<Assignment3-2>

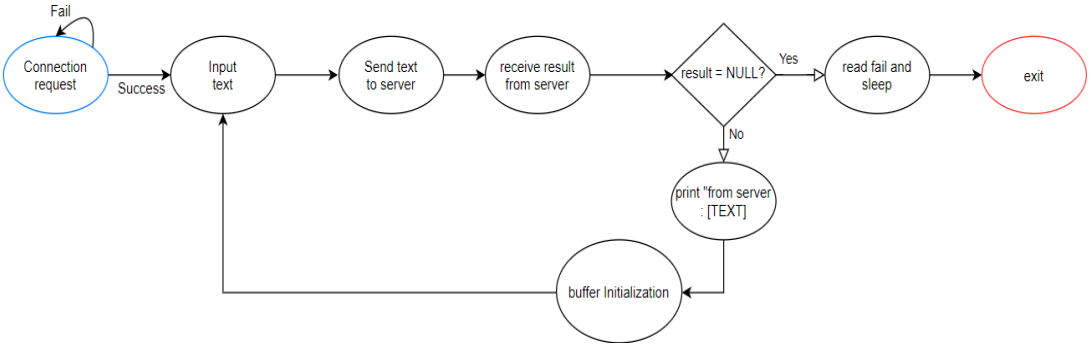
<table>
<tr><td>과제명</td><td colspan="4">Assignment3-2ㅁ
(Advanced echo Server)</td></tr>
<tr><td>과목명</td><td colspan="4">시스템프로그래밍(H020-3-0922-01)(화 5 목 6)</td></tr>
<tr><td>성명</td><td>정승훈</td><td rowspan="2">연락처</td><td>핸드폰</td><td>010–8648-7561</td></tr>
<tr><td>학과</td><td>전자통신공학과</td><td>이메일</td><td>tiktaktok116@naver.com</td></tr>
<tr><td>학번</td><td>2015707003</td><td>지도교수</td><td colspan="2">김태석 교수님</td></tr>
<tr><td>개발기간</td><td colspan="4">2020.05.24
~
2020.05.28</td></tr>
<tr><td rowspan="3">개발 환경</td><td>OS</td><td colspan="3">Linux (Ubuntu 18.04 LTS)</td></tr>
<tr><td>Language</td><td colspan="3">C</td></tr>
<tr><td>Development Tools</td><td colspan="3">Visual Studio Code,
gcc compiler</td></tr>
<tr><td>과제 요구사항
및 구현 내용</td><td colspan="4">1. Client ➔ 서버와의 연결이 성공하면, 아래와 같은 행위를 반복함.
단, 아래 동작 중 하나라도 실패하면 서버와의 연결 종료 후 해당 프로세스 종료.
- 사용자로부터 문자열 입력 받음.
- 서버로 해당 문자열 보냄.
- 서버로부터 문자열 받아서 그대로 출력.
2. Server ➔ Client와 연결된 후, 새로운 프로세스를 생성하여 다음과 같은 작업을 각각 수행.
- Parent Process ➔ 연결된 Client의 정보 출력(IP,PORT) 및 다음 Client와 연결 준비.
- 단, 해당 문자열이 "QUIT" 일 경우, 해당 Client와 연결 종료 및 SIGALRM Signal을 호출하여 해당 프로세스도 종료
- Process가 새로 생성될 때 마다 해당 Process의 PID 출력</td></tr>
</table>

I. Introduction

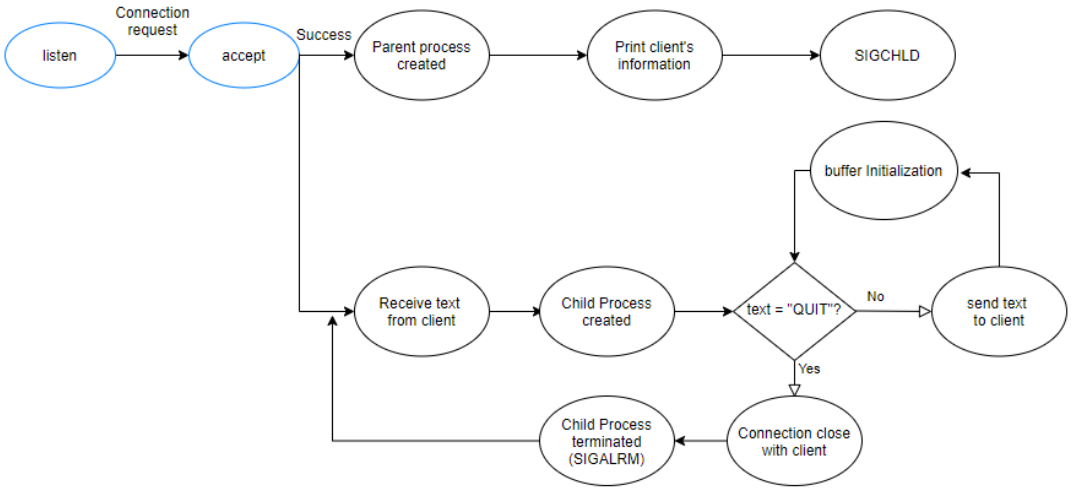
이번 Assignment3-2에서는 Advanced echo server를 구현하는 것이다. client에서 text 내용을 입력하면 그 정보를 server에 보낸 후 text를 그대로 다시 client로 가져와 출력한다. server에서는 client에서 text를 보낼 때 child process를 생성한 후 parent와 child에서 각각의 역할수행이 이루어진다. 보낸 text가 QUIT일 때 Parent에서 child status 판단, Child process terminated 처리를 구현하고 다중 접속할 때도 추가하였다. 쉽게 확인하기 위해서 loopback “127.0.0.1”을 사용하였다.

II. Flow Chart

Client's Flow Chart



Server's Flow Chart



III. Source Code

1) Client

```
1  ////////////////////////////////////////////////////////////////////////////
2  // File Name      : cli.c                                                 //
3  // Date           : 2020/05/24 ~ 2020/05/28                               //
4  // OS             : Ubuntu 18.04.4 LTS                                    //
5  // Student Name   : Seung Hoon Jeong                                      //
6  // Student ID     : 2015707003                                            //
7  // ---------------------------------------------------------------------- //
8  // Title : System Programming Assignment #3-1                             //
9  // Description : Advanced echo Client                                     //
10 ////////////////////////////////////////////////////////////////////////////
11
12 #include <stdio.h>
13 #include <stdlib.h>
14 #include <string.h>
15 #include <unistd.h>
16 #include <arpa/inet.h>
17 #include <sys/types.h>
18 #include <sys/socket.h>
19 #include <netinet/in.h>
20 #include <sys/wait.h>
21 #include <signal.h>
22
23 #define BUF_SIZE 256
24
25 ////////////////////////////////////////////////////////////////////////////////////////////////
26 // Function : int main(int argc, char **argv)                                                 //
27 // ========================================================================================== //
28 // Input: argument on kernel,> [TEXT] or > QUIT                                               //
29 // Output: 1.from server:[TEXT]                                                               //
30 //         2.quit : Program quit                                                              //
31 // ========================================================================================== //
32 // Purpose: Send text and print Receiving data                                                //
33 //          if you input 'QUIT', client connection close after child process end              //
34 ////////////////////////////////////////////////////////////////////////////////////////////////
35 int main(int argc, char **argv)
36 {
37   char buff[BUF_SIZE];
38   int n;
39   int sockfd;
40   struct sockaddr_in serv_addr;
41
42   /* argument count exception handling */
43   if(argc != 3) {
44     printf("Usage :  %s [IP_ADDRESS] [PORT_NUMBER]\n", argv[0]);
45     exit(0);
46   }
47
48   /* open socket */
49   sockfd = socket(AF_INET, SOCK_STREAM, 0);
50
51   memset(&serv_addr, 0, sizeof(serv_addr)); // initialize server socket information struct to zero
52   serv_addr.sin_family = AF_INET;
53   serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
54   serv_addr.sin_port = htons(atoi(argv[2])); // short data(port number) to network byte order
55   /* connect socket */
```

```
56    connect(sockfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
57
58    while(1) {
59    ////////////////////text input//////////////////
60       write(STDOUT_FILENO,"> ", 2);
61       read(STDIN_FILENO, buff, BUF_SIZE);
62    /* remove newline of cmd_buff */
63       if(buff[strlen(buff) - 1] == '\n')
64          buff[strlen(buff) - 1] == '\0';
65       n = strlen(buff);
66       write(sockfd, buff, BUF_SIZE); // send text to server
67       n = read(sockfd, buff, BUF_SIZE); // receive text
68       //printf("%d\n",n);
69    /* if you receive "QUIT", n = 0 */
70       if(n == 0) {
71          bzero(buff,sizeof(buff));
72          sleep(1); // wait for child process terminated
73          break;
74       }
75       else if(n < 0) {
76          perror("read");
77          exit(0);
78       }
79       printf("from server: %s", buff);
80       bzero(buff,sizeof(buff)); // buffer initialization
81       ////////////////////text result////////////////////
82
83    }
84    close(sockfd);
85    return 0;
86 }
87
```

Colored by Color Scripter

2) Server

```c
////////////////////////////////////////////////////////////////////////////
// File Name      : srv.c                                                 //
// Date           : 2020/05/24 ~ 2020/05/28                               //
// OS             : Ubuntu 18.04.4 LTS                                    //
// Student Name   : Seung Hoon Jeong                                      //
// Student ID     : 2015707003                                            //
// ---------------------------------------------------------------------- //
// Title : System Programming Assignment #3-2                             //
// Description : Advanced echo Server                                     //
////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/wait.h>
#include <signal.h>

#define BUF_SIZE 256

void sh_chld(int);
void sh_alrm(int);

////////////////////////////////////////////////////////////////////////////////////////////////////
// Function : int main(int argc, char **argv)                                                     //
// ============================================================================================== //
// Input: x                                                                                       //
// Output: 1.print client's information                                                           //
//         2.check child process created and terminated                                           //
// ============================================================================================== //
// Purpose: print server information                                                              //
////////////////////////////////////////////////////////////////////////////////////////////////////
int main(int argc, char **argv)
{
   char buff[BUF_SIZE];
   int n;
   struct sockaddr_in serv_addr, client_addr;
   int server_fd, client_fd;
   pid_t pid;
   int len;
   int port;
   int client_idx = 0;

   /* check argument count exception handling */
   if(argc !=2) {
      printf("Usage : %s [PORT_NUMBER]\n",argv[0]); // fixed argument format
      exit(0);
   }

   /* open socket */
   server_fd = socket(PF_INET, SOCK_STREAM, 0);

   /* prevent bind error after server terminated */
```

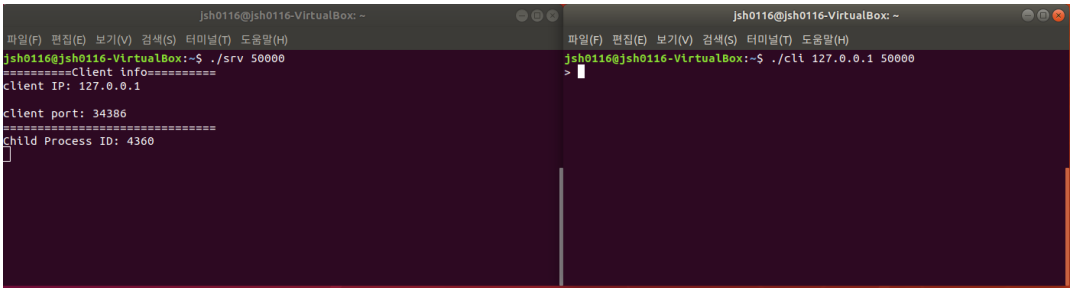
```
58  int option = 1;
59  if(setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &option, sizeof(option)) < 0) {
60     perror("setsockopt");
61     exit(0);
62  }
63
64  memset(&serv_addr, 0, sizeof(serv_addr)); // initialize server socket information struct to zero
65  serv_addr.sin_family = AF_INET;
66  serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
67  serv_addr.sin_port = htons(atoi(argv[1]));
68
69  /* bind socket */
70  bind(server_fd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));
71
72  /* listen socket */
73  listen(server_fd, 5);
74
75  /* Applying signal handler(sh_alrm) for SIGALRM */
76  signal(SIGALRM,sh_alrm);
77  /* Applying signal handler(sh_chld) for SIGCHLD */
78  signal(SIGCHLD,sh_chld);
79
80
81  while(1) {
82     len = sizeof(client_addr);
83     /* accept socket descriptor */
84     client_fd = accept(server_fd, (struct sockaddr*)&client_addr, &len);
85     //printf("new client connected..\n");
86     if(client_idx == 5) {
87        close(client_fd);
88        continue;
89     }
90     client_idx++;
91
92     /* 필요한 소스 삽입(fork() 이용) */
93     if((pid = fork()) < 0) {
94        perror("fork");
95        close(client_fd);
96        continue;
97     }
98
99     /* Child Process */
100    else if(pid == 0) {
101       close(server_fd);
102       sleep(0.5);
103       printf("Child Process ID: %d\n", getpid()); // print child process ID
104       memset(buff,0x00,BUF_SIZE);
105       while((len = read(client_fd, buff, BUF_SIZE)) != 0) {
106          /* when server received text "QUIT" */
107          if(strcmp(buff, "QUIT\n") == 0){
108             close(client_fd); // connection close with client
109             //signal(SIGALRM,sh_alrm);
110             sh_alrm(port); // call SIGALRM, terminate child process
111          }
112          else {
113             write(client_fd, buff, len); // send text to client
114             bzero(buff,sizeof(buff)); // buffer initialization
115
```

```
116             //continue;
117           }
118         }
119         close(client_fd);
120         exit(0);
121 
122       }
123 
124       /* Parent Process */
125       else {
126         close(client_fd);
127         /* print client's information */
128         printf("==========Client info==========\n");
129         printf("client IP: %s\n", inet_ntoa(client_addr.sin_addr)); // display IP by using inet_ntoa
130         printf("\n");
131         printf("client port: %d\n",ntohs(client_addr.sin_port)); // display Port number by using ntohs()
132         printf("==============================\n");
133 
134       }
135     }
136     close(server_fd);
137     return 0;
138 }
139 
140 ////////////////////////////////////////////////////////////////////////////////////////////////////
141 // Function : void sh_chld(int signum)                                                            //
142 // ============================================================================================== //
143 // Input: sigaction                                                                               //
144 // Output: print status of child process                                                          //
145 // ============================================================================================== //
146 // Purpose: Check status of child process                                                         //
147 ////////////////////////////////////////////////////////////////////////////////////////////////////
148 void sh_chld(int signum) {
149     printf("Status of Child process was changed.\n");
150     wait(NULL);
151 }
152 
153 ////////////////////////////////////////////////////////////////////////////////////////////////////
154 // Function : void sh_alrm(int signum)                                                            //
155 // ============================================================================================== //
156 // Input: sigaction                                                                               //
157 // Output: print child process will be terminated                                                 //
158 // ============================================================================================== //
159 // Purpose: ready for terminating child process                                                   //
160 ////////////////////////////////////////////////////////////////////////////////////////////////////
161 void sh_alrm(int signum) {
162     printf("Child Process(PID : %d) will be terminated.\n", getpid());
163     exit(1);
164 }
```

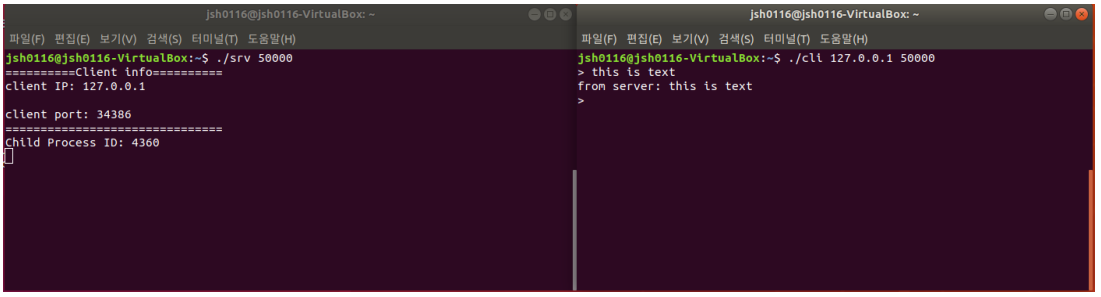
Colored by Color Scripter

IV. Result

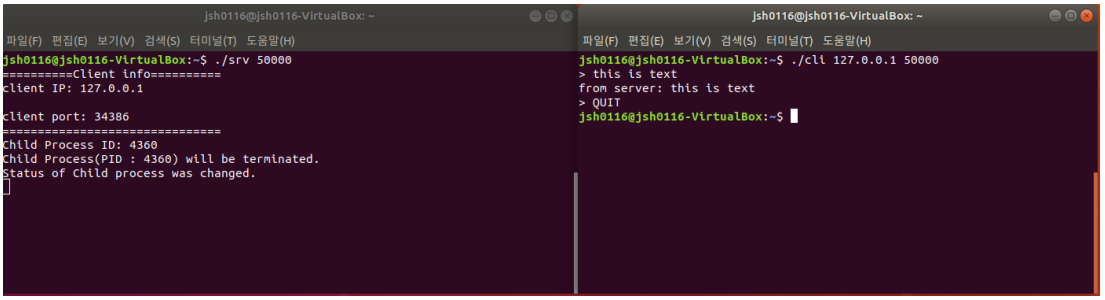
1) Server 실행, client 실행 및 연결 확인



2) Text 전송 및 수신 확인



3) 서버에 QUIT 명령어 전달 후 SIGALRM, SIGCHLD 호출



그 이후 1~2초후에 client 자동 종료.

4) Client 다중 접속한 경우