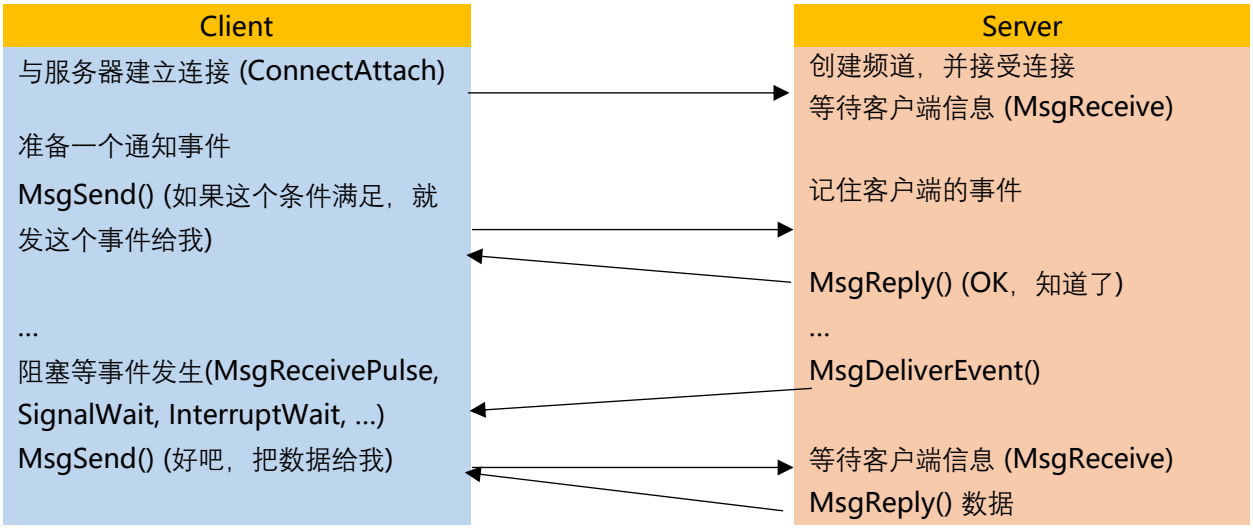
关于 MsgDeliverEvent()

对于 QNX 的 MsgDeliverEvent() 这个内核调用，有不少疑问，分出来细讲一下吧。这个函数的基本于法是这样的：



如上所见，客户端是会需要阻塞等待事件发生的。但这个并不是绝对的，根据事件具体是什么而定。

MsgDelieverEvent()里的事件

MsgDelieverEvent()里的事件其实可以有好几种，具体使用哪种，取决于客户端与服务器端的约定。具体事件是在 sys/siginfo.h 里定义的。

```
#define SIGEV_NONE              0       /* notify */
#define SIGEV_SIGNAL            1       /* notify, signo, value */
#define SIGEV_SIGNAL_CODE       2       /* notify, signo, value, code */
#define SIGEV_SIGNAL_THREAD     3       /* notify, signo, value, code */
#define SIGEV_PULSE             4       /* notify, coid, priority, code,
value */
#define SIGEV_UNBLOCK           5       /* notify */
#define SIGEV_INTR              6       /* notify */
#define SIGEV_THREAD            7       /* notify, notify function,
notify_attributes */
```

具体的定义，可以去 QNX 用户手册里查；一般来说，比较常用的，就是“脉冲”(SIGEV_PULSE)；”信号“(SIGEV_SIGNAL)和中断（SIGEV_INTR）。这里，脉冲需要客户端阻塞在一个事先准备好的频道上(MsgReceive() / MsgReceivePulse())；而中断，可以用 InterruptWait()来等。但是对于“信号”，客户端可以阻塞在 sigwait() / sigwaitinfo() 上，专门等这个事件；但是，也可以跟传统 UNIX 的“信号处理”一样，对于指定的信号用 signal()函数挂一个处理回调，这样客户端不需要专门阻塞在某一点，只要异步处

理信号就可以了。当然需要提醒的是，在多线程编程时代，异步处理也是有很多同步措施需要担心的；所以有时候于其担心自己的程序不知道什么时候会被打断，还不如开一个线程为阻塞点，这样程序也容易一点。

也许会有疑问，既然客户端终究是要阻塞的，那直接开个线程，阻塞到服务器上多好，何必这么复杂地绕来绕去呢？这是因为有些情况下没办法直接阻塞到服务器端，最明显的例子就是 POSIX 的 select() 函数。

Select()函数

Select()这个函数，在单线程 UNIX 程序的时候，应该算是一个非常常用到的操作了。大家有机会看看 Unix 常用程序的源码，在很多场合可以看到这个函数。QNX 在 sys/select.h 里，定义了这个函数。

```
int select( int width,
            fd set * readfds,
            fd set * writefds,
            fd set * exceptfds,
            struct timeval * timeout );
```

具体详细定义大家可以查说明，但基本上的意思就是给出“多个”fd，当这些 fd 可以读（readfds），可以写（writefds），出错（exceptfds）时，返回。timeout 可以是 NULL，表示一直等下去，或者可以给出时间，这样可以在一定的时间后，自动退出 select()。

一个用 select()的程序通常是这样。

```
switch ( n = select( 1 + max( console, serial ), &rfd, 0, 0, &tv ) ) {
      case -1:
        perror( "select" );
        return EXIT FAILURE;
      case  0:
        puts( "select timed out" );
        break;
      default:
        printf( "%d descriptors ready ...\n", n );
        if( FD ISSET( console, &rfd ) )
          puts( " -- console descriptor has data pending" );
        if( FD ISSET( serial, &rfd ) )
          puts( " -- serial descriptor has data pending" );
    }
```

好，现在问题来了，在 QNX 上要怎么实现这个函数？

在 QNX 上实现 select()函数

在 QNX 上，每一个 fd 就是指向一个服务器的“连接号“。如果同时 select() 10 个 fd 的话，如果用”直接开线程去阻塞在服务器上“这种办法，那意味着就要开 10 个线程。fd_set 的标准大小是 1024，而一个 select()可以涵盖 3 个 fd_set，最坏的情况大家可以算算需要开多少线程才能满足要求？

所以这个时候就可以看到 select() 是怎么用 MsgDeliverEvent() 来实现的了。大约是这样:

```
int select( int width,fd set * readfds, fd set * writefds,
            fd set * exceptfds, struct timeval * timeout )
{
        lonotify_t ion;
        structure sigevent evt;

        msg = &ion.msgi ;
        msg->type = _IO_NOTIFY;
        msg->combine_len = sizeof msgi;
        msg->action = _NOTIFY_ACTION_POLLARM;

        /* for each fd, send a _IO_NOTIFY msg to their server */
        for fd in readfds
        {
                SIGEV_SIGNAL_VALUE_INIT(&;msg->event, SIGSELECT, fd);
                msg->flags |= _NOTIFY_COND_INPUT;
                MsgSend(fd, msg, sizeof(*msg), &ion.o, sizeof ion.o)
        }

        for fd in writefds
        {
                ...
        }

        For fd in exceptfds
        {
                ...
        }

        /* all messages send to different servers already, waitfor the reply */
        sigemptyset(&set);
        sigaddset(&set, SIGSELECT);
        if (sigtimedwait(&set, &info, &timeout) == -1)
          return -1;
        fd = info.si_value & _NOTIFY_DATA_MASK;
        if (info.si_value & _NOTIFY_COND_INPUT)
                print( "fd %d is ready to input!\n" , fd);
        if (info.si_value & _NOTIFY_COND_OUTPUT)
                print( "fd %d is ready to output\n" , fd);
        if (info.si_value & _NOTIFY_COND_OBAND)
                print( "fd %d is excepted\n" , fd);
        return 1;
}
```

上面并不是真实的代码，但你可以看到大致是怎么工作的。对于每一个 fd，我们会有一个 SIGEV_SIGNAL_VALUE 事件，fd 作为事件的值；然后整个事件，加上一些 flag，通过 struct _io_notify 被发送到了服务器端。

当所有的服务器都被通知了以后，程序进入一个 sigtimedwait() 阻塞状态，等各个服务器反应。

服务器端，当条件满足后，会 MsgDeliverEvent()那个对应的 SIGEV_SIGNAL_VALUE。其实就是一个 SIGSELECT 发送给客户端。客户端的 sigtimedwait() 就会捕捉到这个信号，然后根据带回来的值，来决定哪个 fd 发生了什么。

select()的性能

上述的这种方案，并不是实际上 QNX 实装的方案。可以看出来这个方案的性能其实是比较差的。每次只有一个 fd 会被唤醒，即使是 httpd 服务器这种有大量 socket 连接（大量 fd），但其实服务器是同一个的时候，传统的操作系统上一次 select()可以有多个 fd 被唤醒。所以在实装上 QNX 实现了 poll()/epoll()，而 select()，则在内部被转化为 poll()实现了。

poll()的实现就不在这里展开了，但基本还是跟上面的先传递一个事件，然后等 MsgDelieverEvent()把事件发回来的方案相似，只是对于指向同一服务器的 fd 做了合并优化，这样就有可能同时得到多个 fd 被唤醒。