



进程间通信(IPC)

SSE202/204: 操作系统原理

苏玉鑫

suyx35@mail.sysu.edu.cn

助教: 龙玉丹 单诗雯 毛晨希 沈志轩 郑灿峰 胡伟峰



- 部分内容来自：上海交通大学并行与分布式系统研究所操作系统课件
 - <https://ipads.se.sjtu.edu.cn/courses/os/>
- 其它参考资料：
 - 清华大学操作系统公开课
 - <https://open.163.com/newview/movie/courseintro?newurl=ME1NSA351>
 - 介绍标准内容，适合考研
 - 南京大学计算机软件研究所
 - <http://jyywiki.cn/OS/2025/>
 - <https://space.bilibili.com/202224425/channel/collectiondetail?sid=192498>
 - 比较有趣

两个进程是如何实现共享内存的?

A

只有线程才共享内存，进程不能共享内存

B

通过修改页表项权限，可以实现共享内存

C

通过修改页表基地址寄存器，可以实现共享内存

D

通过写时拷贝机制，可以实现共享内存

E

通过大页的形式，可以实现共享内存

提交

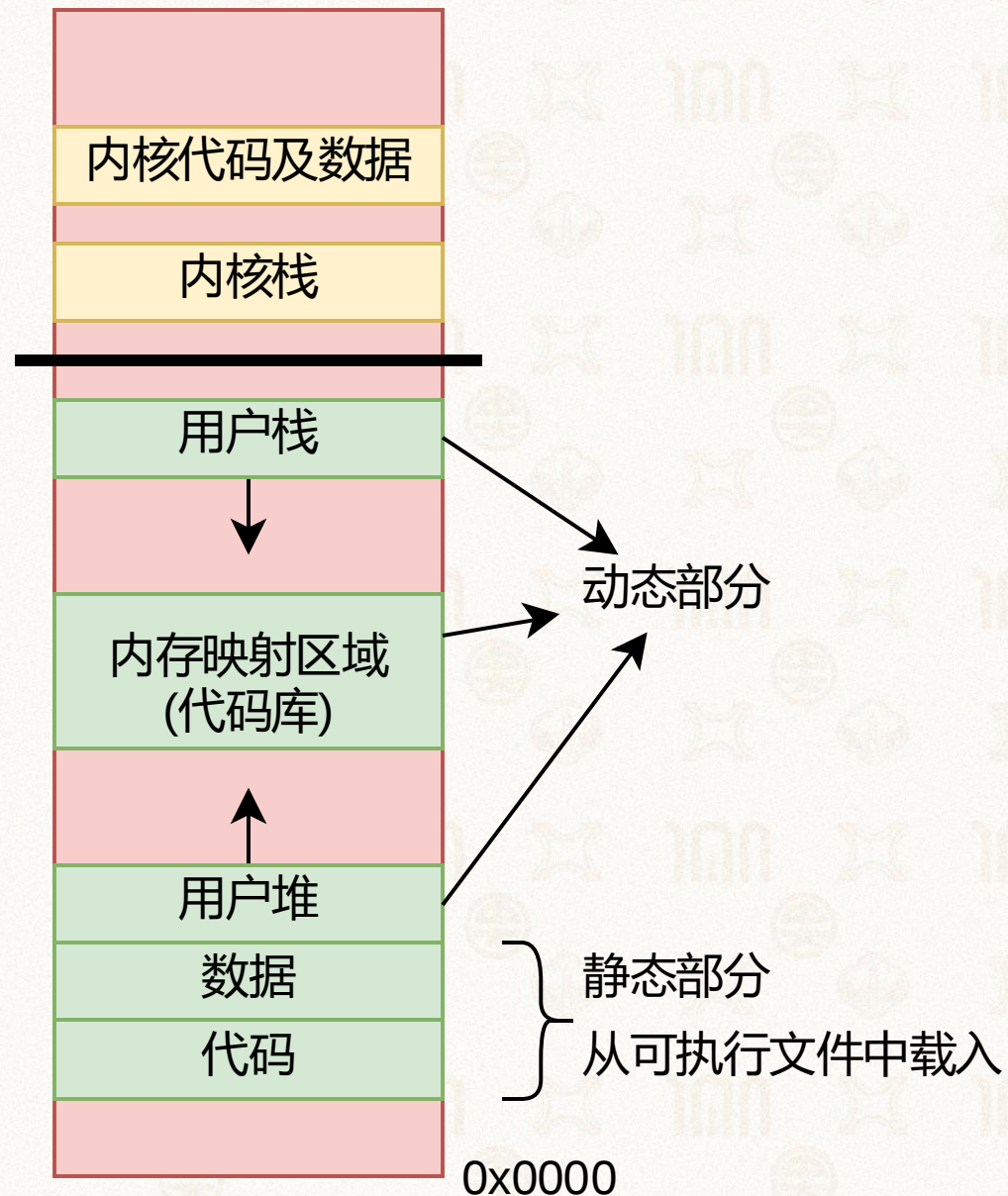
- 通信基础
- 共享内存通信
- 消息传递
- 管道
- 消息队列



进程：运行中的程序



- 进程是计算机程序运行时的抽象
 - 静态部分：程序运行需要的代码和数据
 - 动态部分：程序运行期间的状态（程序计数器、堆、栈.....）
- 进程具有独立的虚拟地址空间
 - 每个进程都具有“独占全部内存”的假象
 - 内核中同样包含内核栈和内核代码、数据





应用程序的功能非常复杂



- 独立进程：一个进程就是一个应用
 - 不会去影响其他进程的执行，也不会被其他进程影响
- 单个应用的功能非常复杂

任务管理器							
文件(F) 选项(O) 查看(V)							
进程 性能 应用历史记录 启动 用户 详细信息 服务							
名称	PID	状态	用户名	CPU	内存(活动的...	体系结构	描述
Acrobat.exe	21288	正在运行	yxsu	00	190,152 K	x86	Adobe Acrobat DC
AcroCEF.exe	10324	正在运行	yxsu	00	3,388 K	x86	Adobe AcroCEF
AcroCEF.exe	20336	正在运行	yxsu	00	876 K	x86	Adobe AcroCEF
acrotray.exe	16516	正在运行	yxsu	00	360 K	x86	AcroTray
AggregatorHost.exe	6436	正在运行	SYSTEM	00	728 K	x64	AggregatorHost.exe
ApplicationFrameHost.e...	13944	正在运行	yxsu	00	5,124 K	x64	Application Frame Hc
audiodg.exe	8808	正在运行	LOCAL SE...	00	4,380 K	x64	Windows 音频设备图
CAJSHost.exe	4676	正在运行	SYSTEM	00	60 K	x86	TTKN@CAJHost
ChsIME.exe	10004	正在运行	yxsu	00	92 K	x64	Microsoft IME
ChsIME.exe	16444	正在运行	SYSTEM	00	1,312 K	x64	Microsoft IME
Code.exe	23500	正在运行	yxsu	00	35,304 K	x64	Visual Studio Code
Code.exe	3324	正在运行	yxsu	00	3,632 K	x64	Visual Studio Code
Code.exe	27952	正在运行	yxsu	00	43,360 K	x64	Visual Studio Code
Code.exe	27144	正在运行	yxsu	00	6,112 K	x64	Visual Studio Code
Code.exe	26480	正在运行	yxsu	00	121,416 K	x64	Visual Studio Code
Code.exe	19608	正在运行	yxsu	00	40,708 K	x64	Visual Studio Code
Code.exe	18752	正在运行	yxsu	00	13,364 K	x64	Visual Studio Code
Code.exe	25048	正在运行	yxsu	00	258,800 K	x64	Visual Studio Code
Code.exe	23492	正在运行	yxsu	00	15,496 K	x64	Visual Studio Code



应用程序的功能非常复杂



1924-2024
中山大學 世紀華誕
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

- 独立进程：一个进程就是一个应用
 - 不会去影响其他进程的执行，也不会被其他进程影响
- 单个应用的功能非常复杂



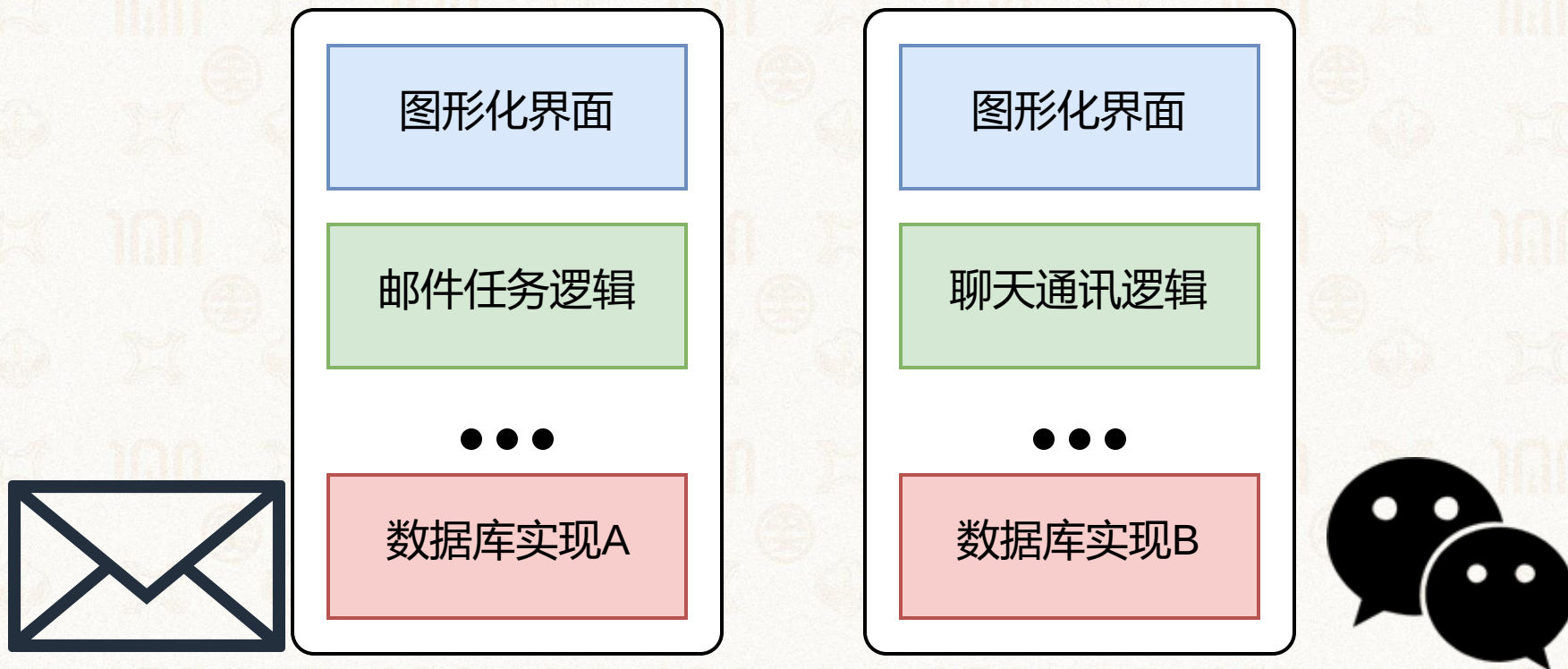


独立进程的问题1: 大量重复实现



1924-2024
中山大學 世纪华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

- 聊天软件和邮件软件都依赖数据库
- 各自实现一份在自己的进程中



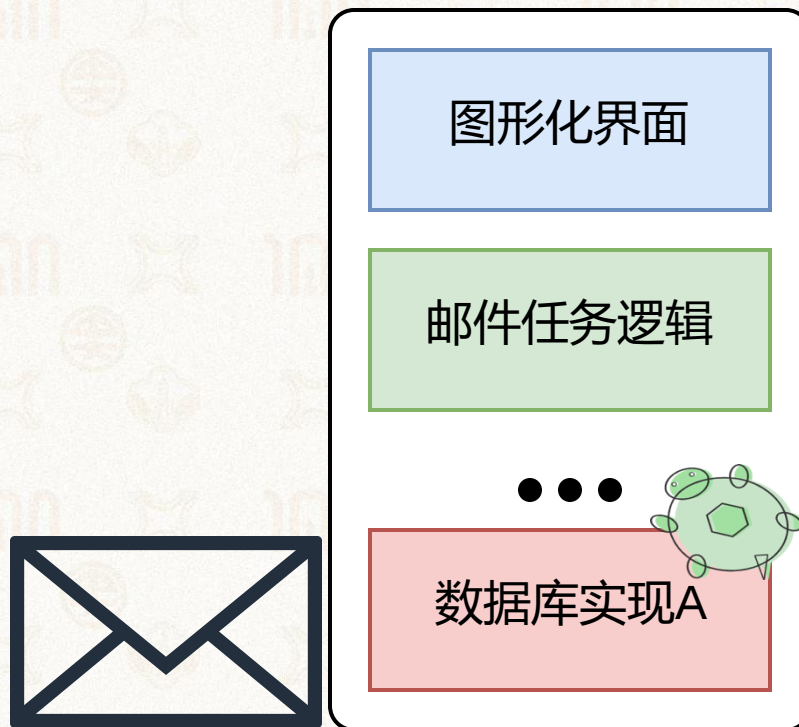


独立进程的问题2: 低效实现



1924-2024
中山大學 世紀華誕
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

- 聊天软件的数据库实现经过精心的优化
- 邮件软件团队的开发重心在其他组件上
 - 借用低效的某数据库开源实现





独立进程的问题3: 没有信息共享

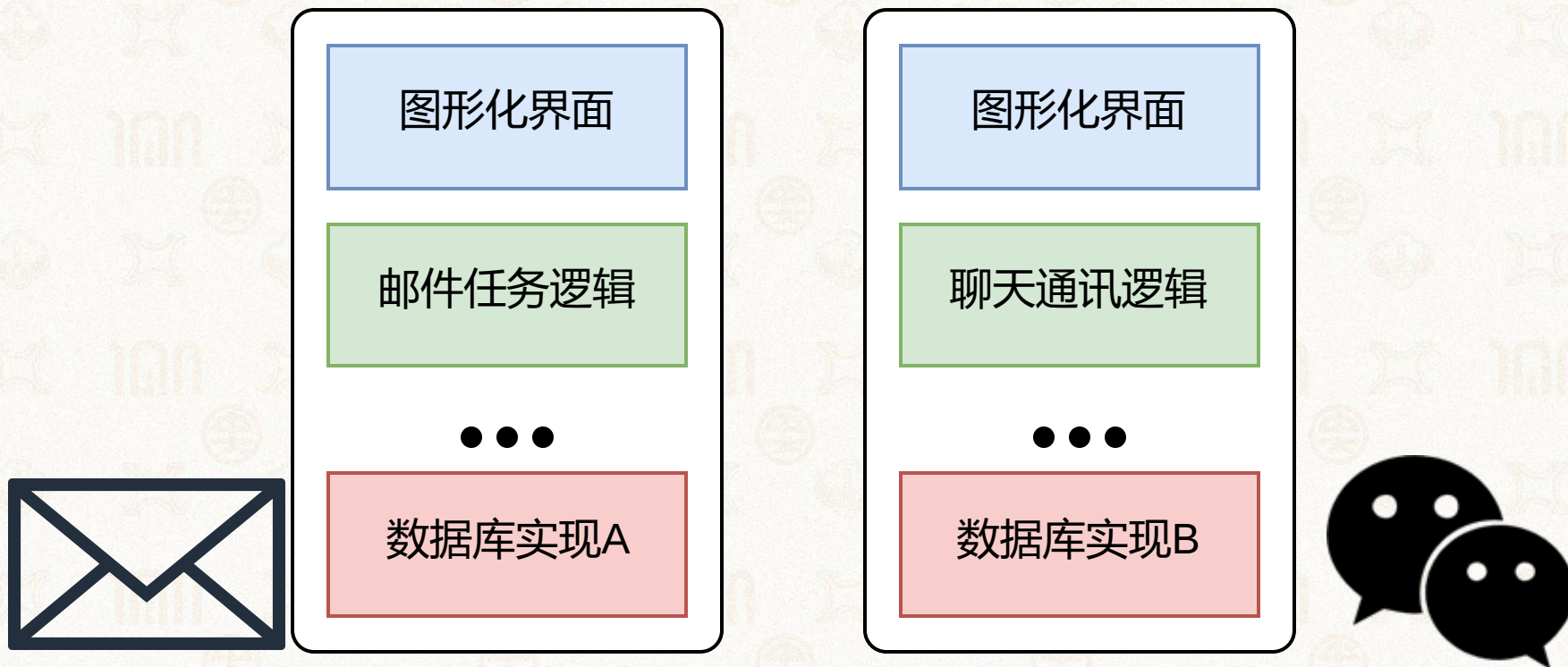


1924-2024
中山大學 世紀華誕
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

➤ 邮件和聊天软件都需要监控系统资源信息

➤ 没有信息共享

- 即使邮件软件已经完成了计算，聊天软件也要重新计算一遍





如果进程可以协作

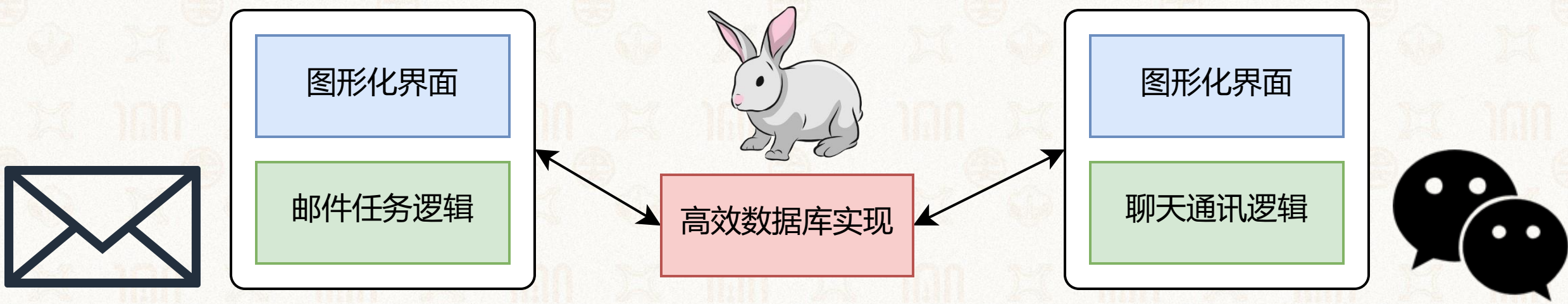


➤ 协作进程

- 和独立进程相反，可以影响其他进程执行或者被影响

➤ 好处

- 模块化: 数据库单独在一个进程中，可以被复用
- 加速计算: 不同进程专注于特定的计算任务，性能更好
- 信息共享: 直接共享已经计算好的数据，避免重复计算





进程间通信 (Inter-process Communication, IPC)

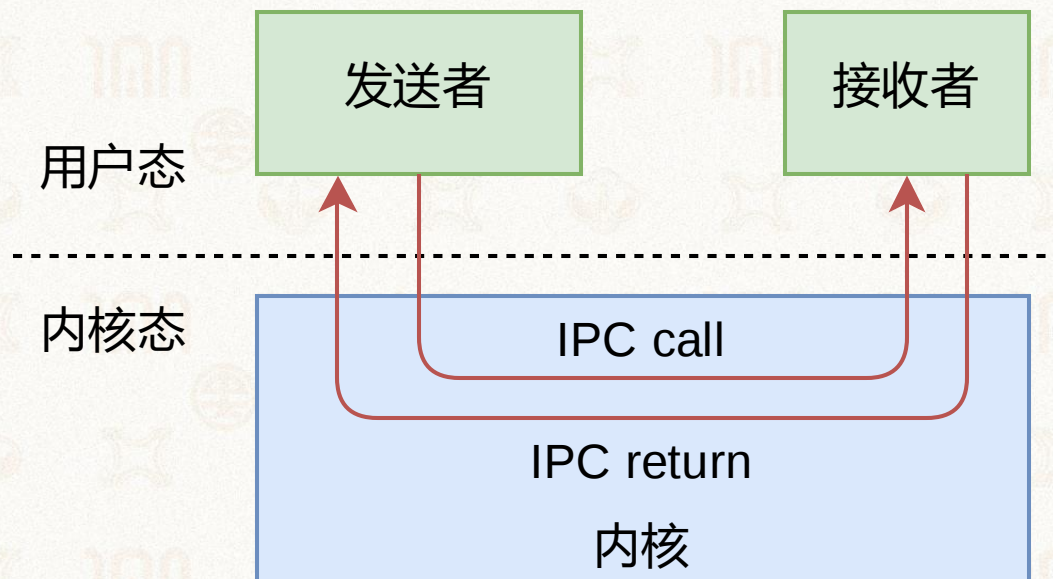


1924-2024
中山大學 世纪华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

➤ 进程协作的达成依赖于进程间通信

➤ 进程间通信:

- 两个(或多个)不同的进程, 通过内核或其他共享资源进行通信, 来传递控制信息或数据
- 交互的双方: 发送者/接收者、客户端/服务端、调用者/被调用者
- 通信的内容一般叫做“消息”





大纲



1924-2024
中山大學 世纪华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

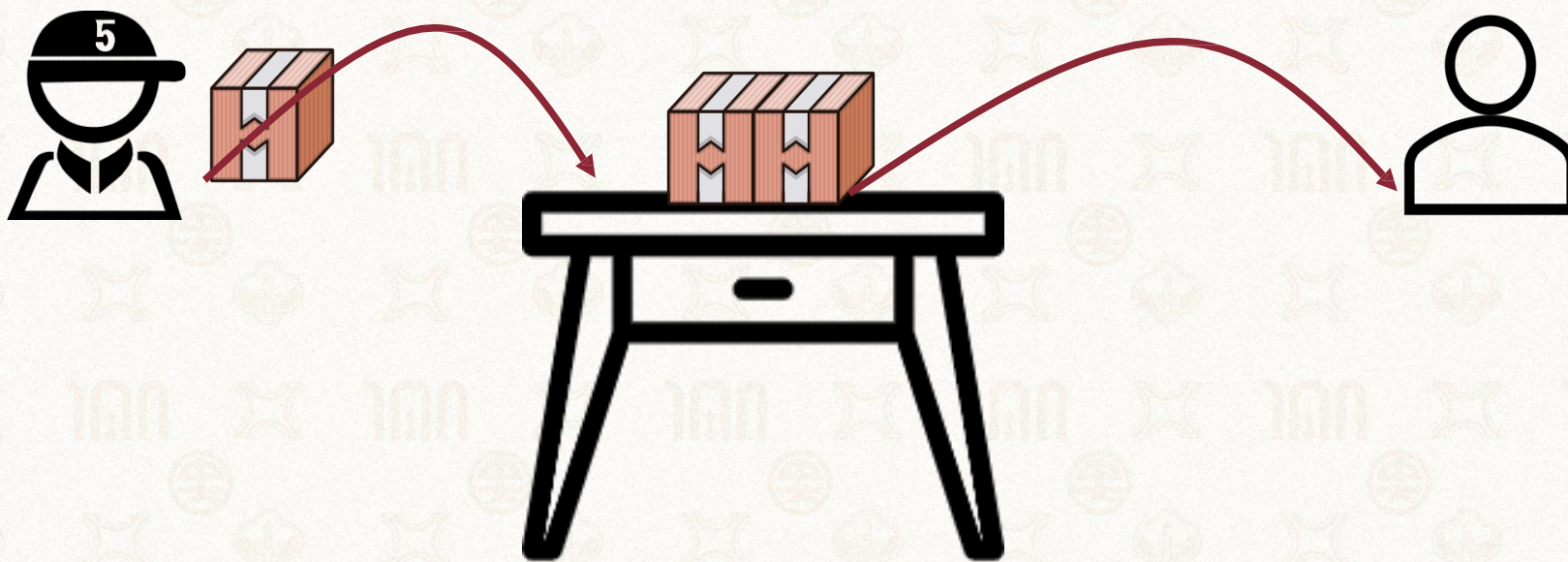
- 通信基础
- 共享内存通信
- 消息传递
- 管道
- 消息队列



共享内存: 共享一块区域



- 小区门口的桌子上允许临时放置快递
- 快递员在桌上放置快递，小明从桌上取快递



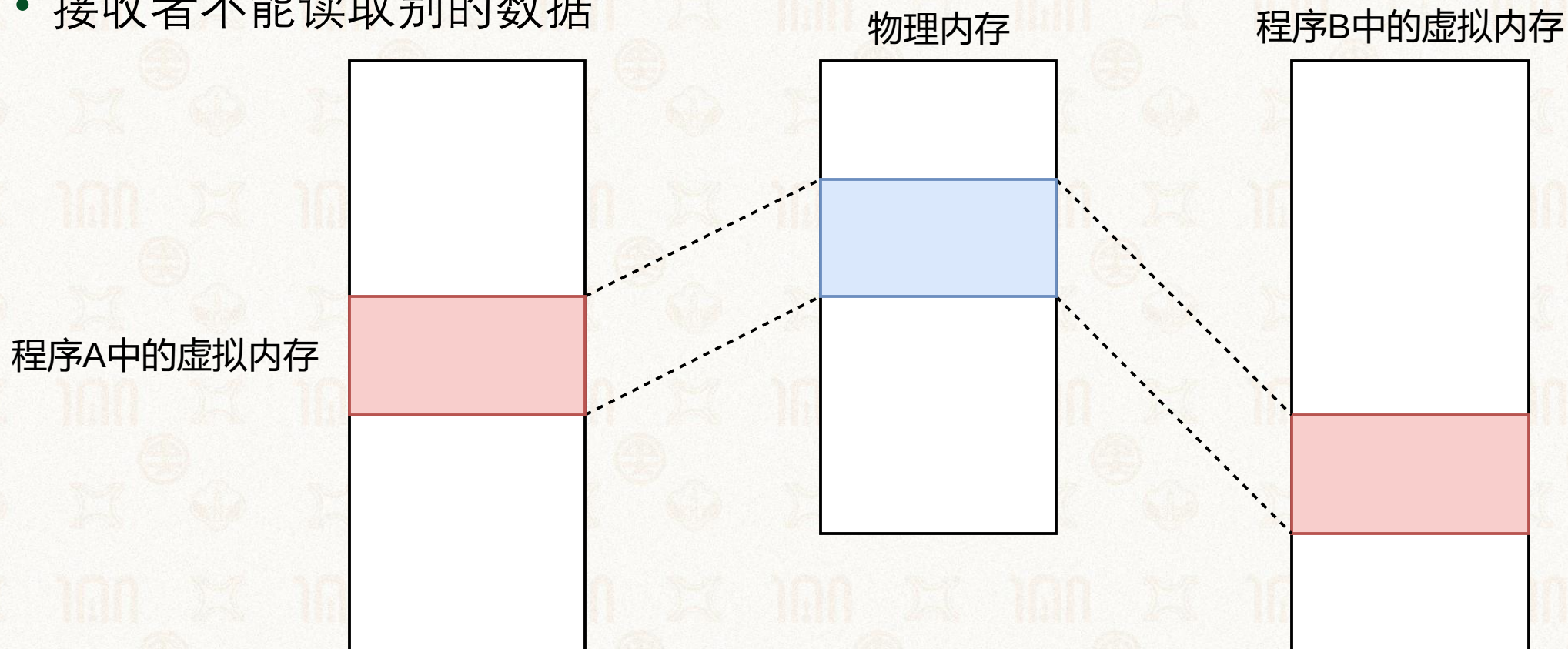
共享的快递桌



共享内存



- 系统内核为两个进程映射共同的内存区域
- 挑战：做好同步
 - 发送者不能覆盖掉未读取的数据
 - 接收者不能读取别的数据



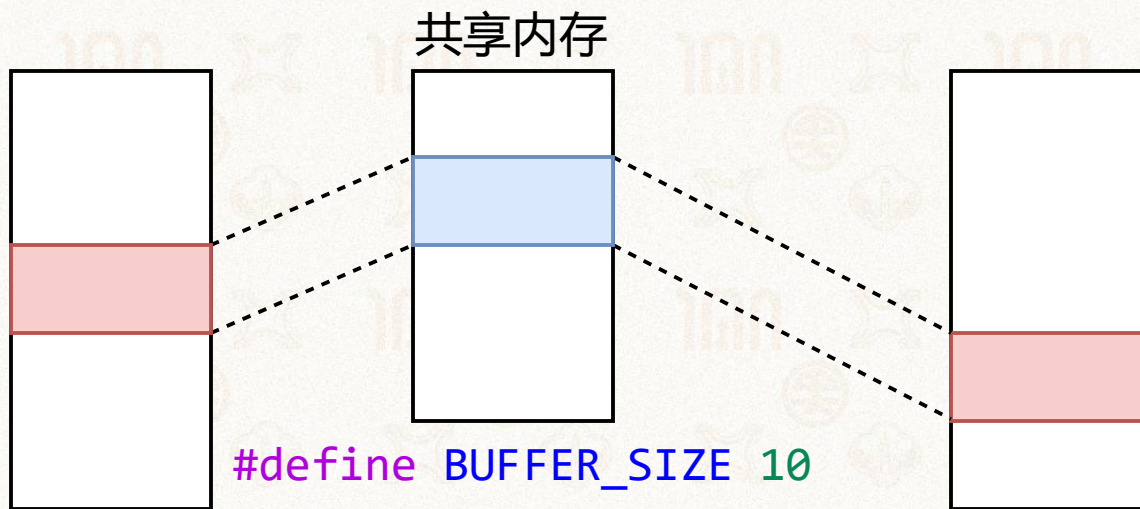


共享内存



1924-2024
中山大学 世纪华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

➤ 基础实现：共享区域



```
typedef struct {  
    // ...  
} item;
```

```
item buffer[BUFFER_SIZE];
```

← 共享数据区域，容量为10

```
int in = 0;  
int out = 0;
```

← 共享状态

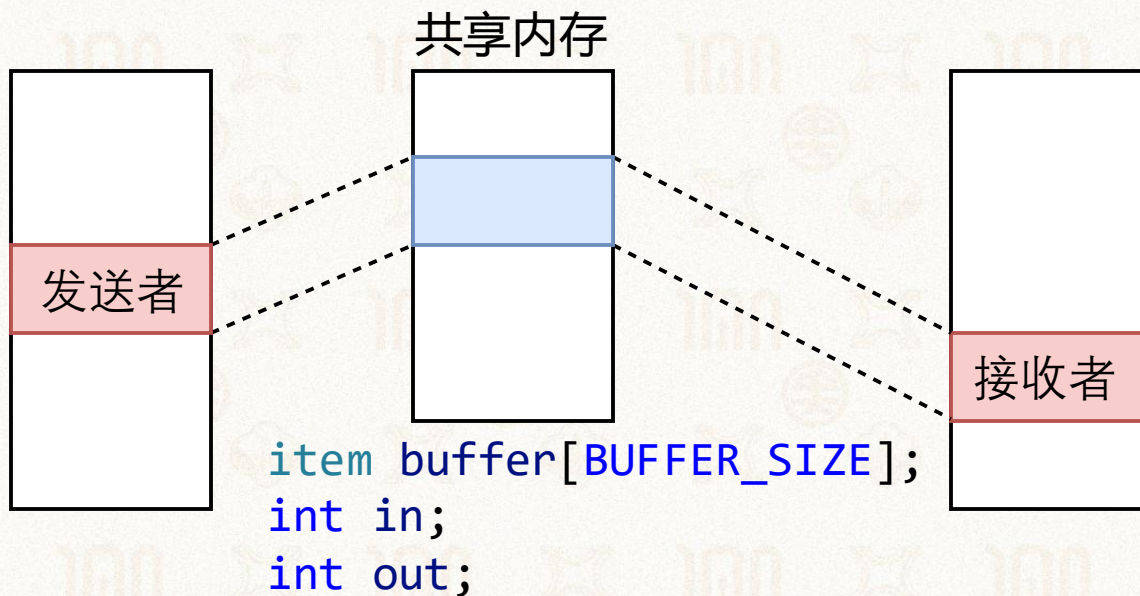


共享内存



1924-2024
中山大學 世紀华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

➤ 基础实现：发送者



```
while(new_package) {  
    // 产生一个item  
    while (((in + 1) % BUFFER_SIZE) == out)  
        ;// 什么也不干，干等着，因为没有多余的空间  
    buffer[in] = item;  
    in = (in + 1) % BUFFER_SIZE;  
}
```

当没有空间时，发送者盲目等待

发送者放置消息

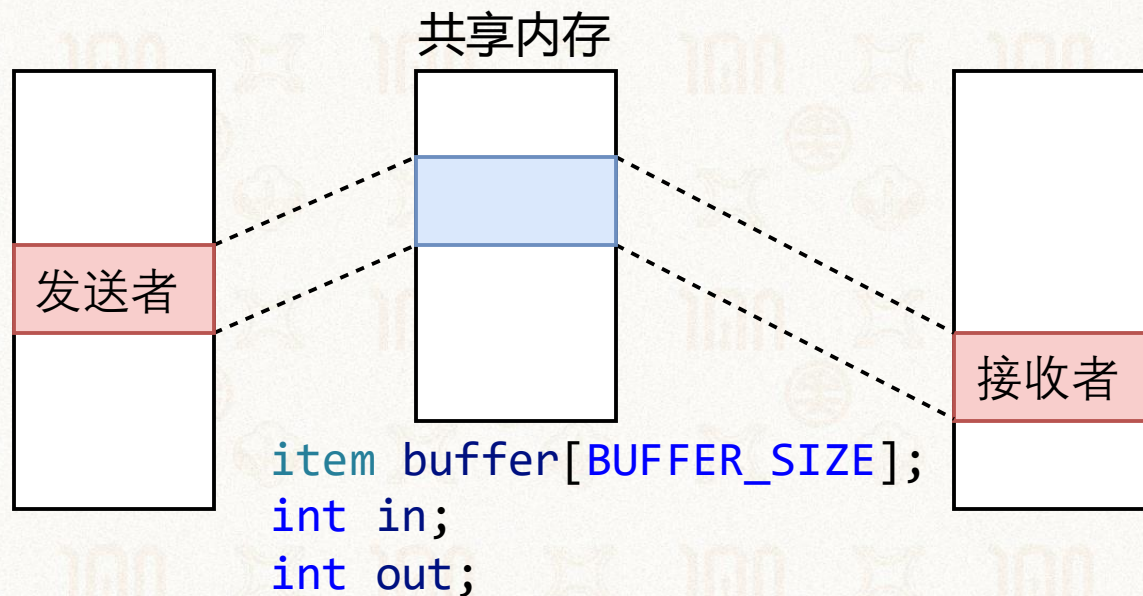


共享内存



1924-2024
中山大学 世纪华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

➤ 基础实现: 接收者



当没有新消息时, 接收者盲目等待

接收者获取消息

```
while (wait_package) {  
    while (in == out)  
        ; // 什么也不干, 没有新item可以被消费的  
        // 从buffer中删除一个item  
    item = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
    return item;  
}
```



共享内存的问题



- 轮询导致资源浪费
- 固定一个检查时间，时延长

```
while(new_package) {  
    // 产生一个item  
    while (((in + 1) % BUFFER_SIZE) == out)  
        ;// 什么也不干，干等着，因为没有多余的空间  
    buffer[in] = item;  
    in = (in + 1) % BUFFER_SIZE;  
}
```

```
while (wait_package) {  
    while (in == out)  
        ; // 什么也不干，没有新item可以被消费的  
    // 从buffer中删除一个item  
    item = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
    return item;  
}
```



大纲



1924-2024
中山大學 世纪华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

- 通信基础
- 共享内存通信
- 消息传递
- 管道
- 消息队列



消息传递



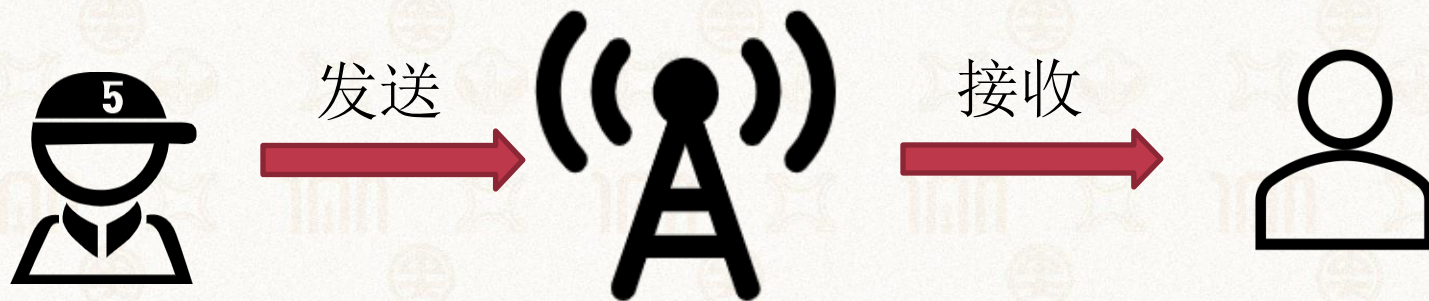
1924-2024
中山大學 世紀华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

➤ 消息系统

- 通过中间层(如内核)保证通信时延, 仍可以利用共享内存传递数据

➤ 好处:

- 低时延 (消息立即转发)
- 不浪费计算资源





消息传递



1924-2024
中山大學 世纪华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

➤ 基本操作:

- 发送消息Send(message)
- 接收消息Recv(message)

➤ 如果两个进程P和Q希望通过消息传递进行通信，需要:

- 建立一个通信连接
- 通过Send/Recv接口进行消息传递



直接通信



➤ 直接通信

- 进程拥有一个唯一标识
- Send(P, message): 给P进程发送一个消息
- Recv(Q, message): 从Q进程接收一个消息

➤ 直接通信下的连接

- 连接的建立是自动的 (通过标识)
- 一个连接唯一地对应一对进程
- 一对进程之间也只会存在一个连接
- 连接可以是单向的，但是在大部分情况下是双向的



直接通信



➤ 发送者

```
while (new_package) {  
    /* Produce an item */  
    while (((in + 1) % BUFFER_SIZE) == out)  
        ; /* nothing, no free buffers */  
  
    buffer[in] = item;  
    in = (in + 1) % BUFFER_SIZE;  
  
    Send(XiaoMing, "Package");  
}
```

➤ 接收者

```
while (wait package) {  
    Recv(Expressman, Msg);  
  
    item = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
  
    return item;  
}
```

Recv会阻塞，直到Send发送消息过来

快递员有好多苦恼：

快递员给小明送快递（特别是到付的快递），但：



小明经常不接听电话，怎么办？

作答



快递员有好多苦恼 (1)



- 快递员执行Send的时候，小明还没有Recv
- 快递员知道小明妈妈经常在家，希望建立一个聊天群，在群里发布快递信息
- 小明不接听时可以拜托妈妈下来拿快递



小明经常不接听电话，怎么办？

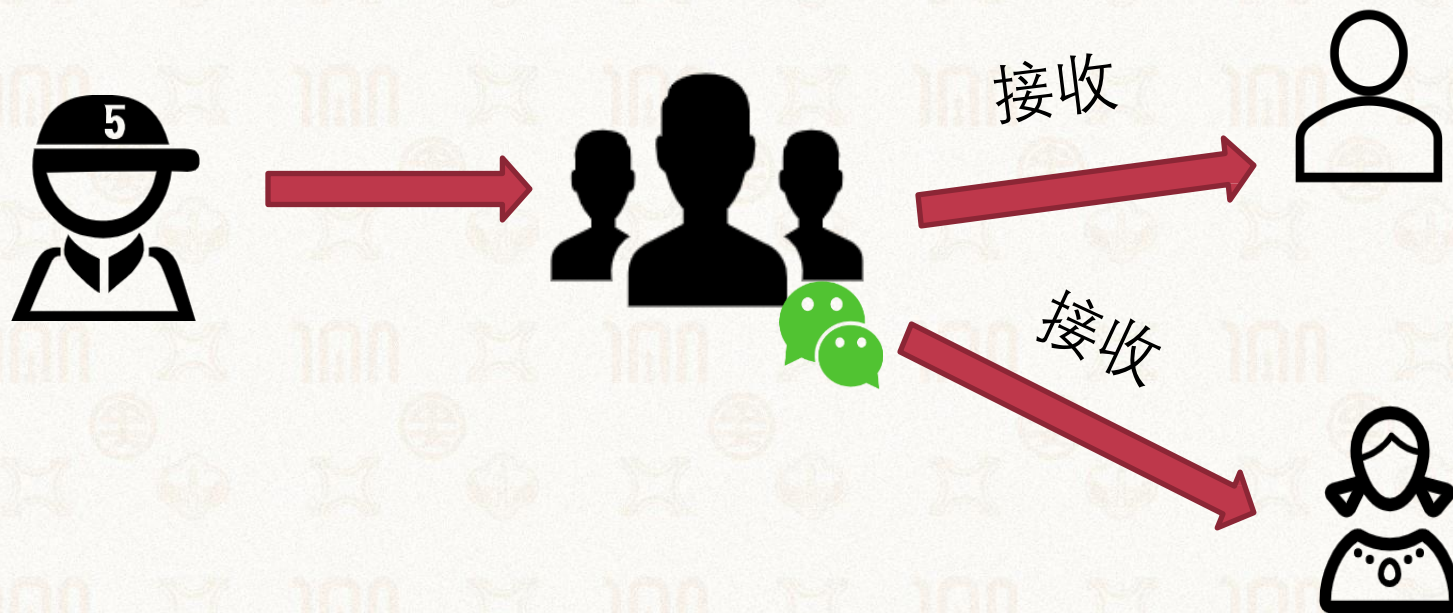


间接通信：用聊天群发布快递信息



➤ 消息的发送和接收需要经过一个“信箱”

- 聊天群 (所有在群内的人都可以接收消息)
- 每个“信箱”有自己唯一的标识符 (这里的群号)
- 发送者往“信箱”发送消息，接收者从“信箱”读取消息





间接通信



➤ 间接通信下的连接

- 进程间连接的建立发生在共享一个信箱时
- 每对进程可以有多个连接 (共享多个信箱)
- 连接同样可以是单向或双向的

➤ 间接进程间通信的操作

- 创建一个新的信箱
- 通过信箱发送和接收消息
- 销毁一个信箱

➤ 原语

- Send(M, message): 给信箱M发送一个消息
- Recv(M, message): 从信箱M接收一个消息



间接通信: 用聊天群发布快递信息



➤ 发送者(快递员)

```
while (new_package) {  
    /* Produce an item */  
    while (( (in + 1) % BUFFER_SIZE) == out)  
        ; /* nothing, no free buffers */  
  
    buffer[in] = item;  
    in = (in + 1) % BUFFER_SIZE;  
  
    Send(Mailbox, "Package");  
}
```

➤ 接收者(小明)

Recv(Mailbox, Msg);

➤ 接收者(小明妈妈)

Recv(Mailbox, Msg);

小明或者妈妈任何一个人只要上聊天群, 就能看到快递信息



快递员有好多苦恼 (2)



快递信息发布到群里，经常是小明和妈妈一起下来了。我都被投诉好几次了，这可怎么办好？

➤ 怎么解决“信箱”共享带来的多接收者的问题呢？



信箱共享的挑战



➤ 信箱的共享

- 进程P1、P2和P3共享一个信箱M
- P1负责发送消息， P2、P3负责接收消息
- 当一个消息发出的时候， 谁会接收到最新的消息呢？

➤ 可能的解决方案

- 让一个连接(信箱)只能被最多两个进程共享， 避免该问题
- 同一时间， 只允许最多一个进程在执行接收信息的操作
- 让消息系统任意选择一个接收者 (需要通知发送者谁是最终接收者)

快递员有好多苦恼：



小明和妈妈回复消息很慢，如果我发完消息等待回复可能要等很久；
如果我放下快递直接走的话事后又可能被投诉，怎么办呢？

作答



快递员有好多苦恼 (3)



小明和妈妈回复消息很慢，如果我发完消息等待回复可能要等很久；
如果我放下快递直接走的话事后又可能被投诉，怎么办呢？

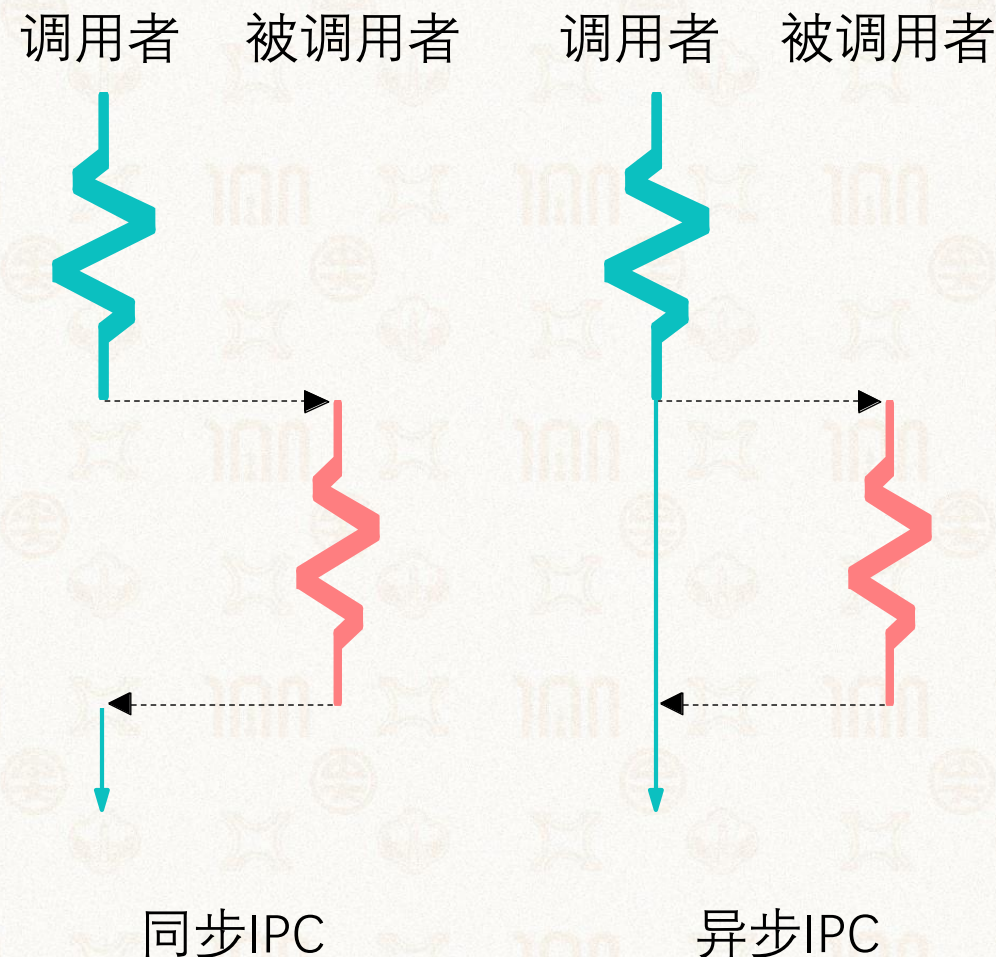
- 快递员想要弄清楚自己是等待消息被确认呢(阻塞)
- 还是发送完消息就赶去送下一个快递呢(非阻塞)?



消息传递的同步与异步



- 消息的传递可以是阻塞的，也可以是非阻塞的
- **阻塞**通常被认为是**同步**通信
 - 阻塞的发送/接收: 发送者/接收者一直处于阻塞状态，直到消息发出/到来
 - 同步通信通常有着更低时延和易用的编程模型 (不会被投诉)
- **非阻塞**通常被认为是**异步**通信
 - 发送者/接收者不等待操作结果，直接返回
 - 异步通信的**带宽**一般更高 (快递员可以送更多的快递)





超时机制



➤ 为了好评，快递员选择：

- 尽可能等待 (同步的通信)
- 但是一旦超过一个值 (如15分钟)，就先带走快递，等下再配送

➤ 超时机制的引入

- Send(A, message, **Time-out**)
 - 超过Time-out限定的时间就返回错误信息
- 两个特殊的超时选项：
 - 一直等待 (阻塞)
 - 不等待 (非阻塞)
- 避免由通信造成的拒接服务攻击等



同步通信和超时机制



➤ 发送者(快递员)

```
while (new_package) {  
    /* Produce an item */  
    while (( (in + 1) % BUFFER_SIZE) == out)  
        ; /* nothing, no free buffers */  
    buffer[in] = item;  
    in = (in + 1) % BUFFER_SIZE;  
  
    if(Send(Mailbox, "Package", "15min") == error)  
        goto retry;  
}
```

➤ 接收者(小明)

Recv(Mailbox, Msg, Time-out);

➤ 接收者(小明妈妈)

Recv(Mailbox, Msg, Time-out);



通信连接的缓冲: 快递桌的作用



➤ 缓冲

- 通信连接可以选择保留住还没有处理的消息

➤ 常见的三种设计

• 零容量

- 通信连接本身不缓冲消息，发送者只能阻塞等待接收者接收消息 (保安不提供快递桌)

• 有限容量

- 连接可以缓冲最多N个消息，当缓冲区满之后发送者只能阻塞等待 (快递只能放在桌上，但是空间有限)

• 无限容量

- 连接可以缓冲系统资源允许下的任意数量的消息，发送者几乎不需要等待 (快递可以放在门口任何位置)



大纲



1924-2024
中山大學 世纪华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

- 通信基础
- 共享内存通信
- 消息传递
- 管道
- 消息队列

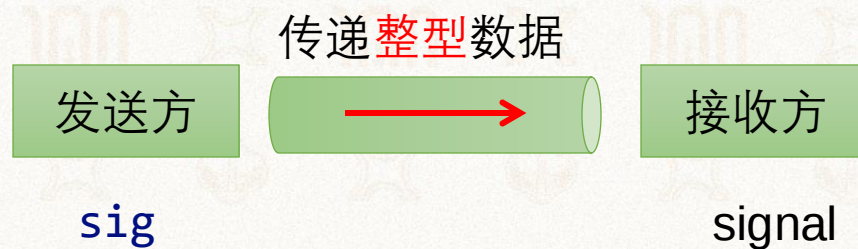


协程之间的通信：管道(channel)



```
package main
import (
    "fmt"
    "time"
)
// 这个函数接收的参数是一个int类型的管道
func longTask(signal chan int) {
    // 不带参数的 for
    // 相当于 while 循环
    for {
        fmt.Println("longTask is running")
        // 接收 signal 通道传值
        v := <-signal
        // 如果接收值为 1, 停止循环
        if v == 1 {
            break
        }
        time.Sleep(1 * time.Second)
    }
    fmt.Println("longTask is finished")
}
```

```
func main() {
    // 声明通道
    sig := make(chan int)
    // 异步调用 longTask
    go longTask(sig)
    // 等待 1 秒钟
    time.Sleep(3 * time.Second)
    // 向通道 sig 传值
    sig <- 1
    // 然后 longTask 会接收 sig 传值, 终止循环
    time.Sleep(1 * time.Second)
}
```





- 管道是Unix等宏内核系统中非常重要的进程间通信机制
- 管道(Pipe): 两个进程间的一根通信通道
 - 一端向里投递, 另一端接收
 - 管道是间接消息传递方式, 通过共享一个管道来建立连接
- 例子:
 - 常见的命令: `ls | grep`

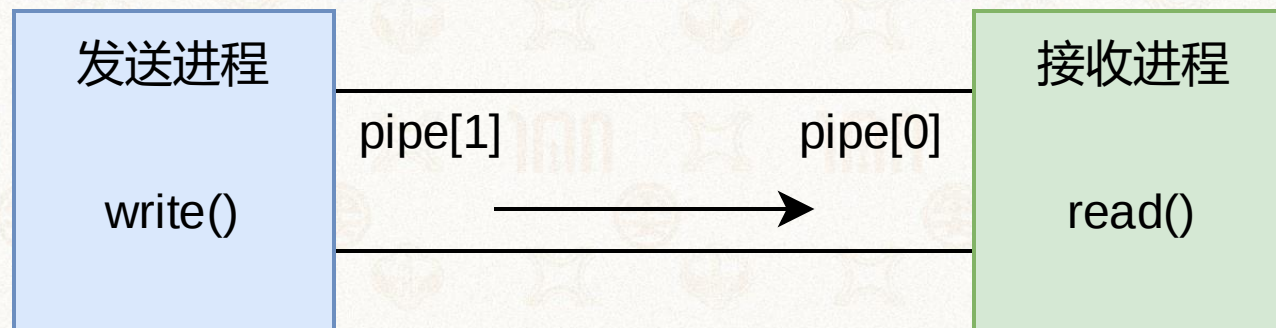
```
yxsu@Dell-T6401:~/os/process$ ls
a.out          fork_demo.c    hello-name.c    test.txt        waitpid_demo.c
execve_demo    fork_readfile  myecho          ucontext_demo.c
execve_demo.c  fork_readfile.c myecho.c        vfork_demo
fork_demo      hello-name     pthread_yield_demo.c vfork_demo.c
yxsu@Dell-T6401:~/os/process$ ls | grep fork
fork_demo
fork_demo.c
fork_readfile
fork_readfile.c
vfork_demo
vfork_demo.c
yxsu@Dell-T6401:~/os/process$
```



➤ 管道的特点:

- 单向通信，当缓冲区满时阻塞
- 一个管道有且只能有两个端口：
 - 一个负责输入 (发送数据)
 - 一个负责输出 (接收数据)
- 数据不带类型，即字节流
- 基于Unix的文件描述符使用

```
int fd[2];  
pipe(fd);  
fd[0]; // 读  
fd[1]; // 写
```





➤ Xv6 (Unix V6)为例: 管道数据结构

连接缓冲区域, 定长

```
struct pipe {  
    struct spinlock lock;  
    char data[PIPESIZE];
```

一个连接对应最多两个
进程

```
    uint nread; // 读取的字节数  
    uint nwrite; // 写入的字节数  
    int readopen; // 文件fd的读模式是开放状态  
    int writeopen; // 文件fd的写模式是开放状态  
};
```



➤ Xv6为例: 管道写操作

```
int pipewrite(struct pipe *p, char *addr, int n) {  
    int i;  
    acquire(&p->lock);  
    for(i = 0; i < n; i++) {  
        while(p->nwrite == p->nread + PIPESIZE) {  
            if(p->readopen == 0 || proc->killed) {  
                release(&p->lock);  
                return -1;  
            }  
            wakeup(&p->nread);  
            sleep(&p->nwrite, &p->lock);  
        }  
        p->data[p->nwrite++ % PIPESIZE] == addr[i];  
    }  
    wakeup(&p->nread);  
    release(&p->lock);  
    return n;  
}
```

检查缓冲区域是否满

如果读者不存在, 那么
返回错误信息

尝试唤醒读者去读消息
, 并将自己阻塞住

往缓冲区域上放围置消息



➤ Xv6为例: 管道读操作

检查是否有消息

阻塞等待消息到来

读取消息

```
int piperead(struct pipe *p, char *addr, int n) {
    int i;
    acquire(&p->lock);
    while(p->nread == p->nwrite && p->writeopen) {
        if(proc->killed) {
            release(&p->lock);
            return -1;
        }
        sleep(&p->nread, &p->lock);
    }
    for(i = 0; i < n; i++) {
        if(p->nread == p->nwrite) {
            break;
        }
        addr[i] = p->data[p->nread++ % PIPESIZE];
    }
    wakeup(&p->nwrite);
    release(&p->lock);
    return i;
}
```



管道的优点与问题



➤ 优点: 设计和实现简单

- 针对简单通信场景十分有效

➤ 问题:

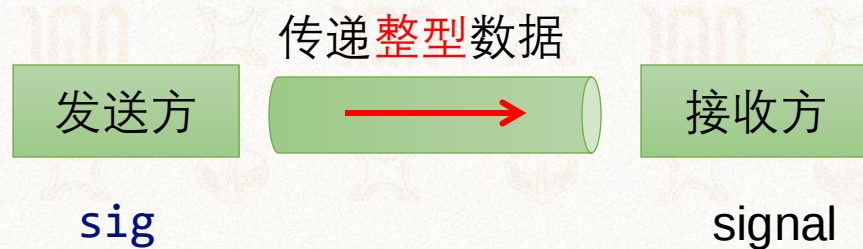
- 缺少消息的类型, 接收者需要对消息内容进行解析
- 缓冲区大小预先分配且固定
- 只能支持单向通信
- 只能支持最多两个进程间通信

主观题 10分 为什么管道不能支持双向通讯?

```
package main
import (
    "fmt"
    "time"
)
func longTask(signal chan int) {
    // 不带参数的 for
    // 相当于 while 循环
    for {
        fmt.Println("longTask is running")
        // 接收 signal 通道传值
        v := <-signal
        // 如果接收值为 1, 停止循环
        if v == 1 {
            break
        }
        time.Sleep(1 * time.Second)
    }
    fmt.Println("longTask is finihsed")
}
```

这个函数接收的参数是一个int类型的管道

```
func main() {
    // 声明通道
    sig := make(chan int)
    // 异步调用 longTask
    go longTask(sig)
    // 等待 1 秒钟
    time.Sleep(3 * time.Second)
    // 向通道 sig 传值
    sig <- 1
    // 然后 longTask 会接收 sig 传值, 终止循环
    time.Sleep(1 * time.Second)
}
```



作答



大纲



1924-2024
中山大學 世纪华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

- 通信基础
- 共享内存通信
- 消息传递
- 管道
- 消息队列

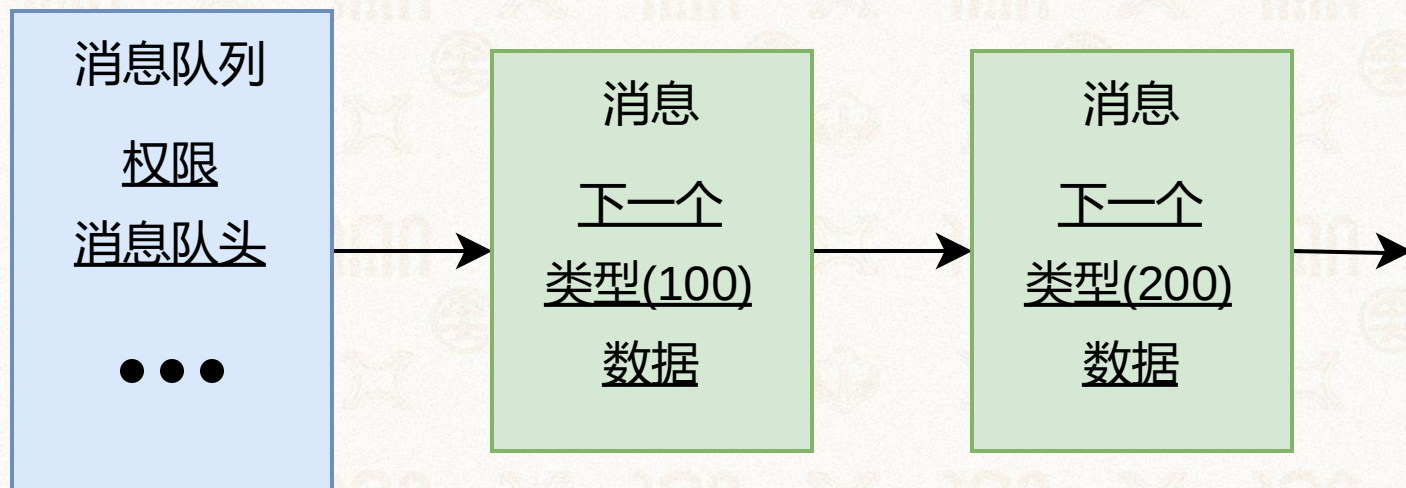


消息队列: 带类型的消息传递



- 消息队列: 以链表的方式组织消息
 - 任何有权限的进程都可以访问队列, 写入或者读取
 - 支持异步通信 (非阻塞)
- 消息的格式: 类型 + 数据
 - 类型: 由一个整型表示, 具体的意义由用户决定
- 消息队列是间接消息传递方式
 - 通过共享一个队列来建立连接

```
Ftok();  
Msgget();  
Msgsnd();  
Msgrcv();  
Msgctl();
```





消息队列：带类型的消息传递



➤ 消息队列的组织

- 基本遵循FIFO (First-In-First-Out)先进先出原则
- 消息队列的写入： 增加在队列尾部
- 消息队列的读取： 默认从队首获取消息

➤ 允许按照类型查询: Recv(A, type, message)

- 类型为0时返回第一个消息 (FIFO)
- 类型有值时按照类型查询消息
 - 如type为正数， 则返回第一个类型为type的消息



消息队列 VS. 管道



➤ 缓存区设计:

- 消息队列: 链表的组织方式, 动态分配资源, 可以设置很大的上限
- 管道: 固定的缓冲区间, 分配过大资源容易造成浪费

➤ 消息格式:

- 消息队列: 带类型的数据
- 管道: 数据 (字节流)

➤ 连接上的通信进程:

- 消息队列: 可以有多个发送者和接收者
- 管道: 两个端口, 最多对应两个进程

➤ 消息的管理:

- 消息队列: FIFO + 基于类型的查询
- 管道: FIFO



大纲



1924-2024
中山大學 世纪华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

- 通信基础
- 共享内存通信
- 消息传递
- 管道
- 消息队列



接下来。。。。



1924-2024
中山大學 世纪华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

- 进程间通信这一章相对不重要
- 接下来的同步原语，操作系统**最重要**的部分
 - 代码题基本都出自这两章



1924-2024
中山大學 世纪华诞
100th ANNIVERSARY
SUN YAT-SEN UNIVERSITY

1924-2024

谢谢

微信: suyuxin

钉钉: 苏玉鑫

B站: <https://space.bilibili.com/502854403>

软工集市课程专区: <https://ssemarket.cn/new/course>

匿名提问箱: <https://suask.me/ask-teacher/106/苏玉鑫>

