Linux 进程间的通信机制在 RoboCup 中的应用

贺建立, 程家兴, 刘慧婷, 鲁 杰
(安徽大学 计算智能与信号处理实验室, 安徽 合肥 230039)

摘 要: Linux 是多任务的操作系统, 进程之间相互可靠的通信是系统安全和稳定的重要保障。在 RoboCup 仿真组中, 每个 Agent 是由多线程来实现的。同样地, Agent 的线程之间可靠地进行通信是至关重要的。文中分析了 Linux 进程之间的一部分通信机制, 包括套接字、信号、互斥量, 并把它们应用于 RoboCup 中, 给出了 RoboCup 系统流程。每个 Agent 用多线程来实现比用多进程来实现大大降低了系统的开销。

关键词: 进程; 套接字; 信号; 互斥量
中图分类号: TN911.7 文献标识码: A 文章编号: 1005-3751(2004)02-0018-03

Application of Linux Interprocess Communication in RoboCup

HE Jian-li, CHENG Jia-xing, LIU Hui-ting, LU Jie
(Intelligent Computing & Signal Processing Laboratory, Anhui University, Hefei 230039, China)

Abstract: Linux is a multi-task operation system. Interprocess communication must be reliable, which ensures system's safety and stabilization. Each agent is made up of multi-thread in RoboCup Simulation. Similarly, agent's interthread reliable communication is vital. In this article, analyzes parts of interprocess communication mechanism, which include socket, signal, mutex variable. Apply them to RoboCup. Furthermore, represent the flow of RoboCup. Each agent who is made up of multi-thread not multi-process greatly reduces system's resource.

Key words: process; socket; signal; mutex variable

Linux 这样的多任务操作系统能够让许多程序同时运行, 一个运行中的程序就是一个进程。进程之间能够相互合作, 它们可以相互发送信息, 能够相互中断, 甚至能够把共享内存段的工作安排好, 但它们不能共享变量。在一个程序里的多个执行线路就叫线程。多个线程之间可以共享变量, 能够同时运行。运用"信号"或者"互斥量"能够达到多线程之间的同步的目的。线程在某些应用软件里有着很高的价值, 比如多线程的数据库服务器等。

RoboCup 是比较测试各种 AI 技术的一个合适领域, 在科研和教学领域里已经产生了广泛的兴趣。在 RoboCup 仿真组中, 采用的是 Client/Server 模型。每个 Client 是一个自治的多线程进程, 通过 UDP/IP 协议与 Server 进行通信。

1 Linux 进程间的通信机制

1.1 套接字

套接字涵盖了计算机网络中的通信, 使客户/服务器系统的开发工作既可以在本地单机上进行, 也可跨网络进行。一台机器上的某个进程可以使用套接字与另一台机器上的某个进程进行通信。首先, 服务器软件必须先创建一个套接字, 这是分配给该服务器进程的一个操作系统资源。接着, 服务器进程会给套接字起名。给本地套接字起的名字是 Linux 文件系统中的一个文件名, 一般放在/tmp 或/usr/tmp 子目录里。网络套接字的名字则是一个与客户所能连接的特定网络有关的服务标识符。给套接字起名字要使用系统调用 bind。然后, 服务器就开始等待有客户连接到这个命名套接字上来。系统调用 listen 的作用是创建一个队列, 来自客户的连接将在这个队列上排队等待服务器的处理。服务器通过系统调用 accept 来接受来自客户的接入连接。当服务器调用 accept 时, 会新创建一个套接字, 它的唯一用途就是与特定的客户进行通信。

套接字的特性是由三个属性确定的, 即域、类型和协议, 另外还有一个被用做其名字的地址[1]。地址的格式会根据域的不同而变化。域定义的是套接字通信中使用的网络介质。最常用的套接字域是 AF_INET, 它对应的是因特网的网络模型, 许多 UNIX 局域网用的都是它, 因特网自身用的也是它。一个套接字可以有多种不同的通信方式, 而每种通信方式又有不同的特性。因特网提供两种截然不同的服务: 流和数据图。流式服务提供的是一个有序的可靠的双向字节流。被发送的数据不会被丢失、复制或

收稿日期: 2003-07-11
作者简介: 贺建立(1974—), 男, 安徽宿松人, 硕士研究生, 研究方向为计算机算法、人工智能。

者先后次序被弄乱; 错误会被自动纠正而不是报告给用户。流式套接字被定义为 SOCK_STREAM 类型, 在 AF_INET 域里通过 TCP/IP 连接实现。SOCK_DGRAM 类型的数据图套接字不需要建立和维护一个连接。它在网络中传输时不会被撤分, 一个数据图就是一条网络消息, 它可能被丢失、复制或者不按先后次序到达目的地。数据图套接字在 AF_INET 域里是通过 UDP/IP 连接实现的, 它提供的是一种无序的不可靠服务。AF_INET 域里的套接字地址是由一个定义在 netine/in.h 头文件里的 sockaddr_in 结构确定的, 它至少应该包括如下的几个成员:

```
struct sockaddr_in {
    short int sin_family; /* AF_INET */
    unsigned short int sin_port ; /* Port number */
    struct in_addr sin_addr ; /* Internet address */
};
```

IP 地址结构 in_addr 被定义为:

```
struct in_addr {
    unsigned long int s_addr;
};
```

1.2 信　号

信号是 Linux 系统响应某些状况而产生的事件, 进程在接收到信号时会采取相应的行动[1]。它们可以明确地由一个进程产生并发送给另一个进程, 用这种办法传递信息或协调操作行为。进程可以通过调用 kill 向包括它本身在内的另一个进程发送信号。也可以通过 alarm 函数调用安排经过预定的时间后出现一个警告钟 SIGALARM 信号。int sigaction (int sig, const struct sigaction * act, struct sigaction * oact) 函数是一个健壮的信号接口。其作用是设置与信号 sig 关联着的操作动作。sigaction 结构是在 signal.h 文件里定义的, 它被用来定义在接收到 sig 指定的信号时将要采取的操作动作, 至少包含以下几个成员:

```
void (*) (int) sa_handler /* function
sigset_t sa_mask /* signals to block in sa_handler
int sa_flags /* signal action modifiers
```

在参数 act 指向的 sigaction 结构里, 函数指针指向一个将在接收到信号 sig 时被调用的函数。sa_mask 域给出的是在调用由参数 sa_handler 指定的函数之前将被添加到该进程的信号掩码里去的一组信号。这是一组将被阻塞且不会被发送给该进程的信号。由 sigaction 设置的处理器所捕捉到的信号在默认的情况下是不会被重置的, 如果希望获得对信号进行重置的效果, 就必须在 sa_flags 域里包含上标志值 SA_RESETHAND。

1.3 互斥量

使用互斥量是多线程程序中的同步访问手段之一。程序员给某个对象加上一把"锁", 每次只允许一个线程去访问它。如果想对代码关键部分的访问进行控制, 必须在进入这段代码之前锁定一把互斥量, 在完成操作之后再打开它。互斥量的作用就象是一把多人共用的钥匙, 谁拿到这把钥匙谁就可以访问受保护的代码段, 以确保某一时刻只有一个线程访问临界区内的代码。

2 解析 RoboCup 中的通信机制

在 RoboCup 仿真组中, 每个 Agent 是一个独立的进程。它把自己的决策行为发送给服务器进程, 服务器进程把球场上的信息以及其它相关信息发送给它。Agent 有两个线程, 主线程和辅助线程。主线程根据当前球场上的信息以及自身的状态作出相应的决策, 并把决策行为送入命令队列。辅助线程从服务器进程接收信息, 并解析信息, 然后更改世界状态, 为主线程做决策提供依据。此外, 辅助线程还负责及时地把命令队列里的命令发送给服务器进程。辅助线程在 main 函数里由下列语句创建: pthread_create(&sense, NULL, sense_callback , &s);

2.1 Agent 与服务器进程的通信

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0)
```

创建一个套接字, 它的域是 AF_INET, 类型是 SOCK_DGRAM, 使用缺省套接字协议。

```
cli_addr.sin_family = AF_INET ;
cli_addr.sin_addr.s_addr = htonl(INADDR_ANY) ;
cli_addr.sin_port = htons(0) ;
```

填写客户套接字地址。

```
bind(sockfd, (struct sockaddr *) &cli_addr, sizeof(cli_addr))
```

给套接字起名字, 将它关联到一个 IP 端口号上去。

```
m_sock.socketfd = sockfd
m_sock.serv_addr.sin_family = AF_INET ;
m_sock.serv_addr.sin_addr.s_addr = inet_addr(host);
m_sock.serv_addr.sin_port = htons(port) ;
```

填写服务器套接字地址。

```
sendto(m_sock.socketfd, msg, n, 0,
        (struct sockaddr *) &m_sock.serv_addr, sizeof(m_sock.serv_addr);
```

信息发送到服务器进程。

```
recvfrom(m_sock.socketfd, msg, maxsize, 0,
        (struct sockaddr *) &serv_addr, &servlen);
```

从服务器进程接收信息。

```
struct sigaction sigact;
sigact.sa_flags = SA_RESTART;
sigact.sa_handler = (void (*) (int)) sigalarmHandler;
sigaction( SIGALRM, &sigact, NULL );
```

设置一个与警告钟信号关联的操作动作 sigalarmHandler。其功能是把命令队列里的命令发送给服务器进程。标志 SA_RESTART 作用是重新启动可中断函数而不是给出 EINTR 错误。警告钟信号的产生是由 setitimer(ITIMER_REAL, &itv, NULL) 设定。函数 setitimer 与函数 alarm 作用是一样的。经过由参数 itv 设定的时间后

产生一个警告钟 SIGALARM 信号。每次 Sense 信息到达后都要调用该函数。每个仿真周期都要把命令发送给服务器进程,发送的时间要确保如果该仿真周期内有 See 信息到达,应该在 See 信息到达之后稍后发送。

2.2 线程间的同步

在函数 waitForNewInformation() 里,有下面的语句:

```
pthread_mutex_lock( &mutex_newInfo);
pthread_cond_timedwait( &cond_newInfo, &mutex_newInfo, &timeout);
pthread_mutex_unlock( &mutex_newInfo);
```

在函数 bool WorldModel:: setTimeLastSeeMessage(Time time) 或函数 bool WorldModel:: setTimeLastSenseMessage(Time time) 里有下面的语句:

```
pthread_mutex_lock( &mutex_newInfo);
bNewInfo = true;
pthread_cond_signal( &cond_newInfo);
pthread_mutex_unlock( &mutex_newInfo);
```

主线程根据世界状态作出相应的决策,把命令保存进命令队列后,调用函数 WM- > waitForNewInformation()。该函数一直处于阻塞状态,直到 timeout 时间到,或者函数 bool WorldModel:: setTimeLastSeeMessage(Time time) 或函数 bool WorldModel:: setTimeLastSenseMessage(Time time) 调用,发出条件信号。这两个函数在辅助线程收到感知信息时被调用。当 timeout 时间到时函数 pthread_cond_timedwait(&cond_newInfo, &mutex_newInfo, &timeout) 返回 ETIMEDOUT,进程结束[2]。整个系统流程如图 1 所示。

3 结束语

在 RoboCup 仿真组中,创建一个新线程比创建一个新进程有更明显的优势。每个 Agent 用两个线程而不是两个进程来实现很大程度上降低了系统的开销。从系统的流程可以看出,主线程和辅助线程以一种密切合作的关系同时运行,而且共享着一些变量,这一点用多进程是不能实现的。主线程和辅助线程通过互斥量和条件信号同步。

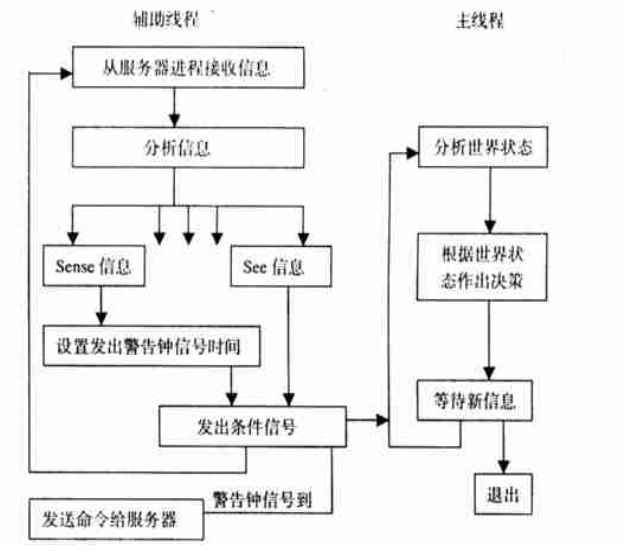

图 1 RoboCup 系统流程

参考文献:

[1] Stones N M R. Linux 程序设计[M]. 北京: 机械工业出版社, 2002.

[2] Stevens W R. 进程间的通信[M]. 北京: 清华大学出版社, 2002.

(上接第 17 页)

CmppSendSingle() 实现。具体做法是短信发送程序定时扫描待发短信表,如果有待发的短信,短信发送程序立即发送一条短信并从待发短信表中删除该条短信,然后短信发送程序会将短信的发送结果写入发送日志表,在主窗口中也会显示该条短信发送的情况。

⑤配置文件管理程序在主窗口上提供系统初始化配置的功能,配置文件管理程序只能在短信接收和发送程序没有启动时使用。

3.2 CMPP API 的加载

发送短信过程中一定要有 CMPPAPI.DLL 的加载和初始化,才能保证信息发送。加载和初始化代码如下[6]:

```
hLIBrarySend= LoadLibrary("CMPPAPI.dll");
InitCmppApi= (INITCMPPAPI) GetProcAddress(hLibrarySend, "InitCMPPAPI");
Char * ppathFileName;
PpathFileName= (LPSTR)(LPCTSTR)"cmppc.ini";
(* InitCmppapi)(ppathFileName);
```

4 结束语

基于 Internet 的短信发送、订阅已被企业和个人广泛接受与使用,文中从介绍 CMPP 协议着手,深入探讨了短信收发系统平台中的关键技术——短信收发监控程序的设计方案。相信此设计方案为 Internet 短信服务程序的开发能提供借鉴作用。

参考文献:

[1] 中国移动通信互联网短信网关接口协议[Z]. 中国移动通信集团公司, 2002.

[2] Short Message Peer to Peer Protocol Specification v3.4[Z]. SMPP Developers Forum, 1999.

[3] 宋国森. TCP/IP 应用指南[M]. 北京: 机械工业出版社, 1996.

[4] 郑 沫. Microsoft SQLServer7.0 开发实例精解[M]. 北京: 北京希望电子出版社, 2000.

[5] 段 兴. Visual C++ 实用程序 100 例[M]. 北京: 人民邮电出版社, 2002.

[6] 梁书斌. Visual C++ 6.0 高级编程[M]. 北京: 清华大学出版社, 1999.