presentation

Nonblocking I/O

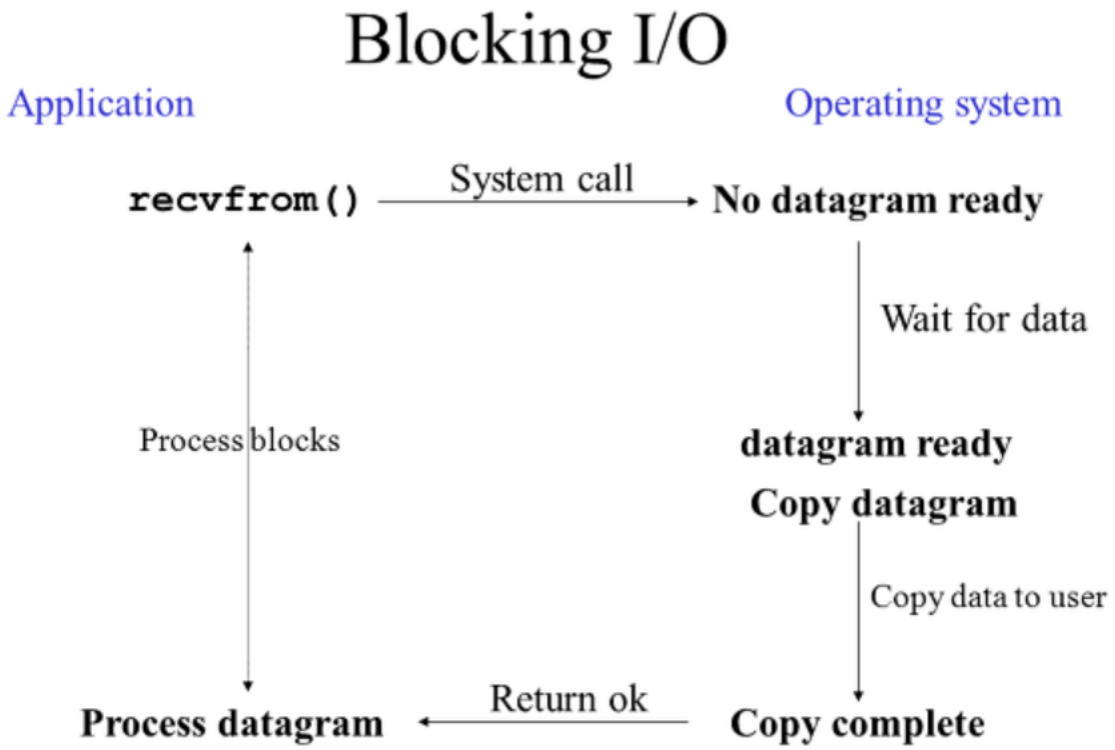
2017707007 강민준

CONTENTS

Blocking

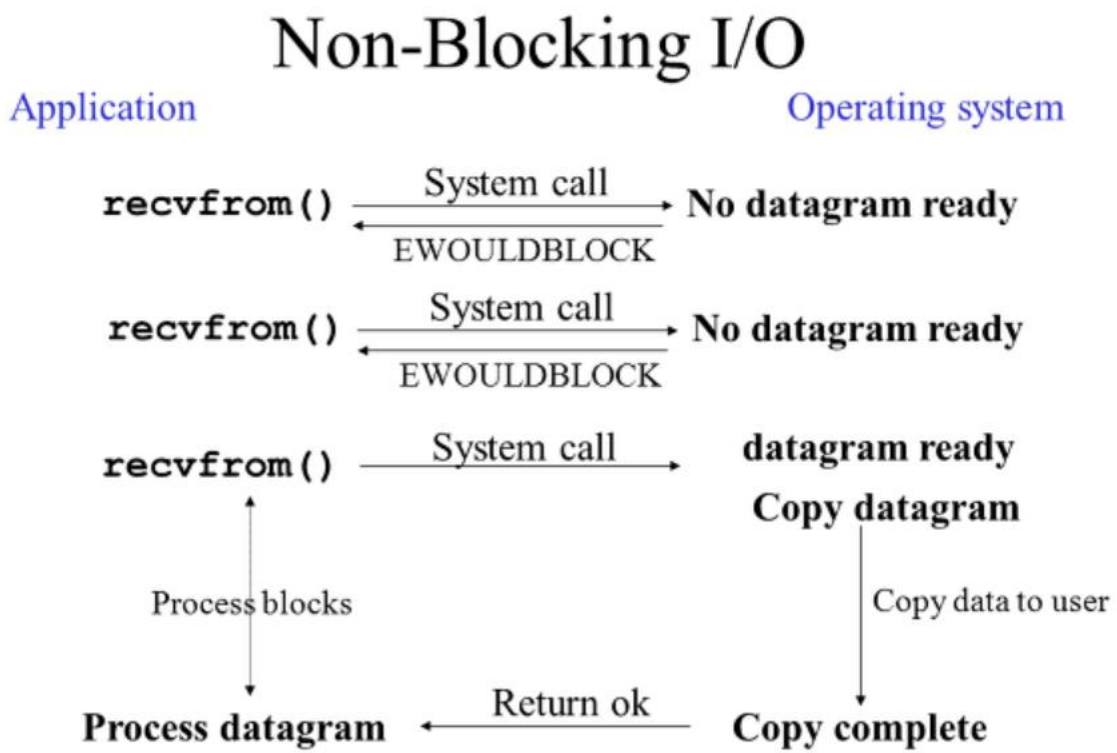
1:1 통신

프로그램이 한가지 작업만을 수행



Nonblocking

1:M 통신
여러 작업 병행
즉시 처리할 수 없어도, system call을 return 하며, block 방지



01 | Introduction

Four categories:

1. **Input** operations
   - read, readv, recv, recvform, recvmsg

2. **Output** operations
   - write, writev, send, sendto, sendmsg

3. **Accept** operation
   - accept

4. **Connect** operation
   - connect

01 | Introduction

Return

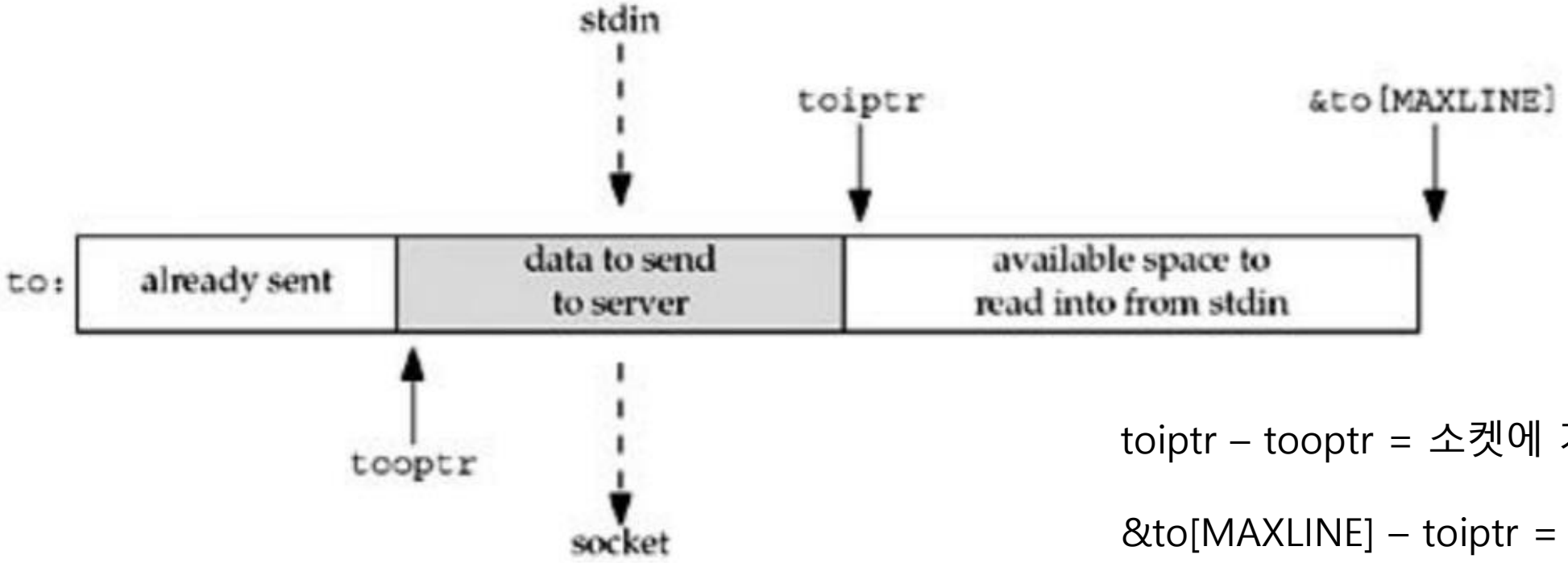
	Blocking	Nonblocking
Read 버퍼가 empty 이면	block	Return : -1 Errno : EWOULDBLOCK
Write 버퍼가 full 이면	block	Return : -1 Errno : EWOULDBLOCK
Accpet Queue가 비어 있으면	block	Return : -1 Errno : EWOULDBLOCK
Connect 완전히 수행 할 때 까지	block	곧바로 return

1. 버퍼의 관리가 매우 복잡
2. 멀티쓰레드 기반에서 nonblocking 사용 시, CPU 사용량 급증

CONTENTS

02 | Read & Write

To(상대방에게) buffer



toiptr – tooptr = 소켓에 기록되는 data bytes

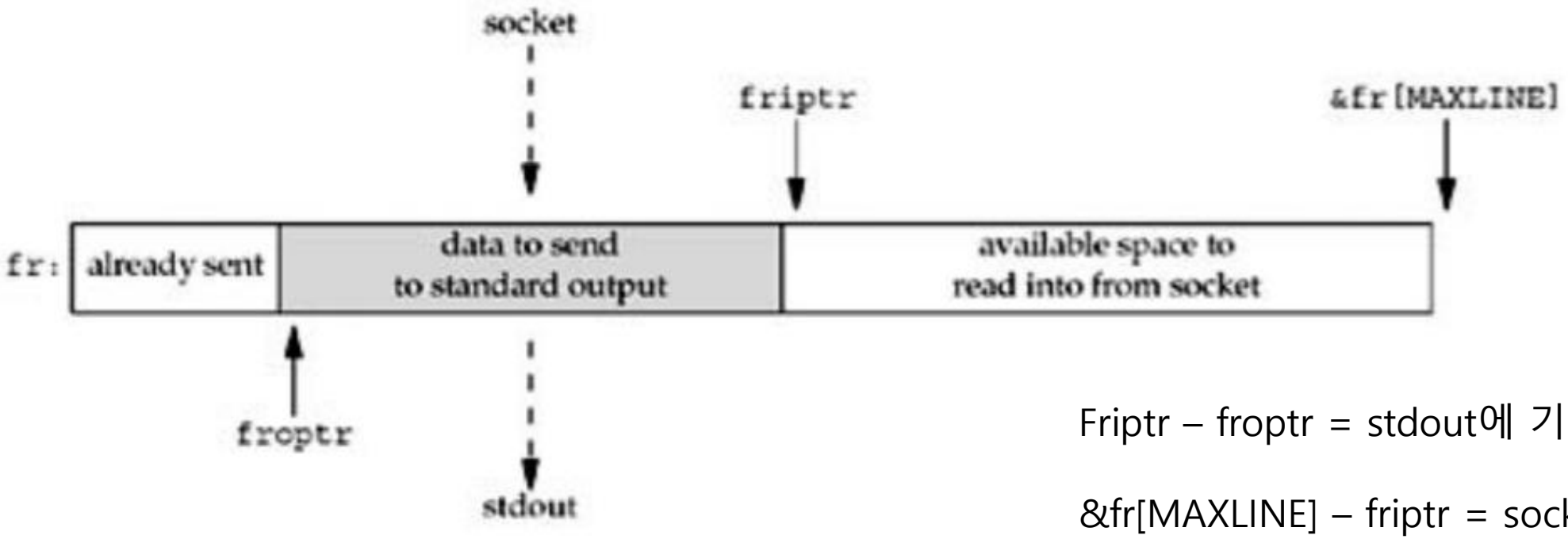
&to[MAXLINE] – toiptr = stdin에서 읽는 bytes

tooptr = toiptr 시, ptr 2개 시작지점으로 초기화

to : 표준 입력에서 서버로 가는 데이터를 포함
toiptr : 포인터는 표준 입력에서 데이터를 읽을 수 있는 다음 바이트를 포함
tooptr: : 소켓에 기록되어야 하는 다음 바이트를 가리 킵니다.

02 | Read & Write

From(누구로부터) buffer



Friptr – froptr = stdout에 기록되는 data bytes

&fr[MAXLINE] – friptr = socket에서 읽는 bytes

froptr = friptr 시, ptr 2개 시작지점으로 초기화

fr : 서버에서 표준 출력으로 가는 데이터를 포함

02 | Read & Write

str_cli

```
1  #include    "unp.h"
2
3  void
4  str_cli(FILE *fp, int sockfd)
5  {
6      int          maxfdp1, val, stdineof;
7      ssize_t      n, nwritten;
8      fd_set       rset, wset;
9      char         to[MAXLINE], fr[MAXLINE];
10     char         *toiptr, *tooptr, *friptr, *froptr;
11
12     val = Fcntl(sockfd, F_GETFL, 0);
13     Fcntl(sockfd, F_SETFL, val | O_NONBLOCK);
14
15     val = Fcntl(STDIN_FILENO, F_GETFL, 0);
16     Fcntl(STDIN_FILENO, F_SETFL, val | O_NONBLOCK);
17
18     val = Fcntl(STDOUT_FILENO, F_GETFL, 0);
19     Fcntl(STDOUT_FILENO, F_SETFL, val | O_NONBLOCK);
20
21     toiptr = tooptr = to;    /* initialize buffer pointers */
22     friptr = froptr = fr;
23     stdineof = 0;
24     maxfdp1 = max(max(STDIN_FILENO, STDOUT_FILENO), sockfd) + 1;
```

int fcntl(int fd, int cmd, int arg);

F_GETFL : 파일 상태 플래그들을 조회한다.(open() 함수 호출 시 설정한 플래그 값들을 반환해준다.)
F_SETFL : 파일 상태 플래그들을 설정한다.

Fcntl을 사용하여 nonblocking으로 설정

12, sockfd 가져오고
13, sockfd에, 12에서 가져온 sockfd | O_NONBLOCK

옵션 입힘 이를 STDIN과 STDOUT에 반복

Select에 사용될 앞서 설명한 ptr 초기화

02 | Read & Write

str_cli

```
26  for (; ; ) {
27      FD_ZERO(&rset);
28      FD_ZERO(&wset);
29      if (stdineof == 0 && toiptr < &to[MAXLINE])
30          FD_SET(STDIN_FILENO, &rset);    /* read from stdin */
31      if (friptr < &fr[MAXLINE])
32          FD_SET(sockfd, &rset);          /* read from socket */
33      if (tooptr != toiptr)
34          FD_SET(sockfd, &wset);          /* data to write to socket */
35      if (froptr != friptr)
36          FD_SET(STDOUT_FILENO, &wset);   /* data to write to stdout */
37
38      Select(maxfdp1, &rset, &wset, NULL, NULL);
```

// read, write set으로 초기화

```
FD_ZERO(fd_set* set);          //fdset을초기화
FD_SET(int fd, fd_set* set);   //fd를 set에 등록
FD_CLR(int fd, fd_set* set);   //fd를 set에서 삭제
FD_ISSET(int fd, fd_set* set);//fd가 준비되었는지 확인
```

읽기 위한

쓰기 위한

29, stdin에서 EOF를 아직 읽지 않고,
"to"버퍼에 최소 1byte의 데이터 공간이 있으면 stdin_FILENO가 rset에 set됨

31, "fr"버퍼에 최소 1byte의 데이터 공간이 있으면 sockfd가 rset에 set됨
33, "to" 버퍼 socket에 쓸 데이터가 있는 경우
35, "fr"버퍼에 표준 출력으로 전송할 데이터가 있는 경우

02 | Read & Write

str_cli – read state

29, stdin에서 EOF를 아직 읽지 않고,
"to"버퍼에 최소 1byte의 데이터 공간이 있으면 stdin_FILENO가 rset에 set됨

Read 상황으로 부름.

```
45      if (FD_ISSET(STDIN_FILENO, &rset)) {
46          if ((n = read(STDIN_FILENO, toiptr, &to[MAXLINE] - toiptr)) < 0) {
47              if (errno != EWOULDBLOCK)
48                  err_sys("read error on stdin");
49          }
50          else if (n == 0) {
51              fprintf(stderr, "%s: EOF on stdin₩n", gf_time());
52              stdineof = 1;           /* all done with stdin */
53              if (tooptr == to)
54                  Shutdown(sockfd, SHUT_WR);/* send FIN */
55
56          }
57          else {
58              fprintf(stderr, "%s: read %d bytes from stdin₩n", gf_time(), n);
59              toiptr += n;            /* # just read */
60              FD_SET(sockfd, &wset);  /* try and write to socket below */
61          }
62      }
```

일반적으로이 조건은 발생 X
EWOULDBLOCK을 리턴하여 읽음을 알림
-> 에러 핸들링

표준 입력이 끝난 상황 (버퍼에 데이터 X)
Flag bit 인 stdineof 를 set 하고,
Server측에 FINISH를 보냄
(shutdown 역할)
-> 종료

read가 데이터를 반환하면 toiptr을 증가.
루프 중, 소켓에 대한 쓰기를 위해 wset를 on 함

02 | Read & Write

str_cli

31, "fr"버퍼에 최소 1byte의 데이터 공간이 있으면 sockfd가 rset에 set됨

```
64          if (FD_ISSET(sockfd, &rset)) {
65              if ((n = read(sockfd, friptr, &fr[MAXLINE] - friptr)) < 0) {
66                  if (errno != EWOULDBLOCK)
67                      err_sys("read error on socket");
68
69              }
70              else if (n == 0) {
71                  fprintf(stderr, "%s: EOF on socket₩n", gf_time());
72                  if (stdineof)
73                      return;     /* normal termination */
74                  else
75                      err_quit("str_cli: server terminated prematurely");
76              }
77              else {
78                  fprintf(stderr, "%s: read %d bytes from socket₩n",
79                      gf_time(), n);
80                  friptr += n;        /* # just read */
81                  FD_SET(STDOUT_FILENO, &wset);   /* try and write below */
82              }
83          }
```

Stdin 을 읽은 상황

Socket 을 읽은 상황

02 | Read & Write

str_cli

33, "to" 버퍼 socket에 쓸 데이터가 있는 경우

```
87          if (FD_ISSET(STDOUT_FILENO, &wset) && ((n = friptr - froptr) > 0)) {
88              if ((nwritten = write(STDOUT_FILENO, froptr, n)) < 0) {
89                  if (errno != EWOULDBLOCK)
90                      err_sys("write error to stdout");
91
92              }
93              else {
94                  fprintf(stderr, "%s: wrote %d bytes to stdout₩n",
95                      gf_time(), nwritten);
96                  froptr += nwritten;     /* # just written */
97                  if (froptr == friptr)
98                      froptr = friptr = fr;   /* back to beginning of buffer */
99              }
100         }
```

표준 출력이 쓰기 가능,
쓰기 바이트 수가 0보다 큰 경우
쓰기가 호출됨
EWOULDBLOCK이 리턴 시, 아무 일 없음.

쓰기에 성공, froptr은 쓴 바이트 수만큼 증가
Outptr == inptr 의 경우, 버퍼 초기값 설정

02 | Read & Write

str_cli

35, "fr"버퍼에 표준 출력으로 전송할 데이터가 있는 경우

이와 유사

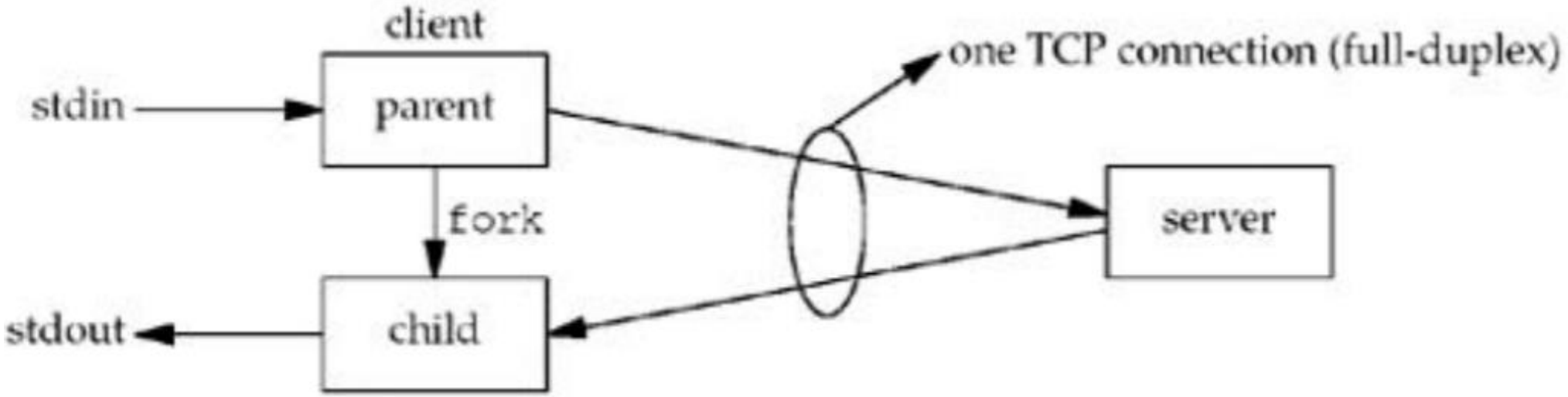
```
102         if (FD_ISSET(sockfd, &wset) && ((n = toiptr - tooptr) > 0)) {
103             if ((nwritten = write(sockfd, tooptr, n)) < 0) {
104                 if (errno != EWOULDBLOCK)
105                     err_sys("write error to socket");
106
107             }
108             else {
109                 fprintf(stderr, "%s: wrote %d bytes to socket₩n",
110                     gf_time(), nwritten);
111                 tooptr += nwritten; /* # just written */
112                 if (tooptr == toiptr) {
113                     toiptr = tooptr = to;   /* back to beginning of buffer */
114                     if (stdineof)
115                         Shutdown(sockfd, SHUT_WR);  /* send FIN */
116                 }
117             }
118         }
119     }
```

쓰기에 성공, froptr은 쓴 바이트 수만큼 증가
Outptr == inptr 의 경우, 버퍼 초기값 설정 뿐 아니라,
EOF가 발생하면 FINISH를 server로 전송

코드가 굉장히 김 -> simple str_cli 설명 (fork 사용)

02 | Read & Write

간단한 str_cli (fork)



자식 process -> server에서 stdout으로 복사
부모 process -> stdin에서 server로 행 복사

(fork 사용으로 인한 종료방법 수정)

기존에는 FIN 코드를 보내는 것으로 종료
이는 불가능, 자식이 종료되며 SIGCHLD를 잡아야함
부모가 실행중이면 SIGTERM

* 이유 (부록참조)

02 | Read & Write

간단한 str_cli (fork)

```
3   void
4   str_cli(FILE *fp, int sockfd)
5   {
5       pid_t pid;
6       char sendline[MAXLINE], recvline[MAXLINE];
7       if ((pid = Fork()) == 0) { /* child: server -> stdout */
8           while (Readline(sockfd, recvline, MAXLINE) > 0)
9               Fputs(recvline, stdout);
10          kill(getppid(), SIGTERM); /* in case parent still running */
11          exit(0);
12      }
13      /* parent: stdin -> server */
14      while (Fgets(sendline, MAXLINE, fp) != NULL)
15          Writen(sockfd, sendline, strlen(sendline));
16
17      Shutdown(sockfd, SHUT_WR); /* EOF on stdin, send FIN */
18      pause();
19      return;
20  }
```

복사가 끝나면 부모 호출이 일시 중지,
SIGTREM을 수신할 때 까지 sleep
여기서 대기가 아닌 kill인 이유는,
예제에 필요한 str_cli에 시간측정을 위함

CONTENTS

03 | connect

Connect 용도

Nonblocking에서 사용하는 3 가지 connect의 사용

1. 3-way handshake를 수행하는 도중, 다른 처리와 겹침
   - 연결에는 하나의 RTT가 필요한데, WAN의 경우 긴 연결타임을 가질 수 있음, 따라서 다른 것을 처리해야함

2. 동시에 여러 연결 가능

3. select를 사용하여 연결이 설정 될 때까지 기다리므로 select에 대한 시간 제한을 지정하여 연결 시간 제한 단축

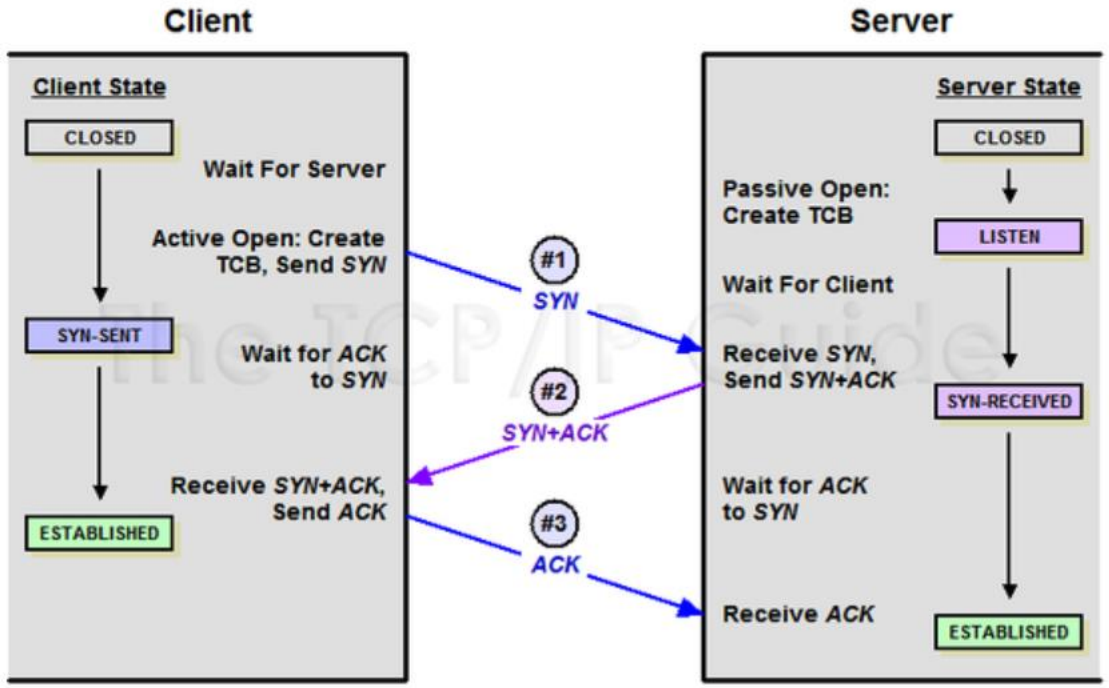
03 | connect

Connect 수행작업

Connect 완전히 수행 할 때 까지	block	곧바로 return

곧 바로 return 하지만,

TCP의 3-way Handshaking



이 과정은 계속 됨

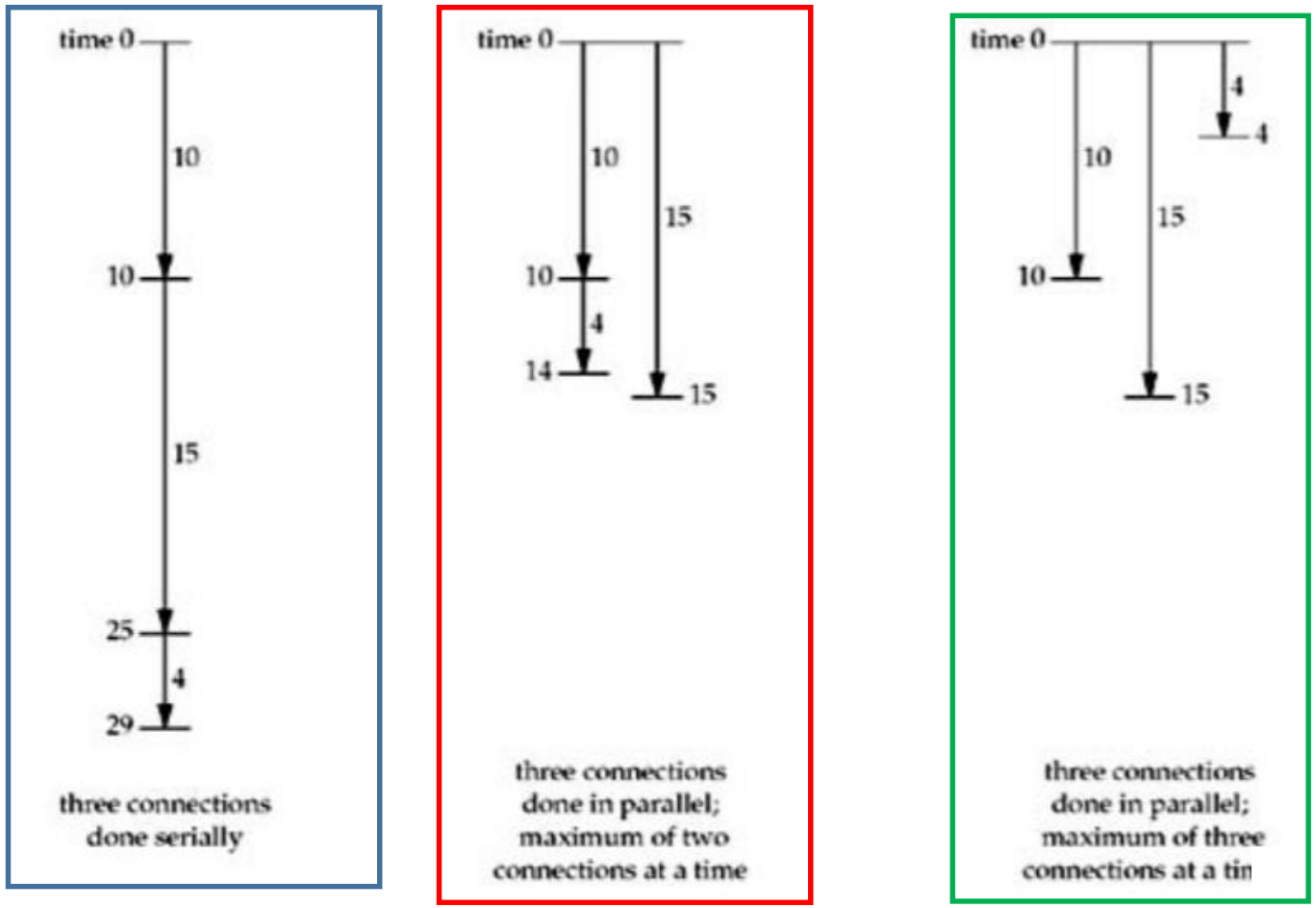
이 후, select를 통해 확인함.

03 | connect

parrel client connect

클라이언트는 nonblocking connect를 사용하여 동시에 여러 페이지를 한 번에 하나씩 직렬로 가져 옴.

Figure 16.12. Establishing multiple connections in parallel.



직렬로 수행
총 29개의 시간단위로, 10, 15, 4개의 시간이 필요하다 가정

두개의 연결을 병렬로 수행
총 시간 15로 이상적인 상황을 나타냄
리소스 점유 경쟁 때문에, 각각의 직렬 time이 길어지더라도 29보다 작음

세개의 연결을 병렬로 수행,
총 시간 15로 2와 같음

03 | connect

예제

socket 구성 후, 무한 루프에 에서 연결요청 계속 수락하는 상황

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define BUF_SIZE 30
void error_handling(char *message);
void read_childproc(int sig);

int main(int argc, char *argv[]) {
    int serv_sock, clnt_sock;
    struct sockaddr_in serv_adr, clnt_adr;

    pid_t pid;
    struct sigaction act;
    socklen_t adr_sz;
    int str_len, state;
    char buf[BUF_SIZE];
    if (argc != 2) {
        printf("Usage : %s <port>Wn", argv[0]);
        exit(1);
    }

    act.sa_handler = read_childproc;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    state = sigaction(SIGCHLD, &act, 0);
    // 1. socket 하나를 생성한다.
    serv_sock = socket(PF_INET, SOCK_STREAM, 0);
    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family = AF_INET;
    serv_adr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_adr.sin_port = htons(atoi(argv[1]));

    // 2. socket에 IP와 Port 번호를 할당한다.
    if (bind(serv_sock, (struct sockaddr*) &serv_adr, sizeof(serv_adr)) == -1)
        error_handling("bind() error");

// 3. 생성한 socket을 server socket(listen socket)으로 등록한다.
    if (listen(serv_sock, 5) == -1)
        error_handling("listen() error");
while (1) {
        adr_sz = sizeof(clnt_adr);
        // 4. 부모 프로세스는 리스닝 소켓으로
        //accept 함수 호출을 통해서 연결요청을 수락한다.
        clnt_sock = accept(serv_sock, (struct sockaddr*)&clnt_adr, &adr_sz);
        if (clnt_sock == -1)
            continue;
        else
            puts("new client connected...");

        close(serv_sock);
    return 0;
}
```

03 | connect

예제

socket 구성 후, 무한 루프에 에서 연결요청 계속 수락하는 상황

```
// 5. 이때 얻게 되는 소켓의 파일 디스크립터
//(클라이언트와 연결된 연결 소켓)를 자식 프로세스를 생성해 넘겨준다.
        pid = fork();                                          //nonblocking state를 가지기 시작한 순간
        if (pid == -1) {                                       //error 핸들링
            close(clnt_sock);
            continue;
        }
        if (pid == 0) {
            close(serv_sock);
// 6. 자식 프로세스는 전달받은 파일 디스크립터를
//바탕으로 서비스를 제공한다.
            while ((str_len = read(clnt_sock, buf, BUF_SIZE)) != 0)
                write(clnt_sock, buf, str_len);

            close(clnt_sock);                                  //종료핸들링을 자식프로세스 close로 핸들링
            puts("client disconnected...");
            return 0;
        }
        else
            close(clnt_sock);
    }
```

Presentation

감사합니다

부록

https://www.cs.huji.ac.il/course/2007/com1/strclinonb.c

부록

기존에는 FIN 코드를 보내는 것으로 종료
이는 불가능, 자식이 종료되며 SIGCHLD를 잡아야함
부모가 실행중이면 SIGTERM

* 이유 (부록참조)

이유 (전문)

We again need to worry about the termination sequence. Normal termination occurs when the EOF on standard input is encountered. The parent reads this EOF and calls shutdown to send a FIN. (The parent cannot call close. See Exercise 16.1.) But when this happens, the child needs to continue copying from the server to the standard output, until it reads an EOF on the socket.

It is also possible for the server process to terminate prematurely (Section 5.12); if this occurs, the child will read an EOF on the socket. If this happens, the child must tell the parent to stop copying from the standard input to the socket (see Exercise 16.2). In Figure 16.10, the child sends the SIGTERM signal to the parent, in case the parent is still running (see Exercise 16.3). Another way to handle this would be for the child to terminate and have the parent catch SIGCHLD, if the parent is still running

요약

부모는 close를 호출 할 수 없고, 자식은 소켓에서 EOF를 읽을 때 까지 server 에서 stdout으로 계속 복사가 일어나 비효율적. 서버가 조기종료 될 수 있는데, 자식은 socket에서 EOF를 계속 읽음 따라서 SIGTERM, SIGCHLD로, 핸들링 필요

부록 – 예제 전체 코드 (Server)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define BUF_SIZE 30
void error_handling(char *message);
void read_childproc(int sig);

int main(int argc, char *argv[]) {
    int serv_sock, clnt_sock;
    struct sockaddr_in serv_adr, clnt_adr;

    pid_t pid;
    struct sigaction act;
    socklen_t adr_sz;
    int str_len, state;
    char buf[BUF_SIZE];
    if (argc != 2) {
        printf("Usage : %s <port>₩n", argv[0]);
        exit(1);
    }

    act.sa_handler = read_childproc;
    sigemptyset(&act.sa_mask);
    act.sa_flags = 0;
    state = sigaction(SIGCHLD, &act, 0);
    // 1. socket 하나를 생성한다.
    serv_sock = socket(PF_INET, SOCK_STREAM, 0);
    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family = AF_INET;
    serv_adr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_adr.sin_port = htons(atoi(argv[1]));

    // 2. socket에 IP와 Port 번호를 할당한다.
    if (bind(serv_sock, (struct sockaddr*) &serv_adr, sizeof(serv_adr)) == -1)
        error_handling("bind() error");

// 3. 생성한 socket을 server socket(listen socket)으로 등록한다.
    if (listen(serv_sock, 5) == -1)
        error_handling("listen() error");

    while (1) {
        adr_sz = sizeof(clnt_adr);
        // 4. 부모 프로세스는 리스닝 소켓으로
        //accept 함수 호출을 통해서 연결요청을 수락한다.
        clnt_sock = accept(serv_sock, (struct sockaddr*)&clnt_adr, &adr_sz);
        if (clnt_sock == -1)
            continue;
        else
            puts("new client connected...");
        // 5. 이때 얻게 되는 소켓의 파일 디스크립터
        //(클라이언트와 연결된 연결 소켓)를 자식 프로세스를 생성해 넘겨준다.
        pid = fork();
        if (pid == -1) {
            close(clnt_sock);
            continue;
        }
        if (pid == 0) {
            close(serv_sock);
        // 6. 자식 프로세스는 전달받은 파일 디스크립터를
        //바탕으로 서비스를 제공한다.
            while ((str_len = read(clnt_sock, buf, BUF_SIZE)) != 0)
                write(clnt_sock, buf, str_len);

            close(clnt_sock);
            puts("client disconnected...");
            return 0;
        }
        else
            close(clnt_sock);
    }
    close(serv_sock);
    return 0;
}

void read_childproc(int sig) {
    pid_t pid;
    int status;
    pid = waitpid(-1, &status, WNOHANG);
    printf("removed proc id: %d ₩n", pid);
}
void error_handling(char *message) {
    fputs(message, stderr);
    fputc('₩n', stderr);
    exit(1);
}
```

부록 – 예제 전체 코드 (client)

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define BUF_SIZE 1024
void error_handling(char *message);

int main(int argc, char *argv[]) {
// 파일 디스크립터를 위한 변수
    int sock;
    char message[BUF_SIZE];
    int str_len;
    struct sockaddr_in serv_adr;

    if (argc != 3) {
        printf("Usage : %s <IP> <port>₩n", argv[0]);
        exit(1);
    }

    // 1. socket 하나를 생성한다.
    sock = socket(PF_INET, SOCK_STREAM, 0);
    if (sock == -1)
        error_handling("socket() error");

    memset(&serv_adr, 0, sizeof(serv_adr));
    serv_adr.sin_family = AF_INET;
    serv_adr.sin_addr.s_addr = inet_addr(argv[1]);
    serv_adr.sin_port = htons(atoi(argv[2]));

    // 2. socket을 이용해 server의 server socket(listen socket)에 연결을 요청한다.
    if (connect(sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr)) == -1)
        error_handling("connect() error!");
    else
        puts("Connected.....");

    while (1) {
        fputs("Input message(Q to quit): ", stdout);
        fgets(message, BUF_SIZE, stdin);

        if (!strcmp(message, "q₩n") || !strcmp(message, "Q₩n"))
            break;

        // 3. 연결된 socket을 통해 server로부터 데이터를 송수신한다.
        write(sock, message, strlen(message));
        str_len = read(sock, message, BUF_SIZE - 1);
        message[str_len] = 0;
        printf("Message from server: %s", message);
    }

    close(sock);
    return 0;
}

void error_handling(char *message) {
    fputs(message, stderr);
    fputc('₩n', stderr);
    exit(1);
}
```

출처 : https://jongmin92.github.io/2019/02/28/Java/java-with-non-blocking-io/

멀티프로세스 부분 발췌 후 주석 추가