단일 프로세스 서버의 한계

iterative server program = tcpfirstsrv.c

```
if(listen(srvSd, 2)<0)
```

iterative한 성질을 갖고있어 두개까지 접속가능하더라도, 처음 연결한 client가 끝날 때까지 기다려야 한다.

tcptestclnt.c

tcpfirstclnt에서 while문 수정

```
while(1){
    fgets(wBuff, BUFSIZ-1, stdin);
    readLen = strlen(wBuff);
    if(readLen<2) continu;e

    write(clntSd, wBuff, readLen-1);
    wBuff[readLen-1]='\0';
    if(!strcmp(wBuff, "END")) break;
}
```

tcpfirstsrv.c에 차례대로 tcpfirstclnt.c, test, test 접속은 가능 하지만 clnt 먼저 처리 하고 test로 넘어옴

Parent/Child Process

fork()

- 자식 프로세스 생성, 부모의 모든 정보를 복사해감

```
int pid;
if(pid=fork()==0){
    //child
}
else{
    //parent
}
```

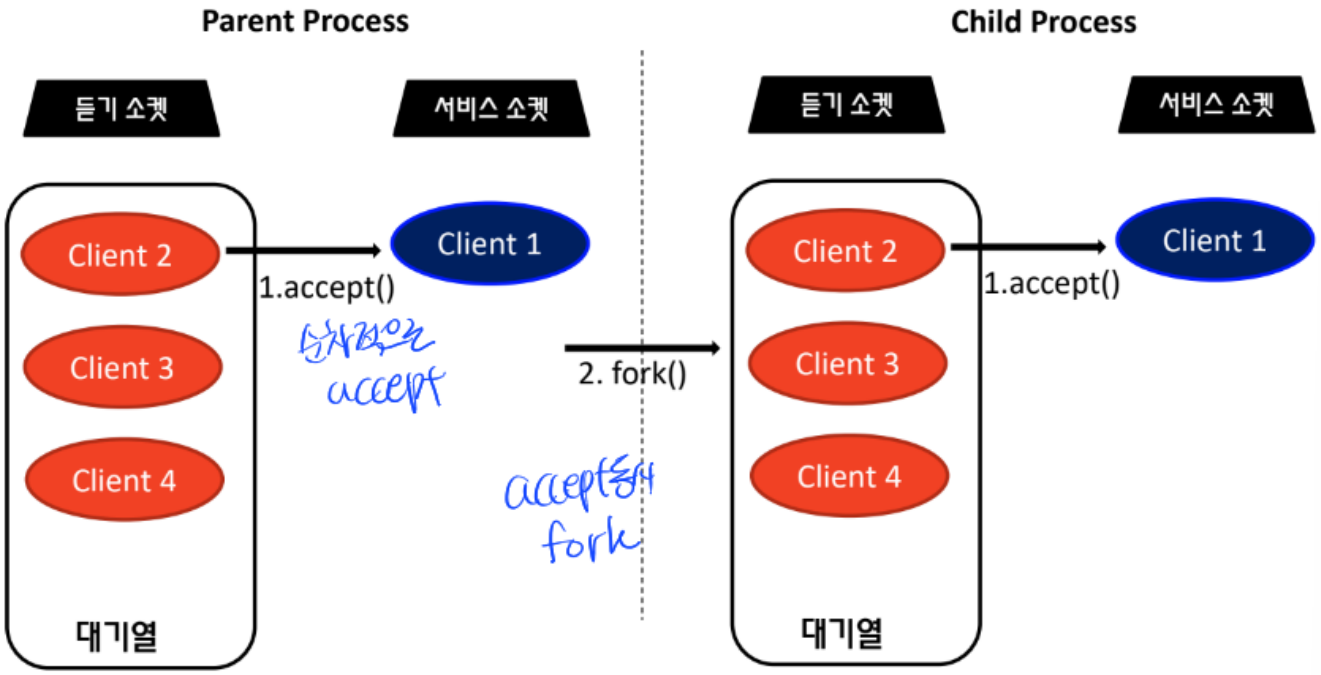
- 프로세스 생성 후 변화한 자원은 공유되지 않음.

Child 프로세스 종료

```
//Parent
int status;
```

```
wait(&status);
WIFEXITED(status); //자식 정상 종료시 true
WEXITSATUS(status); //자식 프로세스의 반환 값
WIFSIGNALED(status); //자식 프로세스가 시그널에 의해 종료한 경우 참
```

Multi-process-based server



- Parent Process Client 순차적으로 accept 후 fork해서 Child로 넘기기
- Child Process fork 된 client accept

Parent의 서비스 소켓과 Child의 듣기 소켓 close --> fork된 프로세스 하나가 하나의 client를 처리

▶ 실습 01: multiProcessTcpServer.c

```c
#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <arpa/inet.h>

void errProc();
void errPrint();
int main(int argc, char** argv)
{
    int srvSd, clntSd;
    struct sockaddr_in srvAddr, clntAddr;
    int clntAddrLen, readLen, strLen;
    char rBuff[BUFSIZ];
    pid_t pid;
```

```
19
20      if(argc != 2) {
21          printf("Usage: %s [port] \n", argv[0]);
22          exit(1);
23      }
24      printf("Server start...\n");
25
26      srvSd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
27      if(srvSd == -1 ) errProc();
28
29      memset(&srvAddr, 0, sizeof(srvAddr));
30      srvAddr.sin_addr.s_addr = htonl(INADDR_ANY);
31      srvAddr.sin_family = AF_INET;
32      srvAddr.sin_port = htons(atoi(argv[1]));
33
34      if(bind(srvSd, (struct sockaddr *) &srvAddr, sizeof(srvAddr)) == -1)
35          errProc();
36      if(listen(srvSd, 5) < 0)
37          errProc();
38      clntAddrLen = sizeof(clntAddr);
39
40      while(1)
41      {
42          //client 연결 받음
43          clntSd = accept(srvSd, (struct sockaddr *)
44                          &clntAddr, &clntAddrLen);
45          if(clntSd == -1) {
46              errPrint();
47              continue;
48          }
49          printf("client %s:%d is connected...\n",
50              inet_ntoa(clntAddr.sin_addr),
51              ntohs(clntAddr.sin_port));
52
53          //자식 프로세스 생성
54          pid = fork();
55          if(pid == 0) { /* child process */
56              //부모의 서버 소켓 닫기
57              close(srvSd);
58
59              //해당 clntSd 하나만 관리 & client 요청 처리
60              while(1) {
61                  readLen = read(clntSd, rBuff, sizeof(rBuff)-1);
62                  if(readLen == 0) break;
63                  rBuff[readLen] = '\0';
64                  printf("Client(%d): %s\n",
65                      ntohs(clntAddr.sin_port),rBuff);
66                  write(clntSd, rBuff, strlen(rBuff));
67              }
68
69              //Client 종료시 프로세스 종료
70              printf("Client(%d): is disconnected\n",
71                  ntohs(clntAddr.sin_port));
72              close(clntSd);
73              return 0;
74          }
75          //fork 에러
```

```
76          else if(pid == -1) errProc("fork");
77
78          else { /*Parent Process*/
79              //clntSd는 자식에게 넘겨줬으니 부모에서는 닫기
80              close(clntSd);
81          }
82      }
83      close(srvSd);
84      return 0;
85  }
86
87  void errProc(const char *str)
88  {
89      fprintf(stderr,"%s: %s \n",str, strerror(errno));
90      exit(1);
91  }
92
93  void errPrint(const char *str)
94  {
95      fprintf(stderr,"%s: %s \n",str, strerror(errno));
96  }
97
                                                    Colored by Color Scripter
```

실습 코드와 tcptestclnt.c 는 잘 동작하지만

문제점

연결하는 client마다 프로세스를 생성하기 때문에 매우 비효율적임 --> 보통은 작업단위로 fork하기 마련

Multi-process-based client

UDP 채팅 서버/클라이언트 프로그램

- 서버 클라이언트별 자식 프로세스 생성
- 클라이언트 부모: 사용자 입력 받아 보내기 자식: 다슨 사용자 입력 받아 출력

▶ 실습 02: ChatServer.c

```
1  #include <stdio.h>
2  #include <sys/socket.h>
3  #include <sys/types.h>
4  #include <netinet/in.h>
5  #include <string.h>
6  #include <errno.h>
7  #include <stdlib.h>
8  #include <arpa/inet.h>
9
10 #define MAX_CLIENT 10
11
12 void errProc(const char*);
13 int checkSockList(struct sockaddr_in *entry, struct sockaddr_in *list, int count);
14
15 int main(int argc, char** argv)
16 {
```

```
17      int mySock,readLen, nRecv, res;
18      char buff[BUFSIZ];
19      char nameBuff[50];
20      char * strAddr;
21      struct sockaddr_in srcAddr, destAddr;
22      socklen_t addrLen;
23      int nClient = 0, i = 0, port;
24      struct sockaddr_in sockets[MAX_CLIENT];
25
26      if(argc != 2) {
27          fprintf(stderr,"Usage: %s Port",argv[0]);
28          return 0;
29      }
30
31      memset(&sockets, 0, sizeof(sockets[0])*MAX_CLIENT);
32
33      //UDP Socket
34      mySock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
35      if(mySock == -1) errProc("socket");
36
37      memset(&srcAddr, 0, sizeof(srcAddr));
38      srcAddr.sin_addr.s_addr = htonl(INADDR_ANY);
39      srcAddr.sin_family = AF_INET;
40      srcAddr.sin_port = htons(atoi(argv[1]));
41
42      res = bind(mySock,(struct sockaddr *) &srcAddr,    sizeof(srcAddr));
43      if(res == -1) errProc("bind");
44      addrLen = sizeof(destAddr);
45
46      while(1)
47      {
48          //접속자로부터 데이터 받으면
49          nRecv = recvfrom(mySock, buff, BUFSIZ-1 , 0,
50              (struct sockaddr *) &destAddr,
51              &addrLen);
52          if(nRecv == -1) errProc("recvfrom");
53
54          //해당 접속자를 list 형태로 저장
55          res = checkSockList(&destAddr, sockets, nClient);
56
57          //처음 접속한 접속자라면
58          if(res == nClient) { /*New Client*/
59              if(res == MAX_CLIENT) continue; /*MAX Client Limit*/
60              else
61              {
62                  memcpy(&sockets[res], &destAddr, sizeof(destAddr));
63                  //현재 접속자수 ++
64                  nClient++;
65              }
66          }
67
68          printf("nClient: %d \n", nClient);
69          strAddr = inet_ntoa(destAddr.sin_addr);
70
71          //nameBuff에 이런 문장 넣기
72          sprintf(nameBuff, "%s: %d>> ", strAddr, ntohs(destAddr.sin_port));
73
```

```
        //list를 통해 자신이 아닌 다른 모든 client에게 메세지 전달
        for(i = 0; i < nClient; i++)
        {
            if(i == res) continue; /* sender == receiver skip */
            //어디서 어디로 보냈는지 명시
            sendto(mySock, nameBuff, strlen(nameBuff), 0,
            (struct sockaddr *) &sockets[i], addrLen);
            //받은 데이터 그대로 보내기

            sendto(mySock, buff, nRecv, 0,
            (struct sockaddr *) &sockets[i], addrLen);
        }
    }
    return 0;
}

/* 리스트 검색 후 인덱스 반환,
리스트에 엔트리가 없을 경우 마지막 인덱스 + 1 반환 */
int checkSockList(struct sockaddr_in *entry,
                    struct sockaddr_in *list, int count)
{
    int i = 0;
    struct sockaddr_in *ptrSockAddr;
    for(i = 0; i < count; i++) {
        ptrSockAddr = list + i;
        //같으면 이미 list에 저장되어 있는 것.
        if(memcmp(ptrSockAddr, entry,
            sizeof(struct sockaddr_in)) == 0)
            return i;
    }
    return i;
}


void errProc(const char* str)
{
    fprintf(stderr,"%s: %s \n", str, strerror(errno));
    exit(1);
}

```

Colored by Color Scripter

▶ 실습 02: ChatClient.c

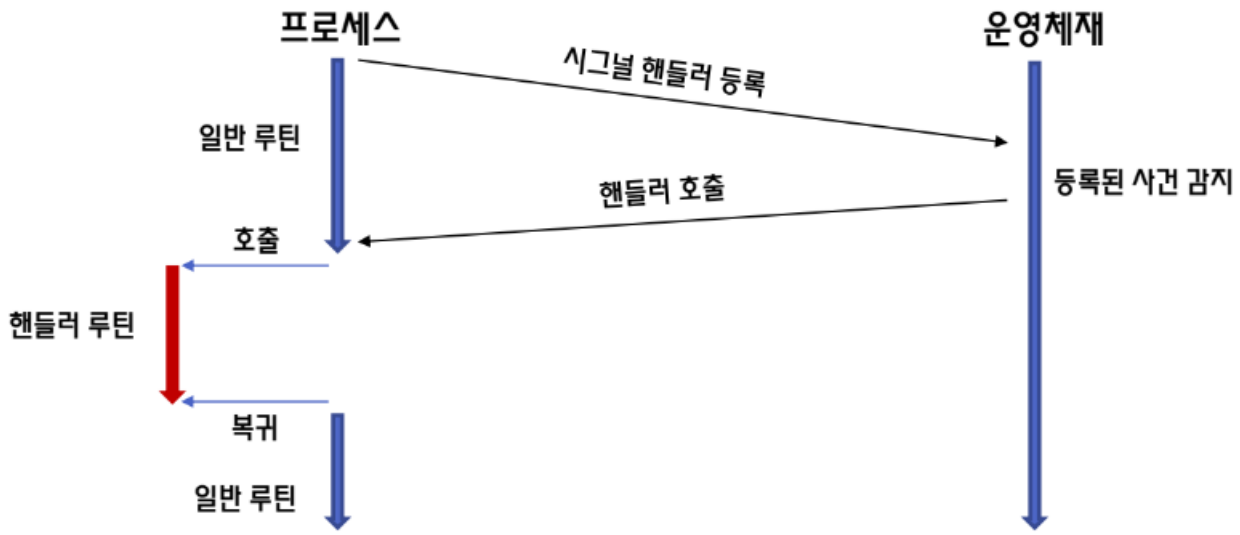
```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <signal.h>
#include <netinet/in.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <arpa/inet.h>


void errProc(const char*);
int main(int argc, char** argv)
{
```

```c
    int mySock,readLen, nSent, nRecv;
    char buff[BUFSIZ];
    char strAddr;
    struct sockaddr_in destAddr;
    socklen_t addrLen;
    pid_t pid;

    mySock = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

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

    pid = fork();
    if(pid == -1) errProc("fork");

    if(pid == 0) { /*child process UDP Recvfrom*/
    //Child에서는 받은 걸 그대로 쓰기
        while(1)
        {
            nRecv = recvfrom(mySock, buff, BUFSIZ-1, 0,
                (struct sockaddr*) &destAddr, &addrLen);
            if(nRecv == -1) errProc("read");
            write(1,buff,nRecv);
        }
    }
    else { /*Parent Process UDP Sendto */
    //부모 프로세스에서는 사용자 입력을 보내기
        while(1)
        {
            fgets(buff, BUFSIZ-1, stdin);
             readLen = strlen(buff);
            if(readLen == 0) errProc("fgets");
            nSent = sendto(mySock, buff, readLen, 0,
                (struct sockaddr*) &destAddr, addrLen);
            if(nSent == -1) errProc("write");
            buff[readLen-1] = '\0';
            if(!strcmp(buff,"END")) break;
        }
        //
        kill(pid, SIGTERM);
    }
    return 0;
}

void errProc(const char* str)
{
    fprintf(stderr,"%s: %s \n", str, strerror(errno));
    exit(1);
}

```

Colored by Color Scripter

IPC (Inter Process Communication)

Signal




```
//SIGINT 발생시 핸들링
signal(SIGINT, sigint_handler);

//SIGINT 발생시 기본 동작 실행 (SIG_DFL)
signal(SIGINT, SIG_DFL);
```

▶ 실습 03: signalbasic.c

```c

#include <stdio.h>
#include <signal.h>

int global_count = 10;

void sigint_handler(int sig)
{
    if(sig == SIGINT)
    {
        printf("Received SIGINT... %d Lives Left \n", global_count--);
    }
    if(global_count <= 0)
    {
        //11번째부터는 정상실행
        signal(SIGINT, SIG_DFL);
    }
}

int main(int argc, char **argv)
{
    int i = 0;
    //시그널 핸들링
    signal(SIGINT, sigint_handler);
    while(1)
    {
```

```
        printf("%d: sleep and awake \n", ++i);
        sleep(5);
    }
    return 1;
}

```

Colored by Color Scripter

```
sigaction(handler, sigaction, sa_flag, sa_mask)
//sa_flag: signal option
//sa_mask: signal mask to apply blockin signal
```

▶ 실습 04: signalsecond.c

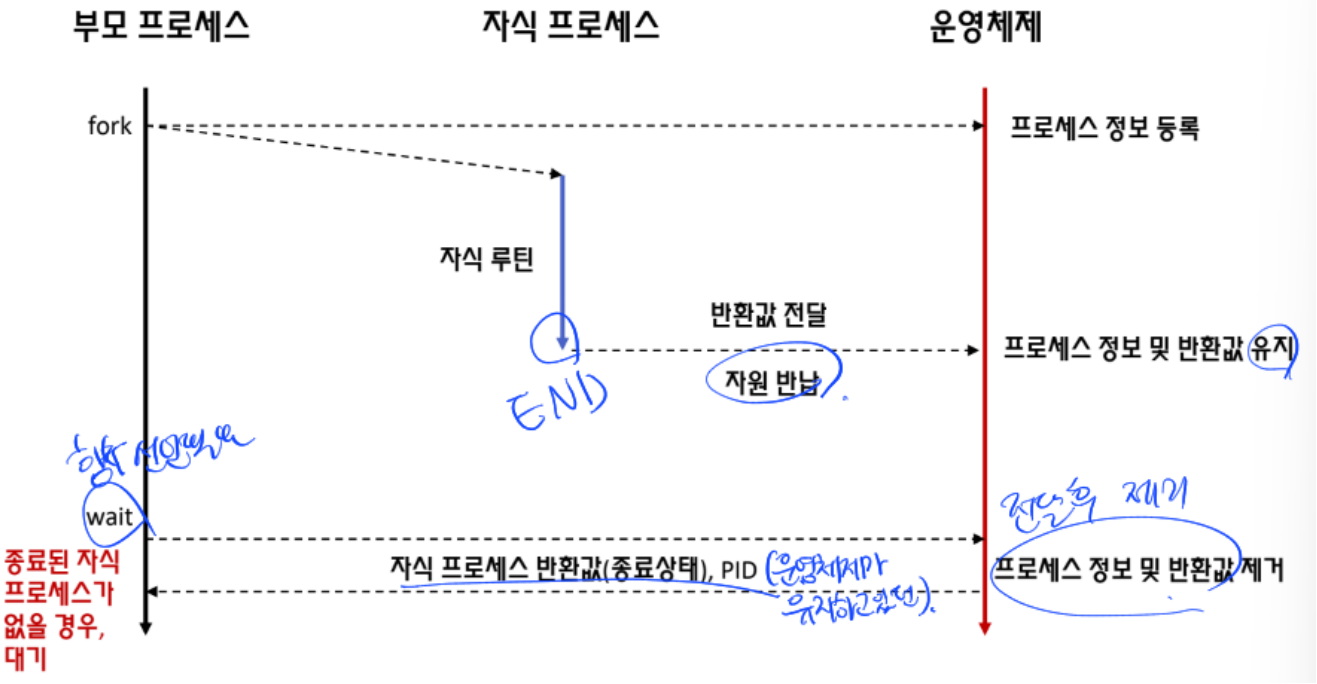
```


#include <stdio.h>
#include <signal.h>

int main(int argc, char **argv)
{
    int i = 5;
    //sigaction 구조체 변수
    struct sigaction new_action, old_action;

    //시그널 발생시 SIG_IGN(무시)
    new_action.sa_handler = SIG_IGN;
    //sa_mask 0으로 초기화
    sigemptyset(&new_action.sa_mask);
    //flag를 0으로
    new_action.sa_flags = 0;


    //SIGINT 기능을 old_action에 저장하고 SIGINT를 new_action 구조체로 바꿈
    sigaction(SIGINT, &new_action, &old_action);
    //--> SIGINT는 ctrl^c 무시

    while(1)
    {
        printf("%d: sleep and awake \n", i--);
        sleep(1);
        if(i == 0) break;
    }
    return 1;
}

```

Colored by Color Scripter

좀비 프로세스

Child Process 끝났어도 호출할 수 있기에 부모의 wait 전에는 존재한다. 부모가 적절한 시점에 wait를 호출해야 한다. -> signal을 이용해 자식이 끝났다고 전달하면 wait하기

❐ 좀비(defunct) 프로세스



시그널 발생

```
//해당 pid에게 signal 발생시킴
kill(pid, signal);
```