2021 年 3 卷 2 期 ISSN: 2661-3751(Print); 2661-376X(Online)

系统集成中进程间通信技术的运用分析与探究

霍金鹏　邸少峰　王永杰
三维通信股份有限公司　浙江　杭州　310012

摘要：进程间的通信技术已经成为更加有效利用网络及个体计算机资源的必备方式。本文对系统集成和进程间通信技术的特性及目的进行了分析，围绕无名管道、命名管道、消息队列、信号量、共享存储（内存）五种常用进程间通信方法的原型程序、通信原理加以介绍，并简单梳理了进程间通信技术应用于系统集成的方式，以供参考。

关键词：系统集成；进程间通信；管道；消息队列；共享存储

引言：进程是计算机系统进行资源分配和调度的基本单位，是操作系统结构的基础 [1]。在早期的计算机结构中，进程是程序的基本执行实体；时至今日，进程已经成为线程的容器。随着计算机性能的逐渐提升，系统在同一时间可执行多个程序，即有多个“进程”。在此种情况下，系统的资源需要共享，故个程序在执行的过程中必然陷入相互“制约”的境地，具体体现便是每个程序都会出现卡顿（实则是程序执行中断）。为了进一步提高系统运行速度，需引入进程间通信技术。

1. 系统集成及进程间通信技术的特性分析

1.1 系统集成

系统集成是一个计算机及网络领域的专用词汇，是指通过结构化和综合布线系统，将多个处于分离状态设备的功能、信息等集中在基于网络的统一控制系统中，目的在于使区域范围内的所有资源构成一个庞大的资源整体，处于网络之中的所有终端设备都可以按照一定的规程集中获取并运用这些资源，进而从根本上提高网络资源利用率。因此，从某种意义上来看，早期系统集成可被视为功能集成。但时至今日，在计算机系统及网络云领域之内，所有能够以数据形式呈现的内容都可以成为“被集成”的对象。换言之，系统集成经过验证之后，已经成为一种提高资源利用率的最有效资源共享模式。为了成功构建侧重模式，解决各类设备、子系统之间的接口、协议等问题已经成为硬件层面的“小问题”；核心问题在于如何使资源共享的过程更加顺畅。比如云端存储资源的数据库及服务器实际上便是系统集成模式下的重要组成部分，在用网高峰时段，大量用户集中访问之下，资源的传输效率必定会受到影响。在此期间，若要尽量维持每一个客户的访问速度，必须运用进程间通信技术。

1.2 进程间通信

“进程”的本质便是“正在运行的程序”。如果一个（子）系统中同一时段只有一个（相对而言，不同程序的“体量”之间存在规模性差异，微小程序产生的任何影响都微乎其微）进程，则不会产生“通信”问题。但若同时存在多个“影响力”相当的进程，则不同进程之间经常需要传播或交换信息。以“远程控制”为例，两个人持有两台独立运行的计算机，其中一人的终端设备出现了一些故障，需要他人借助远程控制协助解决，此间涉及的问题为：①两人两台电脑中各自的进程均处于独立状态，按照常态不能相互访问。②建立连接之后，通过一台电脑控制操作另一台电脑时发生的所有信息交互（可理解为进程交互）均经由“共享内存区”而实现。此时，两台电脑的系统空间中有一部分暂时成为了“公共场所”，多个进程均可以访问。由此便实现了不同电脑之间的进程间通信。在系统集成模式下，类似远程控制的进程间通信行为数量必然激增，故探讨进程间通信技术在系统集成中的有效运用方式很有必要。

2. 进程间通信技术常见的五种通讯方式梳理

进程间通信技术常见的五种信息通讯方式分别为无名管道、命名管道、消息队列、信号量、共享存储（内存）[2]。

2.1 无名管道

无名管道简称“管道”，是 UNIX 操作系统下最常见的进程间通信形式。管道共有三个特点：①半双工形式，即数据只能在一个方向上流动，有固定的读写端（独立）；②只能应用于父子、兄弟等有“亲缘关系”进程之间的通信；③可被视为一种特殊的文件，但不属于任何文件系统，只存在于内存中，可使用常规的读写函数进行操作控制。如上文所述，单一进程中，通信无任何意义，因此，通过无名管道实现进程通信的前提在于必须同时具备父子（兄弟）进程。如下列程序（只列举核心部分，前端调用函数部分未纳入）：

```
 4 int main()
 5 {
 6    int fd[2];  // 两个文件描述符
 7    pid_t pid;
 8    char buff[20];
10    if(pipe(fd) < 0)  // 创建管道
11        printf("Create Pipe Error!\n");
13    if((pid = fork()) < 0)  // 创建子进程
14        printf("Fork Error!\n");
15    else if(pid > 0)  // 父进程
```

```
16  {
17      close(fd[0]); // 关闭读端
18      write(fd[1], " hello world\n ", 12);
19  }
20  else
21  {
22      close(fd[1]); // 关闭写端
23      read(fd[0], buff, 20);
24      printf("%s", buff);
25  }
27  return 0;
28 }
```

若要实现数据从父进程流向子进程，则必须对父进程的读端（fd[0]）和子进程的写端（fd[1]）执行“ 关闭 ”控制操作；若不执行该操作，则数据便会由子进程流向父进程。

2.2 命名管道

命名管道简称“ FIFO ”，同样是一种文件类型，主要特点有二：其一，与无名管道最大的区别在于，可在无关联的进程之间实现数据的交换；其二，通信路径的名称与 FIFO 之间存在一定的关联，进而能够以一种特殊设备文件的形式 在文件系统中储存。此种进程间通信形式的原型程序如下：

```
1 #include <sys/stat.h>
2 // 返回值：成功返回 0，出错返回 -1
3 int mkfifo(const char *pathname, mode_t mode);
```

第三行程序中的 mode 是一个参数，与 open 函数中的 mode 运行机制相同。如果创建了一个命名管道，则一般文件的 I/O 函数便可用于操作控制该管道。如果 open 函数中存在一个命名管道之后，有关“ 非阻塞标志 O_NONBLOCK ”的具体控制区别为：如果 O_NONBLOCK 处于默认状态，即“ 无任何额外指定 ”命令时，处于“ 只读 ”状态的 open 函数需要对某个其他进程执行“ 阻塞 ”作业，如此一来，对应的命名管道便会随之“ 打开 ”，一种特定的进程间通信行为便会成功实施。如果 O_NONBLOCK 经由控制程序而收到了额外的指令，则 open 函数便不会对其他进程进行阻塞。但计算机 C 语言程序与人类大脑的思考方式存在明显差异，具体而言：如果 open 函数是人脑中的一项“ 调度指令 ”，在发现某个或某几个其他进程不需要进行阻塞时，大脑只需停止思考 open 函数（不管了）即可；但在计算机 C 语言程序中，因为 open 函数已经处于“ 待机准备 ”状态，如果放任不管则会引起程序运行错乱。因此，需通过“ 写端 ”体现 open 函数此时所处的状态，目的在于对控制程序整体进行“ 交代 ”。具体体现形式便是：open 函数出错，需要“ 返回 ”，记“ -1 ”，此时 open 函数不会执行阻塞作业，命名管道 FIFO 便会“ 打开 ”。

2.3 消息队列

消息队列以一种“ 链接表 ”的形式存放在控制内核中，一个消息队列通常由一个特定的标识符（具备相应的 ID）加以表示，有助于系统和其他进程识别。此种方式的特点为：①消息队列面向记录，其内的消息在格式及优先级方面均有“ 特权 ”；②消息队列与常规的进程信息发送、接收流程无关，处于独立状态。如果进程终止，则消息队列和内中包含的信息依然会得到保留，不会消失；③通过消息队列，可实现对进程通信信息的随机查询，除了按照常规模式的“ 先进先出 ”次序读取相关信息之外，还可以按照消息的类型实现读取。此种进程间通信的原型程序为：

```
1 #include <sys/msg.h>
2 // 创建或打开消息队列：成功返回队列 ID，失败返回 -1
3 int msgget(key_t key, int flag);
4 // 添加消息：成功返回 0，失败返回 -1
5 int msgsnd(int msqid, const void *ptr, size_t size, int flag);
6 // 读取消息：成功返回消息数据的长度，失败返回 -1
7 int msgrcv(int msqid, void *ptr, size_t size, long type,int flag);
8 // 控制消息队列：成功返回 0，失败返回 -1
9 int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

读取消息队列中的具体内容是，需要使用 msgrcv 函数。其中，type 参数存在下列三种情况：

① type == 0，返回队列中的第一个消息；

② type > 0，返回队列中消息类型为 type 的第一个消息；

③ type < 0，返回队列中消息类型值小于或等于 type 绝对值的消息，如果有多个，则取类型值最小的消息。

当 type 值处于非 0 状态时，mstrcv 函数读取消息队列无需遵从先进先出原则，此种形式即可理解为消息队列的特殊读取优先级。

2.4 信号量

信号量进程间通信模式的实质是一个计数器，可用于实现进程间的互斥与同步，并非用于存储进程间通讯数据[3]。此种模式的特点为：①信号量用于进程间同步，若要实现进程间数据传递必须借助共享内存；②信号量基于操作系统的 PV 操作，程序对信号量的操作均为原子操作；③对信号量的所有 PV 操作都可任意加减正整数，操作空间并非仅限于“ 1 ”；④可支持信号量组。在 Linux 操作系统下，信号量进程间通信的原型程序如下：

```
1 #include <sys/sem.h>
2 // 创建或获取一个信号量组：若成功返回信号量集 ID，失败返回 -1
3 int semget(key_t key, int num_sems, int sem_flags);
4 // 对信号量组进行操作，改变信号量的值：成功返回 0，失败返回 -1
```

5 int semop(int semid, struct sembuf semoparray[], size_t numops);

6 // 控制信号量的相关信息

7 int semctl(int semid, int sem_num, int cmd, ...);

上述程序中的信号量只能收取 0 和 1 两种变量，即二值信号量。需要注意的是，Linux 下的信号量函数在通用信号量数组上进行操作，并非在上述简单的二值信号量上进行操作。上述程序中提及的 semctl 函数中的命令有很多，可根据实际情况选用。

2.5 共享内存

顾名思义，两个以上的进程拥有一个给定的存储区。事实上，此种进程间通信方式是当前应用范围最广，最容易为人所理解的方式。主要特点为：①进程间通信速度最快，原因在于可直接对内存信息进行存储与读取，避免了很多其他流程；②多个进程可在同一时间内同时被操作，故需要进行同步作业；③如上文所述，信号量与共享内存常常结合运用，通过同步信号量的方式对共享内存进行访问。此种模式的原型程序与信号量极其相似，只需将“信号量”字样改为“共享内存”字样即可。

3. 进程间通信技术在系统集成中的运用

将上述五种进程间通信技术应用于系统集成时，程序的运行原理并未出现根本性的改变。当前的主流方式为通过编写“中介程序”，完成集成状态下的进程间通信。通常情况下，编写程序代码时需要使用 View 命令，可通过添加 SendComman 代码的方式，完成对进程间信息的读取、反馈，进而发送相关指令。总体而言，进程间通信技术应用于系统集成时，主要原理是相同的，只是通信的“场所”有所扩大。

结语：综上所述，进程间通信技术并不难理解，单一系统中进程的通信方式与系统集成模式下的通信方式并无本质区别，均需通过相应的程序进行存储与读写调用，可有效提高全网资源利用率。

参考文献：

[1] 肖堃 . 基于加权队列的跨平台进程间通信调度算法 [J/OL]. 吉林大学学报 (工学版):1-6[2021-04-15].

[2] 徐莺 , 江红 , 辜彬 , 等 . 一种进程间通信的自动化测试技术的实现与应用 [J]. 数字技术与应用 ,2019,37(11):12-15.

[3] 付腾 . 系统集成中进程间通信技术的运用探讨 [J]. 通讯世界 ,2019,26(01):42-43.

作者简介：霍金鹏（1982.10-），女，汉族，河北石家庄人，本科，供职于三维通信股份有限公司，研究方向：射频干扰对消技术、通信系统集成。