《openEuler内核编程》

课程讲稿

第四章 第6讲

IPC

软件所制

第四章 第6讲 IPC

**学时：**2学时

**教学目的：**系统学习IPC

**PPT讲稿：**



今天的内容是第6讲，进程间通信IPC。



这一节分5个部分，我们将主要介绍进程通信的概念、通信方式的设计考量、管道、Linux信号机制、dbug等。



首先，进程间通信全程是IPC，Inter-process Communication。

不同进程间进行通信和同步的机制

IPC提供两个基本原语：Send (message)和Receive (message)

进程通信流程：建立通信链路；Send/Recv交换数据

这是进程间通信的架构图。

发送方和接收方通过消息队列、共享内存、管道、邮箱等方式传递消息。

消息发送时，需要指定对方进程pid、文件、端口、消息类型和相关缓冲区

通信方式如图。

消息传递方式：进程A将消息传递到内核，再由内核传递给进程B

共享内存方式：内核分配一块共享内存映射到进程A和进程B的地址空间，A和B直接读写共享内存以发送接收消息。

消息队列是由OS维护的以字节序列为基本单位的间接通信机制

每个消息是一个字节序列，相同标识的消息按先进先出顺序组成一个消息队列

这是消息队列的相关系统调用，包括获取消息队列标示、发送、接收消息、消息队列控制等

共享内存方式是OS将同一块物理内存区域同时映射到多个进程的地址空间的通信机制

每个进程将共享内存区域映射到死有地址空间

优点：快速共享数据

缺点：必须用额外的同步机制来协调数据访问

那线程呢？

同一进程中的线程总是共享相同的内存地址空间

最快的方法

一个进程写另外一个进程立即可见

没有系统调用干预

没有数据复制

不提供同步：需要程序员自己提供同步

共享内存的系统调用。

包括创建共享段、映射到进场地址空间、取消映射、控制共享段

需要信号量等机制协调共享内存的访问冲突

我们来比较一下通信方式

消息队列通过内核地址空间，实现简单（在内存中预设区域），受到内核大小限制=>小消息，每次通信操作需要系统调用（如send，receive），且不易于开发

共享内存通过可用空闲物理内存实现，允许大消息传递（只受限于物理内存），违反了进程的内存保护策略

但是实现复杂：进程需要知道共享内存区域的位置

易于开发：只需要系统调用分配共享内存，读写即可

我们来介绍一下多种通信方式的设计考量

通信方式设计时，要考虑是否缓冲消息，是进行直接还是间接通信，单向、双向，异步还是同步，是通过事件驱动还是主动接受，以及怎样进行例外处理。

首先消息缓冲的设计。

包括无缓冲、有界缓冲和无界缓冲



直接通信需要明确消息目标，接收时也需要明确消息源。

而间接通信通过一个中间人，即mailbox。

通过一个mailbox来进行消息收发。Mailbox是一个唯一的标示。

直接通信时，必须知道接收信息的进程（进程已经执行）；间接通信时，谁接收信息可以在消息发送之后再决定

当两个进程同时要从mailbox接收消息，谁能获得这个消息？消息的排序呢？



直接通信只有接收端有缓冲时，多个进程可能向接收方发送消息，从特定的进程接收消息需要遍历整个缓冲区

每个发送者有一个缓冲区时，每个发送者发送给多个接收者，获取消息仍需要遍历缓冲区



间接通信使用信箱，允许多对多通信，需要打开/关闭信箱，

在信箱需要有一个缓冲区以及互斥锁和条件变量。消息长度不确定，可以把大消息分割成多个小消息

信箱和管道相比，信箱允许多对多通信，管道隐含一个发送一个接收

19:

在消息发送时，同步（阻塞）方式如果资源忙则阻塞，启动数据传输直到源缓冲用完后再阻塞

异步（非阻塞）方式，如果资源忙则阻塞，启动数据传输并且立即返回。结束时需要应用检查状态，通知或者向应用发信号

20:

在消息接收时，同步方式如果有消息则返回数据

异步方式如果有消息则返回数据，如果无消息则返回状态

21:

hrecv(src,type,msg,func) 中消息是函数func的一个参数

消息到达后执行函数

用一个接收线程模拟事件处理，使用事件处理模拟管程

22:

键盘输入是一个典型的例子

需要一个中断处理器，从中断处理函数中生成一个mbox消息

假设键盘设备线程把每个输入字符转换成一个 mbox消息，键盘中断函数怎么和设备线程同步? 设备线程怎么把输入转换成mbox消息?

23:

对于例外处理，当进场结束时

R等待S发来的消息但S已经结束，R会永久阻塞

S发送一个消息给R，但R已经结束，S没有缓冲，永久阻塞

24:

消息丢失时，使用确认ack和超时timeout检测和重传消息

需要接收者每收到一个消息发送一个确认，发送者阻塞知道ack到达或者超时。如果超时发生且没有收到确认，重发消息

问题是会带来重复，或者丢失确认消息

25:

因此，重传必须处理。在接收端消息重复，发送端确认乱序

使用序列号确认是否重复，在接收端删掉重复消息，发送者收到乱序确认时重传

减少确认消息，批量传送确认，接收者发送noack

26:

当消息损坏时，需要进行检测和纠正。

发送端计算整个消息的校验并随消息发送校验和(CRC)，在接收端重新计算校验并和消息中的校验和对比

通过重传 、使用纠错码恢复进行纠正

27:

典型的例子，Socket消息传递

他是TCP和UDP的抽象，通过IP地址和端口进行寻址。

28:

接下里接受管道

29:

进程间基于内存文件的通信机制，子进程从父进程继承文件描述符；缺省文件描述符:0 stdin, 1 stdout, 2 stderr

进程不知道的另一端，可能从键盘、文件、程序读取，也可能写入到终端、文件、程序

30:

管道的相关系统调用包括读写管道，创建管道

31:

这是一个简单的示例。

第一步创建管道，shell为ls创建一个进程，设 置 stdout为 管道写端，为more 创建一个进程, 设置 stdin 为管道读端

32:

Unix谱系下，每个进程都有三个已经打开的“文件”。并非真实的文件，但在unix下，所有对象都可以看作一个文件

即Stdin: 标准输入流；Stdout：标准输出流和Stderr：标准错误流

例如printf 就输出到stdout

每个文件都关联了一个整型的文件描述符

33:

在unix中，当打开一个新文件，这个文件会获得第一个可用的文件描述符

若关闭stdout，再打开一个文件，该文件会有文件描述符 1

因此，printf会以为这个文件时stdout，并将输出写入该文件，且不需要更改ls的任何代码！

34:

下面示例会执行ls –ls 并将其输出写到文件/tmp/stuff中

35:

前一个示例中，执行顺序是关闭stdout；然后打开一个文件，该文件获得描述符 1

如果已经打开了该文件，但他获得了其他的文件描述符？

可以用dup系统调用为一个已经打开的文件创建另一个文件描述符，并从低开始选择一个为使用的文件描述符号

36:

因此前一个例子可以进行重写

37:

Popen函数，是fork函数的一个变体，创建一个双向pipe

Forks and execs一个子进程 （e.g， “ls -a”）；返回管道，其实是一个文件（FILE \*）

父进程和子进程可以通过pipe通信

38:

这个例子就是ls -la输出的实现

39:

接下来我们介绍一下Linux的信号机制

40:

进程间的软件中断通知和处理机制 ，如:SIGKILL, SIGSTOP, SIGCONT等

信号的接收处理包括：捕获(catch)、忽略(Ignore)和屏蔽(Mask):

不足之处是传送的信息量小，只有一个信号类型

41:

这是操作系统内核对信号的实现。

第一步在内核注册信号处理函数。

当发送信号到指定CPU时，内核调用信号处理函数进行处理反馈。

42:

这是个简单的信号使用例子。

43:

为了保障对于以前版本unix内核的兼容，每次都先要初始化重置信号。不重置对于当前版本其实没有影响。

44:

45:

应用通过signal()、sigaction()注册信号处理程序

通过kill()等函数传递信号，或者由内核中硬件异常处理函数触发信号

Signal传递跳转到signal处理函数，类似于中断的处理过程

46:

Signal是异常和catch blocks的底层机制

JVM或其他的运行时系统会设置相关singal处理函数

47:

这是信号异常控制流程。

应用正常秩序过程中，调用信号相关系统调用注册信号处理函数。

信号发生时，触发注册的函数例程进行执行。

48:

信号处理函数可以执行在另外一个栈上，通过sigaltstack()系统调用

类似中断处理，内核将寄存器状态推送到栈上

通过sigreturn()系统调用返回到内核，应用可以修改其私有的栈上寄存器状态

49:

用户态栈段，其中Petcode:是信号处理函数的返回地址

50:

内核通过一小段汇编代码来清理信号处理函数现场

Signal trampoline: Sigreturn()将重制信号处理函数的所有操作

Signal trampoline 代码一半存在于vDSO或者C库中

vDSO（virtual dynamic shared object）是内核自动映射到每个用户态应该地址空间的共享库

51:

对于多线程异常信号处理

第一个可用线程获得信号，大部分处理例程跑在线程的栈上，一个处理例程可以跑在指定的栈上

内核态的线程不执行该处理例程，直到其返回用户态

52:

信号默认处理例程包括Ignore，kill，suspend，continue，dump core，在内核执行

通过signal/sigaction注册新的信号处理例程，覆盖默认。Sigkill，sigstop等不能被覆盖

53:

发送信号，即在任务中标示一个等待的signal

如果有TASK\_INTERRUPTIBLE标示，则被阻塞

当中断或syscall返回时，检查等待的signal，如果有等待的signal，发送。

54:

55:

D-BUS是2002年创建的一种进程间通信机制。是Freedesktop.org项目的一部分，由redhat和freedesktop社区维护。

他是高层次的IPC，包括Multicast & point-to-point，与OS/体系结构/语言无关

56:

通过库libdbus，允许两个应用程序相互连接并交换消息

一个消息总线守护程序，多个应用程序可以连接

通过policy文件制定安全机制

57:

D-Bus的总线bus相当于D-Bus的通信链路，应用之间通过总线进行通信，应用在总线上寻找service

系统总线System Bus：用于内核，系统应用/服务

任务总线Session Bus：用于gnome/kde等应用通信

D-Bus的服务service是提供IPC API的程序，每个服务都有一个reverse domain name结构的标识名称

58:

D-bus的对象相当于通信的地址，每个service的对象都通过对象路径来标示

D-bus的接口定义了对象支持的方法method和信号signal，每个对象包含一个或多个接口

D-bus的信号提供了一对多的发布-订阅机制

与方法返回值类似，D-bus信号可能包含任意数量的数据

与方法不同，信号是完全异步的，并且可以随时由对象发出

59:

应用调用代理上的方法，代理将构造一个方法调用消息给远端的进程

方法调用消息发送到 bus daemon 中

bus daemon 查找目标的 bus name，如果找 到，就把这个方法发送到该进程中

在 dbus 高层接口中，会先检测并转换成对应 的对象的方法，然后再将应答结果转换成应 答消息发给 daemon

bus daemon 接受到应答消息，将把应答消息 直接发给发出调用消息的进程

60:

完