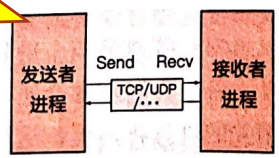


进程通信过程中的问题

- 接收消息的时机和机制？
  - 同步消息 / 异步消息，如何等待消息？
- 消息队列的长度设置？
  - 过大：浪费内存；过小：拆分消息，并导致多次通信
- 消息传递的安全性和可靠性：你真得是你吗？我怎么知道你收到了？
- 系统开销：多次的内存拷贝，指针相关的数据
- 通信方向：单向、双向？
- 参与者：两个进程间、多个进程间？
- 数据抽象：字节流 / 结构化消息 / 数值（信号量、信号） ……

减少轮询，减少内核（CPU）开销

IPC 机制

机制	示例	数据抽象	参与者	方向	内核实现
信号量		计数器	多进程	单向 双向	通过内核维护共享计数器，通过文件权限管理对计数器的访问。
信号		事件编码	多进程	单向	为进程/线程维护信号等待队列。通过用户/组等权限来管理信号的操作。
消息队列		消息	多进程	单向 双向	通过对文件的权限来管理对队列的访问。
共享内存		内存区间	多进程	单向 双向	内核维护共享的内存区间，通过文件的权限来管理对共享内存的管理
管道		字节流	两个进程	单向	以FIFO的缓冲来管理数据，通过文件系统来管理访问权限。
套接字		数据报文	两个进程	单向 双向	利用网络栈来管理通信

- 信号通信机制
- 管道通信机制
- 消息通信机制

3.5.1 信号通信机制

- 信号机制（signal）又称软中断，是一种进程之间进行通信的简单通信机制
- 信号的重要特点是**单向的（异步）事件通知能力**
  - 信号量（semaphore）也可以通知事件，但需要进程主动地查询计数器的状态或者陷入阻塞状态
- 一个进程可以随时发送一个事件到特定的进程/线程
- 接收信号的进程不需要阻塞来等待事件，内核会帮助其切换到响应该信号的处理函数中

3.5.1 信号通信机制

- 典型的信号： Ctrl + C

1. 用户按中断组合键 Ctrl+C；
2. 终端驱动程序收到输入字符，并调用信号系统；
3. 信号系统发送SIGINT信号给shell，shell再把它发送给进程；
4. 进程受到SIGINT信号；
5. 进程撤销。

3.5.1：与硬中断的差别

- 软中断运行在用户态，往往延时较长
- 硬中断运行在核心态，能及时发现

3.5.1：信号的发送

- 可以由用户、进程、内核发送给其他的进程。
  - 用户：通过特殊键（如Ctrl+C），请求内核产生信号
  - 进程：通过系统调用kill(pid,sig)；利用信号与其它进程通信
  - 内核：程序执行出错，如非法段存取、浮点数溢出、非法操作码；通知进程特定事件

- 同步信号：进程执行指令产生的信号称为同步信号

- 异步信号：进程以外的事件所引起的信号称为异步信号

3.5.1：信号处理

- 在 Linux 等 UNIX 系统中，处理信号的时机：
  - 异常、中断处理结束后返回用户态的时刻
  - ret_to_user ：Linux 中从内核返回用户态的函数
  - 此时，内核会检查状态位来判断是否有信号需要处理，如果有，就去先处理该信号事件。

- 软中断运行在用户态，往往延时较长
- 硬中断运行在核心态，能及时发现

3.5.1：信号处理

- 忽略此信号：大多数信号可以被忽略，除了SIGKILL（除了init进程）和SIGSTOP。
- 捕捉信号：通知内核在某种信号发生时，调用一个用户函数。但不能捕捉SIGKILL和SIGSTOP。
- 执行系统默认动作：
  - 终止(+core) [大多数信号如此默认响应，core文件用于调试]
  - 忽略
  - 继续

3.5.1：信号处理

- 信号处理函数
  - 用户可以自定义信号处理函数
  - 需要函数是可重入（reentrant）的:避免使用全局共享变量

在两条语句间，可能会再次收到信号并执行该函数

```
static int counter = 0;


void sig_handler(int signo){
   counter++;
   printf("Got signal, the counter:%d\n",counter);
}
```

3.5.1：Linux下的信号

分类	信号
终止进程	SIGCHLD, SIGHUP, SIGKILL, SIGABRT, SIGSTOP, SIGTSTP, …
例外事件	SIGBUS, SIGSEGV, SIGPWR, SIGFPE, SIGSTKFLT, SIGURG, …
执行系统调用	SIGPIPE, SIGILL, SIGIO, …
终端交互	SIGINT, SIGQUIT, SIGTTIN, SIGTTOU, …
用户进程相关	SIGTERM, SIGALRM, SIGUSR1, SIGUSR2, …
跟踪进程执行	SIGTRAP

- 信号通信机制
- 管道通信机制
- 消息通信机制

3.5.2 示例

Linux 下执行命令时使用管道：

```
who | grep martin | wc -l
```

3.5.2：管道分类

- 匿名管道（unnamedpipe）
  - 是在父进程和子进程之间，或同一父进程的两个子进程之间传输数据的无名字的单向管道
- 命名管道（namedpipe）
  - 是服务器进程和一个或多个客户进程之间通信的单向或双向管道

3.5.2 管道通信机制

- 管道（UNIX, Windows）是连接读写进程，实现他们之间通信的一个特殊文件。属于一种共享文件通信机制
- 管道是一个共享文件，连接读写进程实现通信。写进程往管道一端写入信息，读进程从管道另一端读信息。
- 管道借助于文件系统的机制实现，包括（管道）文件的创建、打开、关闭和读写

3.5.2（续）

- 管道基于文件系统，利用一个打开的共享文件连接两个相互通信的进程，文件作为缓冲传输介质

以字符流方式写入读出，按先进先出方式传送数据

3.5.2：命名管道

- 不同于匿名管道，命名管道可以在不相关的进程之间和不同计算机之间使用，服务器建立命名管道时给它指定一个名字，任何进程都可以通过该名字打开管道的另一端，根据给定的权限和服务器进程通信

3.5.2：命名管道

Terminal 1

```
• $ mkfifo /tmp/test.fifo
• $ echo "hello world!" > /tmp/test.fifo
```

Terminal 2

```
• $ read a < /tmp/test.fifo
• $ echo $a
```

3.5.2：匿名管道

- 通常由父进程创建管道，然后由要通信的子进程继承通道的读端点句柄或写端点句柄，然后实现通信
- 父进程还可以建立两个或更多个继承匿名管道读和写句柄的子进程。这些子进程可以使用管道直接通信，不需要通过父进程。匿名管道是单机上实现子进程标准I/O重定向的有效方法，它不能在网上使用，也不能用于两个不相关的进程之间

3.5.2 匿名管道示例

```
1.  int pipe_fd[2]; //fd[0]只读；fd[1]只写
2.  if(pipe(pipe_fd) < 0) { return -1;} //pipe create error
3.  if((pid=fork())==0) {  //子进程
4.   close(pipe_fd[1]);
5.   read(pipe_fd[0], r_buf, 4);
6.   …
7.   close(pipe_fd[0]);
8.  } else if ( pid > 0){ //父进程
9.   close(pipe_fd[0]);
10.  write(pipe_fd[1], w_buf[i], 4);
     …
11.  close(pipe_fd[1]);
12. }
```

3.5.2：使用管道的注意事项

- 管道机制应具备的三个协调功能：
  - 互斥。一次仅由一个进程读写。（通过测试i_node节点文件的特征位来保证，该特征位就是一个读写互斥标志）
  - 确定对方是否存在
  - 同步。睡眠和唤醒功能。进程在关闭管道的读出或写入时，应唤醒等待写或读此管道的进程

- 信号通信机制
- 管道通信机制
- 消息通信机制

3.5.4 消息通信机制

- 是一种高级的通信方式
- 操作系统隐蔽消息传递的实现细节，简化应用程序编制的复杂性
- 消息传递的复杂性在于：地址空间的隔离，发送进程无法直接将消息复制到接收进程的地址空间中，该工作由操作系统实现
- 提供两个原语：send和receive

3.5.4（续）

- 采用消息传递机制后，进程间通过消息来交换信息
  - 一个正在执行的进程可在任何时刻向另一个正在执行的进程发送消息
  - 一个正在执行的进程也可在任何时刻向正在执行的另一个进程请求消息
- 消息传递不仅具有进程通信的能力，还提供进程同步的能力
  - 如果进程在某一时刻的执行依赖于另一进程的消息或等待其他进程对所发消息的应答，则消息机制与进程的阻塞和释放相联系

3.5.4：通信方式

- 直接通信方式（消息缓冲区）
  - 发送或接收消息的进程必须指出消息发给哪个进程或从哪个进程接收消息。
  - 原语send（P，消息）：把一个消息发送给进程P
  - 原语receive（Q，消息）：从进程Q接收一个消息

3.5.4：通信方式

- 直接通信方式（消息缓冲区）
  - 发送或接收消息的进程必须指出消息发给哪个进程或从哪个进程接收消息。
  - 原语send（P，消息）：把一个消息发送给进程P
  - 原语receive（Q，消息）：从进程Q接收一个消息
- 间接通信方式（信箱）
  - 进程间发送或接收消息通过信箱进行，消息可被理解成信件
  - 信箱是存放信件的存储区域，每个信箱可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、信件位置指针等
  - 信箱体用来存放信件，可分成若干个区，每个区容纳一个信件

3.5.4：间接通信原语

- 原语send（A，信件）：若信箱未满，则把一封信件（消息）发送到信箱A，同时唤醒信件等待者进程，否则发送者阻塞.
- 原语receive（A，信件）：若信箱不空，则从信箱A接收一封信件（消息），同时唤醒等待发送者进程；否则接受者阻塞.

与命名管道的区别？

3.5.4：send的同步方式

- 当发送进程执行send发出一封信件后，它本身的执行可以分两种情况：
  - 阻塞型：等待接收进程回答消息后才继续进行下去
  - 非阻塞型：发出信件后不等回信立即执行下去，直到某个时刻需要接收进程送来的消息时，才对回答信件进行处理

3.5.4：receive的同步方式

- 对于接收进程，执行receive后也可以是阻塞型和非阻塞型的
  - 阻塞型：指直到信件交付完成它都处于等待信件状态，进程处于等待/挂起的状态
  - 非阻塞型：不要求进程等待，以轮询方式查询信箱，有消息才进行处理，否则向调用进程返还控制权

3.5.4：消息通信的同步机制

- 阻塞型send和阻塞型receive：用于进程不设缓冲区的紧密同步方式。两个进程平时都处于阻塞状态，直到有消息传递才被唤醒工作
- 非阻塞型send和阻塞型receive：发送进程不阻塞，可把一个或多个消息发给目标进程，接收进程平时都处于阻塞状态，直到有消息传来才被唤醒。如C/S工作模式
- 非阻塞型send和非阻塞型receive：如能容纳多个消息的消息队列连接的收发进程只有在队列满或空时才有其中之一阻塞