# UC Day13

## 预习课

# 预习内容

## IPC对象

# IPC对象

# IPC对象

- 共享内存、消息队列和信号量，这三种IPC一般被合称为XSI IPC，它们之间有很多相似之处
- 为了实现进程之间的数据交换，系统内核会为参与通信的诸方维护一个内核对象(类似一个结构体变量)，记录和通信有关的各种配置参数和运行时信息，谓之IPC对象
- 系统中的每个IPC对象都有唯一的，非负整数形式的标识符，所有与IPC相关的操作，都需要提供IPC对象标识符

知识讲解

# IPC对象

知识讲解

- 与文件描述符不同，IPC对象标识符不是最小整数。当一个IPC对象被创建，以后又被销毁时，与该类型对象相关的标识符会持续加1，直至达到一个整型数的最大正值，然后又回转到0
- 标识符是IPC对象的内部名。为了使多个合作进程能够在同一个IPC对象上会合，需要提供一个外部名方案。为此使用了键，每个IPC对象都与一个键相关联，于是键就被作为该对象的外部名
- 无论何时，创建或者获取一个IPC对象都必须指定一个键。键的数据类型为key_t，在<sys/types.h>头文件中被定义为int。系统内核负责维护键与标识符的对应关系

# IPC对象

- #include <sys/ipc.h>
- key_t ftok (const char* pathname, int proj_id);
  - 功能：用于合成一个键
  - 参数：pathname 一个真实存在的路径名
    proj_id：项目ID，仅低8位有效，取0到255之间的数
  - 返回值：成功返回可用于创建或获取IPC对象的键，失败返回-1

知识讲解

# IPC对象



- ftok函数用pathname参数调用stat函数，将其输出的stat结构体中的st_dev(设备ID)和st_ino(i节点号)成员与proj_id参数组合来生成键
  - 参与生成键的是设备ID和i节点号，而不是pathname参数字符串本身。假设当前路径为/home/tarena/unixc，则ftok ("/home/tarena/unixc", 123); 与 ftok (".", 123);所返回的键是完全一样的
- 设备ID和i节点号都至少是整型字长的数据，而键也是整型字长，再加上一个字节项目ID，在合成键的过程中难免会丢失一部分信息。因此有时候明明提供的是不同的路径，该函数返回的键却是一样的

# 直播课见

# UC

## C/C++教学体系

# 目录

# 共享内存

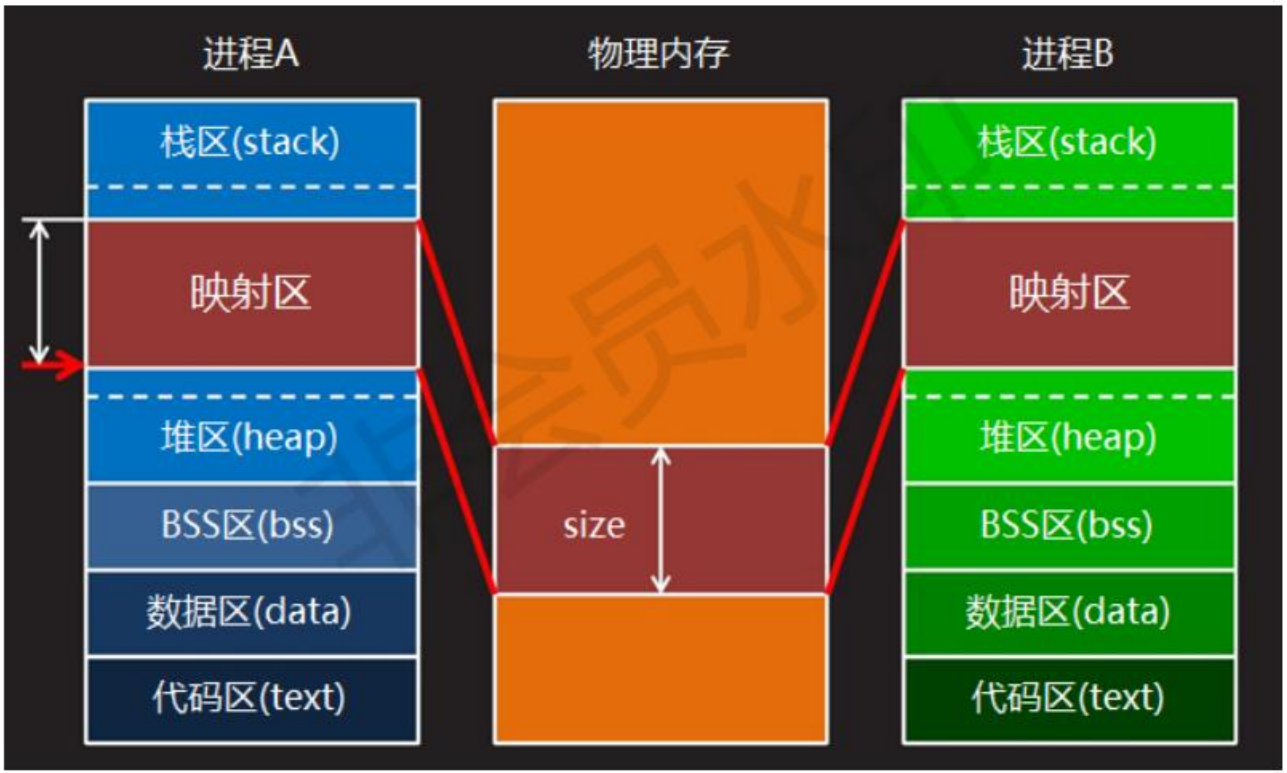
# 共享内存

知识讲解

- 两个或者更多进程，共享同一块由系统内核负责维护的内存区域，其地址空间通常被映射到堆和栈之间
- 多个进程通过共享内存通信，所传输的数据通过各个进程的虚拟内存被直接反映到同一块物理内存中，这就避免了在不同进程和系统内核之间来回复制数据的开销。因此基于共享内存的进程间通信，是速度最快的进程间通信方式。
- 共享内存本身缺乏足够的同步机制，这就需要程序员编写额外的代码来实现

# 共享内存

知识讲解


进程A
物理内存
进程B
栈区(stack)
映射区
堆区(heap)
BSS区(bss)
数据区(data)
代码区(text)
size
栈区(stack)
映射区
堆区(heap)
BSS区(bss)
数据区(data)
代码区(text)

# 共享内存

知识讲解

- #include <sys/shm.h>
- int shmget(key_t key, size_t size, int shmflg);
  - 功能：创建新的或获取已有的共享内存
  - 参数：key：键。
    size：字节数，自动按页取整。
    shmflg：创建标志，可取以下值：
    0 - 获取，不存在即失败。
    IPC_CREAT - 创建，不存在即创建，已存在即获取。
    IPC_EXCL - 排它，不存在即创建，已存在即失败。
    通过位或组合读写权限。
  - 返回值：成功返回共享内存的ID，失败返回-1。

# 共享内存

知识讲解

- #include <sys/shm.h>
- void* shmat(int shmid, void const* shmaddr, int shmflg);
  - 功能：加载共享内存，将物理内存中的共享区域映射到进程用户空间的虚拟内存中。
  - 参数：shmid：共享内存的ID。
    shmaddr：映射到共享内存的虚拟内存起始地址，取NULL，由系统自动选择。
    shmflg：加载标志，可取以下值：
    0 - 以读写方式使用共享。
    SHM_RDONLY - 以只读方式使用共享内存。
  - 返回值：成功返回共享内存的起始地址，失败返回-1。

# 共享内存

知识讲解

- shmat函数负责将给定共享内存映射到调用进程的虚拟内存空间，返回映射区的起始地址，同时将系统内核中共享内存对象的加载计数加一

- 调用进程在获得shmat函数返回的共享内存起始地址以后，就可以象访问普通内存一样访问共享内存中数据。

# 共享内存

知识讲解

- #include <sys/shm.h>
- int shmdt(void const* shmaddr);
  - 功能：卸载共享内存
  - 参数： shmaddr：共享内存的起始地址
  - 返回值：成功返回0，失败返回-1。

  - shmdt函数负责从调用进程的虚拟内存中结束shmaddr所指向的映射区到共享内存的映射，同时将系统内核中共享内存的加载计数减1。

# 共享内存

知识讲解

- #include <sys/shm.h>
- int shmctl(int shmid, IPC_RMID, NULL);
  - 功能：销毁共享内存
  - 参数： shmid：共享内存对象ID
  - 返回值：成功返回0，失败返回-1。

  - 销毁共享内存。其实并非真的销毁，而只是做一个销毁标记，禁止任何进程对该共享内存形成新的加载，但已有的加载依然保留。只有当其使用者们纷纷卸载，直至其加载计数降为0时，共享内存才会真的被销毁

# 共享内存

- 基于共享内存实现进程间通信的编程模型

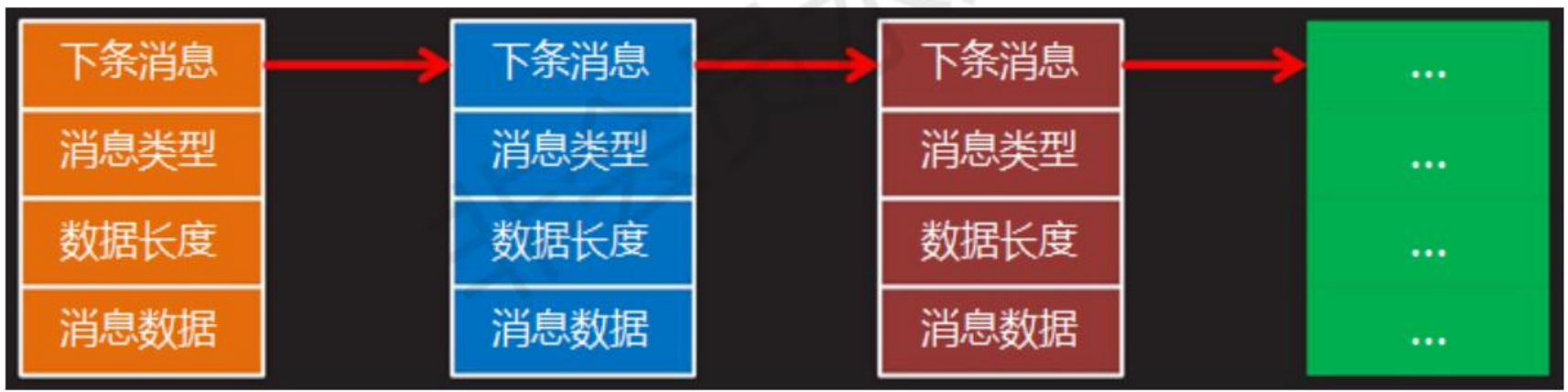
知识讲解

| 步骤 | 进程A | 函数 | 进程B | 步骤 |
|---|---|---|---|---|
| 1 | 创建共享内存 | shmget | 获取共享内存 | 1 |
| 2 | 加载共享内存 | shmat | 加载共享内存 | 2 |
| 3 | 使用共享内存 | strcpy<br>printf<br>... | 使用共享内存 | 3 |
| 4 | 卸载共享内存 | shmdt | 卸载共享内存 | 4 |
| 5 | 销毁共享内存 | shmctl | —— | —— |

# 消息队列

# 消息队列

- 消息队列是一个由系统内核负责存储和管理，并通过消息队列标识符引用的消息链表队列




知识讲解

# 消息队列

知识讲解

- 可以通过msgget函数创建一个新的消息队列或获取一个已有的消息队列。
- 可以通过msgsnd函数向消息队列的尾端追加消息，所追加的消息除了包含消息数据以外，还包含消息类型和数据长度(以字节为单位)。
- 可以通过msgrcv函数从消息队列中提取消息，但不一定非按先进先出的顺序提取，也可以按消息的类型提取



# 消息队列

- 相较于其它几种IPC机制，消息队列具有明显的优势
  - 流量控制：如果系统资源(内存)短缺或者接收消息的进程来不及处理更多的消息，则发送消息的进程会在系统内核的控制下进入睡眠状态，待条件满足后再被内核唤醒，继续之前的发送过程
  - 面向记录：每个消息都是完整的信息单元，发送端是一个消息一个消息地发，接收端也是一个消息一个消息地收，而不象管道那样收发两端所面对的都是字节流，彼此间没有结构上的一致性
  - 类型过滤：先进先出是队列的固有特征，但消息队列还支持按类型提取消息的做法，这就比严格先进先出的管道具有更大的灵活性
  - 天然同步：消息队列本身就具备同步机制，空队列不可读，满队列不可写，不发则不收，无需象共享内存那样编写额外的同步代码

知识讲解

# 消息队列

- 系统限制
  - 可发送消息字节数上限：8192
  - 单条队列消息总字节数上限：16384 (16K)
  - 全系统总消息队列数上限：16
  - 全系统消息总字节数上限：262144 (256K=16x16K)

知识讲解

# 消息队列

知识讲解

- #include <sys/msg.h>
- int msgget(key_t key, int msgflg);
  - 功能：创建新的或获取已有的消息队列
  - 参数：key：键。
    msgflg：创建标志，可取以下值：
    0 - 获取，不存在即失败。
    IPC_CREAT - 创建，不存在即创建，已存在即获取。
    IPC_EXCL - 排它，不存在即创建，已存在即失败。
    通过位或组合读写权限。
  - 返回值：成功返回消息队列的ID，失败返回-1。

# 消息队列

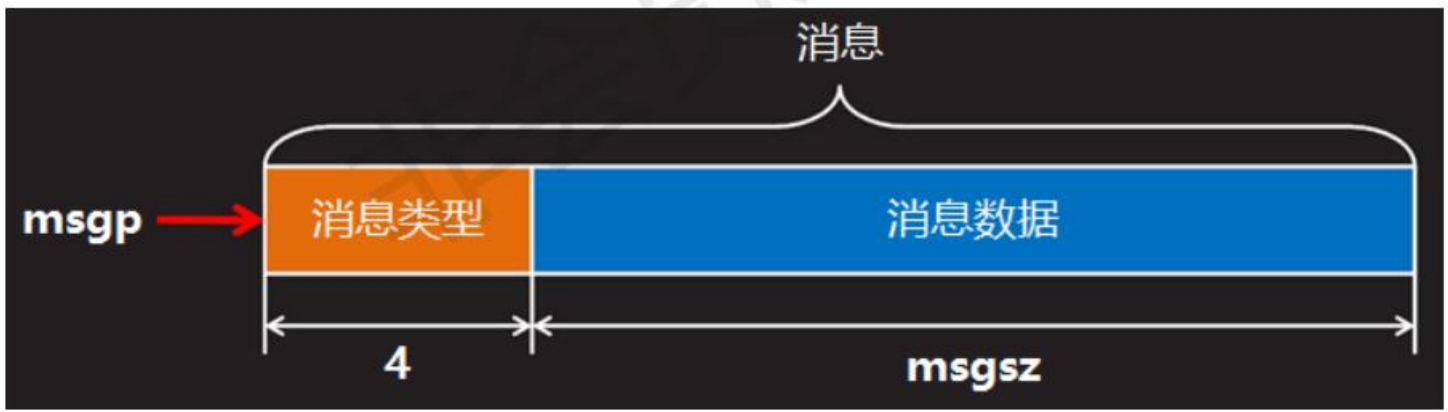
知识讲解

- #include <sys/msg.h>
- int msgsnd(int msgid, void const* msgp, size_t msgsz, int msgflg);
  - 功能：发送消息
  - 参数：msgid：消息队列的ID。
    msgp：指向一个包含消息类型和消息数据的内存块。该内存块的前4个字节必须是一个大于0的整数，代表消息类型，其后紧跟消息数据
    msgsz：期望发送消息数据(不含消息类型)的字节数
    msgflg：发送标志，一般取0即可
  - 返回值：成功返回0，失败返回-1。

# 消息队列

知识讲解

- 注意msgsnd函数的msgp参数所指向的内存块中包含消息类型，其值必须大于0，但该函数的msgsz参数所表示的期望发送字节数中却不包含消息类型所占的4个字节消息



# 消息队列

知识讲解

- 消息队列缺省为阻塞模式，如果调用msgsnd函数发送消息时，超过了系统内核有关消息的上限，该函数会阻塞，直到系统内核允许加入新消息为止。比如有消息因被接收而离开消息队列
- 如果msgflg参数中包含IPC_NOWAIT，则msgsnd函数在系统内核中的消息已达上限的情况下不会阻塞，而是返回-1，同时置errno为EAGAIN。

# 消息队列

知识讲解

- #include <sys/msg.h>
- int msgrcv(int msgid, void* msgp, size_t msgsz, long msgtyp, int msgflg);
  - 功能：接收消息
  - 参数：msgid：消息队列的ID。
    msgp：指向一块包含消息类型(4字节)和消息数据的内存
    msgsz：期望接收消息数据(不含消息类型)的字节数
    msgflg：接收标志，一般取0即可



# 消息队列

知识讲解

- #include <sys/msg.h>
- int msgrcv(int msgid, void* msgp, size_t msgsz, long msgtyp, int msgflg);
  - 参数：msgtyp：消息类型，可取以下值

    0- 提取消息队列的第一条消息

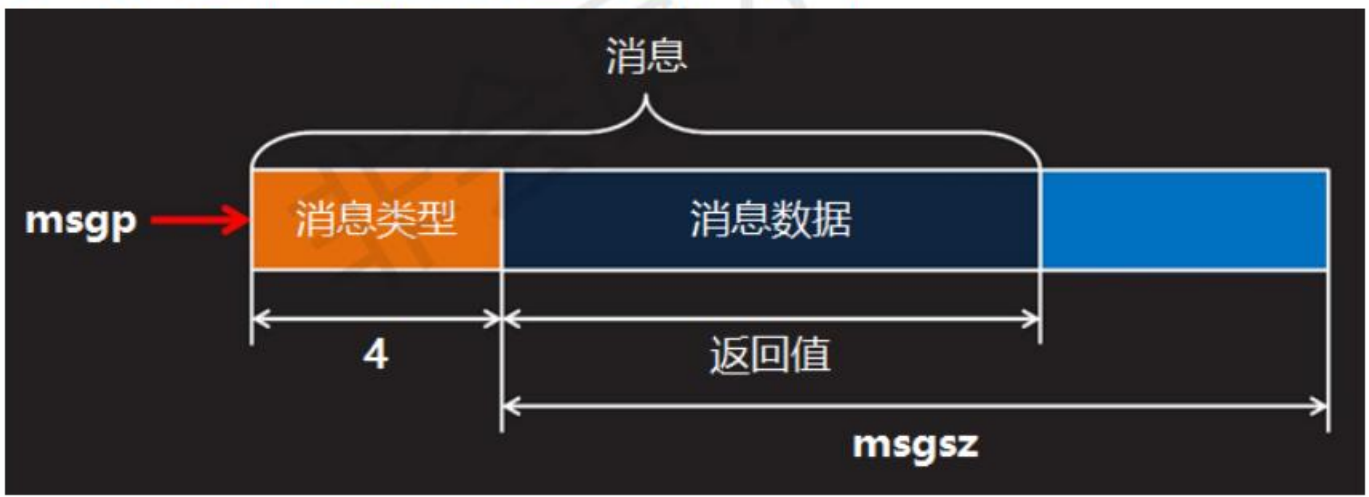
    >0- 若msgflg参数不包含MSG_EXCEPT位，则提取消息队列的第一条类型为msgtyp的消息；若msgflg参数包含MSG_EXCEPT位，则提取消息队列的第一条类型不为msgtyp的消息

    <0- 提取消息队列中类型小于等于msgtyp的绝对值的消息，类型越小的消息越被优先提取
  - 返回值：成功返回实际接收到的消息数据字节数，失败返回-1。

# 消息队列

知识讲解

- 注意msgrcv函数的msgp参数所指向的内存块中包含消息类型，其值由该函数输出，但该函数的msgsz参数所表示的期望接收字节数以及该函数所返回的实际接收字节数中都不包含消息类型所占的4个字节



# 消息队列

知识讲解

- 若存在与msgtyp参数匹配的消息，但其数据长度大于msgsz参数，且msgflg参数包含MSG_NOERROR位，则只截取该消息数据的前msgsz字节返回，剩余部分直接丢弃；但如果msgflg参数不包含MSG_NOERROR位，则不处理该消息，直接返回-1，并置errno为E2BIG
- 若消息队列中有可接收消息，则msgrcv函数会将消息移出消息队列，并立即返回所接收到的消息数据的字节数，表示接收成功，否则此函数会阻塞，直到消息队列中有可接收消息为止。若msgflg参数中包含IPC_NOWAIT位，则msgrcv函数在消息队列中没有可接收消息的情况下不会阻塞，而是返回-1，同时置errno为ENOMSG。

# 消息队列

知识讲解

- #include <sys/msg.h>
- int msgctl(int msgid, IPC_RMID, NULL);
  - 功能：销毁消息队列
  - 参数：msgid：消息队列的ID
  - 返回值：成功返回0，失败返回-1



# 消息队列

- 基于消息队列实现进程间通信的编程模型

知识讲解

| 步骤 | 进程A | 函数 | 进程B | 步骤 |
| --- | --- | --- | --- | --- |
| 1 | 创建消息队列 | msgget | 获取消息队列 | 1 |
| 2 | 发送接收消息 | msgsnd<br>msgrcv | 发送接收消息 | 2 |
| 3 | 销毁消息队列 | msgctl | —— | —— |

# 谢谢

# UC Day013

## 复习课

# IPC命令

# IPC命令

- 查看系统中的IPC对象
  - ipcs -m (memory, 共享内存)
  - ipcs -q (message queue，消息队列)
  - ipcs -s (semphore，信号量集)
  - ipcs -a (all, 所有的)
- 删除系统中的IPC对象
  - ipcrm -m 删除共享内存
  - ipcrm -q 删除消息对列
  - ipcrm -s 删除信号量集

知识讲解



# 下节课见