

# surprisingly slow accept/connect cycle time

**Source:** <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-05/5167.html>

---

**From:** Clifford T. Matthews (*ctm\_at\_ardi.com*)

**Date:** 05/24/05

Date: Tue, 24 May 2005 14:54:05 -0600

To: linux-kernel@vger.kernel.org

While writing some test code, I was surprised to find a couple processes running very slowly. The attached program illustrates this. The program forks and the child attempts to accept 1000 connections. The parent attempts to connect 1000 times. This often takes minutes to run, on 2.4 kernels and 2.6 kernels (including 2.6.12-rc4). If I run this program with "-y", the program will call sched\_yield() after each connect and then the program will take less than a second to run.

I was surprised that the sched\_yield() was needed, because I naively assumed that the scheduler would switch back and forth between processes as they went from runnable to awaiting IO and back.

As such, perhaps the need for a sched\_yield suggests there is a place where scheduling could be improved, but I recognize this is a toy load that may not occur in the real world. My test code runs fine with the sched\_yield, so I'm only pointing this—surprising to me—behavior on the off chance it's useful to someone.

I skim LKML regularly, but very quickly. e-mail is welcome.

—Cliff Matthews <ctm@ardi.com>

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <string.h>
#include <unistd.h>
#include <sys/poll.h>
#include <errno.h>
#include <sched.h>
#include <stdbool.h>
```

```
/*
```

## Linux–Kernel: surprisingly slow accept/connect cycle time

```
* This program can take an amazingly long time to run on a variety of
* Linux kernels. All it does, however, is fork and have the child be
* a server that accepts a bunch of connections, while the parent does
* a bunch of connects.
*
* Adding a sched_yield() after the connect makes the program run quickly.
*
* [ctm@canceled patchbox]$ time ./connect_accept_slowdown
*
* real 3m33.004s
* user 0m0.002s
* sys 0m0.017s
* [ctm@canceled patchbox]$ time ./connect_accept_slowdown -y
*
* real 0m0.064s
* user 0m0.004s
* sys 0m0.039s
*/
```

```
enum
{
    N_CONNECTIONS = 1000,
    PORT = 5065,
};

#define DIE_IF(cond) \
if (cond) \
{ \
    char err[127]; \
    int save_errno; \
    char *mess; \
\
    save_errno = errno; \
    mess = strerror_r (save_errno, err, sizeof err); \
    fprintf (stderr, "%s: %s (%s)\n", __FUNCTION__, #cond, mess); \
    abort (); \
}

static int
socket_or_die (int domain, int type, int protocol)
{
    int retval;

    retval = socket (domain, type, protocol);
    DIE_IF (retval < 0);

    return retval;
}
```

## Linux–Kernel: surprisingly slow accept/connect cycle time

```
static void
connect_or_die (int fd, const struct sockaddr *addrp, socklen_t len)
{
    int sysret;

    do
    {
        sysret = connect (fd, addrp, len);
        DIE_IF (sysret != 0 && errno != ECONNREFUSED);
        if (sysret != 0)
            printf ("retrying\n");
    }
    while (sysret != 0);
}
```

```
static void
bind_or_die (int fd, struct sockaddr *addrp, socklen_t len)
{
    DIE_IF (bind (fd, addrp, len) != 0);
}
```

```
static void
make_address (struct sockaddr *addrp, int port)
{
    struct sockaddr_in in_addr;

    DIE_IF (sizeof in_addr > sizeof *addrp);
    in_addr.sin_family = AF_INET;
    in_addr.sin_port = htons (port);
    in_addr.sin_addr.s_addr = htonl (INADDR_LOOPBACK);
    memcpy (addrp, &in_addr, sizeof in_addr);
}
```

```
static void
listen_or_die (int fd, int backlog)
{
    DIE_IF (listen (fd, backlog) != 0);
}
```

```
static int
accept_or_die (int fd, struct sockaddr *addr, socklen_t *addrlen)
{
    int retval;

    retval = accept (fd, addr, addrlen);
    DIE_IF (retval < 0);

    return retval;
}
```

surprisingly slow accept/connect cycle time

## Linux–Kernel: surprisingly slow accept/connect cycle time

```
}

static void
setsockopt_or_die (int s, int level, int optname, const void *optval,
                  socklen_t optlen)
{
    DIE_IF (setsockopt (s, level, optname, optval, optlen) != 0);
}

static pid_t
fork_or_die (void)
{
    pid_t retval;

    retval = fork ();
    DIE_IF (retval == -1);

    return retval;
}

#define NELEM(x) (sizeof(x) / sizeof ((x)[0]))

static void
do_parent (const struct sockaddr *addrp, bool yield_p)
{
    int sockets[N_CONNECTIONS];
    int i;

    for (i = 0; i < (int) NELEM (sockets); ++i)
    {
        sockets[i] = socket_or_die (PF_INET, SOCK_STREAM, 0);
        connect_or_die (sockets[i], addrp, sizeof *addrp);
        if (yield_p)
            sched_yield ();
    }
}

static void
do_child (struct sockaddr addr)
{
    int accept_fd;
    int sockets[N_CONNECTIONS];
    int on;
    int i;

    accept_fd = socket_or_die (PF_INET, SOCK_STREAM, 0);
    on = 1;
    setsockopt_or_die (accept_fd, SOL_SOCKET, SO_REUSEADDR, &on, sizeof on);
```

surprisingly slow accept/connect cycle time

## Linux-Kernel: surprisingly slow accept/connect cycle time

```
bind_or_die (accept_fd, &addr, sizeof addr);
listen_or_die (accept_fd, 10);
for (i = 0; i < (int) NELEM (sockets); ++i)
{
    socklen_t addr_len;

    sockets[i] = accept_or_die (accept_fd, &addr, &addr_len);
}

}

int
main (int argc, const char *argv[])
{
    struct sockaddr addr;
    pid_t child;
    bool yield_p;

    make_address (&addr, PORT);
    yield_p = (argc >= 2 && (strcmp (argv[1], "-y") == 0 ||
        strcmp (argv[1], "--yield") == 0));

    if ((child = fork_or_die ()) != 0)
        do_parent (&addr, yield_p);
    else
        do_child (addr);

    return 0;
}
-
```

To unsubscribe from this list: send the line "unsubscribe linux-kernel" in the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>