



ISL

• • •

Network Programming #5



ISL (IoT Standard Lab)

Index

1. **Zombie Process**
2. **Signal**
3. **Multi process Server & Client**
4. **Pipe**
5. **Network assignment #5**

01 Zombie Process

Zombie Process Check

```
socket > socket5 > zombie >  zombie.c >  main(int, char * [])
```

```
1  #include <stdio.h>
2  #include <unistd.h>
3
4  int main(int argc, char *argv[])
5  {
6      pid_t pid=fork();
7
8      if(pid==0)    // if Child Process
9      {
10         puts("Hi I'am a child process");
11     }
12     else
13     {
14         printf("Child Process ID: %d \n", pid);
15         sleep(30);
16     }
17
18     if(pid==0)
19         puts("End child process");
20     else
21         puts("End parent process");
22     return 0;
23 }
```

1. 부모가 Block되어 자식의 Return값을 받지 못하는 상태

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/zombie$ ./zombie
Hi I'am a child process
Child Process ID: 6473
End child process
```

실행 순서 보장 X

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/zombie$ ./zombie
Child Process ID: 6327
Hi I'am a child process
End child process
```

실행 순서 보장 X

End parent process

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL
smallldr+	13	0.0	0.0 2612 592 pts/0 S+ 10:59 0:00 sh /mnt/c/Users/KIMSOYONG
smallldr+	18	0.0	0.0 2612 524 pts/0 S+ 10:59 0:00 sh /home/smalldragon/.vsc
smallldr+	22	1.6	0.6 963604 101928 pts/0 Rl+ 10:59 0:11 /home/smalldragon/.vscod
smallldr+	33	0.1	0.2 845012 48424 pts/0 Rl+ 10:59 0:00 /home/smalldragon/.vscod
smallldr+	71	0.0	0.2 586284 41632 pts/1 Ssl+ 10:59 0:00 /home/smalldragon/.vscod
smallldr+	82	0.0	0.2 582480 35072 pts/2 Ssl+ 10:59 0:00 /home/smalldragon/.vscod
smallldr+	110	0.0	0.2 834276 42336 pts/0 Sl+ 10:59 0:00 /home/smalldragon/.vscod
smallldr+	146	0.5	0.7 984416 121944 pts/0 Sl+ 10:59 0:03 /home/smalldragon/.vscod
smallldr+	158	0.0	0.0 10252 5440 pts/3 Ss 10:59 0:00 /bin/bash
smallldr+	201	0.1	0.2 2933608 47496 pts/0 Sl+ 10:59 0:01 /home/smalldragon/.vscod
smallldr+	772	0.0	0.1 6431532 22396 pts/0 Sl+ 11:10 0:00 /home/smalldragon/.vscod
smallldr+	813	0.0	0.0 10008 5080 pts/4 Ss+ 11:10 0:00 /usr/bin/bash
smallldr+	832	0.0	0.0 2492 576 pts/3 S+ 11:10 0:00 ./zombie
smallldr+	833	0.0	0.0 0 0 pts/3 Z+ 11:10 0:00 [zombie] <defunct>
smallldr+	895	0.5	0.0 10008 5024 pts/5 Ss 11:10 0:00 /usr/bin/bash
smallldr+	905	0.0	0.0 10616 3232 pts/5 R+ 11:10 0:00 ps au

```
smalldragon@SD-DESKTOP:~/Workspace$
```

01 Zombie Process

wait()

```
socket > socket5 > zombie > C wait.c > ...
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5
6  int main(int argc, char *argv[])
7  {
8      int status;
9      pid_t pid=fork();
10
11     if(pid==0)
12     {
13         return 3;
14     }
15     else
16     {
17         printf("Child PID: %d \n", pid);
18         pid=fork();
19         if(pid==0)
20         {
21             exit(7);
22         }
23         else
24         {
25             printf("Child PID: %d \n", pid);
26             wait(&status);
27             if(WIFEXITED(status))
28                 printf("Child send one: %d \n", WEXITSTATUS(status));
29
30             wait(&status);
31             if(WIFEXITED(status))
32                 printf("Child send two: %d \n", WEXITSTATUS(status));
33             sleep(30); // Sleep 30 sec.
34         }
35     }
36     return 0;
37 }
```

자식이 정상 종료하면 true

자식이 종료될 때까지 Block

종료된 자식이 반환한 값

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/zombie$ ./wait
Child PID: 1241
Child PID: 1242
Child send one: 3
Child send two: 7
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/zombie$
```

01 Zombie Process

waitpid()

socket > socket5 > zombie > C waitpid.c > ...

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <sys/wait.h>
4
5  int main(int argc, char *argv[])
6  {
7      int status;
8      pid_t pid=fork();
9
10     if(pid==0)
11     {
12         sleep(15);
13         return 24;
14     }
15     else
16     {
17         while(!waitpid(-1, &status, WNOHANG))
18         {
19             sleep(3);
20             puts("sleep 3sec.");
21         }
22
23         if(WIFEXITED(status))
24             printf("Child send %d \n", WEXITSTATUS(status));
25     }
26     return 0;
27 }
```

Non-Block 상태로 임의의 자식 프로세스 종료대기

```
#include <sys/wait.h>
```

```
pid_t waitpid(pid_t pid, int * statloc, int options);
```

→ 성공 시 종료된 자식 프로세스의 ID(또는 0), 실패 시 -1 반환

- pid 종료 확인하고자 하는 자식 프로세스의 ID 전달, 이를 대신해서 -1을 전달하면 wait 함수와 마찬가지로 임의의 자식 프로세스가 종료되기를 기다린다.
- statloc wait 함수의 매개변수 statloc과 동일한 의미로 사용된다.
- options 헤더파일 sys/wait.h에 선언된 상수 WNOHANG을 인자로 전달하면, 종료된 자식 프로세스가 존재하지 않아도 블로킹 상태에 있지 않고, 0을 반환하면서 함수를 빠져 나온다.

G CONSOLE TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/zombie$ ./waitpid
sleep 3sec.
sleep 3sec.
sleep 3sec.
sleep 3sec.
sleep 3sec.
Child send 24
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/zombie$
```

02 Signal

signal()

socket > socket5 > signal > C signal.c > ...

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <signal.h>
4
5  void timeout(int sig)
6  {
7      if(sig==SIGALRM)
8          puts("Time out!");
9
10     alarm(2);
11 }
12 void keycontrol(int sig)
13 {
14     if(sig==SIGINT)
15         puts("CTRL+C pressed");
16 }
17
18 int main(int argc, char *argv[])
19 {
20     int i;
21     signal(SIGALRM, timeout);
22     signal(SIGINT, keycontrol);
23     alarm(2);
24
25     for(i=0; i<3; i++)
26     {
27         puts("wait...");
28         sleep(100);
29     }
30     return 0;
31 }
```

2초 뒤에 SIGALRM 시그널 발생

첫 번째 매개변수에 해당하는
시그널이 발생하면,
두 번째 매개변수에 적힌 함수 호출

alarm()으로 SIGALRM 시그널
발생을 시키면 깨어남

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/signal$ ./signal
wait...
Time out!
wait...
Time out!
wait...
Time out!
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/signal$
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/signal$ ./signal
wait...
^CCTRL+C pressed
wait...
Time out!
wait...
Time out!
```

02 Signal

sigaction()

socket > socket5 > signal > C sigaction.c > ...

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <signal.h>
4
5  void timeout(int sig)
6  {
7      if(sig==SIGALRM)
8          puts("Time out!");
9      alarm(2);
10 }
11
12 int main(int argc, char *argv[])
13 {
14     int i;
15     struct sigaction act;
16     act.sa_handler=timeout;
17     sigemptyset(&act.sa_mask);
18     act.sa_flags=0;
19     sigaction(SIGALRM, &act, 0);
20
21     alarm(2);
22
23     for(i=0; i<3; i++)
24     {
25         puts("wait...");
26         sleep(100);
27     }
28     return 0;
29 }
```

sa_handler에 호출할 함수 저장

sa_mask를 0으로 초기화

```
struct sigaction
{
    void (*sa_handler)(int);
    sigset_t sa_mask;
    int sa_flags;
}
```

sigaction 구조체 변수를 선언해서, 시그널 등록 시 호출될 함수의 정보를 채워서 위의 함수 호출 시 인자로 전달한다. sa_mask의 모든 비트는 0, sa_flags는 0으로 초기화! 이들은 시그널관련 정보의 추가 전달에 사용되는데, 좀비의 소멸을 목적으로는 사용되지 않는다.

```
#include <signal.h>
```

```
int sigaction(int signo, const struct sigaction * act, struct sigaction * oldact);
```

→ 성공 시 0, 실패 시 -1 반환

- signo signal 함수와 마찬가지로 시그널의 정보를 인자로 전달.
- act 첫 번째 인자로 전달된 상수에 해당하는 시그널 발생시 호출될 함수(시그널 핸들러)의 정보 전달.
- oldact 이전에 등록되었던 시그널 핸들러의 함수 포인터를 얻는데 사용되는 인자, 필요 없다면 0 전달.

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/signal$ ./sigaction
wait...
Time out!
wait...
Time out!
wait...
Time out!
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/signal$
```

Multi Process TCP Server

- 시그널을 활용한 좀비 프로세스 대처

```
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);
```

SIGCHLD: 자식이 종료되었음을
알려주는 시그널

```
void read_childproc(int sig)
{
    pid_t pid;
    int status;
    pid=waitpid(-1, &status, WNOHANG);
    printf("removed proc id: %d \n", pid);
}
```

1. Non Blocking으로 종료 신호 대기
2. 자식의 반환 값을 받아 좀비 프로세스 소멸

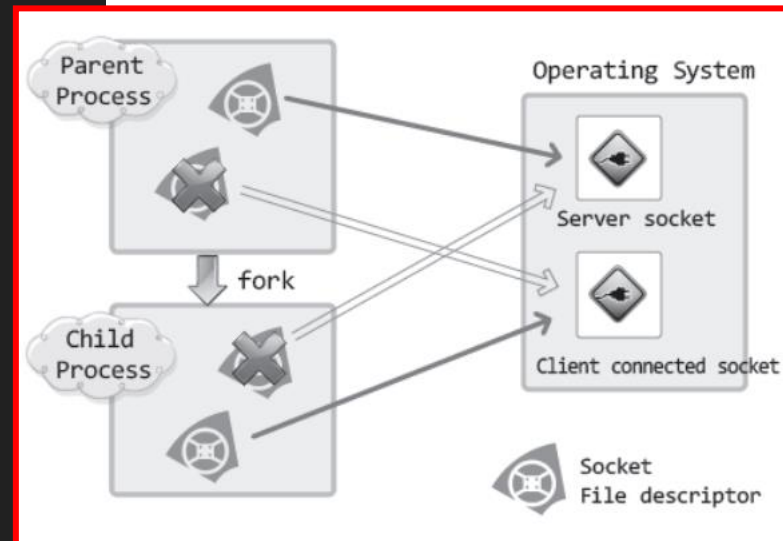
03 Multi Process Server & Client

Multi Process TCP Server

- fork() 이후 소켓 디스크립터 처리

```
while(1)
{
    adr_sz=sizeof(clnt_adr);
    clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_adr, &adr_sz);
    if(clnt_sock==-1)
        continue;
    else
        puts("new client connected...");
    pid=fork();
    if(pid==-1)
    {
        close(clnt_sock);
        continue;
    }
    if(pid==0)
    {
        close(serv_sock);
        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;
```



03 Multi Process Server & Client

Multi Process TCP Server

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/mpserver$ ./eclient 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/socket5/mpserver$
```

Client 1

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/mpserver$ ./eclient 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/socket5/mpserver$
```

Client 2

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/mpserver$ ./mpserver 8080
new client connected...
new client connected...
client disconnected...
removed proc id: 3331
client disconnected...
removed proc id: 3296

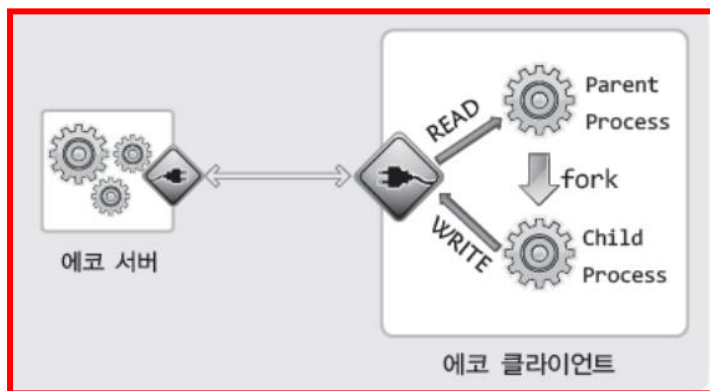
```

Server

03 Multi Process Server & Client

Multi Process TCP Client

- 입출력 루틴 분할



```
sock=socket(PF_INET, SOCK_STREAM, 0);
memset(&serv_addr, 0, sizeof(serv_addr));
serv_addr.sin_family=AF_INET;
serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
serv_addr.sin_port=htons(atoi(argv[2]));

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

pid=fork();
if(pid==0)
    write_routine(sock, buf);
else
    read_routine(sock, buf);

close(sock);
return 0;
```

```
void write_routine(int sock, char *buf)
{
    while(1)
    {
        fgets(buf, BUF_SIZE, stdin);
        if(!strcmp(buf, "q\n") || !strcmp(buf, "Q\n"))
        {
            shutdown(sock, SHUT_WR);
            return;
        }
        write(sock, buf, strlen(buf));
    }
}
```

```
void read_routine(int sock, char *buf)
{
    while(1)
    {
        int str_len=read(sock, buf, BUF_SIZE);
        if(str_len==0)
            return;

        buf[str_len]=0;
        printf("Message from server: %s", buf);
    }
}
```

04 Pipe

pipe() #1

```
socket > socket5 > pipe > C pipe1.c > main(int, char * [])
1  #include <stdio.h>
2  #include <unistd.h>
3  #define BUF_SIZE 30
4
5  int main(int argc, char *argv[])
6  {
7      int fds[2];
8      char str[]="Who are you?";
9      char buf[BUF_SIZE];
10     pid_t pid;
11
12     pipe(fds);
13     pid=fork();
14     if(pid==0)
15     {
16         write(fds[1], str, sizeof(str));
17     }
18     else
19     {
20         read(fds[0], buf, BUF_SIZE);
21         puts(buf);
22     }
23     return 0;
24 }
```

데이터 송신

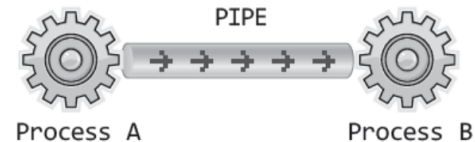
데이터 수신

```
#include <unistd.h>
```

```
int pipe(int fildes[2]);
```

→ 성공 시 0, 실패 시 -1 반환

- fildes[0] 파이프로부터 데이터를 수신하는데 사용되는 파일 디스크립터가 저장된다. 즉, fildes[0]은 파이프의 출구가 된다.
- fildes[1] 파이프로 데이터를 전송하는데 사용되는 파일 디스크립터가 저장된다. 즉, fildes[1]은 파이프의 입구가 된다.



위의 함수가 호출되면, 운영체제는 서로 다른 프로세스가 함께 접근할 수 있는 메모리 공간을 만들고, 이 공간의 접근에 사용되는 파일 디스크립터를 반환한다.

```
smalldragon@SD-DESKTOP: ~/Workspace/socket/socket5/pipe$ ./pipe1
Who are you?
smalldragon@SD-DESKTOP: ~/Workspace/socket/socket5/pipe$
```

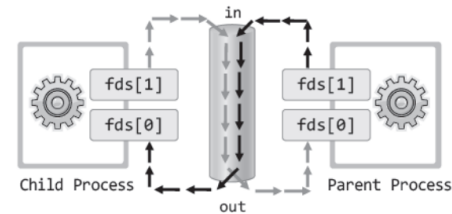
04 Pipe

pipe() #2

```
5  int main(int argc, char *argv[])
6  {
7      int fds[2];
8      char str1[]="Who are you?";
9      char str2[]="Thank you for your message";
10     char buf[BUF_SIZE];
11     pid_t pid;
12
13     pipe(fds);
14     pid=fork();
15
16     if(pid==0)
17     {
18         write(fds[1], str1, sizeof(str1));
19         sleep(2);
20         read(fds[0], buf, BUF_SIZE);
21         printf("Child proc output: %s \n", buf);
22     }
23     else
24     {
25         read(fds[0], buf, BUF_SIZE);
26         printf("Parent proc output: %s \n", buf);
27         write(fds[1], str2, sizeof(str2));
28         sleep(3);
29     }
30     return 0;
31 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/pipe$ ./pipe2
Parent proc output: Who are you?
Child proc output: Thank you for your message
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/pipe$
```



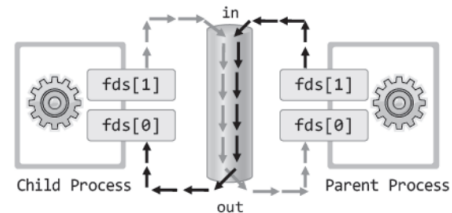
하나의 파일을 이용해서 양방향 통신을 하는 경우, 데이터를 쓰고 읽는 타이밍이 매우 중요해진다. 그런데 이를 컨트롤 하는 것은 사실상 불가능하기 때문에 이는 적절한 방법이 될 수 없다. 왼쪽의 예제에서 sleep 함수의 호출문을 주석처리 해 버리면 문제가 있음을 쉽게 확인할 수 있다.

04 Pipe

pipe() #2

```
5 int main(int argc, char *argv[])
6 {
7     int fds[2];
8     char str1[]="Who are you?";
9     char str2[]="Thank you for your message";
10    char buf[BUF_SIZE];
11    pid_t pid;
12
13    pipe(fds);
14    pid=fork();
15
16    if(pid==0)
17    {
18        write(fds[1], str1, sizeof(str1));
19        read(fds[0], buf, BUF_SIZE);
20        printf("Child proc output: %s \n", buf);
21    }
22    else
23    {
24        read(fds[0], buf, BUF_SIZE);
25        printf("Parent proc output: %s \n", buf);
26        write(fds[1], str2, sizeof(str2));
27        sleep(3);
28    }
29
30    return 0;
31 }
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/pipe$ ./pipe2
Child proc output: Who are you?
```

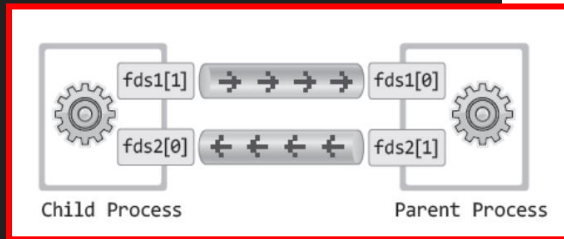


하나의 파이프를 이용해서 양방향 통신을 하는 경우, 데이터를 쓰고 읽는 타이밍이 매우 중요해진다. 그런데 이를 컨트롤 하는 것은 사실상 불가능하기 때문에 이는 적절한 방법이 될 수 없다. 왼쪽의 예제에서 sleep 함수의 호출문을 주석처리 해 버리면 문제가 있음을 쉽게 확인할 수 있다.

04 Pipe

pipe() #3

```
5 int main(int argc, char *argv[])
6 {
7     int fds1[2], fds2[2];
8     char str1[]="Who are you?";
9     char str2[]="Thank you for your message";
10    char buf[BUF_SIZE];
11    pid_t pid;
12
13    pipe(fds1), pipe(fds2);
14    pid=fork();
15
16    if(pid==0)
17    {
18        write(fds1[1], str1, sizeof(str1));
19        read(fds2[0], buf, BUF_SIZE);
20        printf("Child proc output: %s \n", buf);
21    }
22    else
23    {
24        read(fds1[0], buf, BUF_SIZE);
25        printf("Parent proc output: %s \n", buf);
26        write(fds2[1], str2, sizeof(str2));
27        sleep(3);
28    }
29    return 0;
30 }
```



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/pipe$ ./pipe3
Parent proc output: Who are you?
Child proc output: Thank you for your message
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/pipe$
```

04 Pipe

Pipe TCP Server

- 부모 프로세스는 클라이언트 메시지를 받아서 처리하고, 자식 프로세스는 클라이언트의 메시지를 기록

```
30 act.sa_handler=read_childproc;
31 sigemptyset(&act.sa_mask);
32 act.sa_flags=0;
33 state=sigaction(SIGCHLD, &act, 0);

41 if(bind(serv_sock, (struct sockaddr*) &serv_addr, sizeof(serv_addr))==-1)
42     error_handling("bind() error");
43 if(listen(serv_sock, 5)==-1)
44     error_handling("listen() error");
45
46 pipe(fds);
47 pid=fork();
48 if(pid==0)
49 {
50     FILE * fp=fopen("echomsg.txt", "wt");
51     char msgbuf[BUF_SIZE];
52     int i, len;
53
54     for(i=0; i<10; i++)
55     {
56         len=read(fds[0], msgbuf, BUF_SIZE);
57         fwrite((void*)msgbuf, 1, len, fp);
58     }
59     fclose(fp);
60     return 0;
61 }
```

Zombie 처리

Block

Child Process

```
63 while(1)
64 {
65     adr_sz=sizeof(clnt_addr);
66     clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_addr, &adr_sz);
67     if(clnt_sock==-1)
68         continue;
69     else
70         puts("new client connected...");
71
72     pid=fork();
73     if(pid==0)
74     {
75         close(serv_sock);
76         while((str_len=read(clnt_sock, buf, BUF_SIZE))!=0)
77         {
78             write(clnt_sock, buf, str_len);
79             write(fds[1], buf, str_len);
80         }
81
82         close(clnt_sock);
83         puts("client disconnected...");
84         return 0;
85     }
86     else
87         close(clnt_sock);
88 }
89 close(serv_sock);
90 return 0;
```

멀티 프로세스 서버

Echo

Pipe Write

Parent Process


04 Pipe

Pipe TCP Server

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/pipe$ ./pipeserver 8080
new client connected...
removed proc id: 5597
client disconnected...
removed proc id: 5713
^C
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/pipe$
```

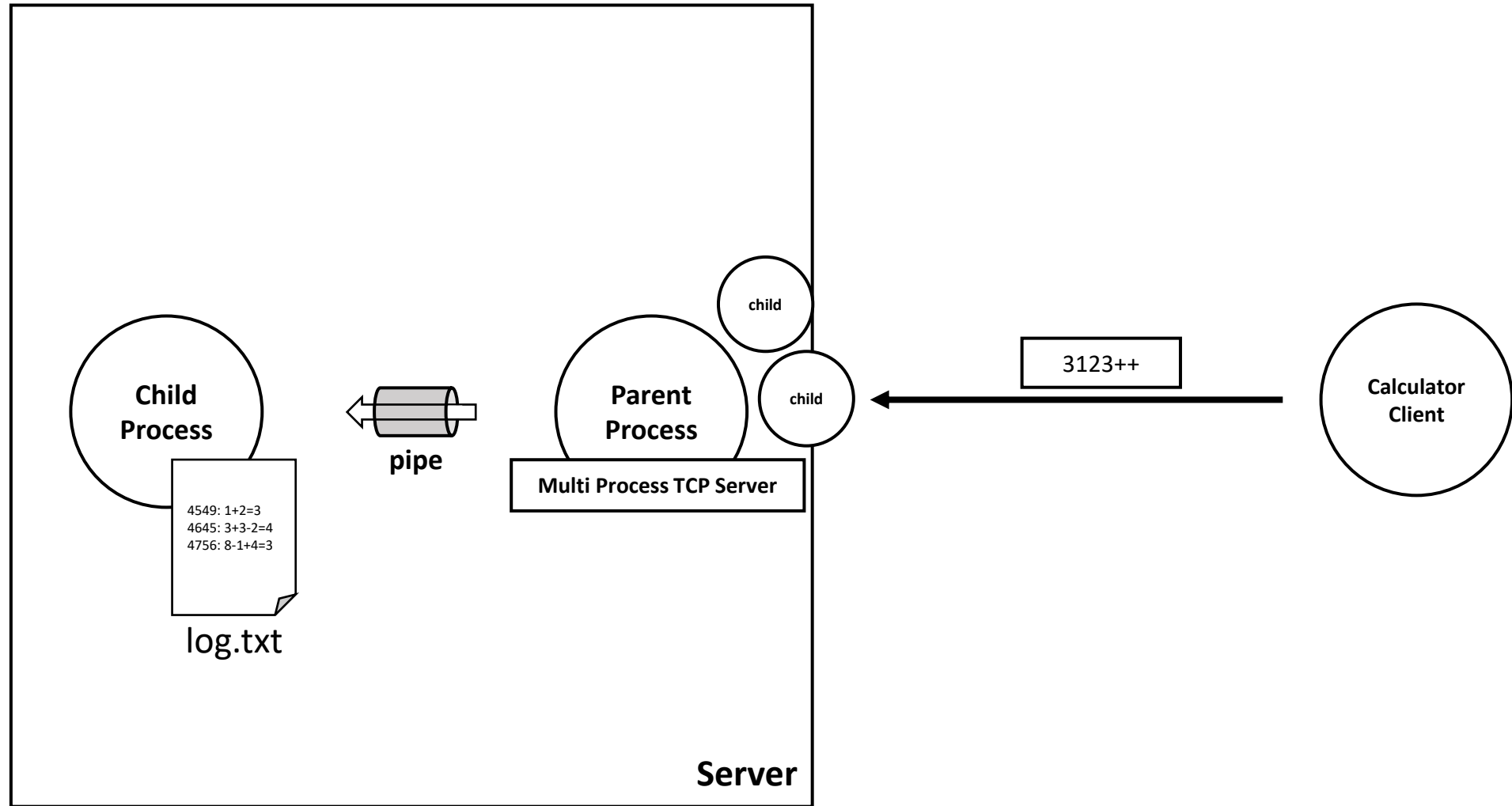
```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/mpserver$ ./eclient 127.0.0.1 8080
Connected.....
Input message(Q to quit): Hello
Message from server: Hello
Input message(Q to quit): How are you?
Message from server: How are you?
Input message(Q to quit): Good
Message from server: Good
Input message(Q to quit): Well
Message from server: Well
Input message(Q to quit): um..
Message from server: um..
Input message(Q to quit): what?
Message from server: what?
Input message(Q to quit): No
Message from server: No
Input message(Q to quit): Gooooood
Message from server: Gooooood
Input message(Q to quit): Wow
Message from server: Wow
Input message(Q to quit): Bye
Message from server: Bye
Input message(Q to quit): q
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/mpserver$
```

socket > socket5 > pipe >  echomsg.txt

```
1 Hello
2 How are you?
3 Good
4 Well
5 um..
6 what?
7 No
8 Gooooood
9 Wow
10 Bye
11
```

04 Pipe

Multi Process Calculator Server



Multi Process Calculator Server

- server.c

1. 서버 실행 시 main 함수의 매개변수로 포트번호를 받아서 실행

Ex) ./server 8080

2. INADDR_ANY와 매개변수로 받은 포트번호로 소켓 바인드 진행

3. sigaction()을 통해 서비스를 제공하는 동안 자식 프로세스가 종료되면, 자식이 종료되었다는 메시지와 함께 그 process id를 출력 (Ex. Removed proc id: 9323)

4. 파이프를 만들고 자식 프로세스를 생성

1. 이 때 자식 프로세스는 "log.txt" 파일을 생성 및 오픈하고 파이프를 통해 들어오는 데이터를 받을 때까지 대기

5. 서버는 Multi process Server 형태로 구현

1. 클라이언트의 요청을 받으면 new client connected...를 표준 출력하고 자식 프로세스를 생성하여 그 클라이언트에게 서비스를 제공할 수 있도록 구현

1. 이 때 서비스는 앞서 했던 Network Programming #2 TCP Calculator 과제와 같음

2. 자식 프로세스는 클라이언트가 요청한 정보를 기반으로 계산하고 결과를 클라이언트에게 보내주고 표준출력 후, 파이프를 통해 앞서 만든 자식 프로세스에게 이 계산

정보를 자신의 process id와 함께 전달하며, 이 정보를 log.txt에 저장함

1. 이 때 보내는 양식은 다음과 같음

2. Ex) process id: 3+4-2=5

6. 클라이언트가 전송한 연산자 수 정보가 char 기준으로 0보다 작은 수를 보냈다면 서버는 Save file(연산자 수) 메시지를 출력하고, 파일을 담당하던 자식 프로세스는 파일을 닫고 종료함. 따라서 이 후에 전송된 계산 요청에 대한 정보는 저장하지 않음

1. 지난 번 과제와는 달리 서버가 종료되지 않음. 자식 프로세스를 종료하였으므로 이후에 받은 계산요청은 기록되지 않음

Multi Process Calculator Server

- client.c (과제2와 동일)

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

Ex) ./client 8080 127.0.0.1 (순서 확인)

2. 매개변수를 활용하여 서버에게 연결 요청

3. 표준입력을 통해 operand count와 이 수만큼 operand를 받고 (operand count)-1 의 수만큼 operator(1바이트)를 입력 받음

1. 이 때 operator는 +, -, *로 한정

2. Operand count를 입력 받을 시 표준출력으로 *Operand count:* 를 출력

3. Operand와 Operator 입력을 받을 시 표준출력으로 각 이름들과 함께 입력을 받는 순으로 0부터 번호를 같이 출력

1. Operand 0: ,Operand 1:

2. Operator 0: ,Operator 1:

4. 표준입력으로 받은 데이터를 char 배열로 받아 한번에 전송

1. 서버에게 operand count를 1바이트로 보내고 이 수만큼 operand를 4바이트로 전송하고, (operation count)-1 의 수만큼 operator(1바이트)를 전송

Ex) (3 | 4, 5, 7 | +, -) 형태를 char 배열을 통해 한번에 write()로 전송

5. 결과를 받게 되면 이를 표준출력으로 Operation result: 와 함께 출력하고 소켓을 닫고 종료

6. operand count를 전송할 때 char 기준 0보다 작거나 같은 값을 전송하는 케이스의 경우, 이 값을 서버에게 전송하고 추가적인 표준입력없이 소켓을 닫고 종료

05 Network assignment #5

Multi Process Calculator Server

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/multi_calc$ ./client 8080 127.0.0.1
Operand count: 2
Operand 0: 1
Operand 1: 2
Operator 0: +
Operation result: 3
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/multi_calc$ ./client 8080 127.0.0.1
Operand count: 0
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/multi_calc$
```


```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/multi_calc$ ./client 8080 127.0.0.1
Operand count: 4
Operand 0: 3
Operand 1: 2
Operand 2: 3
Operand 3: 4
Operator 0: +
Operator 1: -
Operator 2: +
Operation result: 6
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/multi_calc$
```

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/multi_calc$ ./client 8080 127.0.0.1
Operand count: 5
Operand 0: 3
Operand 1: 2
Operand 2: 3
Operand 3: 4
Operand 4: 4
Operator 0: *
Operator 1: +
Operator 2: +
Operator 3: -
Operation result: 9
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/multi_calc$
```

05 Network assignment #5

Multi Process Calculator Server

```
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/multi_calc$ ./server 8080
new client connected...
4594: 1+2=3
removed proc id: 4594
new client connected...
4682: 3+2-3+4=6
removed proc id: 4682
new client connected...
4768: 3*2+3+4-4=9
removed proc id: 4768
new client connected...
Save File(0)
removed proc id: 4531
^C
smalldragon@SD-DESKTOP:~/Workspace/socket/socket5/multi_calc$
```

```
socket > socket5 > multi_calc >  log.txt
```

```
1    4594: 1+2=3
2    4682: 3+2-3+4=6
3    4768: 3*2+3+4-4=9
4
```

Multi Process Calculator Server

- 참고사항

1. 서버가 의도한 것 이외의 값을 받는 케이스를 예외처리할 필요 없음
2. 과제에서 의도한 대로 데이터를 주고받고 이를 출력하는 방식이 아닌, 겉으로 출력 결과만 똑같이 보인다면 점수 없음
3. 빌드 시(gcc) Warning이 발생해서는 안됨. 점수 없음
4. 과제 관련 문의 : thdyd324@gmail.com

- 제출관련

1. 서버 프로그램은 server.c, 클라이언트 프로그램은 client.c로 명명하여 과제 진행
2. 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
```

압축파일명	폴더명	파일명	파일명
2020324067.tar	socket1	server.c	client.c

3. 과제는 10점 만점
4. 제출 기한: 2023.04.28(금) PM 11:59
5. 지각 제출 허용: 2023.05.02(화) PM 11:59 / 하루 늦을 때 마다 2점 씩 감점
지각제출 시 보낼 이메일: eunjia24@gmail.com
6. 기한 안에 아예 제출을 하지 않았을 시 점수 없음