

2019년 1학기 시스템프로그래밍실습 12주차

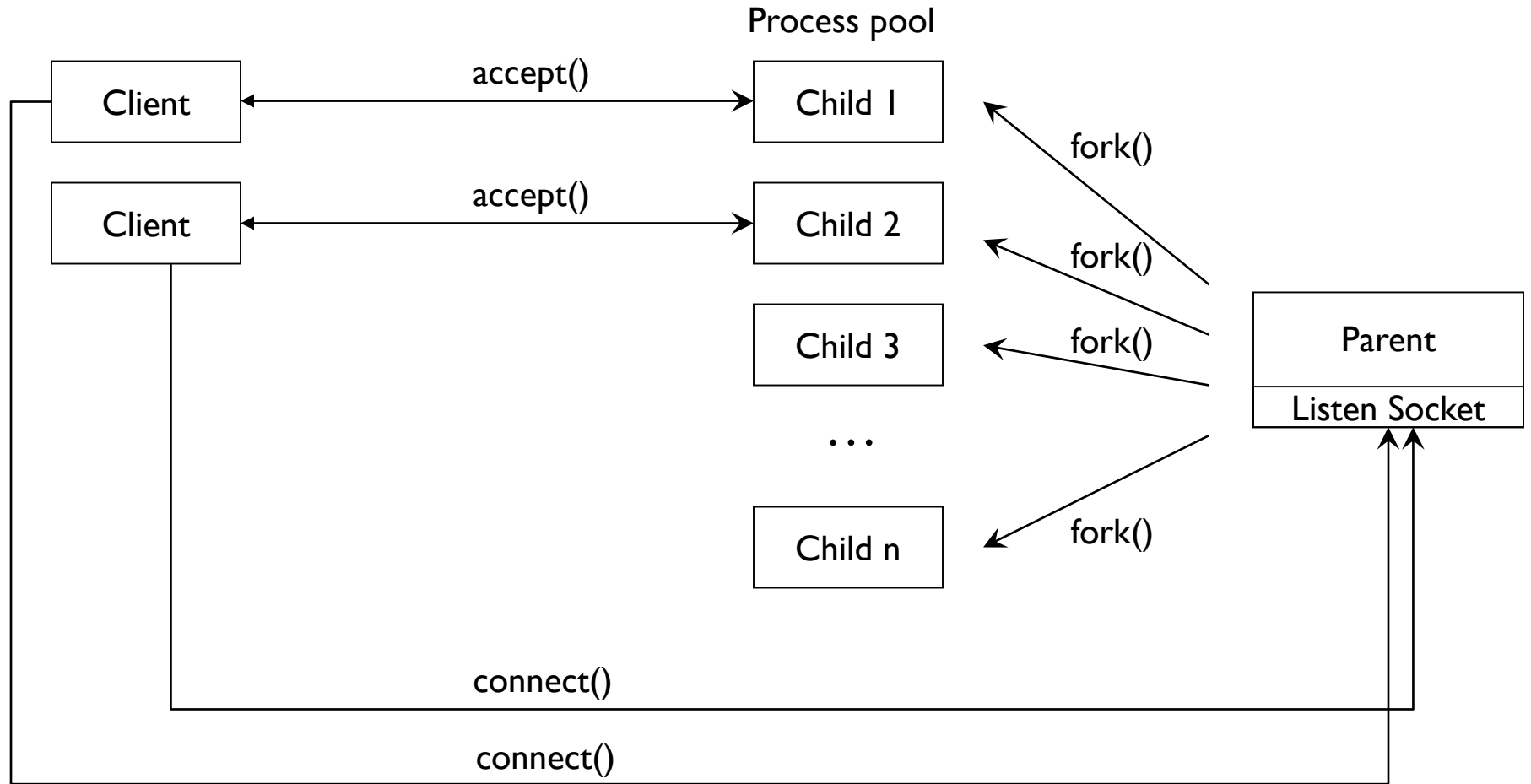
# Pre-forked Web Server

**System Software Laboratory**  
College of Software and Convergence  
Kwangwoon Univ.

# Contents

- **TCP pre-forked server (Assignment 4-1)**
  - Process pre-forking
  - Signal processing
- **Process pool management (Assignment 4-2)**
  - Process scheduling (Max & Min bound)
  - Shared memory, mutex, pthread
- **Mutual Exclusion (Assignment 4-3)**
  - Log file 작성

# Pre-forked server



# Pre-forked server (cont'd)

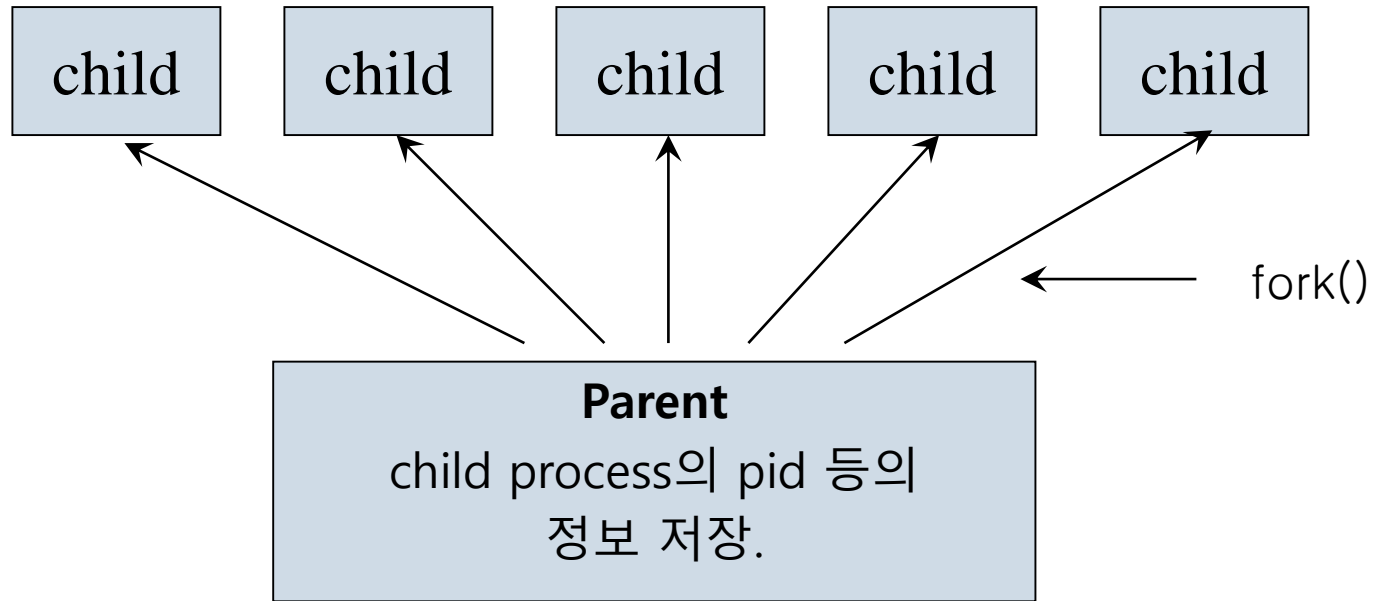
- **Pre-forking child process**

- 미리 일정 수의 Child process를 생성 (fork)
- Request 도착 시, 미리 만들어놓은 프로세스로 서비스 가능
- Request 도착 후에 프로세스를 fork하는 비용을 줄일 수 있음

- **Parent Process**

- Socket()으로 생성한 포트를 각 Child에게 전달
- Child process들을 관리 -> Child들의 pid, 상태 정보 등을 저장

## Pre-forked server (cont'd)



# Pre-forked server (cont'd)

- Child process 생성

```
pid_t child_make(socktet_fd, addrlen) {
    struct node *new;
    int pid;

    if ((pid = fork()) > 0) {                                     // Parent process routine.

        // child process의 정보를 linked list로 저장

        return (pid);
    }
    child_main(listenfd, addrlen);                               // Child process routine.
}
```

# Signal Function

- 어떤 이벤트가 발생하면 이 사실을 프로세스에게 알리는 수단
- **Every signal has a name**
  - These names all begin with the three characters "SIG"
  - Example
    - SIGALRM : 특정 시간 경과 시 발생
    - SIGCHLD : 프로세스 종료 시 발생
    - SIGINT : 인터럽트 키(ctrl+c) 입력 시 발생
    - SIGTERM : 프로세스가 신호를 받고 종료시키는 안전한 방법 (kill)
    - SIGKILL : 프로세스의 의사와 관계없이 바로 프로세스를 종료시키는 신호 (kill -9)
    - SIGUSR1 : User-defined signal 1
    - SIGUSR2 : User-defined signal 2

# Signal Function (cont'd)

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

```
void (*signal(int signo, void (*func)(int))) (int);
```

- **Returns**

- previous disposition of signal (see following) if OK,  
SIG\_ERR on error

- **Signo**

- 시그널의 이름
- Ex) **SIGALRM, SIGCHLD, SIGINT, SIGTERM**

- **void (\*func)(int)**

- signal-handler, 시그널이 발생했을 때 수행할 함수
- SIG\_IGN, 시그널 무시

- **SIGKILL, SIGSTOP**는 Signal handler 등록 및 무시 불가



# kill function

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

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

```
int kill(pid_t pid, int sig);
```

- **Role**

- Send sig to process (pid)

- **Returns**

- On success (at least one signal was sent), zero is returned.  
On error, -1 is returned.

- **sig**

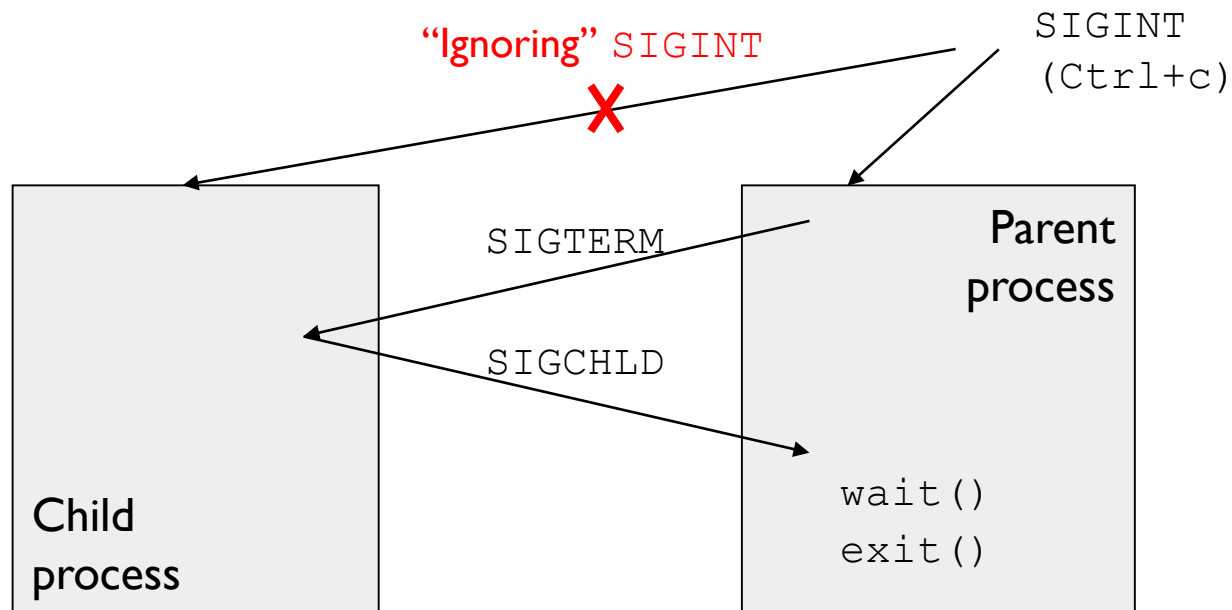
- 발생시킬 시그널 ex) SIGALRM, SIGCHLD, SIGINT, SIGTERM

- **pid**

- 해당 프로세스 ID를 전송 하는 변수

# Signal flow

- Kill Signal에 대한 Handler 작성 (Parent)
  - Ctrl+c로 Parent를 강제 종료 시킬 경우, Parent가 종료되기 전에 모든 Child process들을 종료 시키는 Handler 필요
  - Child process가 Zombie process가 되는 것을 방지
- Signal 발생 순서



# Lab

- **Pre-forked Echo Server**

- 9주차 실습인 “Echo Server”에 다중 접속 기능 구현
  - Echo server에 여러 echo client가 접속되어도, 각각의 echo client와 통신할 수 있도록 구현
  - pre-forked 방식으로 구현
  - client.c는 이전 실습 자료 사용

# Pre-forked echo server

```
#define BUFFSIZE    1024
#define PORTNO      40000

static int maxNchildren; // The maximum # of Child process
static pid_t *pids;      // Keep Child's pid.
static char *buf;

pid_t child_make(int i, int sockfd, int addrlen);
void child_main(int i, int sockfd, int addrlen);

int main(int argc, char *argv[])
{
    struct sockaddr_in server_addr, client_addr;
    int socket_fd, addrlen, i, opt=1;
    buf = (char *) malloc(BUFFSIZE+1);

    if((socket_fd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        printf("Server: Can't open stream socket.\n");
        return 0;
    }

    setsockopt(socket_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
    bzero((char *)&server_addr, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    server_addr.sin_port = htons(PORTNO);
    if(bind(socket_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        printf("Server: Can't bind local address.\n");
        return 0;
    }
}
```

## Pre-forked echo server (cont'd)

```
listen(socket_fd, 5);

maxNchildren = 5;
pids = (pid_t *)malloc(maxNchildren * sizeof(pid_t));
addrlen = sizeof(client_addr);

// Pre-forking routine
for( i=0 ; i < maxNchildren ; i++)
    pids[i] = child_make(i, socket_fd, addrlen); // Parent returns

for ( ; ; )
    pause();
}

pid_t child_make(int i, int sockfd, int addrlen) {
    pid_t pid;

    if ((pid = fork()) > 0)
        return(pid); // Parent move out

    child_main(i, sockfd, addrlen); // Children never returns
}
```

## Pre-forked echo server (cont'd)

```
void child_main(int i, int sockfd, int addrlen)
{
    int client_fd, len_out;
    char buf[BUFSIZE];
    socklen_t clilen;
    struct sockaddr *client_addr;

    client_addr = (struct sockaddr *)malloc(addrlen);
    printf("child %ld starting\n", (long) getpid());

    while(1) {
```

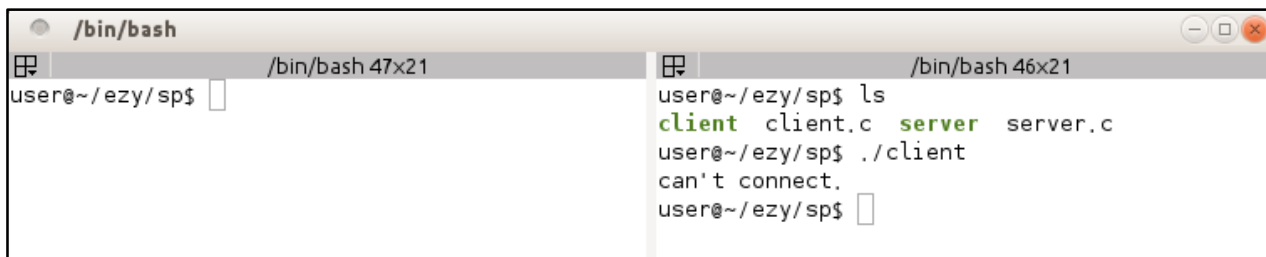
**Fill your codes**

));

```
}
}
```

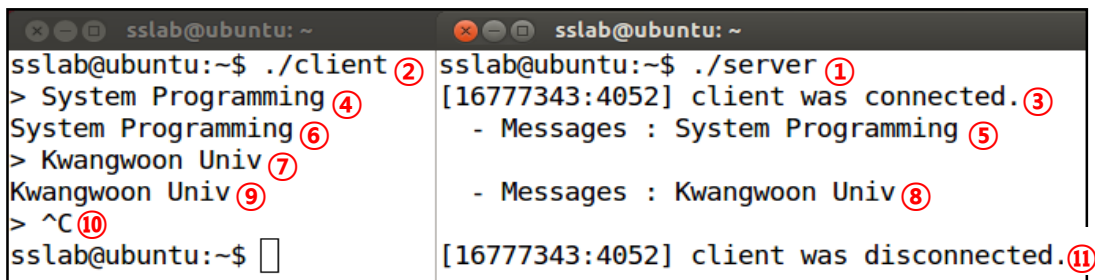
# Execution of Echo Server and Client

- Command 창 : server(1개), client(최대 5개)
- server를 실행하지 않은 상태에서 client 실행 시 **can't connect**.



The image shows two terminal windows. The left window, titled '/bin/bash 47x21', shows a user prompt 'user@~/ezy/sp\$'. The right window, titled '/bin/bash 46x21', shows the following commands and output: 'ls' lists 'client.c' and 'server.c'; './client' results in the error 'can't connect.'.

- 하나의 command 창에서는 server 실행, 다른 command 창에서는 client를 실행
- Client가 정상 실행되면 > 문자가 출력되고 메시지를 입력하면 server로 전달
  - Server는 받은 메시지를 client에게 다시 전달
  - Client는 결국 자신이 보낸 메시지를 받아 출력.



The image shows two terminal windows. The left window, titled 'sslab@ubuntu: ~', shows the client running './client' (2), which then sends 'System Programming' (4) and 'Kwangwoon Univ' (7). The right window, also titled 'sslab@ubuntu: ~', shows the server running './server' (1), which prints '[16777343:4052] client was connected.' (3), receives the message 'System Programming' (5), and prints 'Messages : System Programming' (5). It then receives 'Kwangwoon Univ' (8) and prints 'Messages : Kwangwoon Univ' (8). Finally, it prints '[16777343:4052] client was disconnected.' (11) after the client is interrupted with ^C (10).

※ Server 종료 후 다시 실행할 때  
'Server: Can't bind local address.' 라는  
메시지가 뜨면, 1~2분 정도 후 다시 실행하면 됨

# Execution of Pre-forked echo server

```
splab@ubuntu:~/SP$ ./srv
child 5499 starting
child 5500 starting
child 5501 starting
child 5503 starting
child 5502 starting
a
child 5499 processing request

b
child 5500 processing request

c
child 5501 processing request

d
child 5503 processing request

e
child 5502 processing request

f
child 5500 processing request

g
child 5503 processing request
```

```
splab@ubuntu:~/SP$ ./cli
> a
a
>

splab@ubuntu:~/SP$ ./cli
> b
b
> f
f
>

splab@ubuntu:~/SP$ ./cli
> c
c
>

splab@ubuntu:~/SP$ ./cli
> d
d
> g
g
>

splab@ubuntu:~/SP$ ./cli
> e
e
>
```



2019년 1학기 시스템프로그래밍실습 12주차

# Assignment 4-1

**System Software Laboratory**  
College of Software and Convergence  
Kwangwoon Univ.

# Assignment 4-1

- Assignment 3-3 에 다음 사항을 추가
  - 포트는 Makefile로 생성시 생성 파일에 나오도록 생성
    - Ex) preforked\_server\_33343(본인의포트번호) 이런 형식으로 실행파일 생성.
  - Pre-forked 방식으로 server 구현
    - 5개 child 프로세스를 생성 및 유지
    - 부모 프로세스는 fork()로 자식 프로세스 만들고, terminal에 그 정보를 출력 (“... forked.”)
  - SIGINT 발생 시, 모든 프로세스를 완전하게 종료 시켜야 함
    - 프로세스가 수행해야 할 연산
      - 부모 프로세스
        - (1) 자식 프로세스 상태 출력 (“... terminated.”)
        - (2) 해당 프로세스 상태 출력 (“... terminated.”)
        - (3) 종료
      - 자식 프로세스: 종료
    - 단, zombie 프로세스가 생성되지 않도록 보장해야 함
      - Zombie 프로세스 확인: ps [pid] -> STAT에 Z 출력 시, zombie 프로세스

# Assignment 4-1

- Assignment 3-3 에 다음 사항을 추가 (cont'd)
  - Connection history 출력 (과제 3-3의 연장)
    - 10초마다 자식 프로세스에 연결된 client의 접속 기록(connection history)를 출력해야 함
    - 각 자식 프로세스 마다 최신 history 기록을 저장 및 출력 (자식 프로세스 당 최대 10개)
      - 즉, 5개의 자식 프로세스의 전체 history 개수는 최대 50개
      - History 번호 (“No.”)는 자식 프로세스 마다 개별적으로 유지
    - History 제목은 부모 프로세스가 출력, 하위에 있는 내용은 자식 프로세스가 출력
      - 즉, “... Connection history ...”, “No.”, “IP” 등의 제목 부분은 부모 프로세스에서 출력
    - 부모 프로세스가 history 제목을 출력한 뒤, SIGUSR1 signal을 자식 프로세스에게 전달하여 history 내용 출력을 지시
      - 본 과제에서는 history 내용 출력 관련한 동기화 문제를 고려하지 않음
    - 단, 3-3에서 출력하였던 request 수는 본 과제에서 출력하지 않음
  - 터미널에 간단한 로그 기록을 출력
    - “... Server is started.”
    - “... Socket is created ...”
    - “... Server is terminated”

# Assignment 4-1 (cont'd)

## 결과화면

```
[Wed May 15 18:00:00 2019] Server is started.
[Wed May 15 18:00:00 2019] Socket is created IP : 223.195.34.120, port : 22345

[Wed May 15 18:00:00 2019] 2332 process is forked.
[Wed May 15 18:00:00 2019] 2333 process is forked.
[Wed May 15 18:00:00 2019] 2334 process is forked.
[Wed May 15 18:00:00 2019] 2335 process is forked.
[Wed May 15 18:00:00 2019] 2336 process is forked.
...

===== New client =====
[Wed May 15 18:05:34 2019]
IP : 223.195.33.43
Port : 22343

=====
===== Connection History =====
NO.      IP          PID      PORT      TIME
1        223.195.33.39  2332     22340     Wed May 15 18:03:20 2019
2        223.195.33.43  2333     22343     Wed May 15 18:05:34 2019
```

Server program 시작

Socket 생성 시

Child process 생성 시

Client 접속 시

History 제목 → 부모 프로세스가 출력  
History 내용 → 자식 프로세스가 출력

# Assignment 4-1 (cont'd)

## 결과화면 (cont'd)

```
===== Disconnected client =====
[Wed May 15 18:10:23 2019]
IP : 223.195.33.39
Port : 22340
=====
===== Connection History =====
NO.      IP          PID      PORT      TIME
1        223.195.33.43  2333     22343     Wed May 15 18:03:20 2019

SIGINT signal 발생
[Wed May 15 18:12:20 2019] 2336 process is terminated.
[Wed May 15 18:12:20 2019] 2335 process is terminated.
[Wed May 15 18:12:20 2019] 2334 process is terminated.
[Wed May 15 18:12:20 2019] 2333 process is terminated.
[Wed May 15 18:12:20 2019] 2332 process is terminated.
[Wed May 15 18:12:20 2019] Server is terminated.
```

client 종료 시

server 종료 시,  
client가 먼저 종료되어야 함!

History 제목 → 부모 프로세스가 출력  
History 내용 → 자식 프로세스가 출력

child process 종료 시

server 종료 시

# Assignment 4-1 (cont'd)

- **Code Requirements**

- 이전 과제 부분에 문제가 있는 경우 감점
- 출력 형식에 맞지 않으면 감점
- 소스코드의 50% 이상 주석을 달지 않은 경우 감점
- Copy 적발 시 0점

- **Makefile Requirements**

- 실행 파일이 “`preforked_server_portnumber(본인의포트번호)`”로 생성되도록 Makefile 작성
- 컴파일 도중 warning 발생 시 감점
- “\$ make” 를 통해 실행파일이 생성되지 않는 경우, 0점

# Assignment 4-1 (cont'd)

## ▪ Softcopy Upload

- 제출 파일
  - 보고서 (파일명: 실습요일\_4-1\_학번.pdf)
  - 요일\_4-1\_학번.c , Makefile
- 위 파일들을 압축해서 제출 (파일명: 실습 요일\_4-1\_학번.tar.gz)
  - e.g. 월1,2 → **mon\_4-1\_2018110609.tar.gz**
  - e.g. 화3,4 → **tue\_4-1\_2018110609.tar.gz**
  - e.g. 금5,6 → **fri\_4-1\_2018110609.tar.gz**
- U-Campus의 과제 제출에 **5월 31일(금) 23:59:59까지** 제출
  - U-Campus에 올린 후 다시 다운로드 받아서 **파일에 문제가 없는지 필히 확인**
  - 미리 공지한 바와 같이, **delay 받지 않음 (예외 없음)**
  - **Ubuntu 16.04 64bits** 환경에서 채점

## ▪ 과제 질문 관련

- 해당 과제 출제 담당 조교에게 이메일로 문의 → 김태현 조교 (taehyun9203@gmail.com)
- 과제 제출 마감 당일에는 오후 4시까지 도착한 질문 메일에만 답변

# Assignment 4-1 (cont'd)

## ■ 표지

- 다음의 내용은 필히 기록
- 과제 이름 (e.g. Assignment#4-1)
- 분반 (요일, 담당 교수님)
- 본인 인적 사항 (학번, 이름) 아래의 내용은 보고서에 필히 포함

## ■ 과제 내용

- Introduction : 5줄 이하
- Flow chart : 4주차 자료의 Appendix 참고
- Pseudo code : 4주차 자료의 Appendix 참고
- Result : 수행한 내용을 캡처 이미지와 함께 설명
- Conclusion : 결론 및 고찰

## ■ 보고서 이름은 “실습 요일\_과제명\_학번”으로 수정

- e.g. 월1,2 → mon\_4-1\_2018110609.pdf
- e.g. 화3,4 → tue\_4-1\_2018110609.pdf
- e.g. 금5,6 → fri\_4-1\_2018110609.pdf