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

- IPC概述
- 共享内存系统
- 消息传递系统
- 管道Pipes
- 客户机-服务器系统中的通信
  - 套接字Sockets
  - 远程进程调用RPC

Overview

- **独立进程与合作进程**
  - 在操作系统中并发执行的进程可能是独立进程或合作进程
    - 如果某个进程不能影响或受系统中执行的其他进程的影响，则该进程是独立的。
      - 任何不与其他进程共享数据的进程都是独立的。
    - 如果一个进程可以影响或受到系统中执行的其他进程的影响，那么它就是合作的。
      - 任何与其他进程共享数据的进程都是合作进程。

Overview

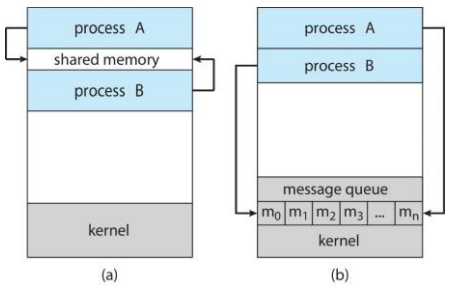
- **独立进程与合作进程**
  - 提供进程合作的原因：
    - 信息共享
      - 多个应用程序对信息的并发访问
    - 计算加速
      - 对于具有多个处理核心的计算机，将特定任务分解为子任务并并行执行可能会加快计算速度。
    - 模块化
      - 以模块化方式构建系统，将系统功能划分为单独的进程或线程
    - 方便
      - 即使是单个用户也可以同时处理多个任务。例如，用户可以同时进行编辑、听音乐和编译。
  - 我们将在后面详细讨论合作进程及其同步（Lecture15-18）

Overview

- **进程间通信**
  - 合作进程需要进程间通信（IPC）机制，允许它们交换数据和信息。
    - 如果两个进程P和Q希望通信，它们需要：
      - 在他们之间建立通信链路
      - 通过发送/接收交换消息
    - 通信链路的实现：
      - 物理链路（例如，共享内存、硬件总线）
      - 逻辑链路（例如，逻辑属性）
    - 实现的问题：
      - 如何建立链路？
      - 链路是否可以与两个以上的进程关联？
      - 每对通信进程之间可以有多少链路？
      - 链路的容量是多少？
      - 链接可以容纳的消息大小是固定的还是可变的？
      - 链接是单向的还是双向的？

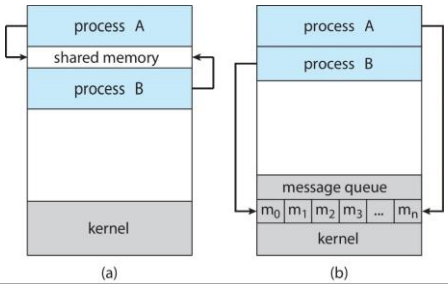
Overview

- **进程间通信**
  - IPC有两种基本模型
    - (a) *共享内存*/*内存共享*
      - 建立了一个由合作进程共享的内存区域。然后，进程可以通过向共享区域读写数据来交换信息。
      - 仅在建立共享内存区域时才需要系统调用。一旦建立了共享内存，所有访问都被视为例行内存访问，不需要内核的帮助。



Overview

- **进程间通信**
  - IPC有两种基本模型
    - (b) *消息传递Message Passing*
      - 通信通过合作进程之间交换的消息进行。
      - 用于交换少量数据
      - 在分布式系统中比共享内存更容易实现
      - 通常使用系统调用实现，因此需要更耗时的内核任务介入
      - 多核系统：具有更好性能，是IPC的首选机制



Shared-memory Systems

- **共享内存系统**
  - 通常，共享内存区域位于创建共享内存段的进程的地址空间中。
  - 希望使用此共享内存段进行通信的其他进程必须将其连接到其地址空间，然后通过读取和写入共享区域中的数据来交换信息。
  - 数据的位置和形式由这些进程确定，不受操作系统的控制。
  - 这些进程还负责确保它们不会同时写入同一位置。
    - 他们大多保持互斥 *mutual exclusion*（Lecture14-19）

Shared-memory Systems

- **共享内存的生产者-消费者问题**
  - 生产者-消费者问题是合作进程的常见范例。
    - 生产者进程生成由消费者进程使用的信息。
    - 这两个进程共享一个FIFO缓冲区，由生产者填充，消费者清空
  - 生产者和消费者同时运行，必须同步，以便消费者不会尝试消费尚未生产的信息块。
  - 可以使用两种类型的缓冲区。
    - 无限缓冲
      - 对缓冲区的大小没有实际限制
      - 如果缓冲区为空，则使用者必须等待；生产者总是可以生产新信息块
    - 有限缓冲
      - 具有固定的缓冲区大小
      - 如果缓冲区为空，则使用者必须等待；如果缓冲区已满，生产者必须等待。

Shared-memory Systems

- **共享内存的生产者-消费者问题**
  - 共享数据

```
#define BUFFER_SIZE 10

typedef struct {
    … …    /* item structure */
} item;

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

  - 共享缓冲区实现为带有两个逻辑指针的循环数组：in和out。
    - in指向缓冲区下一可用空间；out指向缓冲区的第一个可用项。
    - 当in等于out时，缓冲区为空

    - 当((in+1) % BUFFER_SIZE)等于out时，缓冲区已满
    - 此方案最多允许缓冲区中同时包含(BUFFER_SIZE-1)个项目
      - 为什么？

Shared-memory Systems

- **共享内存的生产者-消费者问题**
  - 生产者：

```
item next_produced;
while (true) {
    …… /* 生产一个项目保存在next_produced结构体 */
    while (((in + 1) % BUFFER_SIZE) == out)
        ; /* buffer满，等待 */
    buffer[in] = next_produced;
    in = (in + 1) % BUFFER_SIZE;
}
```

  - 消费者：

```
item next_consumed;
while (true) {
    while (in == out)
        ;     /* buffer空，等待 */
    next_consumed = buffer[out];
    out = (out + 1) % BUFFER_SIZE;
    …… /* 消费next_consumed项目 */
}
```

Examples

- **Linux：共享内存**
  - 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$
```

2^{64}

Examples

- **Linux：共享内存**
  - 生成密钥ID

```
#include <sys/shm.h>
key_t ftok(const char *pathname, int id);
/* key_t类型是int. ftok()将pathname和项目id转换为IPC密钥 */

key_t key = ftok("/home/myshm", 0x27);
if((key == -1) {
    perror("ftok()");
} else
    printf("key = 0x%x\n", key);
```

  - 内核中创造共享内存

```
int shmget(key_t key, int size, int shmflg);
/* shmget()分配共享内存段，大小size的上限是1.9G */

int shmid = shmget(IPC_PRIVATE, 4096, IPC_CREAT|IPC_EXCL|0660);
if(shmid == -1) {
    perror("shmget()");
}
```

Examples

- **Linux：共享内存**
  - Attach进程附上地址

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

void *shmptr = shmat(shmid, 0, 0);
    /* shmaddr=0:附上的地址由内核决定*/
if(shmptr == (void *)(-1))
    perror("shmat()");
```

  - Detach进程分离地址

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

if(shmdt(shmptr) == -1)
    perror("shmdt()");
```

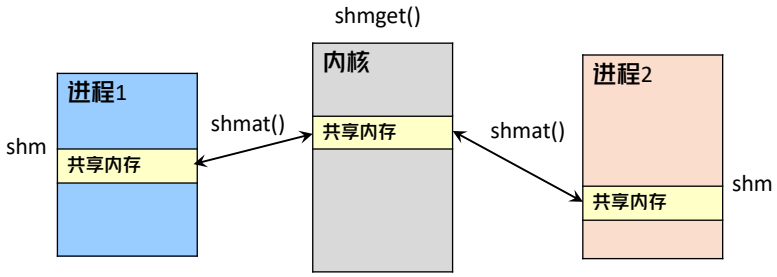
Examples

Linux：共享内存

- Release内核释放

```
int shmctl(int shmid,int cmd,struct shmid_ds *buf);

if (shmctl(shmid, IPC_RMID, 0) == -1)
    perror("shmctl()");
```



Examples

Linux：共享内存

- 演示例子：Single-writer-single-reader问题
  - **算法** 8-0: shmdata.h

```
#define TEXT_SIZE 4*1024  /* = PAGE_SIZE,每条消息的大小*/
#define TEXT_NUM 1      /* 最大消息数 */
/* 总大小不能超过当前shmmax，否则shmget时将产生‘invalid argument’错误 */

#define PERM S_IRUSR|S_IWUSR|IPC_CREAT

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

/* 演示结构，根据需要进行修改 */
struct shared_struct {
    int written; /* flag = 0: 缓冲区可写 ; others: 可读 */
    char mtext[TEXT_SIZE]; /* 用于消息读写的缓冲区 */
};
```

Examples

Linux：共享内存

- 演示例子：Single-writer-single-reader问题
  - 算法 8-1: shmcon.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/shm.h>
#include <fcntl.h>
#include “alg.8-0-shmdata.h"
```

```
int main(int argc, char *argv[])
{
    struct stat fileattr;
    key_t key; /* int类型 */
    int shmid; /* 共享内存的ID */
    void *shmptr;
    struct shared_struct *shared; /* 共享内存的结构 */
    pid_t childpid1, childpid2;
    char pathname[80], key_str[10], cmd_str[80];
    int shmsize, ret;

    shmsize = TEXT_NUM*sizeof(struct shared_struct);
    printf("max record number = %d, shm size = %d\n", TEXT_NUM, shmsize);

    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");
    }
```

读写控制程序shmcon
1. 创建共享内存

Examples

Linux：共享内存

- 演示例子：Single-writer-single-reader问题
  - 算法 8-1: shmcon.c (2)

```
key = ftok(pathname, 0x27); /* 0x27是项目ID，范围0x0001 - 0xffff，取其低8位 */
if(key == -1) {
    ERR_EXIT("shmcon: ftok()");
}
printf(“key generated: IPC key = 0x%x\n”, key); /* 如不用ftok()，需指定key>0 */

shmid = shmget((key_t)key, shmsize, 0666|PERM);
if(shmid == -1) {
    ERR_EXIT("shmcon: shmget()");
}
printf("shmcon: shmid = %d\n", shmid);

shmptr = shmat(shmid, 0, 0); /* 映射共享内存到虚拟地址，*shmaddr=0：内核决定 */

if(shmptr == (void *)-1) {
    ERR_EXIT("shmcon: shmat()");
}
printf("shmcon: shared Memory attached at %p\n", shmptr);

shared = (struct shared_struct *)shmptr;
shared->written = 0;

sprintf(cmd_str, "ipcs -m | grep '%d'\n", shmid);
printf("\n------ Shared Memory Segments ------\n");
system(cmd_str);
```

读写控制程序shmcon
1. 创建共享内存
2. 获得映射地址
3. 设置可写
4. 查询状态

Examples

Linux：共享内存

- 演示例子：Single-writer-single-reader问题
  - 算法 8-1: shmcon.c (3)

```
if(shmdt(shmptr) == -1) {
    ERR_EXIT("shmcon: shmdt()");
}

printf("\n------ Shared Memory Segments ------\n");
system(cmd_str);

sprintf(key_str, "%x", key);
char *argv1[] = {" ", key_str, 0};

childpid1 = vfork();
if(childpid1 < 0) {
    ERR_EXIT("shmcon: 1st vfork()");
}
else if(childpid1 == 0) {
    execv("./alg.8-2-shmread.o", argv1); /* 调用shmread, 参数为IPC key */
}
else {
    childpid2 = vfork();
    if(childpid2 < 0) {
        ERR_EXIT("shmcon: 2nd vfork()");
    }
    else if (childpid2 == 0) {
        execv("./alg.8-3-shmwrite.o", argv1); /* 调用shmwrite, 参数为IPC key */
    }
```

读写控制程序shmcon
1. 创建共享内存
2. 获得映射地址
3. 设置可写
4. 查询状态
5. fork读、写子进程

Examples

Linux：共享内存

- 演示例子：Single-writer-single-reader问题
  - 算法 8-1: shmcon.c (4)

```
    else {
        wait(&childpid1);
        wait(&childpid2);
        /* shmid可被任何已知IPC密钥的进程删除*/
        if (shmctl(shmid, IPC_RMID, 0) == -1) {
            ERR_EXIT("shmcon: shmctl(IPC_RMID)");
        }
        else {
            printf("shmcon: shmid = %d removed \n", shmid);
            printf("\n------ Shared Memory Segments ------\n");
            system(cmd_str);
            printf("nothing found ...\n");
            return EXIT_SUCCESS;
        }
    }
}
}
```

读写控制程序shmcon
1. 创建共享内存
2. 获得映射地址
3. 设置可写
4. 查询状态
5. fork读、写子进程
6. 等待子进程结束
7. 删除共享内存

Examples

Linux：共享内存

- 演示例子：Single-writer-single-reader问题
  - 算法 8-2: shmread.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>
#include <sys/shm.h>
#include "alg.8-0-shmdata.h"
```

```
int main(int argc, char *argv[])
{
    void *shmptr = NULL;
    struct shared_struct *shared;
    int shmid;
    key_t key;

    sscanf(argv[1], "%x", &key);
    printf("%*sshmread: IPC key = 0x%x\n", 30, " ", key);

    shmid = shmget((key_t)key, TEXT_NUM*sizeof(struct shared_struct), 0666|PERM);
    if (shmid == -1) {
        ERR_EXIT("shread: shmget()");
    }

    shmptr = shmat(shmid, 0, 0);
    if(shmptr == (void *)-1) {
        ERR_EXIT("shread: shmat()");
    }
    printf("%*sshmread: shmid = %d\n", 30, " ", shmid);
    printf("%*sshmread: shared memory attached at %p\n", 30, " ", shmptr);
    printf("%*sshmread process ready ...\n", 30, " ");

    shared = (struct shared_struct *)shmptr;
```

读程序shmread
1. 创建共享内存
2. 获得映射地址

Examples

Linux：共享内存

- 演示例子：Single-writer-single-reader问题
  - 算法 8-2: shmread.c (2)

```
    while (1) {
        while (shared->written == 0) {
            sleep(1); /* 没有消息，等待 ... */
        }
        printf("%*sYou wrote: %s\n", 30, " ", shared->mtext);
        shared->written = 0;
        if (strncmp(shared->mtext, "end", 3) == 0) {
            break; /* 收到end表示结束读写 */
        }
    } /* 使用shared->written实现进程同步是不可靠的 */

    if (shmdt(shmptr) == -1) {
        ERR_EXIT("shmread: shmdt()");
    }

    sleep(1);
    exit(EXIT_SUCCESS);
}
```

读程序shmread
1. 创建共享内存
2. 获得映射地址
3. 循环等待读消息
4. 读到end终止

Examples

Linux：共享内存

- 演示例子：Single-writer-single-reader问题
  - 算法 8-3: shmwrite.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <string.h>
#include <sys/shm.h>
#include "alg.8-0-shmdata.h"
```

```
int main(int argc, char *argv[])
{
    void *shmptr = NULL;
    struct shared_struct *shared = NULL;
    int shmid;
    key_t key;

    char buffer[BUFSIZ + 1]; /* 8192bytes, 用于保存stdin标准输入 */

    sscanf(argv[1], "%x", &key);
    printf("shmwrite: IPC key = 0x%x\n", key);

    shmid = shmget((key_t)key, TEXT_NUM*sizeof(struct shared_struct), 0666|PERM);
    if (shmid == -1) {
        ERR_EXIT("shmwite: shmget()");
    }

    shmptr = shmat(shmid, 0, 0);
    if(shmptr == (void *)-1) {
        ERR_EXIT("shmwrite: shmat()");
    }
    printf("shmwrite: shmid = %d\n", shmid);
    printf("shmwrite: shared memory attached at %p\n", shmptr);
    printf("shmwrite precess ready ...\n");

    shared = (struct shared_struct *)shmptr;
```

写程序shmwrite
1. 创建共享内存
2. 获得映射地址

Examples

Linux：共享内存

- 演示例子：Single-writer-single-reader问题
  - 算法 8-3: shmwrite.c (2)

```
    while (1) {
        while (shared->written == 1) {
            sleep(1); /* 消息还未读取，等待 ... */
        }

        printf("Enter some text: ");
        fgets(buffer, BUFSIZ, stdin);
        strncpy(shared->mtext, buffer, TEXT_SIZE);
        printf("shared buffer: %s\n",shared->mtext);
        shared->written = 1;  /* 消息准备好了 */

        if(strncmp(buffer, "end", 3) == 0) {
            break; /* 输入end表示结束读写 */
        }
    }
       /* detach the shared memory */
    if(shmdt(shmptr) == -1) {
        ERR_EXIT("shmwrite: shmdt()");
    }

    sleep(1);
    exit(EXIT_SUCCESS);
}
```

写程序shmwrite
1. 创建共享内存
2. 获得映射地址
3. 循环等待写消息
4. 写end以终止

```
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-3-shmwrite.o alg.8-3-shmwrite.c
isscgy@ubuntu:/mnt/os-2020$ gcc alg.8-1-shmcon.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out /home/isscgy/myshm
max record number = 1, shm size = 4100
key generated: IPC key = 0x27011c6c
shmcon: shmid = 43
shmcon: shared Memory attached at 0x7fbfc39fb000

------ Shared Memory Segments ------
0x00000000 8          isscgy     600        33554432   2          dest
0x27011c6c 43         isscgy     666        4100       1

------ Shared Memory Segments ------
0x00000000 8          isscgy     600        33554432   2          dest
0x27011c6c 43         isscgy     666        4100       0
                              shmread: IPC key = 0x27011c6c
                              shmread: shmid = 43
                              shmread: shared memory attached at 0x7f455fc52000
                              shmread process ready ...
shmwrite: IPC key = 0x27011c6c
shmwrite: shmid = 43
shmwrite: shared memory attached at 0x7f4c7a45b000
shmwrite process ready ...
Enter some text: Hello World!
shared buffer: Hello World!

                              You wrote: Hello World!

Enter some text: end
shared buffer: end

                              You wrote: end

shmcon: shmid = 43 removed

------ Shared Memory Segments ------
0x00000000 8          isscgy     600        33554432   2          dest
nothing found ...
isscgy@ubuntu:/mnt/os-2020$
```

```
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-3-shmwrite.o alg.8-3-shmwrite.c
isscgy@ubuntu:/mnt/os-2020$ gcc alg.8-1-shmcon.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out /home/isscgy/myshm
max record number = 1, shm size = 4100
key generated: IPC key = 0x27011c6c
shmcon: shmid = 43
shmcon: shared Memory attached at 0x7fbfc39fb000

------ Shared Memory Segments ------
0x00000000 8          isscgy     600        33554432   2          dest
0x27011c6c 43         isscgy     666        4100       1

------ Shared Memory Segments ------
0x00000000 8          isscgy     600        33554432   2          dest
0x27011c6c 43         isscgy     666        4100       0
                              shmread: IPC key = 0x27011c6c
                              shmread: shmid = 43
                              shmread: shared memory attached at 0x7f455fc52000
                              shmread process ready ...
shmwrite: IPC key = 0x27011c6c
shmwrite: shmid = 43
shmwrite: shared memory attached at 0x7f4c7a45b000
shmwrite process ready ...
Enter some text: Hello World!
shared buffer: Hello World!

                              You wrote: Hello World!

Enter some text: end
shared buffer: end

                              You wrote: end

shmcon: shmid = 43 removed

------ Shared Memory Segments ------
0x00000000 8          isscgy     600        33554432   2          dest
nothing found ...
isscgy@ubuntu:/mnt/os-2020$
```

Key的前8位是项目ID的低8位，
后面由文件路径信息决定。
一个文件可生成0xFF个Key

```
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-3-shmwrite.o alg.8-3-shmwrite.c
isscgy@ubuntu:/mnt/os-2020$ gcc alg.8-1-shmcon.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out /home/isscgy/myshm
max record number = 1, shm size = 4100
key generated: IPC key = 0x27011c6c
shmcon: shmid = 43
shmcon: shared Memory attached at 0x7fbfc39fb000

------ Shared Memory Segments ------
0x00000000 8          isscgy     600        33554432   2          dest
0x27011c6c 43         isscgy     666        4100       1

------ Shared Memory Segments ------
0x00000000 8          isscgy     600        33554432   2          dest
0x27011c6c 43         isscgy     666        4100       0
                              shmread: IPC key = 0x27011c6c
                              shmread: shmid = 43
                              shmread: shared memory attached at 0x7f455fc52000
                              shmread process ready ...
shmwrite: IPC key = 0x27011c6c
shmwrite: shmid = 43
shmwrite: shared memory attached at 0x7f4c7a45b000
shmwrite process ready ...
Enter some text: Hello World!
shared buffer: Hello World!

                              You wrote: Hello World!

Enter some text: end
shared buffer: end

                              You wrote: end

shmcon: shmid = 43 removed

------ Shared Memory Segments ------
0x00000000 8          isscgy     600        33554432   2          dest
nothing found ...
isscgy@ubuntu:/mnt/os-2020$
```

注意到内存地址均不相同

```
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-3-shmwrite.o alg.8-3-shmwrite.c
isscgy@ubuntu:/mnt/os-2020$ gcc alg.8-1-shmcon.c
isscgy@ubuntu:/mnt/os-2020$ ./a.out /home/isscgy/myshm
max record number = 1, shm size = 4100
key generated: IPC key = 0x27011c6c
shmcon: shmid = 43
shmcon: shared Memory attached at 0x7fbfc39fb000

------ Shared Memory Segments ------
0x00000000 8          isscgy     600        33554432   2          dest
0x27011c6c 43         isscgy     666        4100       1

------ Shared Memory Segments ------
0x00000000 8          isscgy     600        33554432   2          dest
0x27011c6c 43         isscgy     666        4100       0
                              shmread: IPC key = 0x27011c6c
                              shmread: shmid = 43
                              shmread: shared memory attached at 0x7f455fc52000
                              shmread process ready ...
shmwrite: IPC key = 0x27011c6c
shmwrite: shmid = 43
shmwrite: shared memory attached at 0x7f4c7a45b000
shmwrite process ready ...
Enter some text: Hello World!
shared buffer: Hello World!

                              You wrote: Hello World!

Enter some text: end
shared buffer: end

                              You wrote: end

shmcon: shmid = 43 removed

------ Shared Memory Segments ------
0x00000000 8          isscgy     600        33554432   2          dest
nothing found ...
isscgy@ubuntu:/mnt/os-2020$
```

Examples

- **POSIX共享内存**
  - POSIX系统有几种IPC机制，包括共享内存和消息传递。
  - POSIX共享内存使用内存映射文件组织，该文件将共享内存区域与/dev/shm/中的文件关联。
  - 对于内存共享，进程必须首先使用shm_open()系统调用创建共享内存对象：

```
int shm_open(const char *path, int flags, mode_t mode);
```

    - 实例

```
fd = shm_open(name, O_CREAT|O_RDWR, 0666);
```

    - *path:* 共享内存对象的名称。希望访问此共享内存的进程必须以此名称引用对象。
    - *flags:* 共享内存对象尚不存在则创建（O_CREAT）；该对象打开以进行读取和写入（O_RDWR）。
    - *mode:* 共享内存对象的文件访问权限。
    - 成功调用 shm_open() 将返回共享内存对象的整数文件描述符。

Examples

- **POSIX共享内存**
  - 建立对象后，可使用 ftruncate() 函数配置对象的大小（字节）。

```
int ftruncate(int fd, off_t length);
```

    - 例如，下面的调用将对象的大小设置为4096字节。

```
ftruncate(fd, 4096);
```

  - 最后， mmap() 函数的作用是建立一个包含共享内存对象的内存映射文件。它将返回内存映射文件的指针。

```
void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset);
```

  - 支持的OS：Linux 2.4及更高版本、FreeBSD、…
  - 编译命令

```
gcc -lrt filename.c
```

Examples

POSIX共享内存

- 演示POSIX共享内存API的生产者-消费者问题
  - 算法 8-4: shmpthreadcon.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <sys/mman.h>

#include "alg.8-0-shmdata.h"
```

```
/* gcc -lrt */
int main(int argc, char *argv[])
{
    char pathname[80], cmd_str[80];
    struct stat fileattr;
    int fd, shmsize, ret;
    pid_t childpid1, childpid2;

    if(argc < 2) {
        printf("Usage: ./a.out filename\n");
        return EXIT_FAILURE;
    }

    fd = shm_open(argv[1], O_CREAT|O_RDWR, 0666);
        /* /dev/shm/filename 可读可写共享对象，不存在时将创建 */
    if(fd == -1) {
        ERR_EXIT("con: shm_open()");
    }
    system("ls -l /dev/shm/");

    shmsize = TEXT_NUM*sizeof(struct shared_struct);
    ret = ftruncate(fd, shmsize); /* 设置大小 */
    if(ret == -1) {
        ERR_EXIT("con: ftruncate()");
    }
```

Examples

POSIX共享内存

- 演示POSIX共享内存API的生产者-消费者问题
  - 算法 8-4: shmpthreadcon.c (2)

```
    char *argv1[] = {" ", argv[1], 0};
    childpid1 = vfork();
    if(childpid1 < 0) {
        ERR_EXIT("shmpthreadcon: 1st vfork()");
    }
    else if(childpid1 == 0) {
        execv("./alg.8-5-shmproducer.o", argv1); /* 子进程调用生产者程序 */
    }
    else {
        childpid2 = vfork();
        if(childpid2 < 0)
            ERR_EXIT("shmpthreadcon: 2nd vfork()");
        else if (childpid2 == 0)
            execv("./alg.8-6-shmconsumer.o", argv1); /* 子进程调用消费者程序 */
        else {
            wait(&childpid1); /* 父进程等待子进程 */
            wait(&childpid2);
            ret = shm_unlink(argv[1]);
            if(ret == -1) {
                ERR_EXIT("con: shm_unlink()");
            } /* 只要知道文件名，共享对象可被任意进程删除 */
            system("ls -l /dev/shm/");
        }
    }
    exit(EXIT_SUCCESS);
}
```

Examples

POSIX共享内存

- 演示POSIX共享内存API的生产者-消费者问题
  - 算法 8-5: shmproducer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>

#include "alg.8-0-shmdata.h"
```

```
/* gcc -lrt */
int main(int argc, char *argv[])
{
    int fd, shmsize, ret;
    void *shmptr;
    const char *message_0 = "Hello World!";

    fd = shm_open(argv[1], O_RDWR, 0666); /* /dev/shm/filename 文件作为共享对象 */
    if(fd == -1) {
        ERR_EXIT("producer: shm_open()");
    }

    shmsize = TEXT_NUM*sizeof(struct shared_struct);
    shmptr = (char *)mmap(0, shmsize, PROT_READ|PROT_WRITE, MAP_SHARED, fd, 0);
    if(shmptr == (void *)-1) {
        ERR_EXIT("producer: mmap()");
    }

    sprintf(shmptr,"%s",message_0); /* 写消息Hello World! */
    printf("produced message: %s\n", (char *)shmptr);

    return EXIT_SUCCESS;
}
```

Examples

POSIX共享内存

- 演示POSIX共享内存API的生产者-消费者问题
  - 算法 8-6: shmconsumer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>

#include "alg.8-0-shmdata.h"
```

```
/* gcc -lrt */
int main(int argc, char *argv[])
{
    int fd, shmsize, ret;
    void *shmptr;

    fd = shm_open(argv[1], O_RDONLY, 0444);
    if(fd == -1) {
        ERR_EXIT("consumer: shm_open()");
    }

    shmsize = TEXT_NUM*sizeof(struct shared_struct);
    shmptr = (char *)mmap(0, shmsize, PROT_READ, MAP_SHARED, fd, 0);
    if(shmptr == (void *)-1) {
        ERR_EXIT("consumer: mmap()");
    }

    printf("consumed message: %s\n", (char *)shmptr); /* 读消息 */
    return EXIT_SUCCESS;
}
```

Examples

- **POSIX共享内存**
  - 演示POSIX共享内存API的生产者-消费者问题
    - **算法** 8-6: shmconsumer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/mman.h>
```

```
/* gcc -lrt */
int main(int argc, char *argv[])
{
```

```
isscgy@ubuntu:/mnt/os-2020$ gcc alg.8-4-shmpthreadcon.c -lrt
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-5-shmproducer.o alg.8-5-shmproducer.c -lrt
isscgy@ubuntu:/mnt/os-2020$ gcc -o alg.8-6-shmconsumer.o alg.8-6-shmconsumer.c -lrt
isscgy@ubuntu:/mnt/os-2020$ ./a.out myshm
total 0
-rw-r--r-- 1 isscgy isscgy 0 Mar 21 21:50 myshm
produced message: Hello World!
consumed message: Hello World!
total 0
isscgy@ubuntu:/mnt/os-2020$
```

生成myshm共享文件对象

```
        ERR_EXIT("mmap()");
    }

    printf("consumed message: %s\n", (char *)shmptr);
    return EXIT_SUCCESS;
}
```