上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

SJTU

L2-2. 进程间通信

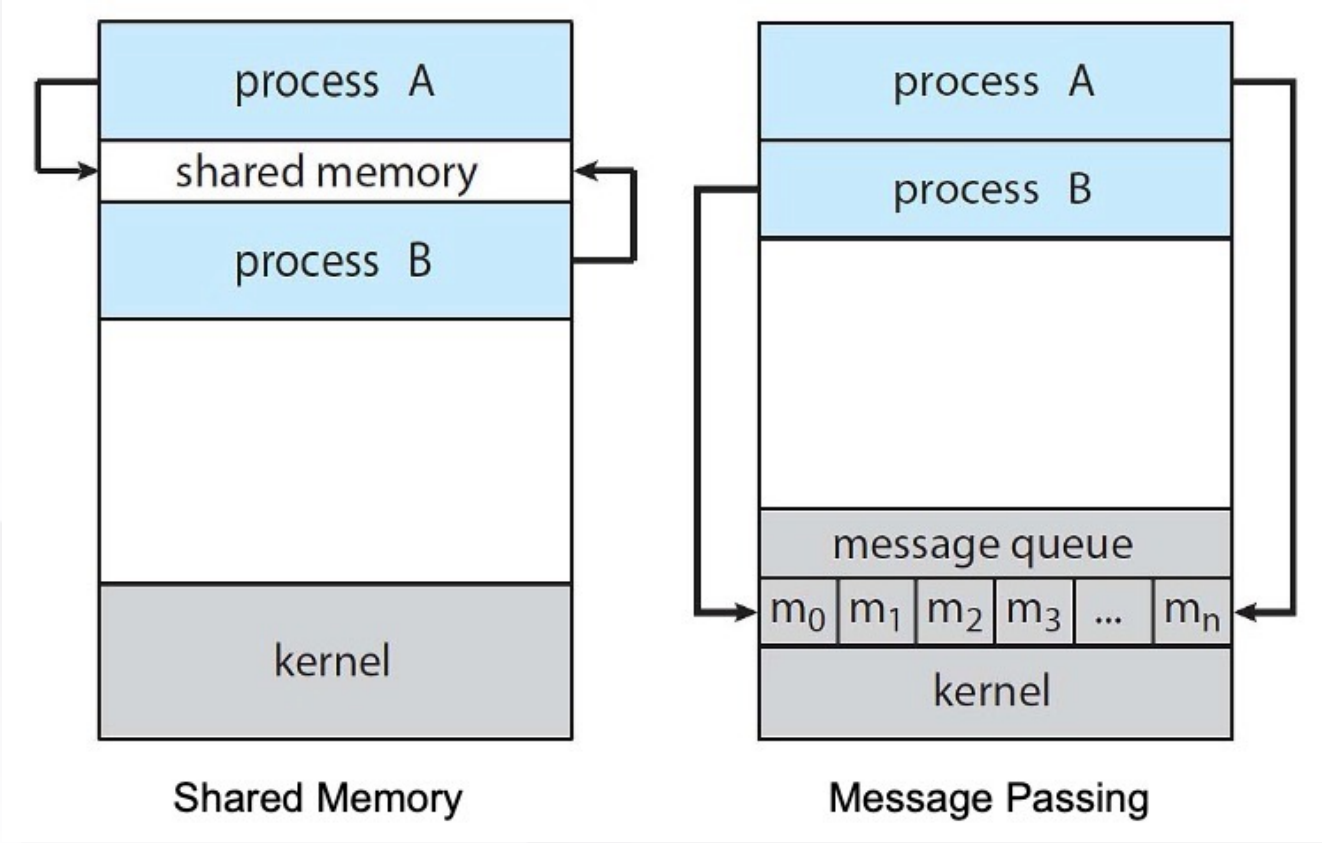
宋卓然

上海交通大学计算机系

songzhuoran@sjtu.edu.cn

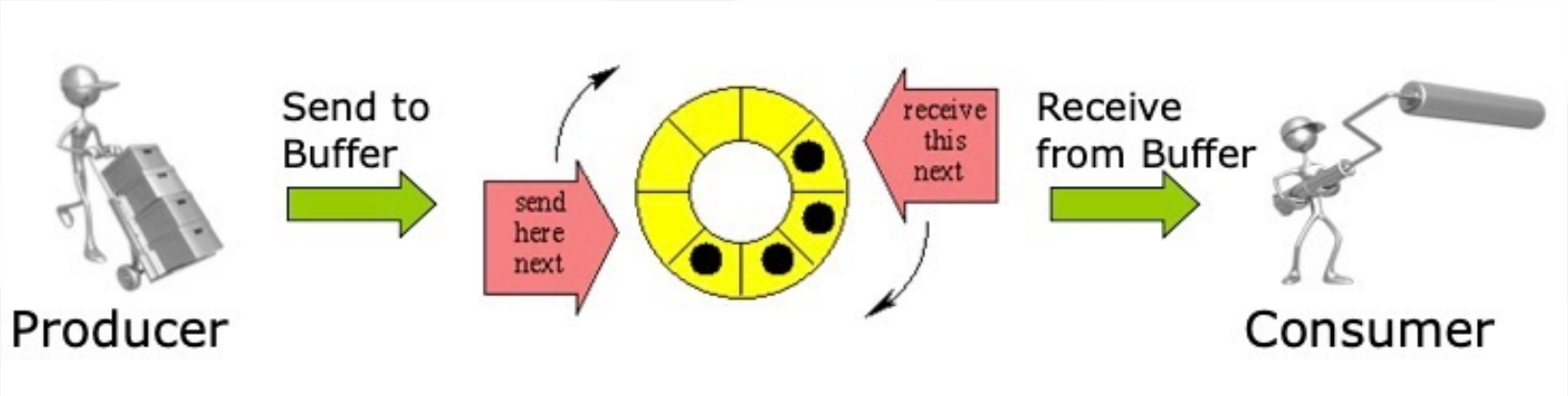
进程间通信

- 进程间可以独立也可以协作
- 需要让进程协作的原因：
  - 信息共享
  - 计算加速
  - 模块化
  - 方便
- 协作方式：
  - 共享内存
  - 消息传递



共享内存系统 生产者消费者问题

- 生产者生产信息，消费者消费信息
  - 有一个可用的缓冲区，以被生产者填充和被消费者清空
- 缓冲区
  - 无界缓冲区，不限制缓冲区大小
  - 有界缓冲区，假设固定大小的缓冲区



生产者消费者问题 共享内存

- 共享数据

```
#define BUFFER_SIZE 10
typedef struct {
    . . .
} item;

item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

生产者消费者问题 共享内存

Producer

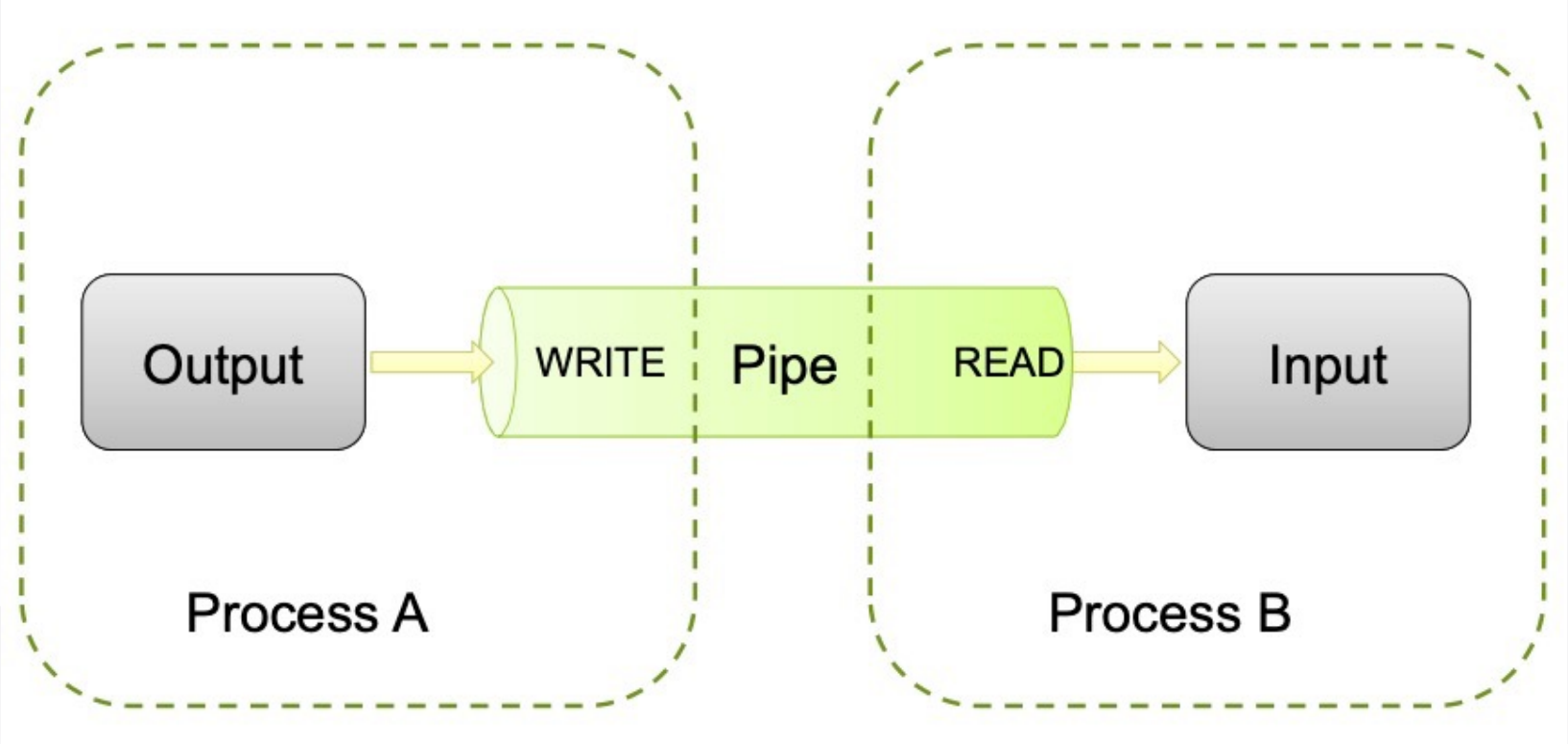
```
while (true) {
  /* Produce an item */
 while (((in + 1) % BUFFER_SIZE)  == out)
    ;   /* do nothing -- no free buffers */
   buffer[in] = item;
   in = (in + 1) % BUFFER_SIZE;
}
```

Consumer

```
while (true) {
  while (in == out)
        ; // do nothing
    // remove an item from the buffer
    item = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
      return item;
}
```

进程间通信-消息传递　管道

- 普通管道允许两个进程进行单向通信
  - 一端写
  - 一端读



进程间通信-消息传递　管道　实例

- First, create a pipe and check for errors

```
int mypipe[2];
if (pipe(mypipe)) {
    fprintf (stderr, "Pipe failed.\n");
    return -1;
}
```

mypipe[0]	read-end
mypipe[1]	write-end

- Second, fork your threads
- Third, close the pipes you don't need in that thread
  - reader should close(mypipe[1]);
  - writer should close(mypipe[0]);

进程间通信-消息传递　管道　实例

- Fourth, the writer should write the data to the pipe
  - write(mypipe[1],&c,1);
- Fifth, the reader reads from the data from the pipe:
  - while (read(mypipe[0],&c,1)>0) {

```
//do something, loop will exit when WRITER closes pipe
}
```

- Sixth, when writer is done with the pipe, close it
  - close(mypipe[1]); //EOF is sent to reader
- Seventh, when reader receives EOF from closed pipe, close the pipe and exit your polling loop
  - close(mypipe[0]); //all pipes should be closed now

进程间通信-消息传递

- 消息传递提供一种通信机制，以便允许进程不必通过共享地址空间来实现通信和同步
- 通过调用原语进行进程间通信
  - send(message)
  - receive(message)
- 如果进程P和Q需要通信， 那么它们必须互相发送消息和接收消息，它们之间要有通信链路

进程间通信-消息传递

- 有几个方法，用于逻辑实现链路和操作send()、receive()
  - 直接或间接的通信
  - 同步或异步的通信
  - 自动或显式的缓冲

进程间通信-消息传递

- 直接通信：需要通信的每个进程必须明确指定通信的接收者或发送者
  - send(P, message):向进程P发送message
  - receive(Q, message):从进程又接收message
- 属性
  - 在需要通信的每对进程之间，自动建立链路。进程仅需知道对方身份就可进行交流
  - 每个链路只与两个进程相关
  - 每对进程之间只有一个链路

进程间通信-消息传递

- 间接通信：通过邮箱或端口来发送和接收消息
  - send(A, message):向邮箱A发送message
  - receive(A, message):从邮箱A接收message
- 属性
  - 只有在两个进程共享一个邮箱时，才能建立通信链路
  - 一个链路可以与两个或更多进程相关联
  - 两个通信进程之间可有多个不同链路，每个链路对应于一个邮箱

进程间通信-消息传递

- 消息传递可以是阻塞或非阻塞的，也称为同步或异步的
  - 阻塞发送：发送进程阻塞，直到消息由接收进程或邮箱所接收
  - 非阻塞发送：发送进程发送消息，并且恢复操作
  - 阻塞接收：接收进程阻塞，直到有消息可用
  - 非阻塞接收：接收进程收到一个有效消息或空消息

进程间通信 实例 POSIX共享内存

```
#include <stdio.h>
#include <stlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
/* the size (in bytes) of shared memory object */
const int SIZE 4096;
/* name of the shared memory object */
const char *name = "OS";
/* strings written to shared memory */
const char *message_0 = "Hello";
const char *message_1 = "World!";

/* shared memory file descriptor */
int shm_fd;
/* pointer to shared memory obect */
void *ptr;
```

```
    /* create the shared memory object */
    shm_fd = shm_open(name, O_CREAT | O_RDRW, 0666);

    /* configure the size of the shared memory object */
    ftruncate(shm_fd, SIZE);

    /* memory map the shared memory object */
    ptr = mmap(0, SIZE, PROT_WRITE, MAP_SHARED, shm_fd, 0);

    /* write to the shared memory object */
    sprintf(ptr,"%s",message_0);
    ptr += strlen(message_0);
    sprintf(ptr,"%s",message_1);
    ptr += strlen(message_1);

    return 0;
}
```

采用POSIX共享内存API的生产者进程

进程间通信 实例 POSIX共享内存

```
#include <stdio.h>
#include <stlib.h>
#include <fcntl.h>
#include <sys/shm.h>
#include <sys/stat.h>

int main()
{
/* the size (in bytes) of shared memory object */
const int SIZE 4096;
/* name of the shared memory object */
const char *name = "OS";
/* shared memory file descriptor */
int shm_fd;
/* pointer to shared memory obect */
void *ptr;

    /* open the shared memory object */
    shm_fd = shm_open(name, O_RDONLY, 0666);

    /* memory map the shared memory object */
    ptr = mmap(0, SIZE, PROT_READ, MAP_SHARED, shm_fd, 0);

    /* read from the shared memory object */
    printf("%s",(char *)ptr);

    /* remove the shared memory object */
    shm_unlink(name);

    return 0;
}
```

采用POSIX共享内存API的消费者进程