OPERATING SYSTEMS

Interprocess Communication

Operating Systems

郑贵锋 博士
中山大学计算机学院
zhenggf@mail.sysu.edu.cn
https://gitee.com/code_sysu

中山大學 SUN YAT-SEN UNIVERSITY

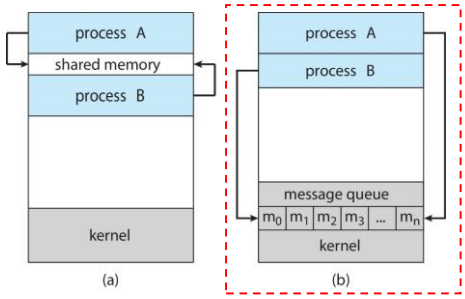
目录

Message-passing Systems

- **消息传递系统**
  - 消息传递是为协作进程提供的一种机制，用于相互通信并同步其操作，而无需求助于共享变量。
    - 通信通过协作进程之间交换的消息进行。
    - 用于交换少量数据
    - 通常使用系统调用实现，因此需要更耗时的内核干预任务
    - 在分布式系统中比共享内存更容易实现
    - 在分布式环境中特别有用，其中通信进程可能位于通过网络连接的不同计算机上。

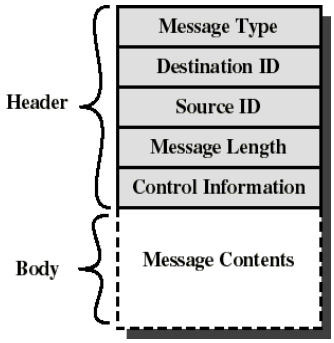
Message-passing Systems

- **消息传递系统**
  - 消息传递工具至少提供两个原语(*primitive*):
    - send(destination, message)或send(message)
    - receive(source, message)或receive(message)
  - 消息大小是固定的或可变的。
    - 可变大小的消息需要更复杂的系统级实现，但编程任务变得更简单。
      - 这是操作系统设计中常见的一种权衡。



Message-passing Systems

- **消息格式**
  - 标题和正文
  - 控制信息
    - 如果缓冲区空间不足怎么办
    - 序列号
    - 优先级
  - 排队规则：通常是FIFO，但也可以包括优先级。



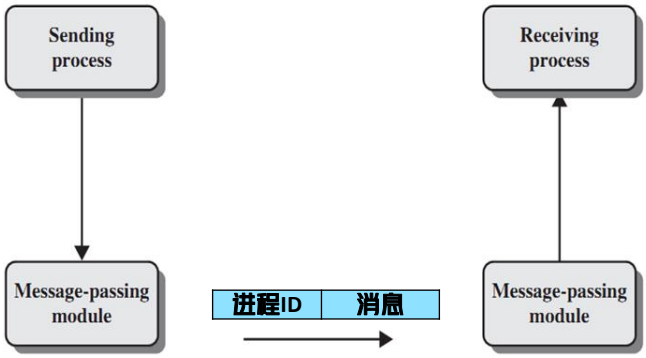
Message-passing Systems

- **消息格式**
  - Linux消息结构
    - 仅包含消息类型

```
struct msg_st {
    long int msg_type;
    char mtext[TEXT_SIZE];
};
```

Message-passing Systems

- **基本消息传递原语**



Message-passing Systems

- **逻辑通信链路**
  - 如果两个进程想要相互通信、发送消息和接收消息，那么它们之间必须存在通信链路。
  - 通信链路可以以多种方式实现。
    - 我们在这里关注的不是链路的物理实现，而是它的逻辑实现。
  - 逻辑实现链路和发送/接收操作的几种方法：
    - 直接或间接通信
    - 同步或异步通信
    - 自动或显式缓冲

Message-passing Systems

- **直接通信**
  - 对称：通信双方相互明确指定接收者或发送者
    send(destination_process_name, message)
    receive(source_process_name, message)
  - 或不对称：接收者不需要为发送者命名。变量id设置为任意接收到的消息的进程的名称。
    send(destination_process_name, message)
    receive(id, message)
  - 属性
    - 在需要通信的每对进程之间自动建立链路。进程只需知道彼此的身份即可进行通信。
    - 链路正好与两个进程关联。
    - 在每对进程之间，只存在一个链路。

Message-passing Systems

- **直接通信**
  - 直接通信的缺点
    - 直接通信使用特定的源/目标进程标识符。但可能提前指定来源是不可能的。
    - 对称和非对称方案中的另一个缺点是生成进程定义的有限模块化。
      - 更改进程标识符可能需要检查所有其他进程定义。
      - 必须找到对旧标识符的所有引用，以便将它们修改为新标识符。
      - 一般来说，这种必须明确指定标识符的硬编码技术，比间接通信技术要差。

Message-passing Systems

- **间接通信**
  - 消息通过邮箱或端口发送和接收。
  - 邮箱可以抽象地看作是一个对象
    - 进程可以将消息放入其中
    - 进程可以从中接收和删除消息。
  - 每个邮箱都有一个唯一的标识。
    - 例如，POSIX消息队列使用整数值来标识邮箱。一个进程可以通过多个不同的邮箱与另一个进程通信，但两个进程只有在具有共享邮箱ID时才能通信。

      send(mailbox_A, message)

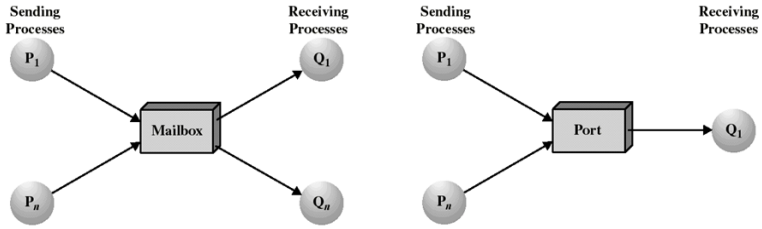
      receive(mailbox_A, message)
  - 属性
    - 只有当一对进程的两个成员都有一个共享邮箱时，才会在该对进程之间建立链路。
    - 一个链路可能与两个以上的进程相关联。
    - 在每对通信进程之间，可能存在多个不同的链路，每个链路对应一个邮箱。

Message-passing Systems

- **间接通信**
  - 假设进程P1、P2和P3都共享邮箱A。进程P1向A发送消息，而P2和P3都从A执行receive()（并删除接收到的消息）。哪个进程将接收P1发送的消息？答案取决于我们选择以下哪种方法：
    - 允许链路最多与两个进程关联。
    - 一次最多允许一个进程执行 receive() 操作。
    - 允许系统任意选择哪个进程将接收消息（即，P2或P3，但不是两者都将接收消息）。系统可以定义一个算法来选择哪个进程将接收消息。
      - 例如，*Round-Robin*，进程轮流接收消息。
    - 系统可以根据发送方识别对应的接收方
  - 邮箱可能由进程或操作系统拥有。必须考虑邮箱的所有权和接收权限。

Message-passing Systems

- **间接通信**
  - 邮箱 vs. 端口
    - 邮箱可以是一对发送方/接收方的专用邮箱。
    - 同一邮箱可以在多个发件人和收件人之间共享：
      - 然后，操作系统可能允许使用消息类型（用于选择）。
    - 端口是与一个接收方和多个发送方关联的邮箱。
      - 用于客户端/服务器应用程序-接收器是服务器



Message-passing Systems

- **间接通信**
  - 邮箱 vs. 端口
    - 邮箱和端口的所有权
      - OS代表进程创建邮箱（进程成为所有者）。
      - 应所有者的请求或所有者终止时，邮箱将被销毁。
      - 端口通常由接收进程创建和拥有。
      - 当接收器终止时，端口被销毁。

Message-passing Systems

- **消息传递系统中的同步**
  - 实现两个原语send()和receive()有不同的设计选项。
  - 消息传递可以是阻塞的，也可以是非阻塞的。阻塞被认为是同步的，而非阻塞被认为是异步的。
    - 阻塞发送
      - 发送进程阻塞，直到接收进程或邮箱接收到消息。
    - 阻塞接收
      - 接收器阻塞，直到有消息可用。
    - 非阻塞发送
      - 发送进程发送消息并继续。
    - 非阻塞接收
      - 接收方检索有效消息或空消息。

Message-passing Systems

- **消息传递系统中的同步**
  - 对于发送者
    - 更自然的是，在发出“send”后不被阻塞：
      - 发送者可以向多个目的地发送多条消息。
      - 发送者通常期望收到消息确认。
        - 如果接收者出现故障
  - 对于接收者
    - 发出接收请求后被阻塞更为自然。
      - 接收者通常在继续之前需要这些信息，但如果发送方进程在发送前失败，接收者可能会被无限期堵塞。

消息传递系统中的同步

- 这里有三种组合是有意义的：
  - 阻塞发送，阻塞接收
    - 在发送方和接收方之间有一个汇聚点rendezvous
  - 非阻塞发送，非阻塞接收
  - 非阻塞发送，阻塞接收
    - 最受欢迎
- 示例：阻塞发送，阻塞接收
  - 在收到消息之前，这两个进程都会被阻塞
  - 在通信链路未缓存（无消息队列）时发生
  - 提供紧密同步（汇聚点）。
- 示例：非阻塞发送、阻塞接收
  - 服务器进程向其他进程提供服务/资源。在继续之前，它将需要预期的信息。
- *后面我们将详细讨论同步问题*

缓存

- 无论通信是直接的还是间接的，由通信进程交换的消息都驻留在临时队列中。基本上，此类队列可以通过三种方式实现：
  - 零容量
    - 没有缓存的消息系统
    - 链路中不能有任何消息处于等待。
    - 发送者必须等待接收者收到消息（rendezvous）。
  - 有限容量
    - 队列的长度为有限的n。
    - 如果链路已满，发送者必须等待。
  - 无限容量
    - 队列的长度可能是无限的。
    - 发送者从不等待。

Examples

- **Linux：消息传递**
  - Linux IPCs限制
    - 内核级限制可以在/etc/sysctl.conf中重新定义

```
isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$ ipcs -l

------ Messages Limits --------
max queues system wide = 32000
max size of message (bytes) = 8192
default max size of queue (bytes) = 16384

------ Shared Memory Limits --------
max number of segments = 4096
max seg size (kbytes) = 18014398509465599
max total shared memory (kbytes) = 18014398509481980
min seg size (bytes) = 1

------ Semaphore Limits --------
max number of arrays = 32000
max semaphores per array = 32000
max semaphores system wide = 1024000000
max ops per semop call = 500
semaphore max value = 32767

isscgy@ubuntu:/mnt/hgfs/VM-Shared/OS-test$
```

Examples

- **Linux：消息传递**
  - 消息结构

```c
struct msg_st {
    long int msg_type;
    char mtext[TEXT_SIZE];
};

struct msqid_ds {
    struct ipc_perm msg_perm;
    time_t msg_stime;     /* 上次 msgsnd 时间 */
    time_t msg_rtime;     /* 上次 msgrcv 时间 */
    time_t msg_ctime;     /* 上次 change 时间 */
    msgqnum_t msg_qnum;   /* 队列中当前的消息数  */
    msglen_t msg_qbytes; /* 队列中允许的最大字节数 */
    pid_t msg_lspid;      /* 上次msgsnd的PID */
    pid_t msg_lrpid;      /* 上次msgrcv的PID */
};
```

Examples

- **Linux：消息传递**
  - 相关API

```
key_t ftok(char *pathname, char proj_id)

int msgget(key_t key, int msgflg)

int msgsnd(int msqid, struct msgbuf *msgp, int msgsz, int msgflg)

int msgrcv(int msqid, struct msgbuf *msgp, int msgsz, long msgtyp, int msgflg)

int msgctl(int msqid, int cmd, struct msqid_ds *buf)
```

Examples

- **Linux：消息传递**
  - 演示Linux消息传递API的发送和接收进程。
    - **算法** 9-0: msgdata.h

```
#define TEXT_SIZE 512
/* 考虑以下配置
------ Messages Limits --------
max queues system wide = 32000
max size of message (bytes) = 8192
default max size of queue (bytes) = 16384
------------------------------------------
消息大小设置为512，消息总数为16384/512=32
如果我们采用最大尺寸8192，则消息数为16384/8192=2（这是不合理的）
*/

/* 消息结构 */
struct msg_struct {
    long int msg_type;
    char mtext[TEXT_SIZE]; /* 二进制数据 */
};

#define PERM S_IRUSR|S_IWUSR|IPC_CREAT

#define ERR_EXIT(m) \
    do { \
        perror(m); \
        exit(EXIT_FAILURE); \
    } while(0)
```

Examples

- **Linux：消息传递**
  - 演示Linux消息传递API的发送和接收进程。
    - 算法 9-1: msgsnd.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/msg.h>
#include <sys/stat.h>
#include <fcntl.h>

#include “alg.9-0-msgdata.h”
```

```
int main(int argc, char *argv[])
{
    struct msg_struct data;

    long int msg_type;
    char buffer[TEXT_SIZE], pathname[80];
    int msqid, ret, count = 0;
    key_t key;
    FILE *fp;
    struct stat fileattr;

    if(argc < 2) {
        printf("Usage: ./a.out pathname\n");
        return EXIT_FAILURE;
    }
    strcpy(pathname, argv[1]);

    if(stat(pathname, &fileattr) == -1) {
        ret = creat(pathname, O_RDWR);
        if (ret == -1) {
            ERR_EXIT("creat()");
        }
        printf("shared file object created\n");
    }
```

Examples

- **Linux：消息传递**
  - 演示Linux消息传递API的发送和接收进程。
    - 算法 9-1: msgsnd.c (2)

```
    key = ftok(pathname, 0x27); /* project_id 可以是任意非零整数 */
    if(key < 0) {
        ERR_EXIT("ftok()");
    }

    printf("\nIPC key = 0x%x\n", key);

    msqid = msgget((key_t)key, 0666 | IPC_CREAT);
    if(msqid == -1) {
        ERR_EXIT("msgget()");
    }

    fp = fopen("./msgsnd.txt", "rb");
    if(!fp) {
        ERR_EXIT("source data file: ./msgsnd.txt fopen()");
    }

    struct msqid_ds msqattr;
    ret = msgctl(msqid, IPC_STAT, &msqattr);
    printf("number of messages remainded = %ld, empty slots = %ld\n", msqattr.msg_qnum,
16384/TEXT_SIZE-msqattr.msg_qnum);
    printf("Blocking Sending ... \n");
```

Examples

- **Linux：消息传递**
  - 演示Linux消息传递API的发送和接收进程。
    - 算法 9-1: msgsnd.c (3)

```
while (!feof(fp)) {
    ret = fscanf(fp, "%ld %s", &msg_type, buffer);
    if (ret == EOF) break;
    printf("%ld %s\n", msg_type, buffer);

    data.msg_type = msg_type;
    strcpy(data.mtext, buffer);

    ret = msgsnd(msqid, (void *)&data, TEXT_SIZE, 0);
        /* 0: 阻塞发送，当消息队列满时等待 */
    if(ret == -1) {
        ERR_EXIT("msgsnd()");
    }
    count++;
}

printf("number of sent messages = %d\n", count);

fclose(fp);
system("ipcs -q");
exit(EXIT_SUCCESS);
}
```

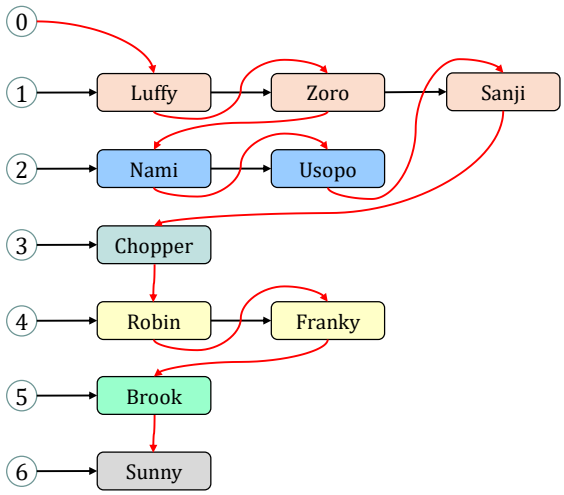
Examples


```
isscgy@ubuntu:/mnt/os-2020$ gcc alg.9-1-msgsnd.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out /home/isscgy/myshm

IPC key = 0x27011c6c
number of messages remainded = 0, empty slots = 32
Blocking Sending ...
1 Luffy
1 Zoro
2 Nami
2 Usopo
1 Sanji
3 Chopper
4 Robin
4 Franky
5 Brook
6 Sunny
number of sent messages = 10

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x27011c9e 1          isscgy     666        0            0
0x27011c6c 4          isscgy     666        5120         10

isscgy@ubuntu:/mnt/os-2020$
```

发送消息进程a.out：
发送了10条不同类型的消息

Examples

- **Linux：消息传递**
  - 演示Linux消息传递API的发送和接收进程
    - 消息队列



Examples

- **Linux：消息传递**
  - 演示Linux消息传递API的发送和接收进程
    - 算法 9-2: msgrcv.c (1)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/msg.h>
#include <sys/stat.h>

#include “alg.9-0-msgdata.h”
```

```c
    /* Usage: ./b.out pathname msg_type */
int main(int argc, char *argv[])
{
    key_t key;
    struct stat fileattr;
    char pathname[80];
    int msqid, ret, count = 0;
    struct msg_struct data;
    long int msgtype = 0;   /* 0 - 任意消息类型 */

    if(argc < 2) {
        printf("Usage: ./b.out pathname msg_type\n");
        return EXIT_FAILURE;
    }
    strcpy(pathname, argv[1]);
    if(stat(pathname, &fileattr) == -1) {
        ERR_EXIT("shared file object stat error");
    }
    if((key = ftok(pathname, 0x27)) < 0) {
        ERR_EXIT("ftok()");
    }
    printf("\nIPC key = 0x%x\n", key);
```

Examples

- **Linux：消息传递**
  - 演示Linux消息传递API的发送和接收进程
    - 算法 9-2: msgrcv.c (2)

```
msqid = msgget((key_t)key, 0666); /* 不新建消息队列 */
if(msqid == -1) {
    ERR_EXIT("msgget()");
}

if(argc < 3)
    msgtype = 0;
else {
    msgtype = atol(argv[2]);
    if (msgtype < 0)
        msgtype = 0;
}    /* 决定消息类型msgtype (分类号) */
printf("Selected message type = %ld\n", msgtype);

while (1) {
    ret = msgrcv(msqid, (void *)&data, TEXT_SIZE, msgtype, IPC_NOWAIT);
    /* 非阻塞接收 */
    if(ret == -1) { /* 此类消息结束 */
        printf("number of received messages = %d\n", count);
        break;
    }

    printf("%ld %s\n", data.msg_type, data.mtext);
    count++;
}
```

Examples

- **Linux：消息传递**
  - 演示Linux消息传递API的发送和接收进程
    - 算法 9-2: msgrcv.c (3)

```
struct msqid_ds msqattr;
ret = msgctl(msqid, IPC_STAT, &msqattr);
printf("number of messages remainding = %ld\n", msqattr.msg_qnum);

if(msqattr.msg_qnum == 0) {
    printf("do you want to delete this msg queue?(y/n)");
    if (getchar() == 'y') {
        if(msgctl(msqid, IPC_RMID, 0) == -1)
            perror("msgctl(IPC_RMID)");
    }
}

system("ipcs -q");
exit(EXIT_SUCCESS);
}
```

Examples

- **Linux：消息传递**
  - 演示Linux消息传递API的发送和接收进程
    - 算法 9-2: msgrcv.c (3)

```
isscgy@ubuntu:/mnt/os-2020$ gcc -o b.out alg.9-2-msgrcv.c
isscgy@ubuntu:/mnt/os-2020$ ./b.out /home/isscgy/mysh
shared file object stat error: No such file or directory
isscgy@ubuntu:/mnt/os-2020$ ./b.out /home/isscgy/myshm 2

IPC key = 0x27011c6c
Selected message type = 2
2 Nami
2 Usopo
number of received messages = 2
number of messages remainding = 8

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x27011c9e 1          isscgy     666        0            0
0x27011c6c 5          isscgy     666        4096         8

isscgy@ubuntu:/mnt/os-2020$
```

接收消息进程b.out：
接收了2条类型为2的消息

Examples

- **Linux：消息传递**

```
isscgy@ubuntu:/mnt/os-2020$ ./b.out /home/isscgy/myshm 0

IPC key = 0x27011c6c
Selected message type = 0
1 Luffy
1 Zoro
1 Sanji
3 Chopper
4 Robin
4 Franky
5 Brook
6 Sunny
number of received messages = 8
number of messages remainding = 0
do you want to delete this msg queue?(y/n)y

------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
0x27011c9e 1          isscgy     666        0            0

isscgy@ubuntu:/mnt/os-2020$
```

接收消息进程b.out：
接收了所有类型为0（任意类型）的消息；
最后删除了消息队列；

Examples

- **POSIX：消息传递**

```
#include <mqueue.h>
```

  - **打开、关闭和取消链路**

```
mqd_t mq_open(const char *name, int oflag, mode_t mode, struct mq_attr *attr );
/* return the mqdes, or -1 if failed */
```

mqd_t mqID;

mqID = mq_open("/anonymQueue", O_RDWR | O_CREAT, 0666, NULL);

```
mqd_t mq_close(mqd_t mqdes);
mqd_t mq_unlink(const char *name); /* return -1 if failed */
```

  - **收发**

```
mqd_t mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len, unsigned
msg_prio); /* return 0, or -1 if failed */
```

mq_send(mqID, msg, sizeof(msg), i)

```
mqd_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len, unsigned
*msg_prio); /* return the number of char received, or -1 if failed */
```

mq_attr mqAttr;

mq_getattr(mqID, &mqAttr);

mq_receive(mqID, buf, mqAttr.mq_msgsize, NULL)

Examples

- **POSIX：消息传递**
  - **请注意，Linux ipcs实用程序与POSIX ipcs实用程序不完全兼容。**
  - **POSIX中的消息队列、共享内存和信号量不能由SystemV bash命令查询，如**

```
$ipcs -q
```

Examples

- **Windows：消息传递**
  - Windows的消息传递工具：高级本地程序调用（LPC）
  - 仅在同一系统上的进程之间通信
  - 使用端口对象（如邮箱）建立和维护通信通道
  - 通信过程如下：
    1. 客户端打开子系统连接端口对象的句柄。
    2. 客户端发送一个连接请求。
    3. 服务器创建两个专用通信端口，并将其中一个端口的句柄返回给客户端。
    4. 客户端和服务器使用相应的端口句柄发送消息或回调，并侦听回复。

Examples

- **Windows：消息传递**
  - Windows中的本地程序调用