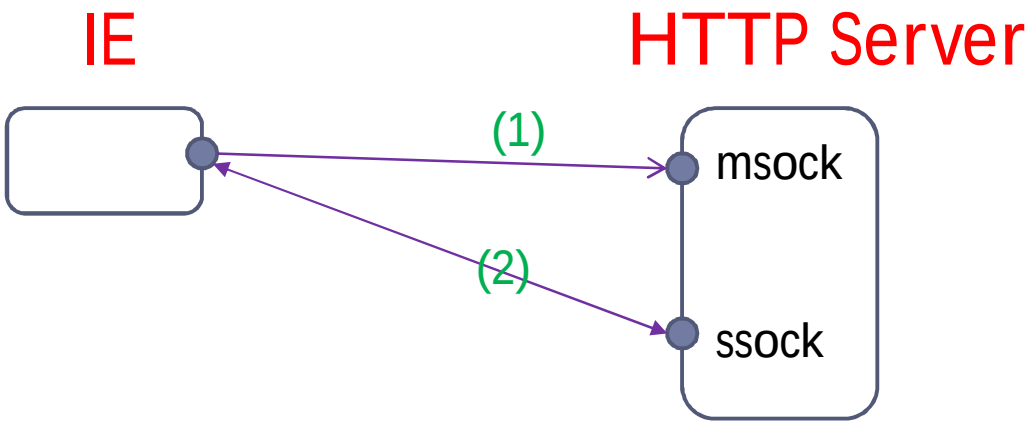
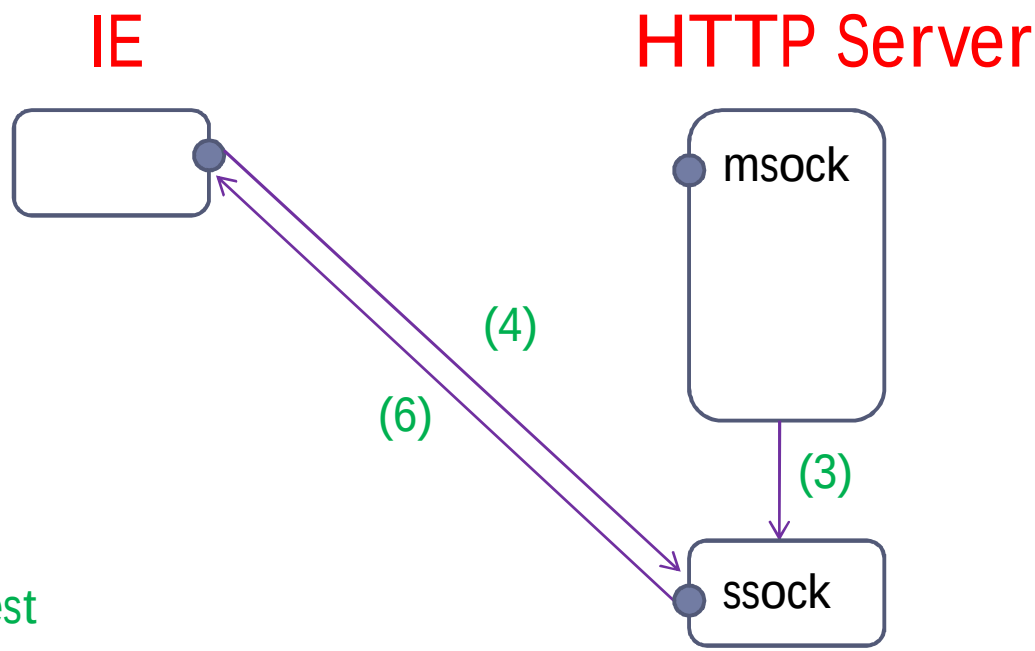
HTTP

Overview



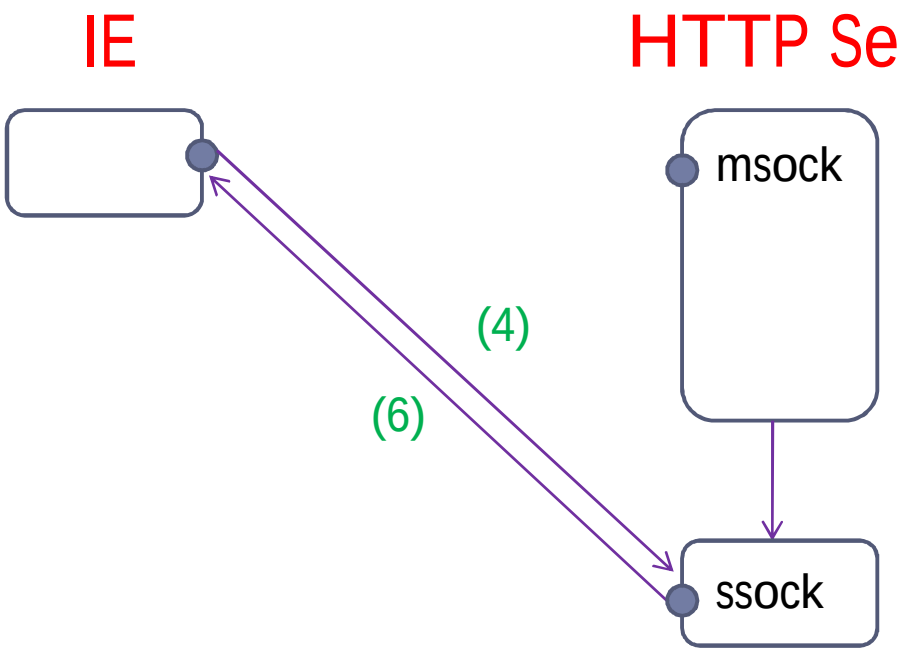
(1) : Connection Request (default : port 80)
(2) : Connection Establish

Overview



(3) : fork
(4) : HTTP request
(6) : HTTP response

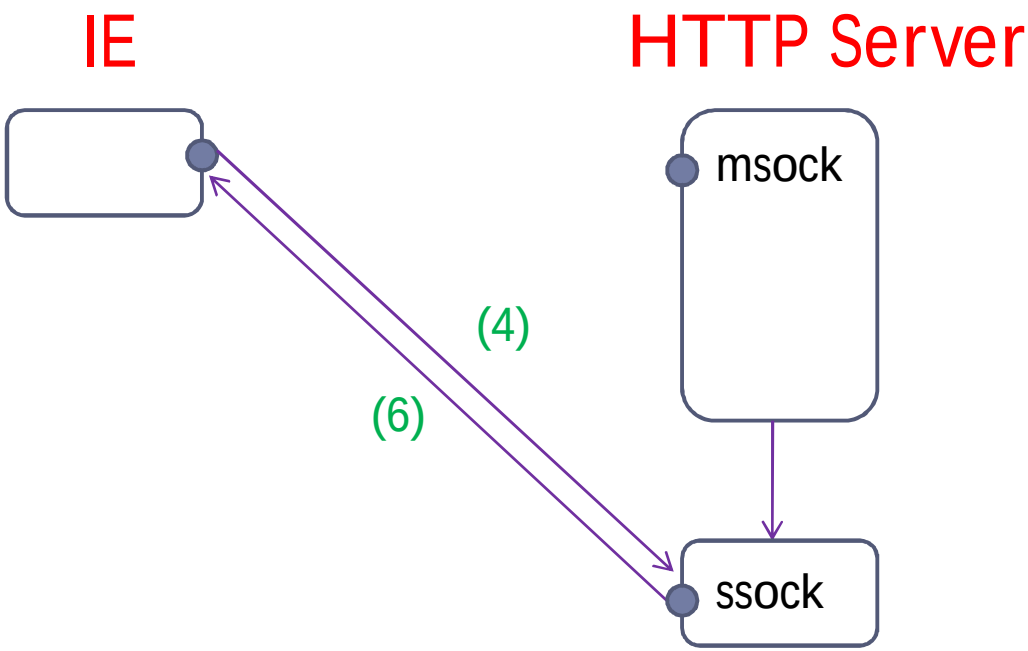
HTTP request (4)



GET http://java.csie/test.cgi?a=aa&b=bb HTTP/1.1
(header)
(blank line)
(data)

Content-Type: text/html
Cache-Control: max-age=0
…

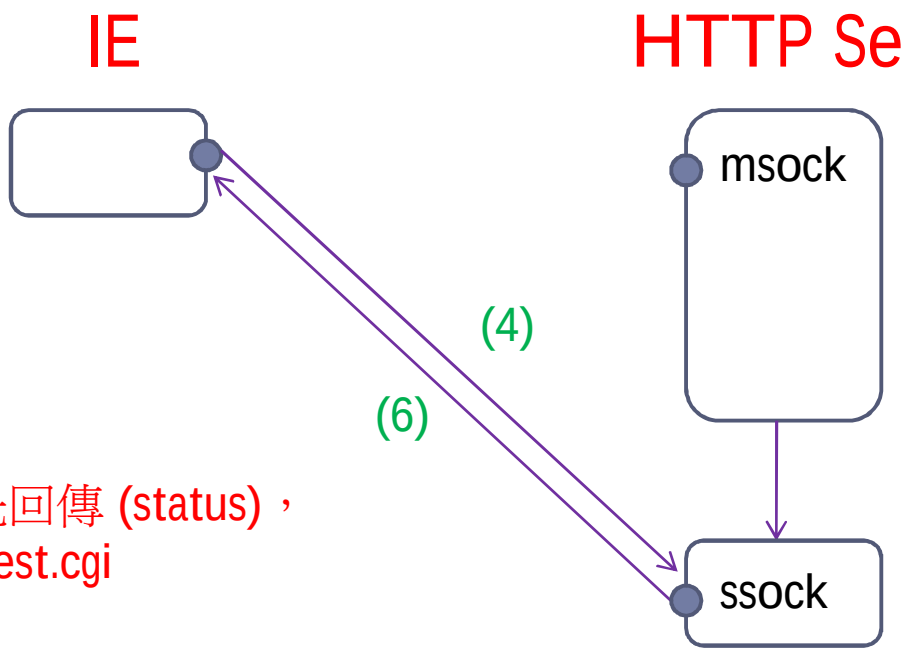
Process request



GET http://java.csie/test.cgi?a=aa&b=bb HTTP/1.1

HTTP Server 須 parse request，
然後將 a=aa&b=bb 放入環境變數 QUERY_STRING 中 (GET method)

HTTP response (6)



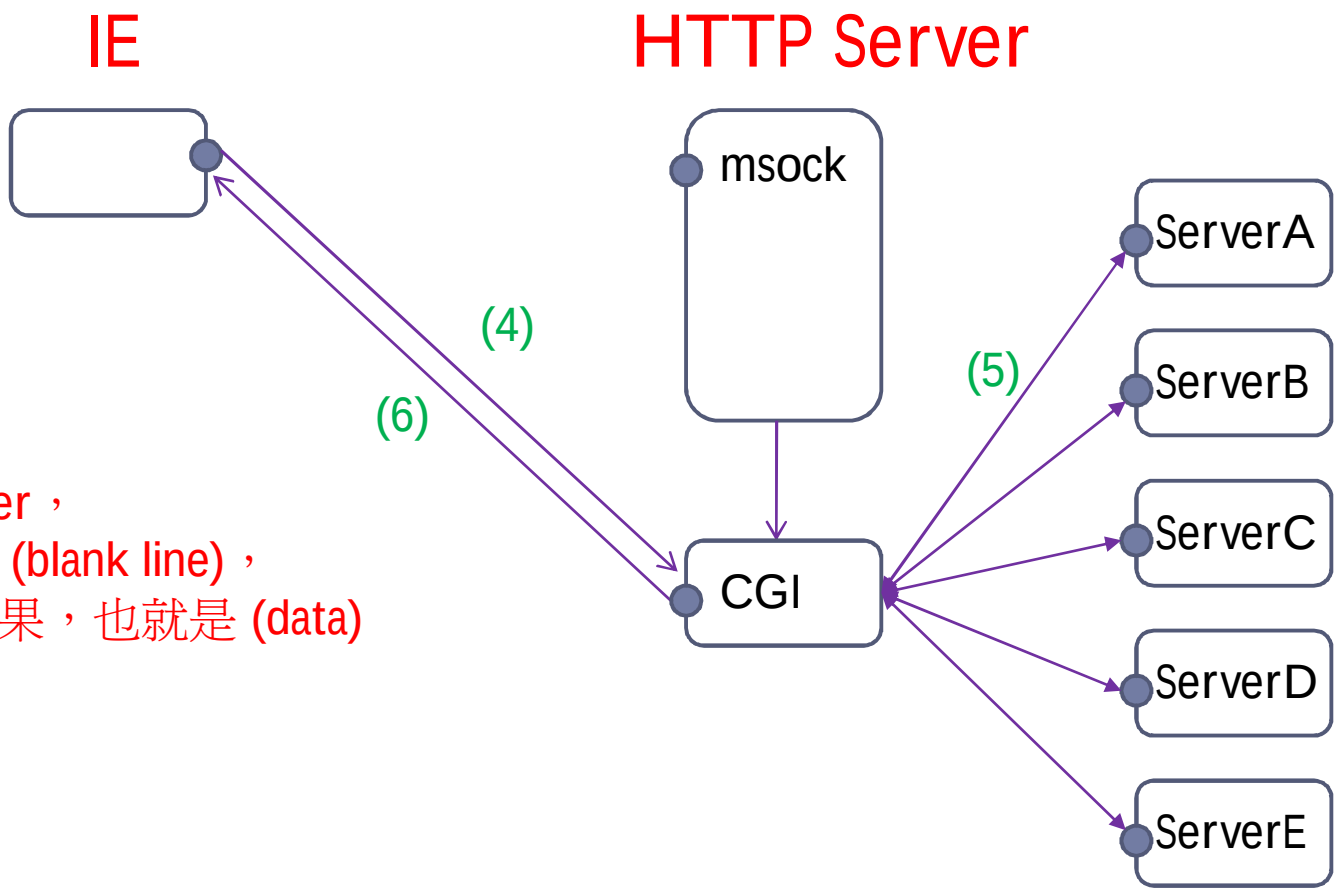
接著 HTTP Server 先回傳 (status)，
然後再 fork & exec test.cgi

(status)
(header)
(blank line)
(data)

HTTP/1.1 200 OK
HTTP/1.1 403 Forbidden
HTTP/1.1 404 Not Found

Status Code Definitions
http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

HTTP response (5)

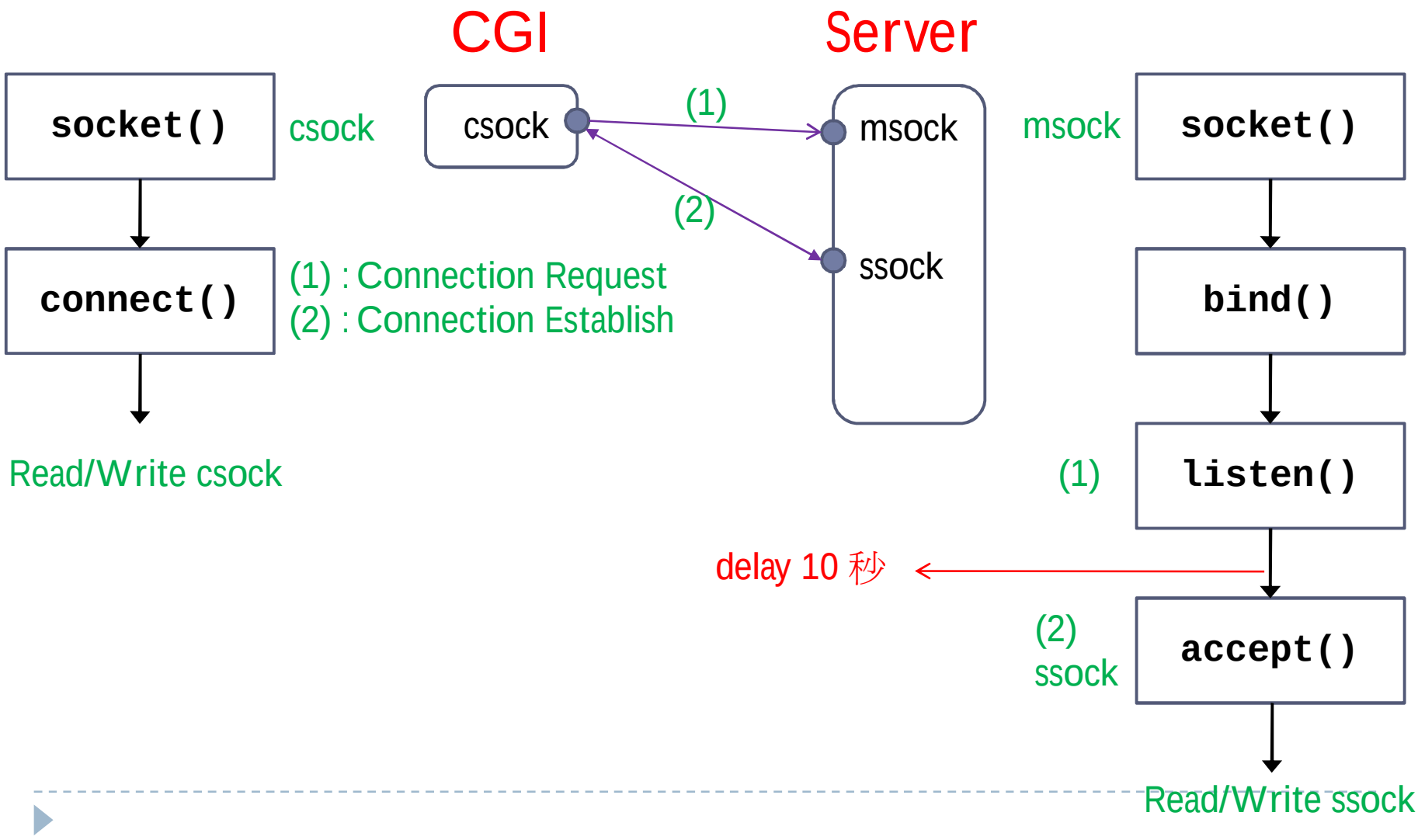


CGI 程式：
connect 到 5 台 server，
然後先回傳 (header) (blank line)，
接著再回傳 (5) 的結果，也就是 (data)

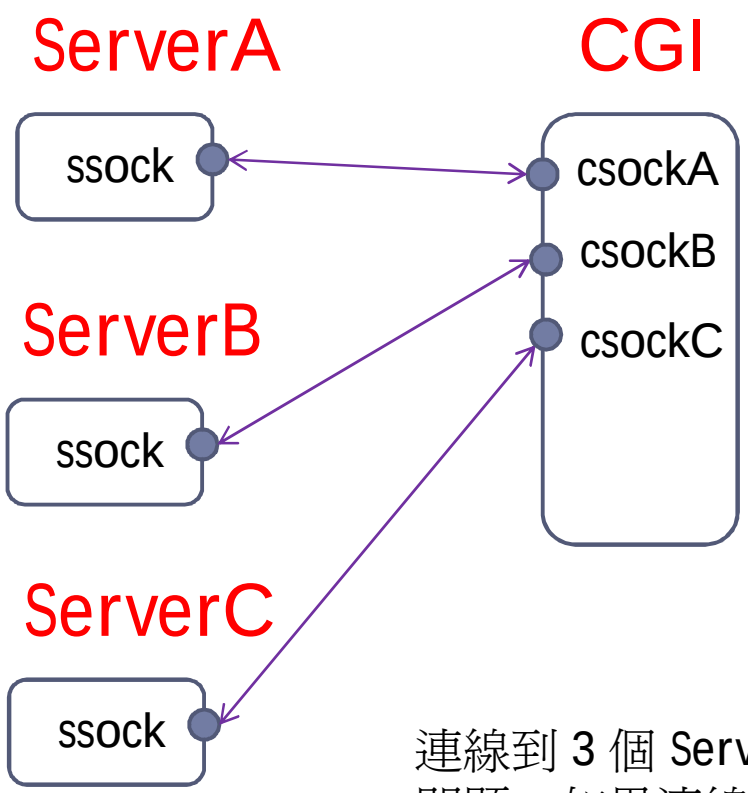
(status)
(header)
(blank line)
(data)

Single Process Model - Concurrent Client

Connect to One Server



Connect to Many Server




```
…
connect(csockA);
connect(csockB);
connect(csockC);

if (ServerA is connected)
        write_request(csockA);
if (ServerB is connected)
        write_request(csockB);
if (ServerC is connected)
        write_request(csockC);
```

連線到 3 個 Server，
問題：如果連線到很慢的 ServerB (or delay 10 秒)
à ServerB 的 connect 會等很久
à 結果造成其它正常 ServerA 的 request 和 ServerC 的 connect/request 也一起等很久

Concurrent Client

- 解決 connect 和 Read/Write 會 block 的問題
- 解決方法
  - non-blocking I/O
    - fcntl
  - select

select

- int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);
- nfds
  - select 監聽的 fd 最大值
    - 一般爲三組 fd_set 中最大值加一 或 FD_SET 的 size
- readfds
  - select 監聽的可讀 fd 集合
- writefds
  - select 監聽的可寫 fd 集合
- exceptfds
  - select 監聽的異常 fd 集合
- timeout
  - select() 的 timeout 時間
    - NULL：不設定 timeout

NON-Blocking I/O

- int fcntl(int fd, int cmd, ...);
- E.g.
  - int flag = fcntl(sockfd, F_GETFL, 0);
  - fcntl(sockfd, F_SETFL, flags | O_NONBLOCK);

Concurrent Client Overview

```
int flag = fcntl(csockA, F_GETFL, 0);
fcntl(csockA, F_SETFL, flags | O_NONBLOCK);

if ( (n = connect (csockA, …)) < 0) {
    if (errno != EINPROGRESS) return (-1);
}

fd_set rfds; /* readable file descriptors*/
fd_set wfds; /* writable file descriptors*/
fd_set rs; /* active file descriptors*/
fd_set ws; /* active file descriptors*/

int conn = 1; /* 連線數量 */
nfds = FD_SETSIZE;
FD_ZERO(&rfds); FD_ZERO(&wfds); FD_ZERO(&rs); FD_ZERO(&ws);
FD_SET(csockA, &rs);
FD_SET(csockA, &ws);
rfds = rs; wfds = ws;
```

Concurrent Client Overview

```
#define F_CONNECTING 0
#define F_READING 1
#define F_WRITING 2
#define F_DONE 3
```

```
int statusA = F_CONNECTING;

while (conn > 0) {
    memcpy(&rfds, &rs, sizeof(rfds)); memcpy(&wfds, &ws, sizeof(wfds));

    if ( select(nfds, &rfds, &wfds, (fd_set*)0, (struct timeval*)0) < 0 ) errexit();

    if (statusA == F_CONNECTING &&
        (FD_ISSET(csockA, &rfds) || FD_ISSET(csockA, &wfds)))
    {
        if (getsockopt(csockA, SOL_SOCKET, SO_ERROR, &error, &n) < 0 ||
            error != 0) {
            // non-blocking connect failed
            return (-1);
        }
        statusA = F_WRITING;
        FD_CLR(csockA, &rs);
    }
```

Concurrent Client Overview

```
#define F_CONNECTING 0
#define F_READING 1
#define F_WRITING 2
#define F_DONE 3
```

```
    else if (statusA == F_WRITING && FD_ISSET(csockA, &wfds) ) {
        n = write(…);  NeedWrite -= n;
        if (n <= 0 || NeedWrite <= 0) {
            // write finished
            FD_CLR(csockA, &ws);
            statusA = F_READING;
            FD_SET(csockA, &rs);
        }
    }
    else if (statusA == F_READING && FD_ISSET(csockA, &rfds) ) {
        n = read(…);
        if (n <= 0) {
            // read finished
            FD_CLR(csockA, &rs);
            statusA = F_DONE ;
            conn--;
        }
    }
}
```