ISL

Network Programming #6

So-Yong Kim

Index

select()

```c
int main(int argc, char *argv[])
{
    fd_set reads, temps;
    int result, str_len;
    char buf[BUF_SIZE];
    struct timeval timeout;

    FD_ZERO(&reads);
    FD_SET(0, &reads);

    while(1)
    {
        temps=reads;
        timeout.tv_sec=5;
        timeout.tv_usec=0;
        result=select(1, &temps, 0, 0, &timeout)
        if(result==-1)
        {
            puts("select() error!");
            break;
        }
        else if(result==0)
        {
            puts("Time-out!");
        }
        else
        {
            if(FD_ISSET(0, &temps))
            {
                str_len=read(0, buf, BUF_SIZE);
                buf[str_len]=0;
                printf("message from console: %s", buf);
            }
        }
    }
    return 0;
}
```

```c
#include <sys/select.h>
#include <sys/time.h>

int select(
int maxfd, fd_set *readset, fd_set *writeset, fd_set *exceptset, const struct timeval * timeout);
```

➜ 성공 시 0 이상, 실패 시 −1 반환

- maxfd — 검사 대상이 되는 파일 디스크립터의 수.
- readset — fd_set형 변수에 '수신된 데이터의 존재여부'에 관심 있는 파일 디스크립터 정보를 모두 등록해서 그 변수의 주소 값을 전달한다.
- writeset — fd_set형 변수에 '블로킹 없는 데이터 전송의 가능여부'에 관심 있는 파일 디스크립터 정보를 모두 등록해서 그 변수의 주소 값을 전달한다.
- exceptset — fd_set형 변수에 '예외상황의 발생여부'에 관심이 있는 파일 디스크립터 정보를 모두 등록해서 그 변수의 주소 값을 전달한다.
- timeout — select 함수호출 이후에 무한정 블로킹 상태에 빠지지 않도록 타임아웃(time-out)을 설정하기 위한 인자를 전달한다.
- 반환 값 — 오류발생시에는 −1이 반환되고, 타임 아웃에 의한 반환 시에는 0이 반환된다. 그리고 관심대상으로 등록된 파일 디스크립터에 해당 관심에 관련된 변화가 발생하면 0보다 큰 값이 반환되는데, 이 값은 변화가 발생한 파일 디스크립터의 수를 의미한다.

```c
struct timeval
{
    long tv_sec;      // seconds
    long tv_usec;     // microseconds
}
```

select()

```c
int main(int argc, char *argv[])
{
    fd_set reads, temps;
    int result, str_len;
    char buf[BUF_SIZE];
    struct timeval timeout;

    FD_ZERO(&reads);
    FD_SET(0, &reads);

    while(1)
    {
        temps=reads;
        timeout.tv_sec=5;
        timeout.tv_usec=0;
        result=select(1, &temps, 0, 0, &timeout)
        if(result==-1)
        {
            puts("select() error!");
            break;
        }
        else if(result==0)
        {
            puts("Time-out!");
        }
        else
        {
            if(FD_ISSET(0, &temps))
            {
                str_len=read(0, buf, BUF_SIZE);
                buf[str_len]=0;
                printf("message from console: %s", buf);
            }
        }
    }
    return 0;
}
```

```c
fd_set reads, temps;
```

```c
/* fd_set for select and pselect.  */
typedef struct
  {
    /* XPG4.2 requires this member name.  Otherwise avoid the name
       from the global namespace.  */
#ifdef __USE_XOPEN
    __fd_mask fds_bits[__FD_SETSIZE / __NFDBITS];
# define __FDS_BITS(set) ((set)->fds_bits)
#else
    __fd_mask __fds_bits[__FD_SETSIZE / __NFDBITS];
# define __FDS_BITS(set) ((set)->__fds_bits)
#endif
  } fd_set;
```

```c
typedef long int __fd_mask;
```

long int __fds_bits[16] = 128bytes → 1024bits
(8bytes)

= 1024개

비트 값의 변화에 따라 그 파일 디스크립터에서 이벤트가 발생하였는지 알 수 있음

select()

```c
int main(int argc, char *argv[])
{
    fd_set reads, temps;
    int result, str_len;
    char buf[BUF_SIZE];
    struct timeval timeout;

    FD_ZERO(&reads);
    FD_SET(0, &reads);

    while(1)
    {
        temps=reads;
        timeout.tv_sec=5;
        timeout.tv_usec=0;
        result=select(1, &temps, 0, 0, &timeout);
        if(result==-1)
        {
            puts("select() error!");
            break;
        }
        else if(result==0)
        {
            puts("Time-out!");
        }
        else
        {
            if(FD_ISSET(0, &temps))
            {
                str_len=read(0, buf, BUF_SIZE);
                buf[str_len]=0;
                printf("message from console: %s", buf);
            }
        }
    }
    return 0;
}
```

파일 디스크립터 0(stdin)을 관찰 대상으로 설정

파일 디스크립터 0(stdin)에서 이벤트가 발생하였는가?

select()

```c
int main(int argc, char *argv[])
{
    fd_set reads, temps;
    int result, str_len;
    char buf[BUF_SIZE];
    struct timeval timeout;

    FD_ZERO(&reads);
    FD_SET(0, &reads);

    while(1)
    {
        temps=reads;
        timeout.tv_sec=5;
        timeout.tv_usec=0;
        result=select(1, &temps, 0, 0, &timeout);
        if(result==-1)
        {
            puts("select() error!");
            break;
        }
        else if(result==0)
        {
            puts("Time-out!");
        }
        else
        {
            if(FD_ISSET(0, &temps))
            {
                str_len=read(0, buf, BUF_SIZE);
                buf[str_len]=0;
                printf("message from console: %s", buf);
            }
        }
    }
    return 0;
}
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/select$
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/select$ ./select
Time-out!
Time-out!
Time-out!
Time-out!
Time-out!
Hi
message from console: Hi
Time-out!
Good Bye!
message from console: Good Bye!
Time-out!
Time-out!
Time-out!
^C
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/select$
```

01 select()

Multiplexing Server

```c
int main(int argc, char *argv[])
{
    int serv_sock, clnt_sock;
    struct sockaddr_in serv_adr, clnt_adr;
    struct timeval timeout;
    fd_set reads, cpy_reads;

    socklen_t adr_sz;
    int fd_max, str_len, fd_num, i;
    char buf[BUF_SIZE];
    if(argc!=2) {
        printf("Usage : %s <port>\n", argv[0]);
        exit(1);
    }

    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]));

    if(bind(serv_sock, (struct sockaddr*) &serv_adr, sizeof(serv_adr))==-1)
        error_handling("bind() error");
    if(listen(serv_sock, 5)==-1)
        error_handling("listen() error");

    FD_ZERO(&reads);
    FD_SET(serv_sock, &reads);
    fd_max=serv_sock;
```

서버 소켓의 파일 디스크립터를 등록

```c
    while(1)
    {
        cpy_reads=reads;
        timeout.tv_sec=5;
        timeout.tv_usec=5000;

        if((fd_num=select(fd_max+1, &cpy_reads, 0, 0, &timeout))==-1)
            break;

        if(fd_num==0)
            continue;

        for(i=0; i<fd_max+1; i++)
        {
            if(FD_ISSET(i, &cpy_reads))
            {
                if(i==serv_sock)     // connection request!
                {
                    adr_sz=sizeof(clnt_adr);
                    clnt_sock=
                        accept(serv_sock, (struct sockaddr*)&clnt_adr, &adr_sz);
                    FD_SET(clnt_sock, &reads);
                    if(fd_max<clnt_sock)
                        fd_max=clnt_sock;
                    printf("connected client: %d \n", clnt_sock);
                }
                else    // read message!
                {
                    str_len=read(i, buf, BUF_SIZE);
                    if(str_len==0)    // close request!
                    {
                        FD_CLR(i, &reads);
                        close(i);
                        printf("closed client: %d \n", i);
                    }
                    else
                    {
                        write(i, buf, str_len);    // echo!
                    }
                }
            }
        }
    }
    close(serv_sock);
    return 0;
```

서버 소켓의 파일 디스크립터 + 1 까지의 범위 관찰

클라이언트 소켓까지 관찰할 수 있도록 설정
파일 디스크립터는 오름차순으로
생성되므로, 새로 만들어진 소켓 파일
디스크립터를 관찰 대상에 포함하기 위한
작업

클라이언트 소켓의 파일 디스크립터를 등록

클라이언트 소켓 등록 해제

01 select()

Multiplexing Server

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/select$ ./selectserv 8080
connected client: 4
connected client: 5
closed client: 4
closed client: 5
```

Server

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/select$ ./selectcli 127.0.0.1 8080
Connected...........
Input message(Q to quit): Hello
Message from server: Hello
Input message(Q to quit): q
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/select$
```

Client 1

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/select$ ./selectcli 127.0.0.1 8080
Connected...........
Input message(Q to quit): Hi
Message from server: Hi
Input message(Q to quit): q
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/select$
```

Client 2

02 send() & recv()

MSG_OOB

```
write(sock, "123", strlen("123"));
sleep(1);
send(sock, "4", strlen("4"), MSG_OOB);
sleep(1);
write(sock, "567", strlen("567"));
sleep(1);
send(sock, "890", strlen("890"), MSG_OOB);
sleep(1);
close(sock);
return 0;
```

MSG_OOB 옵션을 주면 긴급 메시지로 전송되며, 이 메시지를 수신하면 SIGURG 시그널이 발생함

Client(oob_send)

```
30      act.sa_handler=urg_handler;
31      sigemptyset(&act.sa_mask);
32      act.sa_flags=0;
```

시그널 처리를 위한 sigaction() 설정

```
44      serv_adr_sz=sizeof(serv_adr);
45      recv_sock=accept(acpt_sock, (struct sockaddr*)&serv_adr, &serv_adr_sz);
46
47      fcntl(recv_sock, F_SETOWN, getpid());
48      state=sigaction(SIGURG, &act, 0);
49
        while((str_len=recv(recv_sock, buf, sizeof(buf), 0))!= 0)
        {
            if(str_len==-1)
                continue;
            buf[str_len]=0;
            puts(buf);
        }
        close(recv_sock);
        close(acpt_sock);
        return 0;
    }
```

SIGURG는 소켓 소유자를 실행중인 프로세스로 전환해야 처리가능

```
smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week8$ ./oob_recv 8080
123
Urgent message: 4
567
Urgent message: 0
89
smalldragon@smalldragon-desktop:~/Workspace/network_programming_2023/week8$
```

```
62  void urg_handler(int signo)
63  {
64      int str_len;
65      char buf[BUF_SIZE];
66      str_len=recv(recv_sock, buf, sizeof(buf)-1, MSG_OOB);
67      buf[str_len]=0;
68      printf("Urgent message: %s \n", buf);
69  }
```

Server(oob_recv)

02 send() & recv()

MSG_PEEK | MSG_DONTWAIT

```c
if(connect(sock, (struct sockaddr*)&send_adr, sizeof(send_adr))==-1)
    error_handling("connect() error!");

write(sock, "123", strlen("123"));
close(sock);
return 0;
```

Client

```c
recv_adr_sz=sizeof(recv_adr);
recv_sock=accept(acpt_sock, (struct sockaddr*)&recv_adr, &recv_adr_sz);

while(1)
{
    str_len=recv(recv_sock, buf, sizeof(buf)-1, MSG_PEEK|MSG_DONTWAIT);
    if(str_len>0)
        break;
}

buf[str_len]=0;
printf("Buffering %d bytes: %s \n", str_len, buf);

str_len=recv(recv_sock, buf, sizeof(buf)-1, 0);
buf[str_len]=0;
printf("Read again: %s \n", buf);
close(acpt_sock);
close(recv_sock);
return 0;
```

MSG_PEEK: 입력 버퍼에서 읽어도, 읽은 부분이 없어지지 않고 유지

MSG_DONTWAIT: Non-Blocking으로 대기

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/send_recv$ ./peekserv 8080
Buffering 3 bytes: 123
Read again: 123
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/send_recv$
```

Server

03 writev() & readv()

writev()

```c
int main(int argc, char *argv[])
{
    struct iovec vec[2];
    char buf1[]="ABCDEFG";
    char buf2[]="1234567";
    int str_len;

    vec[0].iov_base=buf1;
    vec[0].iov_len=3;
    vec[1].iov_base=buf2;
    vec[1].iov_len=4;

    str_len=writev(1, vec, 2);
    puts("");
    printf("Write bytes: %d \n", str_len);
    return 0;
}
```

1: stdout

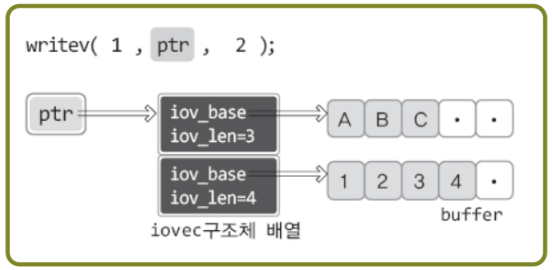
```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/iovec$ ./writev
ABC1234
Write bytes: 7
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/iovec$
```

```c
#include <sys/uio.h>

ssize_t writev(int filedes, const struct iovec * iov, int iovcnt);
```

➜ 성공 시 전송된 바이트 수, 실패 시 -1 반환

- filedes: 데이터 전송의 목적지를 나타내는 소켓의 파일 디스크립터 전달, 단 소켓에만 제한된 함수가 아니기 때문에, read 함수처럼 파일이나 콘솔 대상의 파일 디스크립터도 전달 가능하다.
- iov: 구조체 iovec 배열의 주소 값 전달, 구조체 iovec의 변수에는 전송할 데이터의 위치 및 크기 정보가 담긴다.
- iovcnt: 두 번째 인자로 전달된 주소 값이 가리키는 배열의 길이정보 전달.

둘 이상의 영역에 나뉘어 저장된 데이터를 묶어서 한번의 함수호출을 통해서 보낼 수 있다.




```c
struct iovec
{
    void * iov_base;  // 버퍼의 주소 정보
    size_t iov_len;   // 버퍼의 크기 정보
}
```

03 writev() & readv()

readv()

```c
int main(int argc, char *argv[])
{
    struct iovec vec[2];
    char buf1[BUF_SIZE]={0,};
    char buf2[BUF_SIZE]={0,};
    int str_len;

    vec[0].iov_base=buf1;
    vec[0].iov_len=5;
    vec[1].iov_base=buf2;
    vec[1].iov_len=BUF_SIZE;

    str_len=readv(0, vec, 2);
    printf("Read bytes: %d \n", str_len);
    printf("First message: %s \n", buf1);
    printf("Second message: %s \n", buf2);
    return 0;
}
```

0: stdin

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/iovec$ ./readv
I like TCP/IP socket programming~
Read bytes: 34
First message: I lik
Second message: e TCP/IP socket programming~
```

5글자

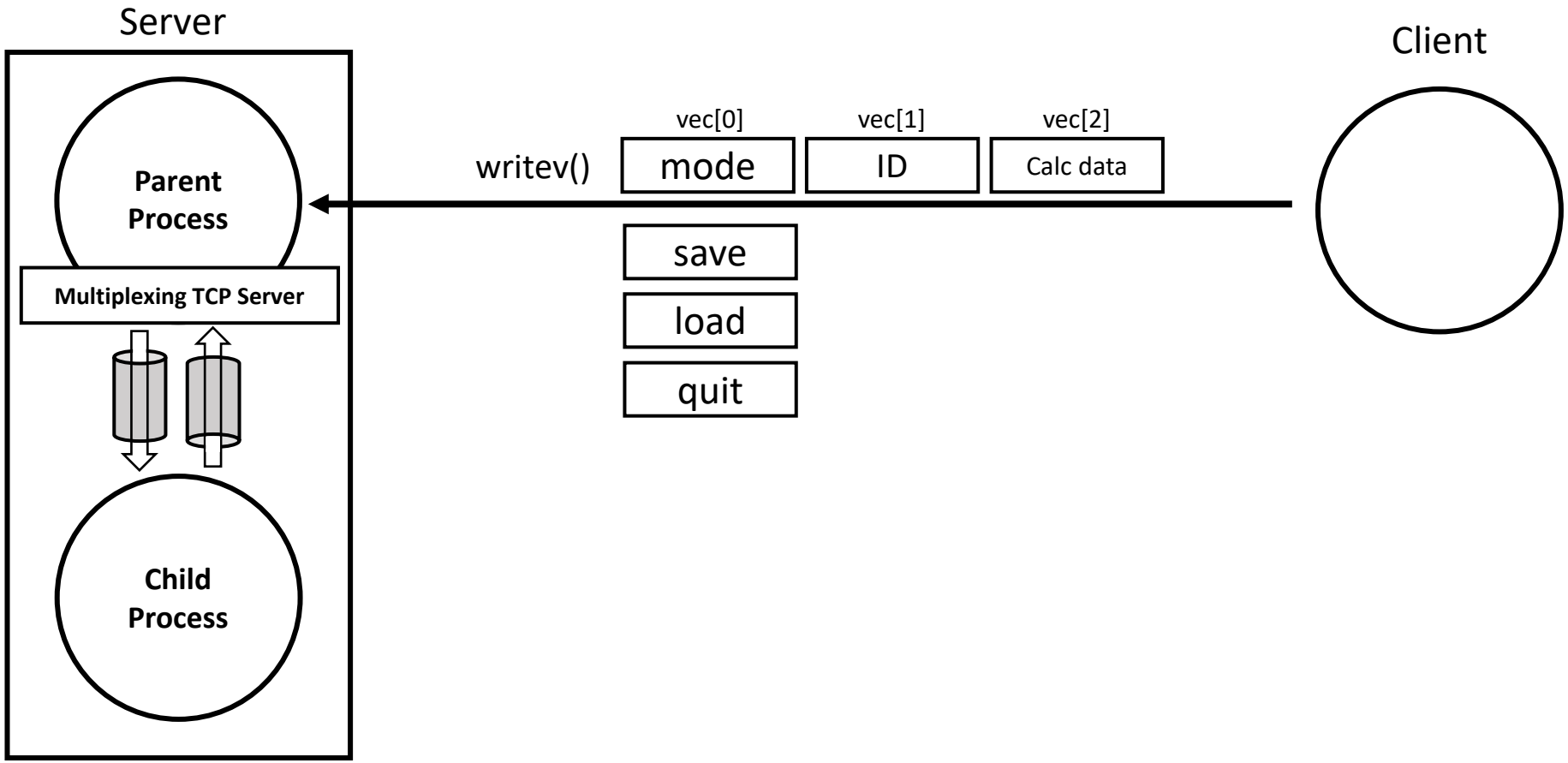
```c
#include <sys/uio.h>

ssize_t readv(int filedes, const struct iovec * iov, int iovcnt);
```

➜ 성공 시 수신된 바이트 수, 실패 시 -1 반환

- filedes — 데이터를 수신할 파일(혹은 소켓)의 파일 디스크립터를 인자로 전달.
- iov — 데이터를 저장할 위치와 크기 정보를 담고 있는 iovec 구조체 배열의 주소 값 전달.
- iovcnt — 두 번째 인자로 전달된 주소 값이 가리키는 배열의 길이정보 전달.

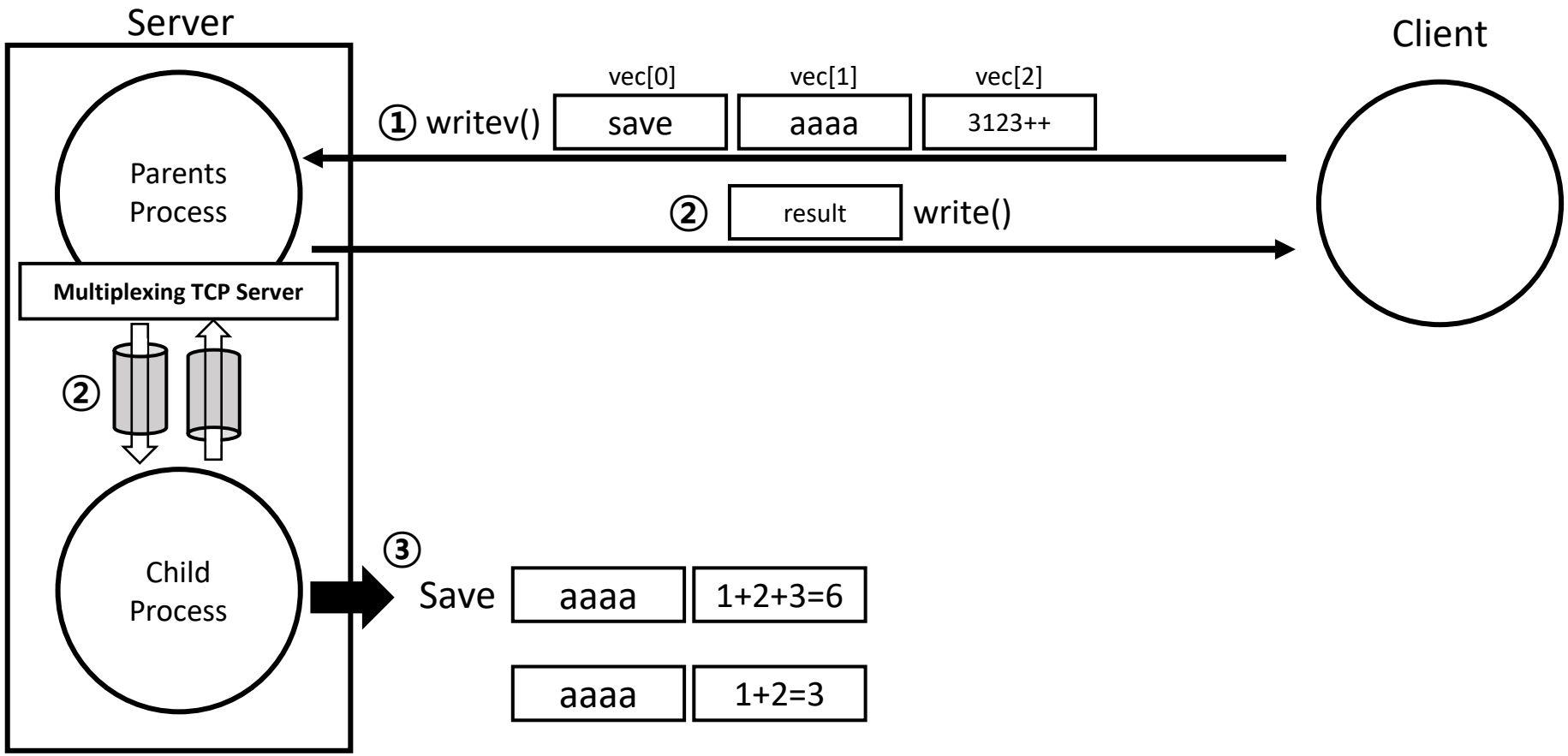
단 한번의 함수호출을 통해서 입력되는 데이터를 둘 이상의 영역에 나눠서 저장이 가능하다.

Multiplexing Calculator Server



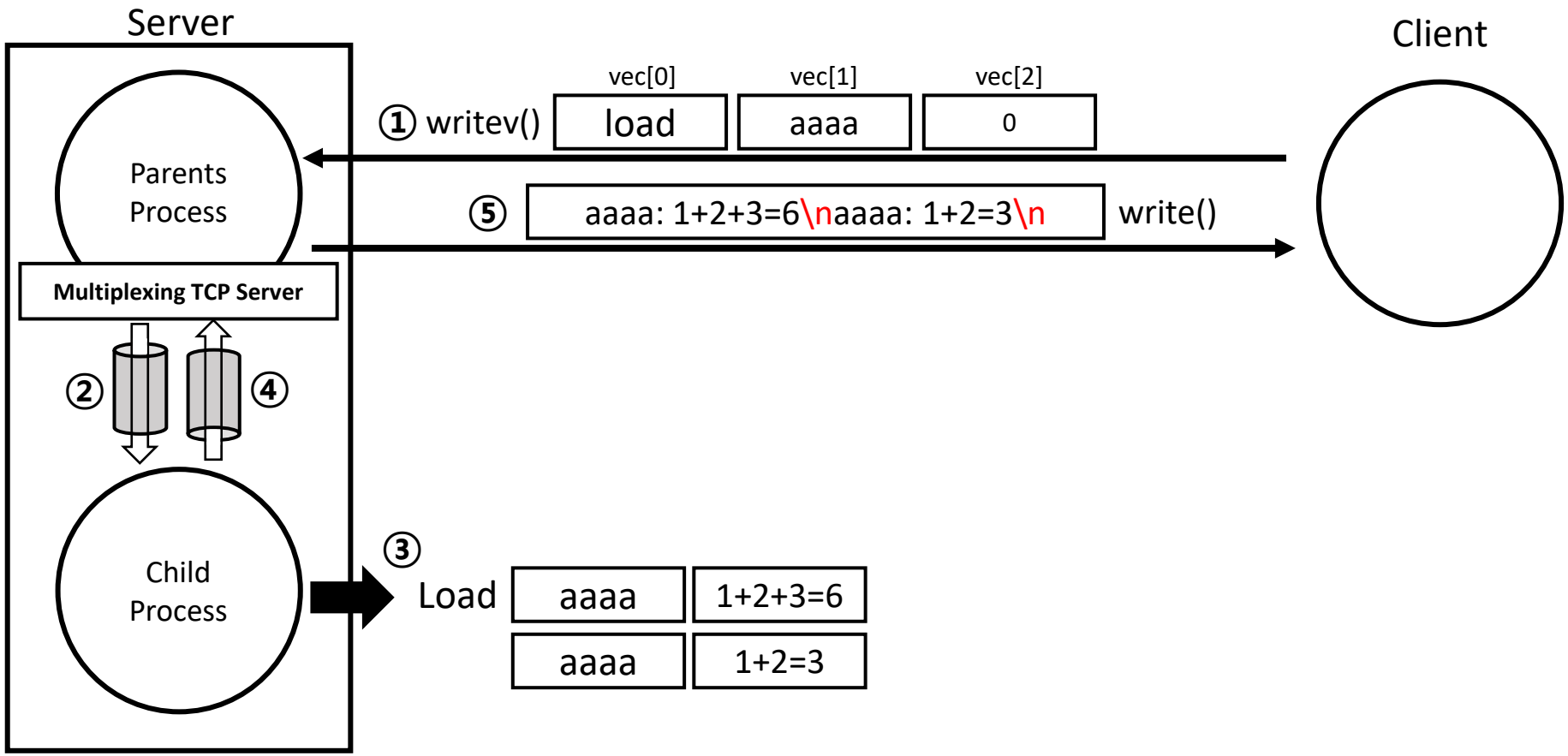
Multiplexing Calculator Server

- **save**



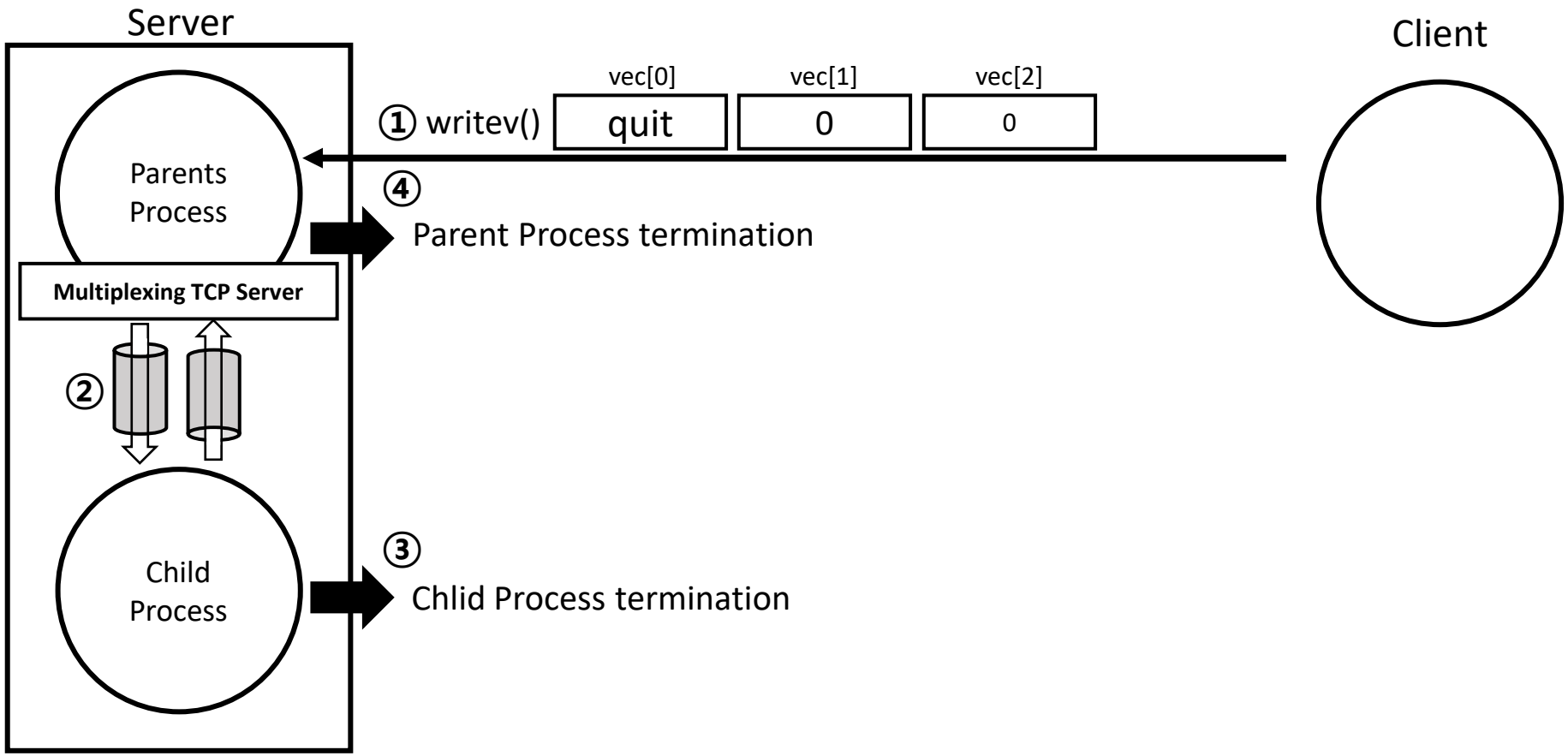
Multiplexing Calculator Server

- **load**



Multiplexing Calculator Server

- **quit**



04 Network assignment #6

Multiplexing Calculator Server

- **Result**

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./server 8080
connected client: 8
closed client: 8
save to aaaa
connected client: 8
save to aaaa
closed client: 8
connected client: 8
load from aaaa
closed client: 8
connected client: 8
quit
removed proc id: 16333
closed client: 8
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$
```

```
Operand count: 3
Operand 0: 1
Operand 1: 2
Operand 2: 3
Operator 0: +
Operator 1: +
Operation result: 6
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./client 8080 127.0.0.1
Mode: save
ID: aaaa
Operand count: 2
Operand 0: 3
Operand 1: 5
Operator 0: *
Operation result: 15
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./client 8080 127.0.0.1
Mode: load
ID: aaaa
aaaa: 1+2+3=6
aaaa: 3*5=15
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./client 8080 127.0.0.1
Mode: quit
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./server 8080
connected client: 8
closed client: 8
save to aaaa
connected client: 8
load from bbbb
closed client: 8
connected client: 8
load from aaaa
closed client: 8
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./client 8080 127.0.0.1
Mode: save
ID: aaaa
Operand count: 3
Operand 0: 1
Operand 1: 2
Operand 2: 3
Operator 0: +
Operator 1: +
Operation result: 6
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./client 8080 127.0.0.1
Mode: load
ID: bbbb
Not exist
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./client 8080 127.0.0.1
Mode: load
ID: aaaa
aaaa: 1+2+3=6
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$
```

04 Network assignment #6

Multiplexing Calculator Server

- 처리사항

1. mode가 의도한 문자 외의 값을 받을 시 아래와 같이 처리

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./client 8080 127.0.0.1
Mode: ddddd
supported mode: save load quit
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./client 8080 127.0.0.1
Mode: d
supported mode: save load quit
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$
```

2. ID의 길이가 4(bytes)가 아닐 시 아래와 같이 처리

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./client 8080 127.0.0.1
Mode: save
ID: aaaaa
Error: ID length must be 4
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$
```

04 Network assignment #6

Multiplexing Calculator Server

- 처리사항

3. Load하였을 때 해당 ID로 기록된 데이터가 없다면 Not exist를 전송

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./server 8080
connected client: 8
load from aaaa
closed client: 8
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./client 8080 127.0.0.1
Mode: load
ID: aaaa
Not exist
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$
```

4. Count count가 overflow가 일어나는 값을 받았다면, 클라이언트는 close()하고 종료

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./server 8080
connected client: 8
closed client: 8
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./client 8080 127.0.0.1
Mode: save
ID: aaaa
Operand count: 0
Overflow will happen(0)
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$
```

04 Network assignment #6

Multiplexing Calculator Server

- **client.c**

1. 클라이언트 실행 시 main 함수의 매개변수로 포트번호와 서버의 IP주소를 받아서 실행

   Ex) ./client 8080 127.0.0.1 (순서 확인)

2. 서버와 연결 전에 "Mode: "를 표준 출력하고 표준 입력으로 Mode 정보를 입력 받음

   1. 이 때 save, load, quit 이외의 값이 들어오면 아래와 같이 출력하고 종료

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./client 8080 127.0.0.1
Mode: ddddd
supported mode: save load quit
```

3. Mode가 Save와 load 였다면, "ID: "를 표준출력하고 id 정보를 입력받음

   1. 이 때 ID 값은 항상 문자열길이가 4가 되도록 받게하며, 이 조건에 맞지 않으면 아래와 같이 출력하고 종료

4. Mode가 Save일 경우 다음과 같이 동작함

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket6/multiplex_server$ ./client 8080 127.0.0.1
Mode: save
ID: aaaaa
Error: ID length must be 4
```

   1. Iovec을 활용하여 첫 번째 배열에는 mode, 두 번째 배열에는 ID를 넣음
   2. 세 번째 배열에는 기존 Network Programming #2 과제와 같이 계산 정보를 넣고 이를 writev()로 전송함
      1. 이 때 계산 정보 입력 양식 역시 Network Programming #2와 동일함
      2. 연산자 수 정보가 char 기준으로 0보다 작은 값을 받을 경우, Overflow will happen(연산자 수 정보)를 표준 출력하고 소켓을 닫고 종료함
   3. 서버에게 계산 결과를 받게 되면 이를 표준출력으로 Operation result: 와 함께 출력하고 소켓을 닫고 종료

04 Network assignment #6

Multiplexing Calculator Server

- **client.c**

1. Mode가 Load 일 경우 다음과 같이 동작함
    1. Save와 마찬가지로 Iovec의 첫 배열에 mode, 두 번째 배열에 ID를 입력하고, 세 번째 배열에 경우 사용하지 않으므로 특정한 값을 넣지 않고 writev()로 전송함
    2. 서버가 ID에 해당하는 계산 결과 정보들을 문자열로 보내주면 이를 read()로 받아 출력하고 소켓을 닫고 종료
2. Mode가 Quit일 경우 다음과 같이 동작함
    1. Iovec 첫 배열에 mode 정보를 입력하고, 두 번째 및 세 번째 배열은 이용하지 않으므로 값을 넣지 않고 writev()로 전송함
    2. 소켓을 닫고 종료

04 Network assignment #6

Multiplexing Calculator Server

- **server.c**

1. 서버 실행시 포트번호를 main 함수의 매개 변수로 받고 실행

   Ex) ./server 8080

2. pipe()를 두 개 생성하고, fork()를 진행함
   1. 자식 프로세스는 파이프를 통해 부모 프로세스가 보내는 데이터를 받을 때까지 대기함
   2. 부모 프로세스는 자식 프로세스가 좀비가 되지 않게 하기 위해 sigaction을 활용하여 자식 프로세스가 종료될 경우 “removed proc id: id값”을 표준 출력하도록 설정함
3. 부모 프로세스는 select()를 활용하여 Multiplexing 형태로 서버를 구축함
   1. Select()의 timeout은 5초로 설정
   2. 클라이언트가 연결되면 “connected client : file descriptor”를 표준 출력함
   3. 클라이언트와 연결이 종료되면 “closed client: file descriptor”를 표준 출력함
4. 클라이언트로부터 readv()를 통해 mode, ID, 계산 데이터 정보를 받음
5. Mode가 save일 경우 “save to ID정보”를 표준 출력하고 다음과 같이 동작함
   1. 계산 데이터를 기반으로 계산을 하고, 계산 정보와 계산 결과 값을 아래와 같은 수식의 형태로 저장을 함
      1. Ex) 3123++ -> 1+2+3=6
   2. 수식 형태의 문자열 데이터와 ID 정보를 파이프를 통해 자식 프로세스에게 전달함
   3. 자식 프로세스는 이 정보를 저장하고 다음 명령을 대기함
      1. 파일은 사용하지 않고 문자열 배열 형태로 이 데이터들을 저장해 둠 (즉, 서버 프로그램이 새로 실행되면 새롭게 저장 시작함)

04 Network assignment #6

Multiplexing Calculator Server

- **server.c**

1. Mode가 Load일 경우 “load from ID정보”를 표준 출력하고 다음과 같이 동작함
    1. Mode와 ID 정보를 파이프로 자식 프로세스에게 전달함
    2. ID 정보를 기반으로 현재 해당 ID로 계산했던 모든 정보를 찾아서 하나의 문자열 형태로 만들어 부모 프로세스에게 전달함

        Ex) aaaa : 1+2+3=6₩naaaa: 1+2=3₩n
    3. 부모 프로세스는 이 문자열 데이터를 클라이언트에게 전송함
        1. 만약 데이터가 없었을 경우 Not exist를 전송함
2. Mode가 Quit일 경우 “quit”을 표준 출력하고 다음과 같이 동작함
    1. 부모 프로세스는 파이프를 통해 mode 정보(quit)을 넘겨줌
    2. 자식 프로세스는 이 정보를 받으면 자신을 종료함
    3. 부모 프로세스도 더 이상 클라이언트의 요청을 받지 않고 종료함
    4. 자식 프로세스는 파이프를 통해 부모 프로세스가 보내는 데이터를 받을 때까지 대기함

04 Network assignment #6

Multiplexing Calculator Server

- 참고사항

1. 과제에서 의도한 대로 데이터를 주고받고 이를 출력하는 방식이 아닌, 겉으로 출력 결과만 똑같이 보인다면 점수 없음
2. 과제 관련 문의 : thdyd324@gmail.com

- 제출관련

1. 서버 프로그램은 server.c, 클라이언트 프로그램은 client.c로 명명하여 과제 진행
2. 빌드 시(gcc) Warning이 발생해서는 안됨
3. 제출 시 파일들을 “자신의 학번.tar” 파일로 제출

   Ex) 2020324067.tar

   ~/Workspace/socket1/(server.c, client.c)

```
smalldragon@DESKTOP-PMPPMHH:~/Workspace$ tar cvf 2020324067.tar -C socket1 server.c client.c
server.c
client.c
```

압축파일명 폴더명 파일명 파일명

4. 과제는 10점 만점
5. 제출 기한: 2023.05.05(금) PM 11:59
6. 지각 제출 허용: 2022.05.09(화) PM 11:59 / 하루 늦을 때 마다 2점 씩 감점

   지각제출 시 보낼 이메일: pjm9562@naver.com
7. 기한 안에 아예 제출을 하지 않았을 시 점수 없음