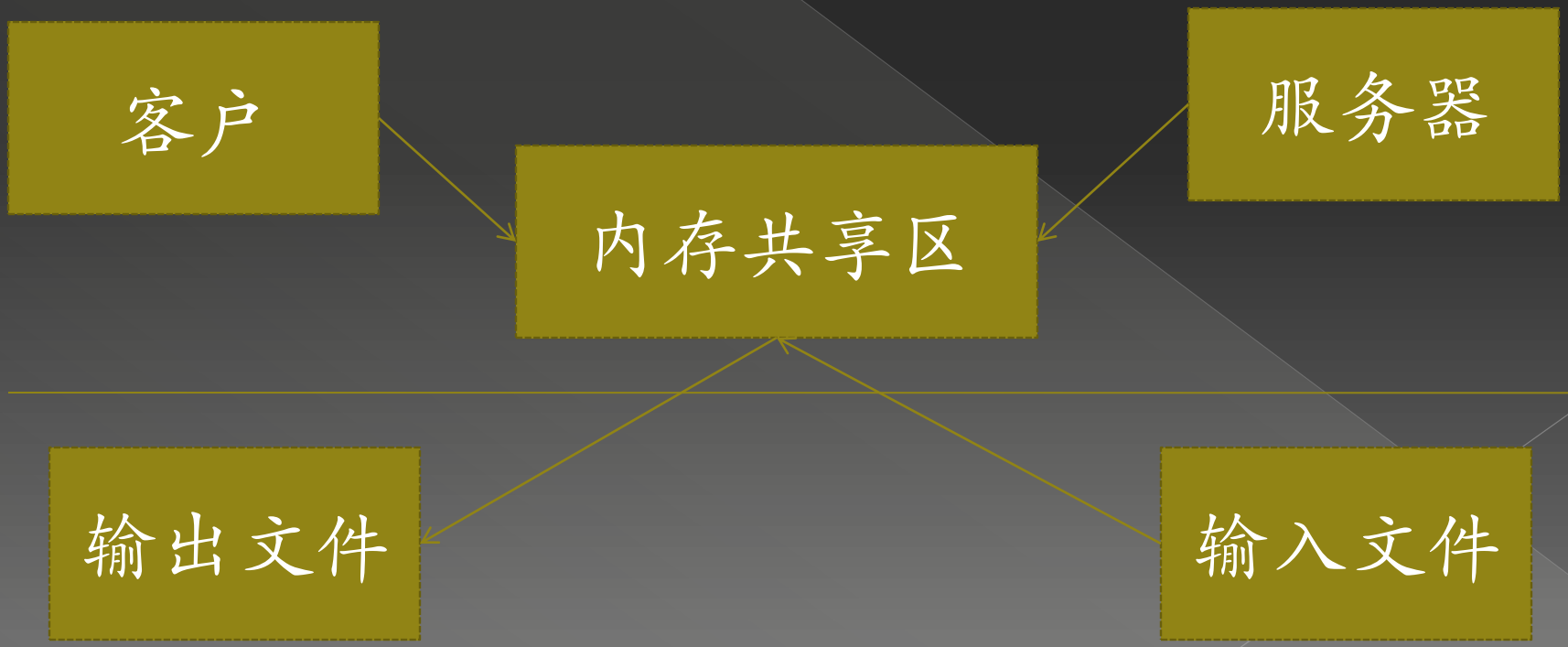
共享内存

武汉众嵌

共享内存

- 共享内容区是可用IPC形式中最快的方式。进程可以直接读写内存，而不需要任何数据的拷贝，进程间数据的传递就不再涉及内核。



共享内存

- Linux内核支持多种共享内存方式，如mmap()系统调用，SystemV 共享内存，PosiX共享内存。本节主要介绍mmap系统调用和SystemV共享内存API的原理即应用。

共享内存——内存映射

- mmap()系统调用使得进程通过映射一个普通文件实现共享内存。等待进程处理完后需要调用munmap函数来解除一个映射关系。如果要保存处理结果，需要调用msync使数据写到磁盘上
- 普通文件被映射到进程地址空间后，进程可以向访问普通内存一样对文件进行访问，不必再调用read()，write()等操作。
- 注意：mmap()实现共享内存也是其主要应用之一，它本身提供了不同于一般对普通文件的访问方式，进程可以像读写内存一样对普通文件的操作。

共享内存——内存映射

- Mmap函数:
  void* mmap (void* addr, size_t len, int prot , int flags, int fd, off_t offset)
- 返回值：成功返回映射到内存中的首地址，否则返回MAP_FAILED ;
- 参数addr指定文件应被映射到进程空间的起始地址。
- len是映射到调用进程地址空间的字节数。
- prot 参数指定共享内存的访问权限。可取如下几个值的或：PROT_READ（可读），PROT_WRITE （可写），PROT_EXEC（可执行），PROT_NONE（不可访问）。一般情况下设置为读写访问。
- flags由以下几个常值指定：MAP_SHARED , MAP_PRIVATE , MAP_FIXED，其中，MAP_SHARED , MAP_PRIVATE必选其一。
- fd为即将映射到进程空间的文件描述字，一般由open()返回，同时，fd可以指定为-1(须指定flags参数中的MAP_ANON)。
- offset参数一般设为0，表示从文件头开始映射。

共享内存——内存映射

- 解除映射关系函数munmap():
  int munmap(void * addr, size_t len)
- 该调用在进程地址空间中解除一个映射关系，addr是调用mmap()时返回的地址，len是映射区的大小。
- 课堂问题：当映射关系解除后，对原来映射地址的访问将导致什么结果?

- 实例一：两个进程通过映射普通文件实现共享内存通信
- //////test1.c
- typedef struct
- {
- 	char name[4];
- 	int age;
- }people;
- int main(int argc,char** argv){
- 	int fd,i;
- 	people* p_map ;
- 	char temp;
- 	fd = open(argv[1],O_CREAT|O_RDWR|O_TRUNC,0777);
- 	lseek(fd,sizeof(people)*10-1,SEEK_SET);
- 	write(fd,"",1);

- p_map =(people*)mmap(NULL,sizeof(people)*10,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);//MAP_PRIVATE
- if(p_map == (people*)-1)
- {
- perror("map err");
- }
- close(fd);
- printf("ptr:%x,_SC_PAGE_SIZE:%ld\n",p_map,_SC_PAGE_SIZE);

```
    temp = 'a';
    for(i=0;i<10;i++){
            temp += 1;
            memcpy((*(p_map+i)).name,&temp,2);
            (*(p_map+i)).age = 20+i;
            printf("%d\n",i);
            sleep(1);
    }
    printf("initalize over.\n");
    //sleep(10);
    munmap(p_map,sizeof(people)*10);
    printf("umap ok.\n");
}
```

```
//test2.c
typedef struct
{
	char name[4];
	int age;
}people;
int main(int argc,char** argv)
{
	int fd,i;
	people* p_map;
	if((fd = open(argv[1],O_RDWR)) < 0)
	{
		perror("open failed.");
		exit(1);
	}
```

```
    p_map = (people*)mmap(NULL,sizeof(people)*10,PROT_READ|PROT_WRITE,MAP_SHARED,fd,0);
    if(p_map == (people*)-1)
    {
        perror("mmap:");
        exit(1);
    }
    printf("ptr:%x\n",p_map);
    for(i=0;i<10;i++)
    {
        printf("name:%s age %d;\n",(*(p_map+i)).name,p_map[i].age);
    }
}
```

共享内存——内存映射

- 如果我们修改了处于内存映射到某个文件的内存区中的内容，那么内核将在稍后某个时刻相应的更新文件。然而有时候我们希望确信硬盘上的文件内存与内存映射区的内容一致怎么办?
- 同步函数msync():
  int msync (void * addr , size_t len, int flags);
- 其中addr和len参数通常指代内存映射区。
- Flags为执行同步的方式：MS_ASYNC(异步执行写)/MS_SYNC(同步执行写)/MS_INVALIDATE(使高速缓存的数据失效)。

- 实例二：父子进程通过匿名映射实现共享内存：
- typedef struct{
- 	char name[4];
- 	int age;
- }people;
- int main(int argc,char** argv){
- 	pid_t pid;
- 	int i;
- 	people* p_map;
- 	char temp;
- 	p_map = (people*)mmap(NULL,sizeof(people)*10,PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANONYMOUS,-1,0);
- 	if((pid=fork()) == 0){
- 		sleep(2);
- 		for(i=0;i<5;i++)printf("child read: the %d people's age is %d\n",i+1,p_map[i].age);
- 		(*p_map).age = 100;
- 		munmap(p_map,sizeof(people)*10);
- 		exit(0);

```
    }else if(pid > 0){
            temp = 'a';
            for(i=0;i<5;i++){
                    temp += 1;
                    memcpy(p_map[i].name,&temp,2);
                    p_map[i].age = 20+i;
            }
            sleep(5);
            printf("parent read: the first people's age is %d\n",p_map[0].age);
            munmap(p_map,sizeof(people)*10);
            printf("umap ok.\n");
    }
}
```

共享内存——system V

- 对于System V共享内存，主要有以下几个API：shmget()、shmat()、shmdt()以及shmctl();
- Shmget函数创建一个新的共享内存区，或者访问一个已经存在的共享内存区。
- 当打开一个共享内存区后，通过shmat函数把它附接到调用进程的地址空间。
- 当进程完成某个共享内存区的使用时，使用shmdt断开和共享内存区的关联。
- Shmctl函数提供了对一个共享内存区的多种操作。

共享内存

- 函数shmget:
  int shmget(key_t key,size_t size,int flags);
- 返回值：成功返回共享内存区标识符，失败返回-1;
- 参数key既可以是ftok的返回值，也可以是IPC_PRIVATE。
- 参数size是以字节为单位指定内存区大小。当函数用来创建一个新共享内存区时，必须指定一个不为0的size值，若打开已存在的共享内存区，size为0;
- Flag是共享内存区的权限值。它可以是IPC_CREAT或IPC_EXCL的组合。

共享内存

- 关联函数shmat：
  void* shmat(int shmid,const void* shmaddr,int flag)
- 返回：成功返回映射区的起始地址，出错返回-1；
- Shmid是由shmget返回的标识符。
- 如果shmaddr是一个空指针，那么系统替调用者选择地址。如果shmaddr是一个非空指针，则返回地址决定于

共享内存

- 断开共享内存区函数:
  int shmdt(const void* shmaddr)
- 当一个进程终止时，它当前附接着的所有共享内存区都自动断接。
- 注意：本函数并不删除所指定的共享内存区。

共享内存

- Shmctl提供了对一个共享内存的多种操作。
  Int shmctl(int shmid,int cmd,struct shmid_ds* buf);
- 该函数提供了三个命令:
  - IPC_RMID 从系统中删除由shmid标识的共享内存区。
  - IPC_SET 给所指出的共享内存区设置shmid_ds结构的以下三个成员:
    shm_perm.uid,shm_perm.gid,shm_perm.mode，它们的值来自buf参数中对应的成员。
  - IPC_STAT 通过buf参数向调用者返回指定共享内存当前的shmid_ds结构。

```
/////  test1.c
typedef struct{
        char name[4];
        int age;
} people;
main(int argc, char** argv)
{
        int shm_id,i;
        key_t key;
        char temp;
        people *p_map;
        key = ftok(argv[1],1);
        if(key==-1)
                perror("ftok error");
```

```
        shm_id=shmget(key,4096,IPC_CREAT|0666);
        if(shm_id==-1)
        {
                perror("shmget error");
                return;
        }
        p_map=(people*)shmat(shm_id,NULL,0);
        temp='a';
        for(i = 0;i<10;i++)
        {
                temp+=1;
                memcpy((*(p_map+i)).name,&temp,1);
                (*(p_map+i)).age=20+i;
        }
        if(shmdt(p_map)==-1)
                perror(" detach error ");
}
```

```
/////test2.c
typedef struct{
        char name[4];
        int age;
} people;
main(int argc, char** argv)
{
        int shm_id,i;
        key_t key;
        people *p_map;
        key = ftok(argv[1],1);
        if(key == -1)
                perror("ftok error");
        shm_id = shmget(key,4096,IPC_CREAT|0666);
```

```
    if(shm_id == -1)
    {
            perror("shmget error");
            return;
    }
    p_map = (people*)shmat(shm_id,NULL,0);
    for(i = 0;i<10;i++)
    {
            printf( "name:%s",(*(p_map+i)).name );
            printf( "\tage %d\n",(*(p_map+i)).age );
    }
    if(shmdt(p_map) == -1)
            perror(" detach error ");
}
```