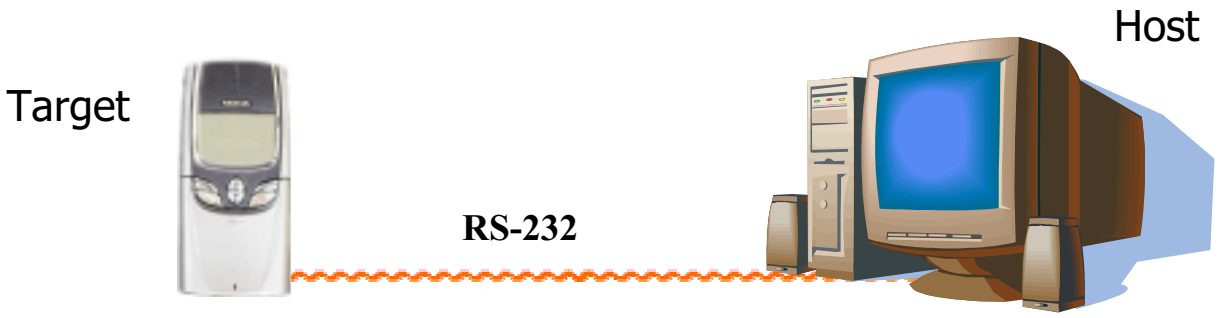
# Tracing Linux Networking Through Remote Kernel Debug

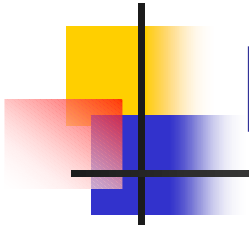
Written by Tsai Ping Tsai
ie855160@csie.fju.edu.tw
High Speed Lab 2000/08/14

# What situation we use Remote debug?

- debug a system kernel
- a limited resource target
  - low memory
  - no monitor

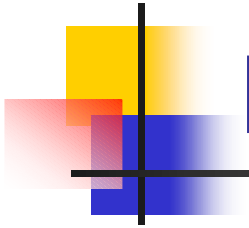



圖一：遠端除錯示意圖



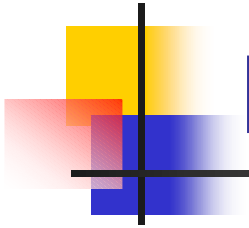
# How to use Remote Debug(cont.)

- Two way to use GDB to remote debug
  - use gdbserver
  - use stub.c
- If we want to debug an operating system, we can't use gdbserver



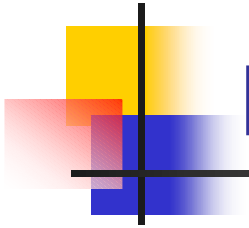
# Remote debug – Use gdbserver

- Use gdbserver
  - The gdbserver is packet with GDB but not compiler when you compiler the GDB.We must compiler gdbserver .
  - ./configure
  - cd gdb/gdbserver
  - Make
- Command
  - Target:「gdbserver /dev/com1 ping –c1 eva」
  - Host: 「target remote /dev/ttyS0」(in GDB)



# Remote debug – Use stub.c

- Use stub.c
  - Client:compiler the program with –g and link with stub.c . Then just run the program.
  - Server:target remote /dev/ttyS0
  - We must write some code for stub.c such as getchar(),putchar()…

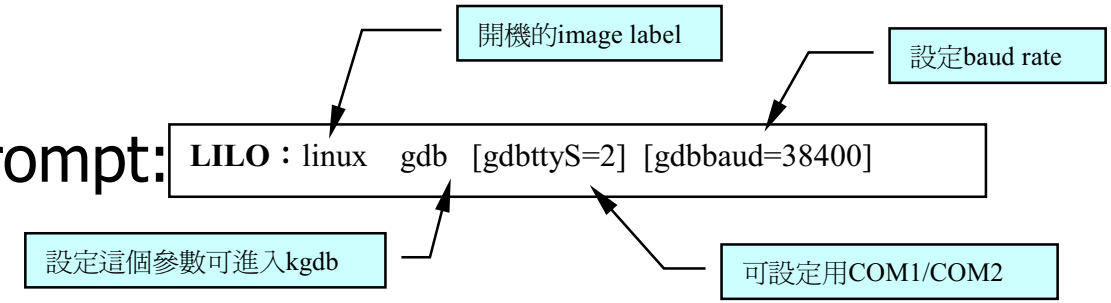


# Remote Kernel debug – Use kGDB(cont.)

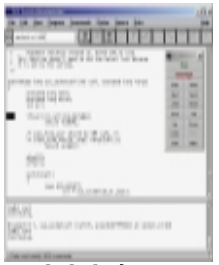
- Use kGDB
  - kGDB is a kernal patch for GDB to provide Linux to remote debug on i386 kernal.
- Step to install
  - cd /usr/src/linux
  - patch -p0 < kgdb0.2-2.2.12
  - make xconfig  (or "make menuconfig")
  - **[at "Kernel hacking" turn on "Kernel support for GDB"]**
  - make bzImage  (or as you normally do)
- Then you can use remote debug on boot time

# Remote Kernel debug – Use kGDB

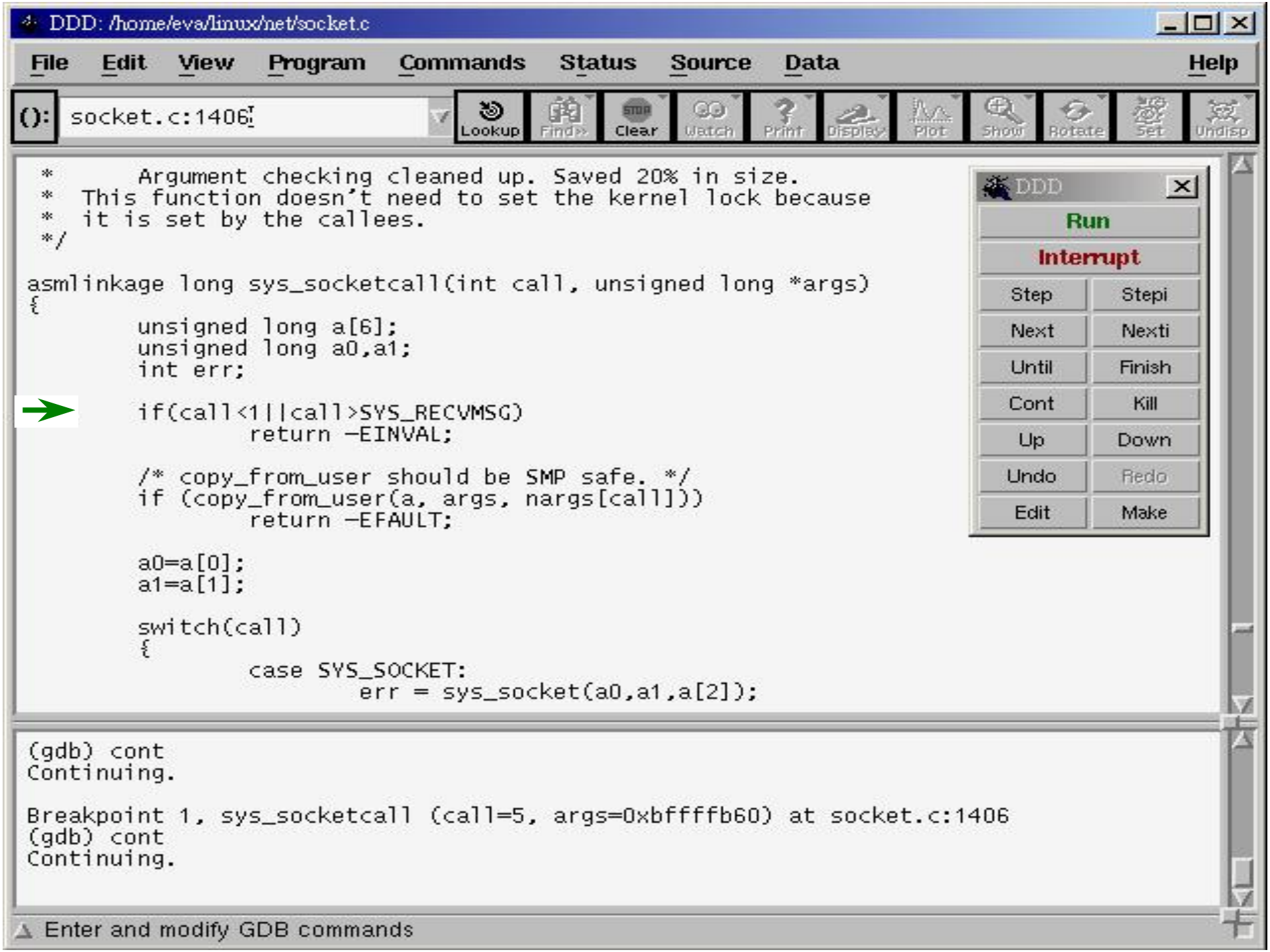
- Target(kgdb)
  - at boot prompt:



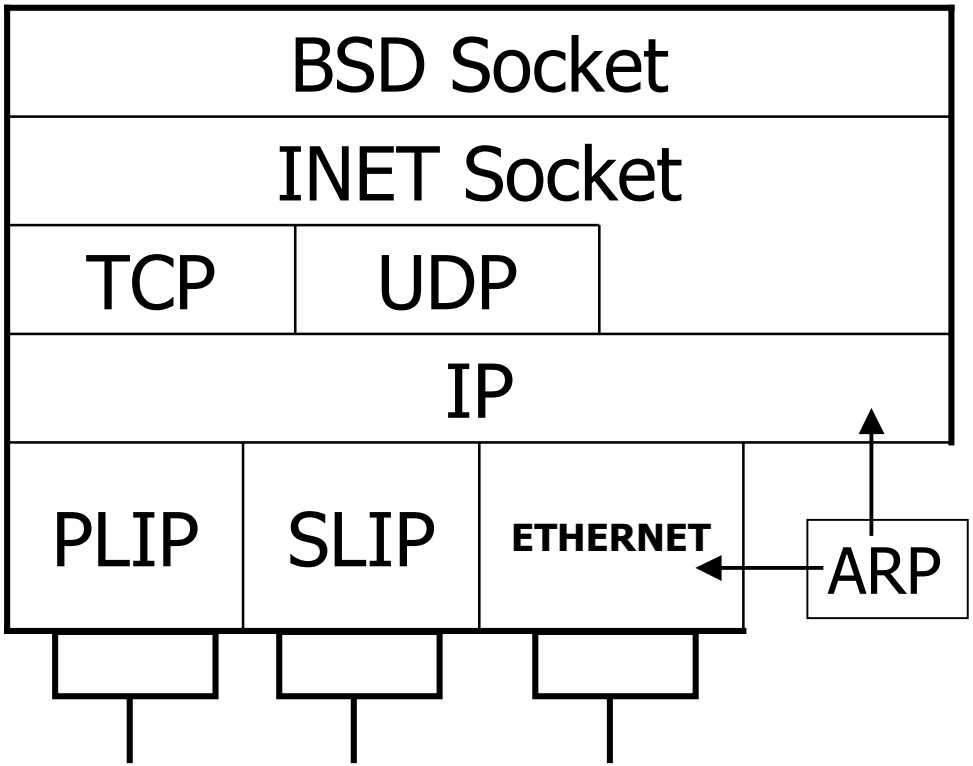

- Host(gdb/ddd)
  - Load symbol: FILE /usr/src/linux/vmlinux
  - set baud rate:set remotebaud 38400
  - change target: target remote /dev/ttyS3
  - continue booting: cont
  - interrupt target: interrupt
  - set break point: break sys_socketcall



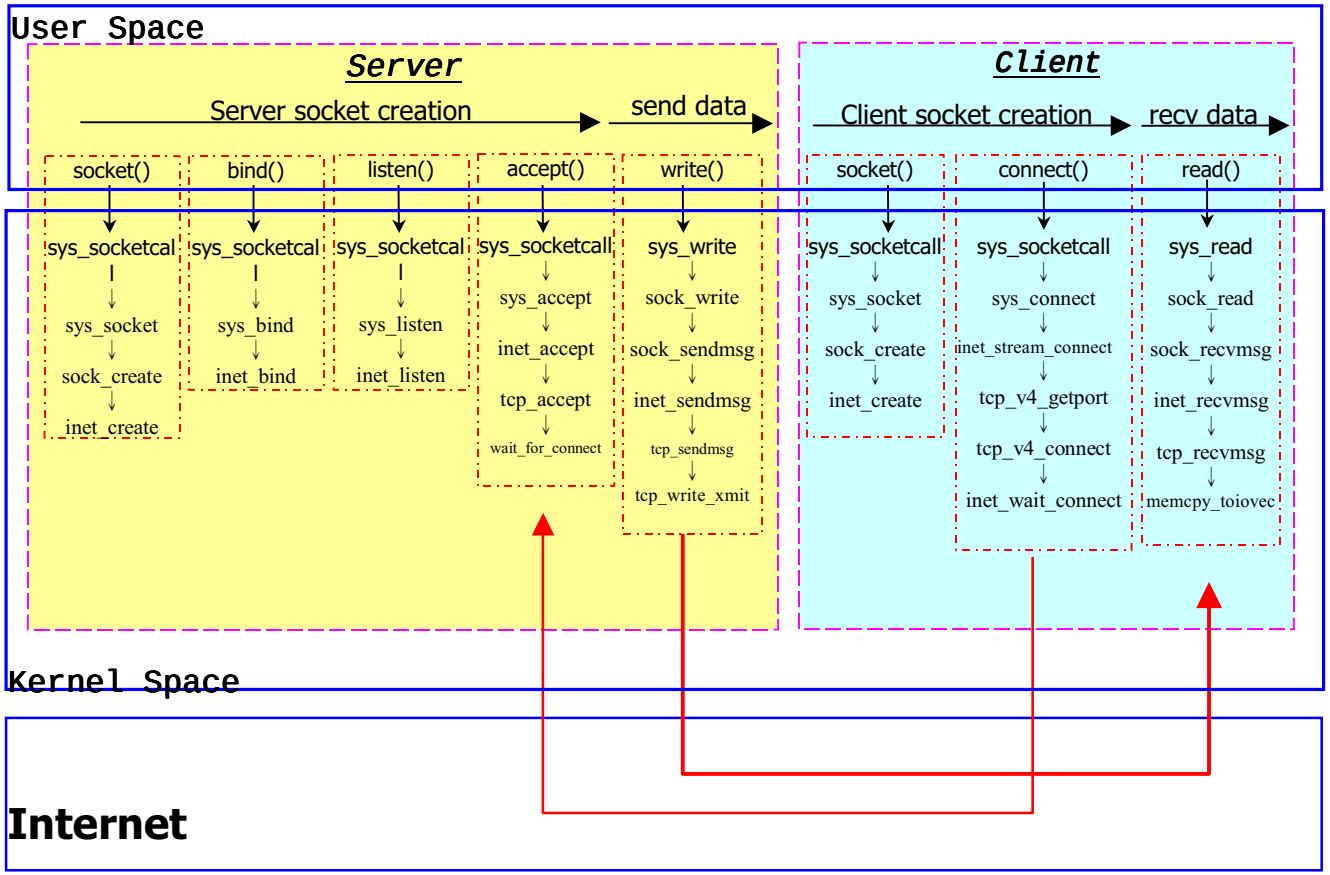
ddd介面


DDD: /home/eva/linux/net/socket.c
File Edit View Program Commands Status Source Data Help
(): socket.c:1406
Lookup Find>> Clear Watch Print Display Plot Show Rotate Set Undisp
* Argument checking cleaned up. Saved 20% in size.
* This function doesn't need to set the kernel lock because
* it is set by the callees.
*/
asmlinkage long sys_socketcall(int call, unsigned long *args)
{
unsigned long a[6];
unsigned long a0,a1;
int err;
if(call<1||call>SYS_RECVMSG)
return -EINVAL;
/* copy_from_user should be SMP safe. */
if (copy_from_user(a, args, nargs[call]))
return -EFAULT;
a0=a[0];
a1=a[1];
switch(call)
{
case SYS_SOCKET:
err = sys_socket(a0,a1,a[2]);
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
(gdb) cont
Continuing.
Breakpoint 1, sys_socketcall (call=5, args=0xbffffb60) at socket.c:1406
(gdb) cont
Continuing.
Enter and modify GDB commands

# Linux TCP/IP Networking Layers



# Client/Server package's flow chart




Kernel Space

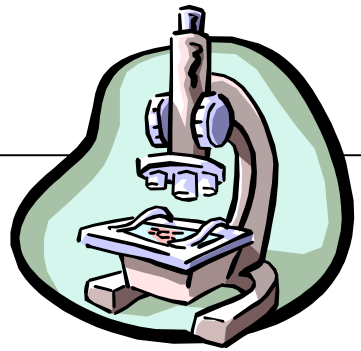
# system call -- write

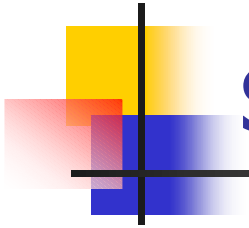
User Space

```
write(sockfd,&ch,1);  //send a character to client
```

Kernel Space

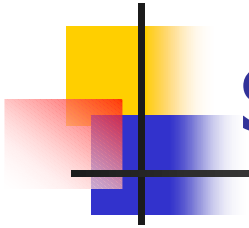
sys_write()
sock_write()
sock_sendmsg()
inet_sendmsg()
tcp_do_sendmsg()
tcp_send_skb()
tcp_write_xmit()
tcp_transmit_skb()





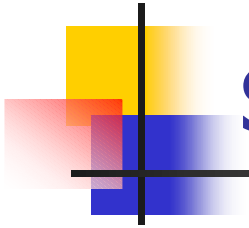
# sys_write()

```
asmlinkage ssize_t sys_write(unsigned int fd, const char * buf, size_t count)
{
    file = fget(fd);
    if (file) {
        if (file->f_mode & FMODE_WRITE) {
            struct inode *inode = file->f_dentry->d_inode;
            ret = locks_verify_area(FLOCK_VERIFY_WRITE, inode, file,
                    file->f_pos, count);
            if (!ret) {
                    ssize_t (*write)(struct file *, const char *, size_t, loff_t *);
                    ret = -EINVAL;
            if (file->f_op && (write = file->f_op->write) != NULL)
                            ret = write(file, buf, count, &file->f_pos);
```



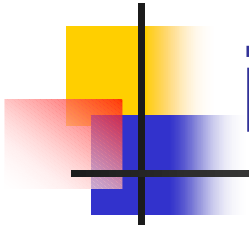
# sock_write()

```
static ssize_t sock_write(struct file *file, const char *ubuf,
{
    sock = socki_lookup(file->f_dentry->d_inode);

    msg.msg_name=NULL;
    msg.msg_namelen=0;
    msg.msg_iov=&iov;
    msg.msg_iovlen=1;
    msg.msg_control=NULL;
    msg.msg_controllen=0;
    msg.msg_flags=!(file->f_flags & O_NONBLOCK) ? 0 : MSG_DONTWAIT;
    return sock_sendmsg(sock, &msg, size);
}
```



# sock_sendmsg()

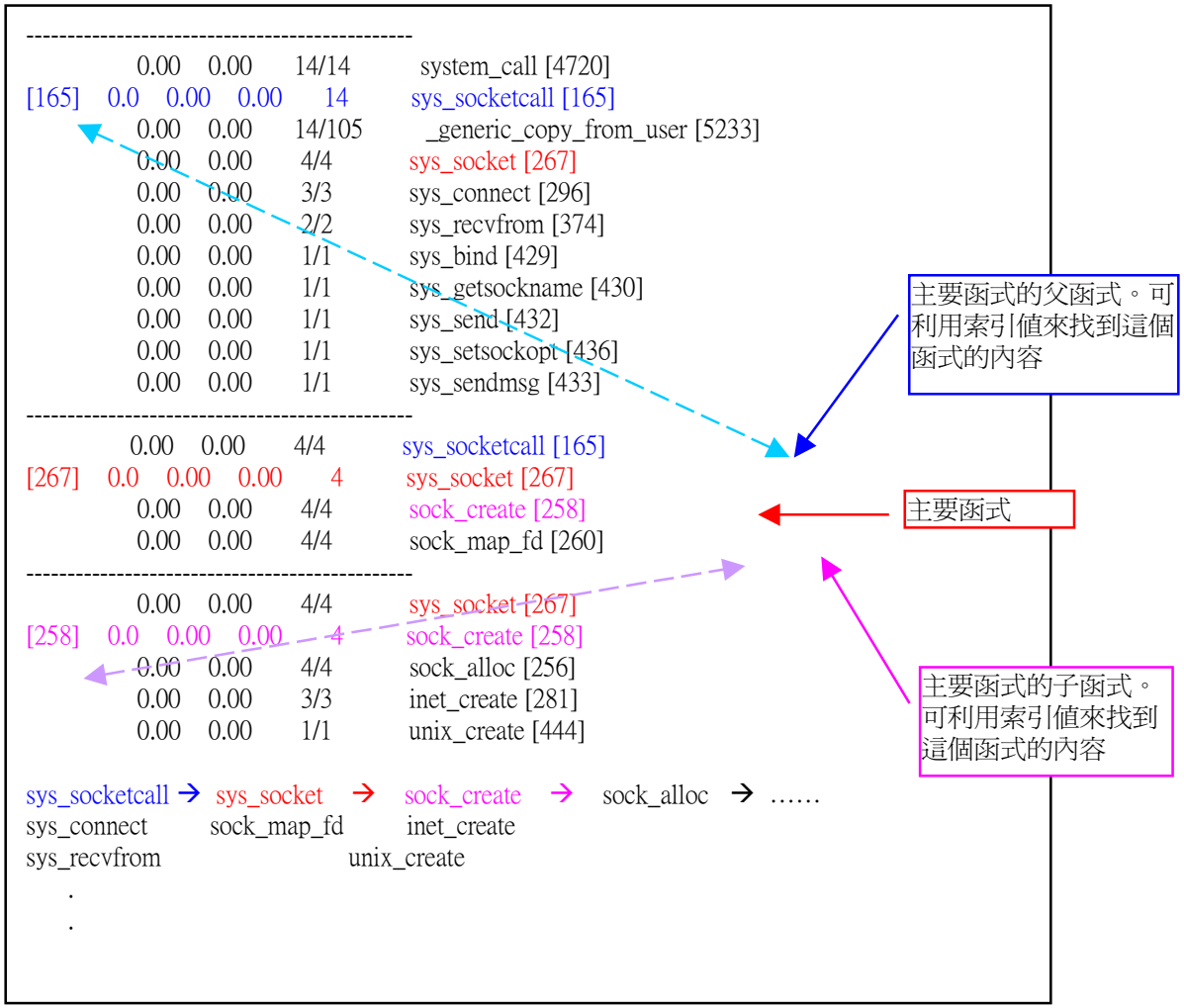
```
int sock_sendmsg(struct socket *sock, struct msghdr *msg, int size)
{
    int err;
    struct scm_cookie scm;

    err = scm_send(sock, msg, &scm);
    if (err >= 0) {
        err = sock->ops->sendmsg(sock, msg, size, &scm);
        scm_destroy(&scm);
    }
    return err;
}
```

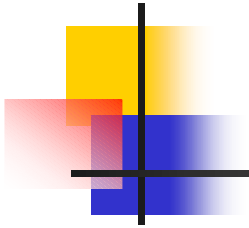


# inet_sendmsg()

```
int inet_sendmsg(struct socket *sock, struct msghdr *msg, int size,
                 struct scm_cookie *scm)
{
    struct sock *sk = sock->sk;

    /* We may need to bind the socket. */
    if (sk->num==0 && inet_autobind(sk) != 0)
        return -EAGAIN;

    return sk->prot->sendmsg(sk, msg, size);
}
```

# KernProf's Report





# 時間到了！