Network Programming: Ch. 16: Nonblocking I/O

Li-Hsing Yen
NYCU

Ver. 1.0.0

Ch. 16: Nonblocking I/O

- Nonblocking reads and writes
- Buffers enabling overlapped nonblocking I/O
- Nonblocking `connect`

Socket Calls That May Block

- By default, sockets are blocking
- Four categories of blocking socket calls
  - Input operations (`read`, `readv`, `recv`, `recvfrom`, `recvmg`)
  - Output operations (`write`, `writev`, `send`, `sendto`, `sendmsg`)
  - Accepting incoming connections (`accept`)
  - Initiating outgoing connections (`connect`)

第六章的str_cli

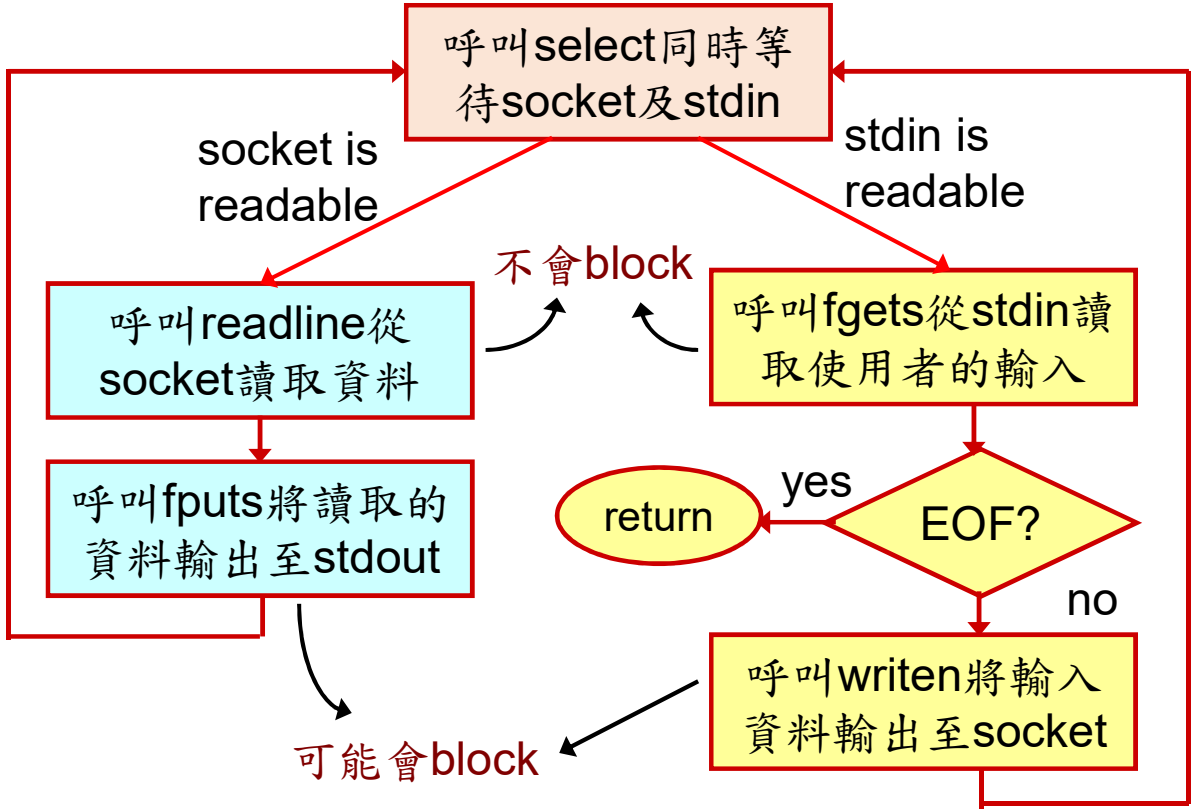
select/strcliselect01.c

```
FD_SET(fileno(fp), &rset);
FD_SET(sockfd, &rset);
maxfdp1 = max(fileno(fp), sockfd) + 1;
Select(maxfdp1, &rset, NULL, NULL, NULL);
if (FD_ISSET(sockfd, &rset)) {  /* socket is readable */
        if (Readline(sockfd, recvline, MAXLINE) == 0)
                err_quit("str_cli: server terminated prematurely");
        Fputs(recvline, stdout);
}
if (FD_ISSET(fileno(fp), &rset)) {  /* input is readable */
        if (Fgets(sendline, MAXLINE, fp) == NULL)
                return;         /* all done */
        Writen(sockfd, sendline, strlen(sendline));
}
```

can block if standard output is slow

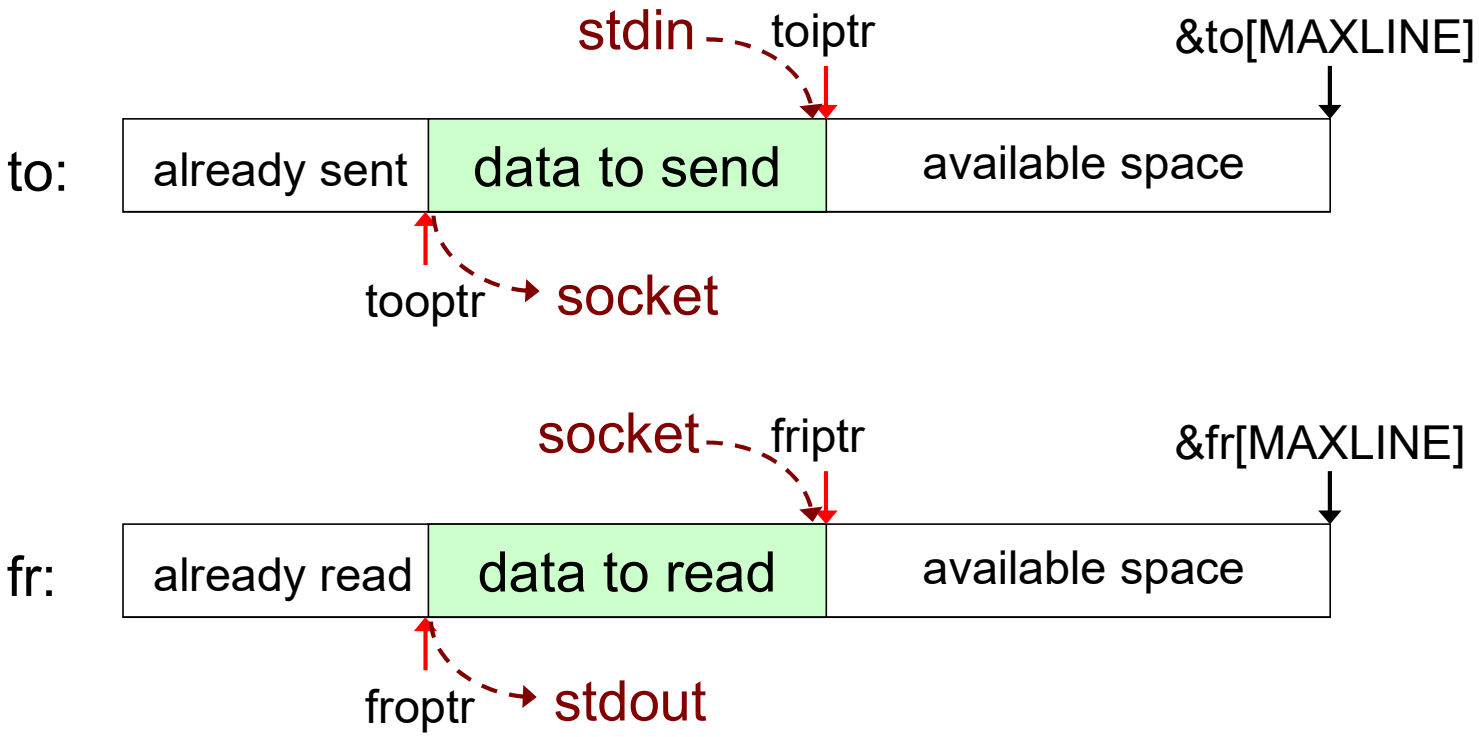
can block if the socket send buffer is full

第六章的str_cli


呼叫select同時等待socket及stdin
socket is readable
stdin is readable
不會block
呼叫readline從socket讀取資料
呼叫fgets從stdin讀取使用者的輸入
呼叫fputs將讀取的資料輸出至stdout
return
yes
EOF?
no
呼叫writen將輸入資料輸出至socket
可能會block

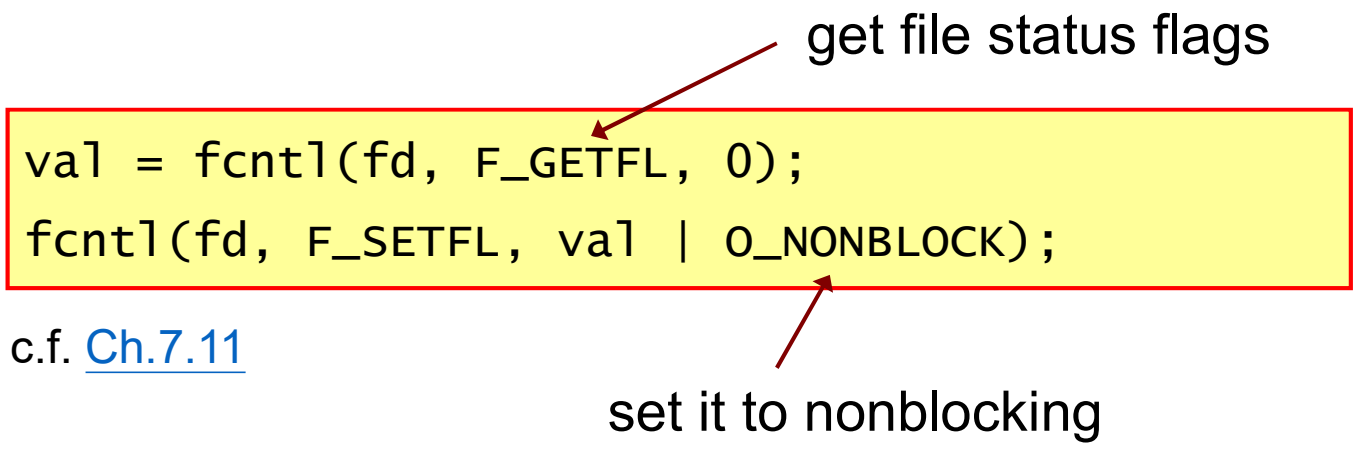
改成純Nonblocking的作法

- 由client程式自行維護兩個queues



Nonblocking Standard Input/Output

- Standard input與output分別改用`read`與`write`(原先為fgets與fputs)，配合select
- 用`fcntl`將blocking descriptor改為non-blocking



c.f. Ch.7.11

將socket, stdin, stdout改為nonblocking

nonblock/strclinonb.c

```
void str_cli(FILE *fp, int sockfd)
{
        ...
```

socket

```
    val = Fcntl(sockfd, F_GETFL, 0);
    Fcntl(sockfd, F_SETFL, val | O_NONBLOCK);
```

stdin

```
    val = Fcntl(STDIN_FILENO, F_GETFL, 0);
    Fcntl(STDIN_FILENO, F_SETFL, val | O_NONBLOCK);
```

stdout

```
    val = Fcntl(STDOUT_FILENO, F_GETFL, 0);
    Fcntl(STDOUT_FILENO, F_SETFL, val | O_NONBLOCK);
```

設定select參數

nonblock/strclinonb.c


```
maxfdp1 = max(max(STDIN_FILENO, STDOUT_FILENO), sockfd) + 1;
for ( ; ; ) {
    FD_ZERO(&rset);
    FD_ZERO(&wset);
    if (stdineof == 0 && toiptr < &to[MAXLINE])
        FD_SET(STDIN_FILENO, &rset);
    if (friptr < &fr[MAXLINE])
        FD_SET(sockfd, &rset);
    if (tooptr != toiptr)
        FD_SET(sockfd, &wset);
    if (froptr != friptr)
        FD_SET(STDOUT_FILENO, &wset);

    Select(maxfdp1, &rset, &wset, NULL, NULL);
```

client尚未關閉至server的連結

to尚有空間

fr尚有空間

to尚有資料未寫入socket

fr尚有資料未寫入stdout

test讀或寫是否ready

測試stdin是否可read

nonblock/strclinonb.c

```
if (FD_ISSET(STDIN_FILENO, &rset)) {
    if ( (n = read(STDIN_FILENO, toiptr, &to[MAXLINE] - toiptr)) < 0) {
    if (errno != EWOULDBLOCK)
        err_sys("read error on stdin");
    } else if (n == 0) {
    fprintf(stderr, "%s: EOF on stdin\n", gf_time());
    stdineof = 1;                   /* all done with stdin */
    if (tooptr == to)
        Shutdown(sockfd, SHUT_WR);/* send FIN */
    } else {
    fprintf(stderr, "%s: read %d bytes from stdin\n", gf_time(), n);
    toiptr += n;                /* # just read */
    FD_SET(sockfd, &wset);    /* try and write to socket below */
    }
}
```

最多可再讀入的byte數

從stdin讀到了EOF

to已無未送出之資料

從stdin中讀入to的資料要寫入至socket

測試socket是否可read

nonblock/strclinonb.c

```
if (FD_ISSET(sockfd, &rset)) {
    if ( (n = read(sockfd, friptr, &fr[MAXLINE] - friptr)) < 0) {
    if (errno != EWOULDBLOCK)
        err_sys("read error on socket");
    } else if (n == 0) {
    fprintf(stderr, "%s: EOF on socket\n", gf_time());
    if (stdineof)
        return;       /* normal termination */
    else
        err_quit("str_cli: server terminated prematurely");
    } else {
    fprintf(stderr, "%s: read %d bytes from socket\n", gf_time(), n);
    friptr += n;       /* # just read */
    FD_SET(STDOUT_FILENO, &wset);
} }
```

最多可再讀入的byte數

從socket讀到了EOF

client之前已關閉送資料至server的連結

早夭

從socket中讀入fr的資料要寫入至stdout

測試stdout是否可以write且fr中有資料需要write

```
if (FD_ISSET(STDOUT_FILENO, &wset) && ( (n = friptr - froptr) > 0)) {
    if ( (nwritten = write(STDOUT_FILENO, froptr, n)) < 0) {
    if (errno != EWOULDBLOCK)
        err_sys("write error to stdout");
    } else {
        fprintf(stderr, "%s: wrote %d bytes to stdout\n", gf_time(),
        nwritten);
        froptr += nwritten;        /* # just written */
        if (froptr == friptr)
        froptr = friptr = fr;
    }
}
```

fr中資料的byte數

當fr中的資料已全部被寫出至stdout

將input及output指標歸零

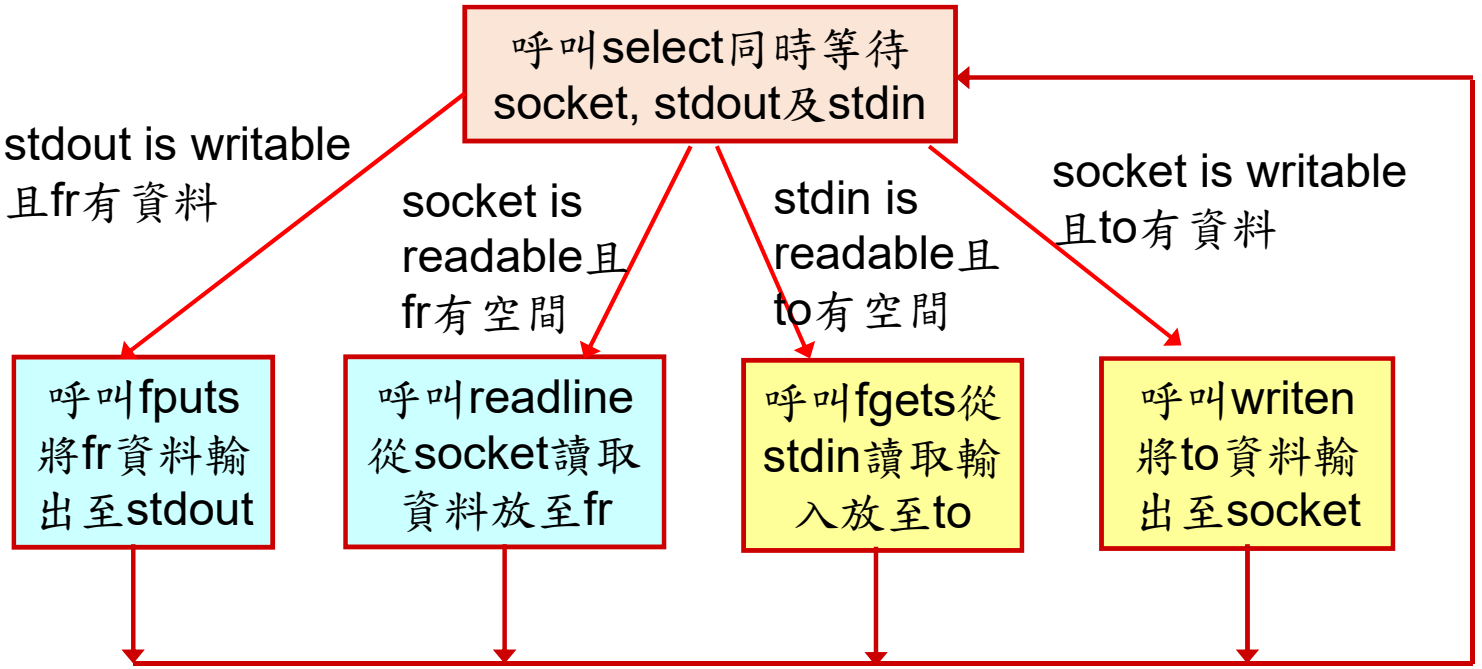
測試sockfd是否可以write且to中有資料需要write

```
if (FD_ISSET(sockfd, &wset) && ( (n = toiptr - tooptr) > 0)) {
    if ( (nwritten = write(sockfd, tooptr, n)) < 0) {
        if (errno != EWOULDBLOCK)
            err_sys("write error to socket");
    } else {
        fprintf(stderr, "%s: wrote %d bytes to socket\n",
                        gf_time(), nwritten);
        tooptr += nwritten;  /* # just written */
        if (tooptr == toiptr) {
            toiptr = tooptr = to;
            if (stdineof)
                Shutdown(sockfd, SHUT_WR);
        }
    }
}
```

to中資料的byte數 >0 則需要write

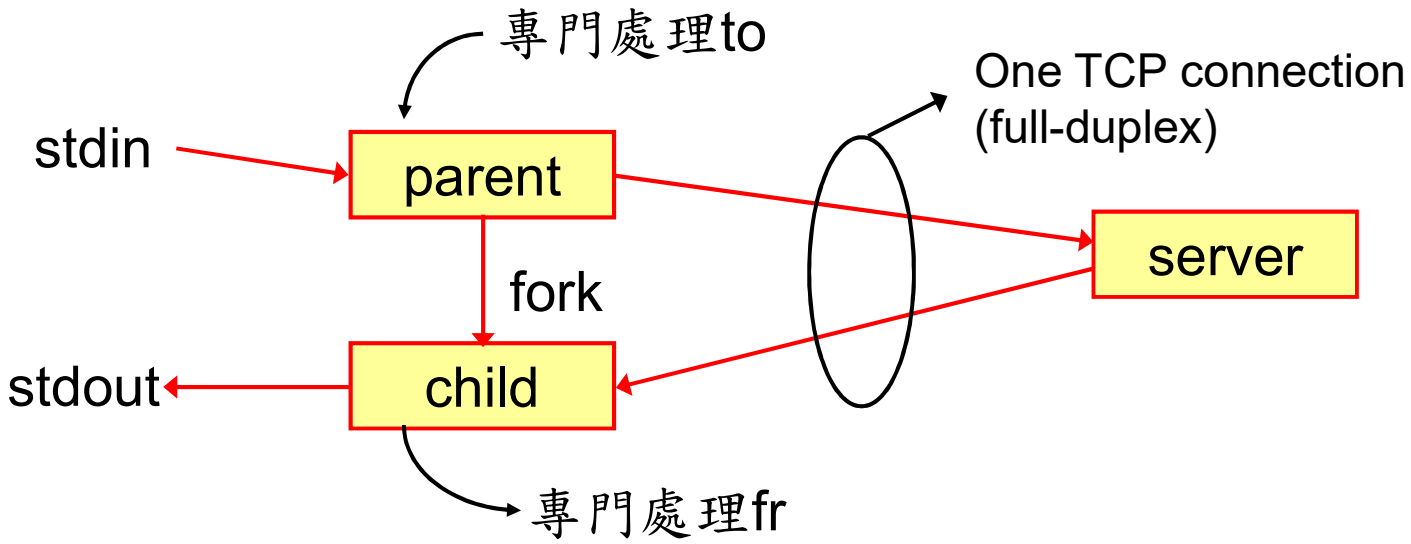
當to中的資料已全部被寫出至socket

純Nonblocking的`str_cli`



A Simpler Version of `str_cli`

- It will be simpler to split the application into processes or threads



Two-Process Approach

- The parent and child are sharing the same socket descriptor
- Each process reads data from one input stream and writes to one output stream
- no need for nonblocking I/O in each process
  - If there is no data to read from the input stream, there is nothing to write to the corresponding output stream
- 但兩個process先結束者要讓對方也可結束

```
void str_cli(FILE *fp, int sockfd)
{
        pid_t    pid;
        char     sendline[MAXLINE], recvline[MAXLINE];

        if ( (pid = Fork()) == 0) {
                while (Readline(sockfd, recvline, MAXLINE) > 0)
                        Fputs(recvline, stdout);
                kill(getppid(), SIGTERM);
                exit(0);
        }
        while (Fgets(sendline, MAXLINE, fp) != NULL)
                Writen(sockfd, sendline, strlen(sendline));
        Shutdown(sockfd, SHUT_WR);
        pause();
        return;
}
```

blocking

child

server已關閉它至client的連結。child送給parent SIGTERM信號終止parent

Child自行結束

parent

blocking

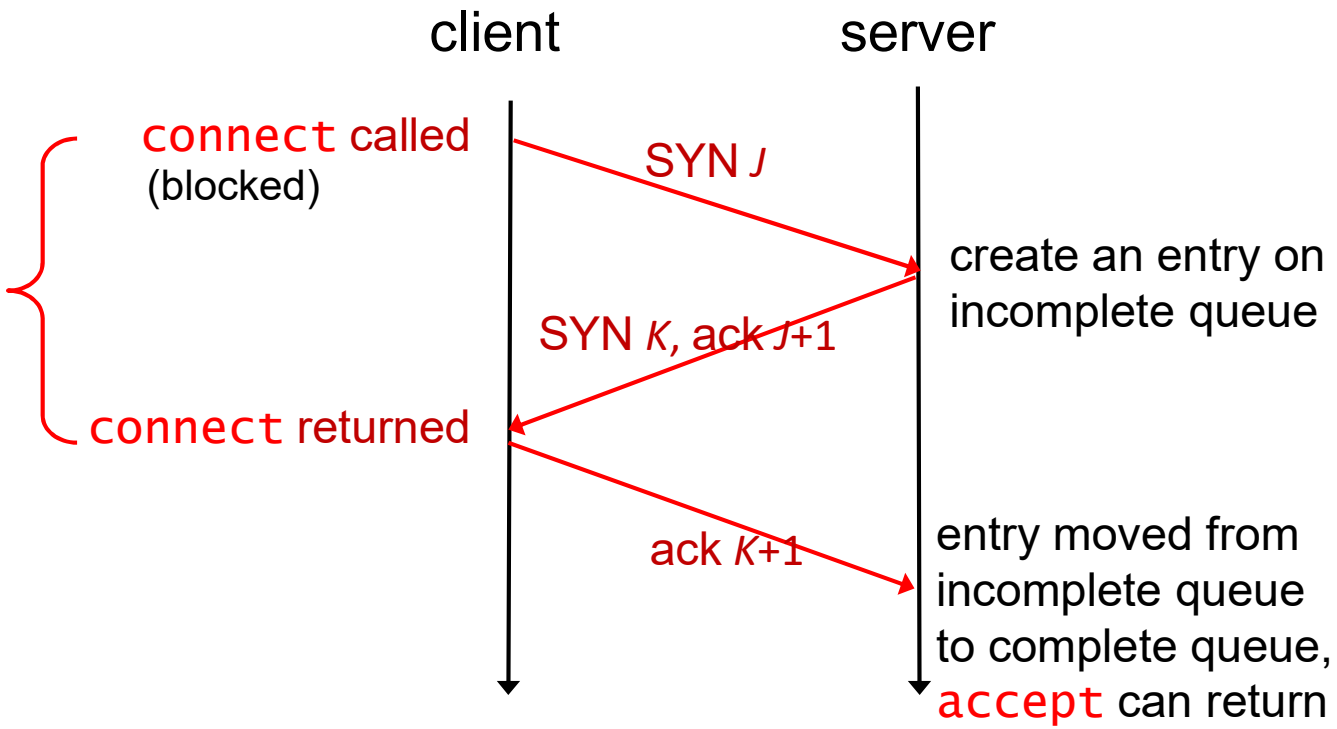
暫停. 等待server的回應

user按下EOF

如果Parent先不正常結束，Child再讀到EOF…

- Parent已不存在時，getppid()傳回1，即init process 的process id
  - 當parent死亡而child仍存在時，init繼承成為child的parent
- 當child沒有superuser權力時，是不允許送SIGTERM給init的(這代表要shutdown)
- 如果child有superuser權力時…

Blocking `connect`



Non-blocking `connect()`

- Returns without waiting for the completion of the three-way handshake
- Three uses for a non-blocking `connect`
  - Overlap other process with the three-way handshake
  - Establish multiple connections at the same time
  - Shorten the timeout for the `connect`

設定socket為non-blocking後，呼叫connect的return值...

- 可能return 0 (成功!)
  - 當server和client在同一台機器時
- 可能return -1,且errno =EINPROGRESS
  - kernel正在進行three-way handshake程序
  - 應用程式可趁此時進行其它工作
  - 稍後再測three-way handshake程序成功與否
- return -1,且errno為其它值⇒ error

Non-blocking connect的函數

```
Int connect_nonb(int sockfd, const SA *saptr, socklen_t salen, int nsec)
{
        int             flags, n, error;
        socklen_t       len;
        fd_set          rset, wset;
        struct timeval  tval;

        flags = Fcntl(sockfd, F_GETFL, 0);
        Fcntl(sockfd, F_SETFL, flags | O_NONBLOCK);
        error = 0;
        if ( (n = connect(sockfd, (struct sockaddr *) saptr, salen)) < 0)
                if (errno != EINPROGRESS)
                        return(-1);
        if (n == 0)
            goto done;  /* connect completed immediately */
```

暫存socket目前的file status

暫時將socket設為nonblocking

errno不為EINPROGRESS時return

Check for immediate completion

connect return -1,且errno = EINPROGRESS。接下來呢?

- 需要判斷connect成功與否的方法
  - When the connection completes successfully, the descriptor becomes writable
  - 如果server已經送來data，此socket還會變成readable
  - However, when the connection establishment encounters an error, the descriptor becomes both readable and writable
  - 因此無法僅僅以socket是否readable/writable作為connect是否成功的判斷依據

決定connect是否成功

- socket尚未readable/writable ⇒ connect 尚未成功
- 至少等到socket變成readable/writable時(次頁)，再進一步判斷connect是否成功
- 以getsockopt取回socket error來判斷是一種方法(如下下頁所示)
- 亦可呼叫getpeername，若傳回-1且errno = ENOTCONN表connect失敗
- 還有其它方法(read, connect again)

Nonblocking connect

```
FD_ZERO(&rset);
FD_SET(sockfd, &rset);
wset = rset;
tval.tv_sec = nsec;
tval.tv_usec = 0;

                                        select傳回0表timeout
if ( (n = Select(sockfd+1, &rset, &wset, NULL,
                                  nsec ? &tval : NULL)) == 0) {
        close(sockfd);      /* timeout */
        errno = ETIMEDOUT;
        return(-1);
}
```

用select測試socket是否readable/writable

```
        if (FD_ISSET(sockfd, &rset) || FD_ISSET(sockfd, &wset)) {
                len = sizeof(error);
                if (getsockopt(sockfd, SOL_SOCKET, SO_ERROR,
&error, &len) < 0)
                        return(-1);
        } else
                err_quit("select error: sockfd not set");
done:
        Fcntl(sockfd, F_SETFL, flags); /* restore file status flags */
        if (error) {
                close(sockfd);          /* just in case */
                errno = error;
                return(-1);
        }
        return(0);
}
```

3/3

readable或writable

傳回0表getsockopt成功

取回socket error

放入error

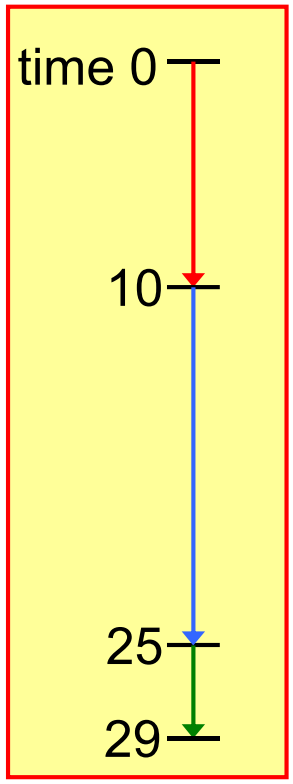
回存原來的file status

error不為0表connect不成功。此時將取回的錯誤代碼放到errno中供讀取

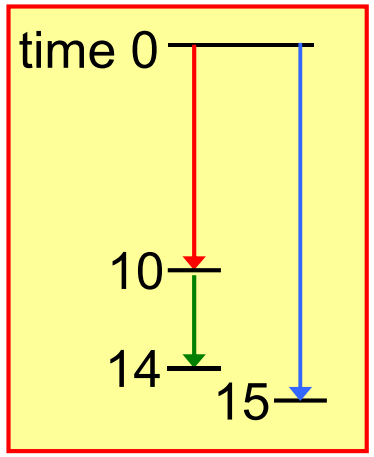
Non-blocking connect: Web Client

- 網頁中通常有多個URL連結
- 瀏覽器可用non-blocking connect方式同時建立多個連結向相關伺服器取回所需的資源
- 這樣做比用blocking connect一個URL接一個URL依序向伺服器取資源有效率

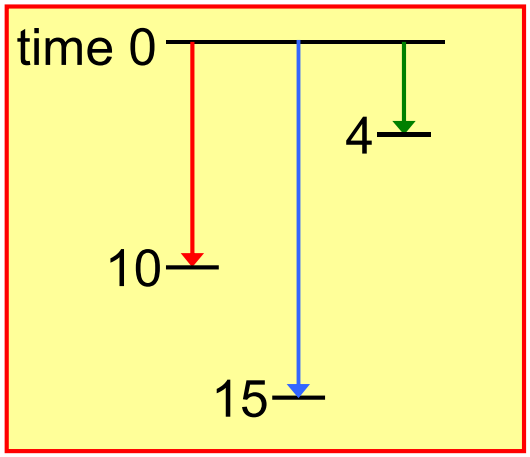
Establishing Multiple Connections in Parallel



serially

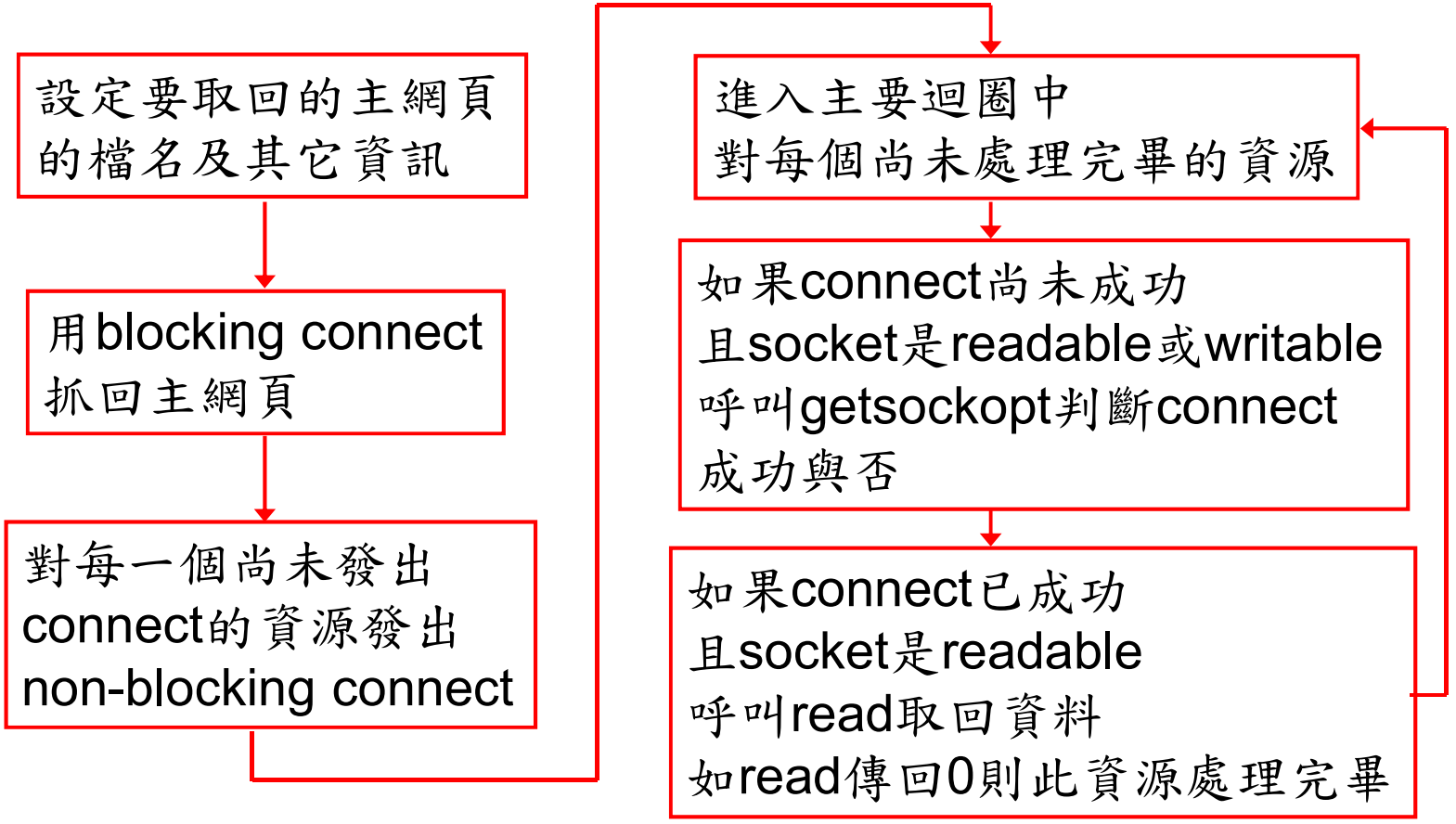


in parallel with max 2 connections at a time



in parallel with max 3 connections at a time

Flow for Web Client



Non-blocking accept()

- 使用select()，當listening socket被select()回傳readable時，server再呼叫accept
  - 理論上此時呼叫accept時不會被block，所以listening socket為blocking socket也無所謂
  - 但在three-way handshake完成後，server尚未呼叫accept前，client TCP可能送出RST，導致completed connection從queue中被移出
  - 可會導致server呼叫accept時被block! (Berkeley 系列的Unix)

解決之道

- Always set a listening socket non-blocking and
- ignore the following errors on the subsequent call to `accept()`: EWOULDBLOCK, ECONNABORTED, EPROTO, and EINTR

前三者都是client aborts the connection before server accepts it. 再次重新呼叫select等待下個連線請求即可