

2025년 1학기 시스템프로그래밍실습 6주차

Multiple-Processing

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

1st Assignment's Descriptions

- **Assignment 1-1**

- 표준입력(STDIN)으로부터 URL 입력
- SHA-1 Algorithm을 사용하여 textual URL을 Hashed URL로 변환
- Hashed URL을 이용하여 Directory와 File 생성

- **Assignment 1-2**

- 시스템으로부터 현재 시간 구함
- Log file을 생성
- Log file에 입력 URL과 현재 시간 기록

- **Assignment 1-3**

- Multiple Processing

Process

- A program in execution
- **Process identifiers**
 - Every process has a unique process ID (a nonnegative integer) → i.e. PID
- **fork()를 이용한 child process 생성**
 - Parent process
 - fork()를 호출하여 자기 자신을 복제
 - Child process
 - fork()에 의하여 생성된 process

getpid & getppid system call

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

pid_t getpid(void);
pid_t getppid(void);
```

- **Get process identification**
- **Returns**
 - getpid : returns the process ID (PID) of the calling process
 - getppid : returns the process ID of the parent of the calling process

fork system call (1/2)

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

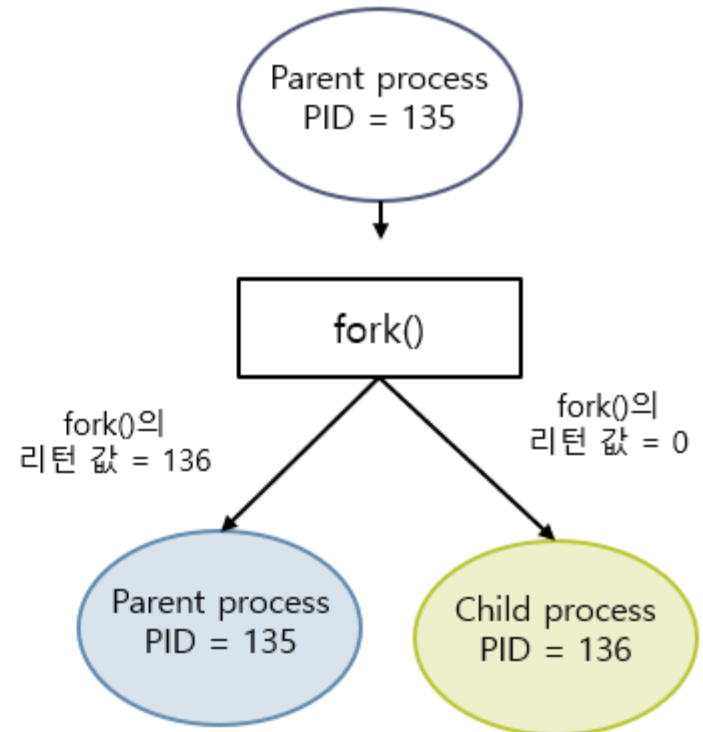
pid_t fork(void);
```

■ Introduction

- Create a child process
 - The only way in Unix to create a new process
- Called once but returns twice

■ Returns

- Child process : 0
- Parent process : process ID of the new child
- Error : -1



fork system call (2/2)

- **Descriptions**

- The child gets a **copy of the parent's data and stack**.
 - Data section : global variables, ...
 - Stack section : local variables, parameters, return values of a function, ...
- The child and parent continue to execute with the instruction that follows the call to fork.
- The parent and child can each execute **different sections of code** at the same time.

```
pid_t PID = fork();
if( PID == 0 ){
    child_work();
    /* Child 프로세스 실행 코드 */
}
else{
    parent_work();
    /* Parent 프로세스 실행 코드 */
}
```

Process Termination

- **SIGCHLD signal**

- Child 프로세스의 상태가 변경되면 해당 signal이 parent 프로세스에게 전달됨 (i.e. terminate, stop, or continue)
- 기본 동작: 무시 (ignorance)

- **wait() and waitpid()**

- Parent process는 wait() 또는 waitpid() 함수를 사용하여 child process의 종료 status를 catch

- **Zombie process**

- Parent process가 child process를 기다리지 않고 종료되어 child process의 resource를 반납하지 못하는 경우

wait & waitpid system call (1/2)

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

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

- **Wait for state change in a child**
- **Returns**
 - process ID if OK, or -1 an error
- **Parameters**
 - pid : process ID to wait, 임의의 자식 process를 기다릴 경우 -1
 - statloc : 종료한 프로세스의 상태를 저장하는 변수
 - options
 - WNOHANG : return immediately if no child has exited
 - WUNTRACED : also return if a child has stopped

wait & waitpid system call (2/2)

- 모든 **child process**가 실행 중일 경우
 - 해당 지점에서 대기 (block)
 - waitpid() 에서 options에 WNOHANG을 명시한 경우
 - child process가 종료되지 않은 경우 waitpid() 즉시 return
- **Child process가 호출 시점에 이미 종료된 경우 (i.e. zombie process)**
 - child process의 종료 status를 즉시 return
 - child process가 사용한 resource를 모두 release
- **Child process가 없을 경우**
 - return error

실습 1 – fork()

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

int main()
{
    char buf[255];
    pid_t pid;
    int status;

    printf("%% ");
    while (fgets(buf, 255, stdin) != NULL) {
        buf[strlen(buf) - 1] = 0;
        if ( (pid = fork()) < 0 )
            fprintf(stderr, "fork error");
        else if (pid == 0) { /* child */
            printf("%s\n", buf);
            return 0;
        }
        /* parent */
        if ( (pid = waitpid(pid, &status, 0)) < 0 )
            fprintf(stderr, "waitpid error\n");
        printf("%% ");
    }
    return 0;
}
```

```
sslab@ubuntu:~/sslab$ ./a.out
% a
a
% b
b
% c
c
% sslab@ubuntu:~/sslab$ <ctrl + D>
```

실습 2 – zombie process

```
#include <sys/types.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
    pid_t pid;

    if ( (pid = fork()) < 0 )
        fprintf(stderr, "fork error");
    else if (pid == 0) /* child */
        exit(0);

    sleep(4);

    system("ps");

    exit(0);
}
```

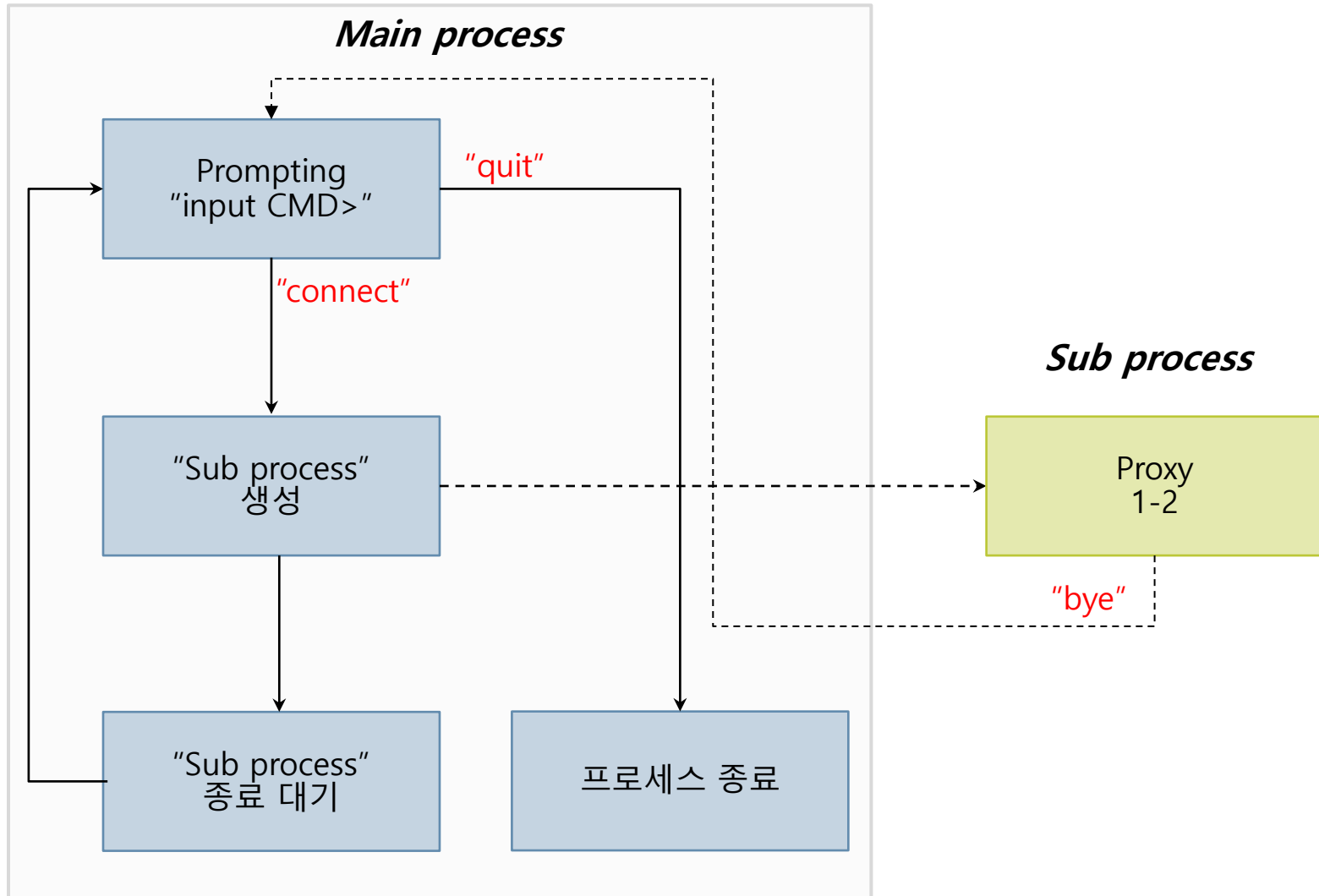
```
sslab@ubuntu:~/sslab$ ./a.out
  PID TTY          TIME CMD
 3123 pts/4        00:00:00 bash
 4059 pts/4        00:00:00 a.out
 4060 pts/4        00:00:00 a.out <defunct>
 4061 pts/4        00:00:00 sh
 4062 pts/4        00:00:00 ps
sslab@ubuntu:~/sslab$
```

2025년 1학기 시스템프로그래밍실습

Proxy #1-3

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

Proxy 1-3(1/4)



Proxy 1-3(2/4)

- **Concurrent Server Implementation with `fork()`**
- **Main process**
 - 사용자 요청 처리를 위한 새로운 프로세스("Sub process")를 생성하고, 관리
 - 동작:
 - 터미널에 `[(pid)]input CMD>`를 출력하고, 사용자의 명령어 입력을 대기
 - `[(pid)]` → `getpid()` 통해 얻은 값 사용
 - **`connect` 명령어 입력**
 - 새로운 프로세스를 생성하고, 해당 프로세스의 종료까지 대기
 - **`quit` 명령어 입력 시**
 - 프로세스 종료
 - 동작 시간, 생성한 child 프로세스 수 정보에 대한 log를 logfile에 아래와 같이 출력 해야 함
 - ****SERVER** [Terminated] run time: 20 sec. #sub process: 2**

```
sslab@ubuntu:~$ ./proxy_cache
[3933]input CMD> connect
[3934]input URL> bye
[3933]input CMD> quit
sslab@ubuntu:~$
```

Proxy 1-3(3/4)

- **Concurrent Server Implementation with `fork()`**
- **Sub process**
 - 사용자의 URL을 입력 받고, Proxy 1-2에서의 연산을 수행
 - 동작:
 - **터미널에 `[(pid)]input URL>`를 출력하고, 사용자의 명령어 입력을 대기**
 - **`[(pid)] → getpid()` 통해 얻은 값 사용**
 - **bye 입력**
 - 해당 프로세스 종료
 - **URL 입력**
 - Proxy 1-2에서의 연산 수행
 - SHA-1 function (input_url to hashed_url)
 - Check: HIT or MISS
 - Manipulate cache directory
 - logging
 - **참고 사항**
 - 이전 "Sub process"가 생성한 cache file도 유지
 - Log file은 1개만 유지

```
sslab@ubuntu:~$ ./proxy_cache
[3933]input CMD> connect
[3934]input URL> www.kw.ac.kr
[3934]input URL> www.google.com
[3934]input URL> bye
[3933]input CMD> connect
[3935]input URL> bye
[3933]input CMD> quit
```

Proxy 1-3(4/4)

- Example

```
sslab@ubuntu:~$ ./proxy_cache
```

```
[3933]input CMD> connect
```

```
[3934]input URL> www.kw.ac.kr
```

```
[3934]input URL> www.google.com
```

```
[3934]input URL> bye
```

```
[3933]input CMD> connect
```

```
[3935]input URL> www.kw.ac.kr
```

```
[3935]input URL> www.naver.com
```

```
[3935]input URL> bye
```

```
[3933]input CMD> quit
```

```
sslab@ubuntu:~$ cat ~/logfile/logfile.txt
```

```
[Miss]www.kw.ac.kr-[2022/03/26, 23:25:22]
```

```
[Miss]www.google.com-[2022/03/26, 23:25:25]
```

```
[Terminated] run time: 11 sec. #request hit : 0, miss : 2
```

```
[Hit]e00/0f293fe62e97369e4b716bb3e78fababf8f90-[2022/03/26, 23:25:30]
```

```
[Hit]www.kw.ac.kr
```

```
[Miss]www.naver.com-[2022/03/26, 23:25:33]
```

```
[Terminated] run time: 7 sec. #request hit : 1, miss : 1
```

```
**SERVER** [Terminated] run time: 20 sec. #sub process: 2
```

```
sslab@ubuntu:~$
```


Report Requirements

- **Ubuntu 20.04.6 Desktop 64bits 환경에서 채점**
- **Copy 발견 시 0점 처리**
- **보고서 구성**
 - **보고서 표지**
 - 수업 명, 과제 이름, 담당 교수님, 학번, 이름, 강의 시간 필히 명시
 - 과제 이름 → Proxy 1-2
 - **아래의 내용은 보고서에 필히 포함**
 - Introduction
 - 과제 소개 – 4줄 이상(background 제외) 작성
 - Flow Chart
 - 코드 작성 순서도
 - Pseudo code
 - 알고리즘
 - 결과화면
 - 수행한 내용을 캡처 및 설명
 - 고찰
 - 과제를 수행하면서 느낀 점 작성
 - Reference
 - 과제를 수행하면서 참고한 내용을 구체적으로 기록
 - 강의자료만 이용한 경우 생략 가능

Report Requirements

- **Softcopy Upload**

- **제출 파일**

- 보고서 + 소스파일 하나의 압축 파일로 압축하여 제출(tar.xz)
 - 1)보고서:
 - 보고서를 **pdf로 변환**하여 제출
 - 보고서 이름은 *Proxy1-3_수강분류코드_학번* 이름 으로 작성
 - 2)C 파일 명:
 - **proxy_cache.c**
 - **Comment 작성(Appendix 내용 참고)**
 - 3)Makefile:
 - 실행파일명: **proxy_cache**
 - C 파일명, 실행파일명 지정한 이름 외 다른 명으로 작성 시 감점

- **tar.xz 압축 방법**

- **(Appendix 내용 참고)**

- 컴파일은 무조건 Makefile(makefile)을 이용한 make로 함.
 - Makefile(makefile) 없거나 실행 불가시 0점
 - 파일 압축 오류 시, 0점 처리

Report Requirements

- 실습 수업을 수강하는 학생인 경우

- 실습 과목에 과제를 제출(.tar.xz)

- 이론 과목에 간단한 .txt 파일로 제출

실습수업때 제출했습니다.

2022-08-29 오후 3:58

텍스트 문서

0KB

- 이론 과목에 .txt 파일 미 제출 시 감점

- .tar.xz 파일로 제출 하지 않을 시 감점

- 예시-이론 월5 수6 수강하는 학생인 경우

- 보고서: Proxy1-3_A_2025123456_홍길동.pdf

- 압축 파일 명: Proxy1-3_A_2025123456_홍길동.tar.xz

수강요일	이론1 월5수6	이론2 목4	실습1 목12
수강분류 코드	A	B	C

- 과제 제출

- KLAS – 강의 과제 제출

- 2025년 4월 17일 목요일 23:59까지 제출

- 딜레이 받지 않음

- 제출 마감 시간 내 미제출시 해당 과제 0점 처리

- 교내 서버 문제 발생 시, 메일로 과제 제출 허용