

2022년 1학기 시스템프로그래밍실습 13주차

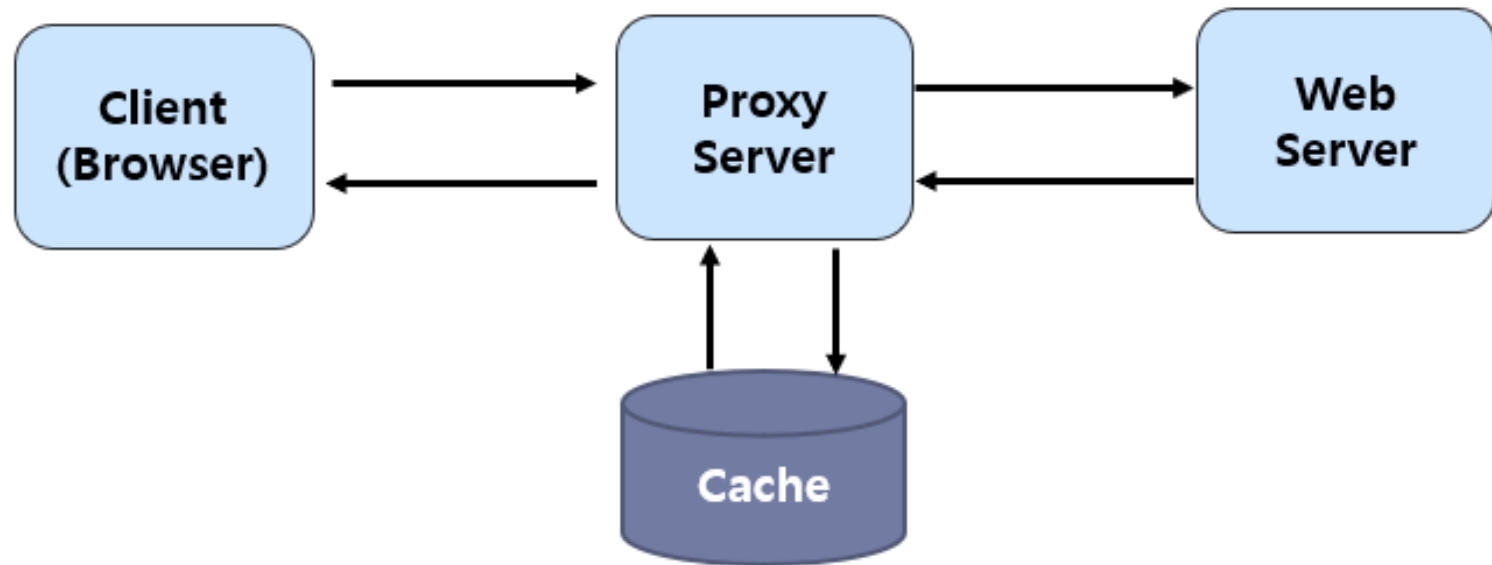
Thread

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

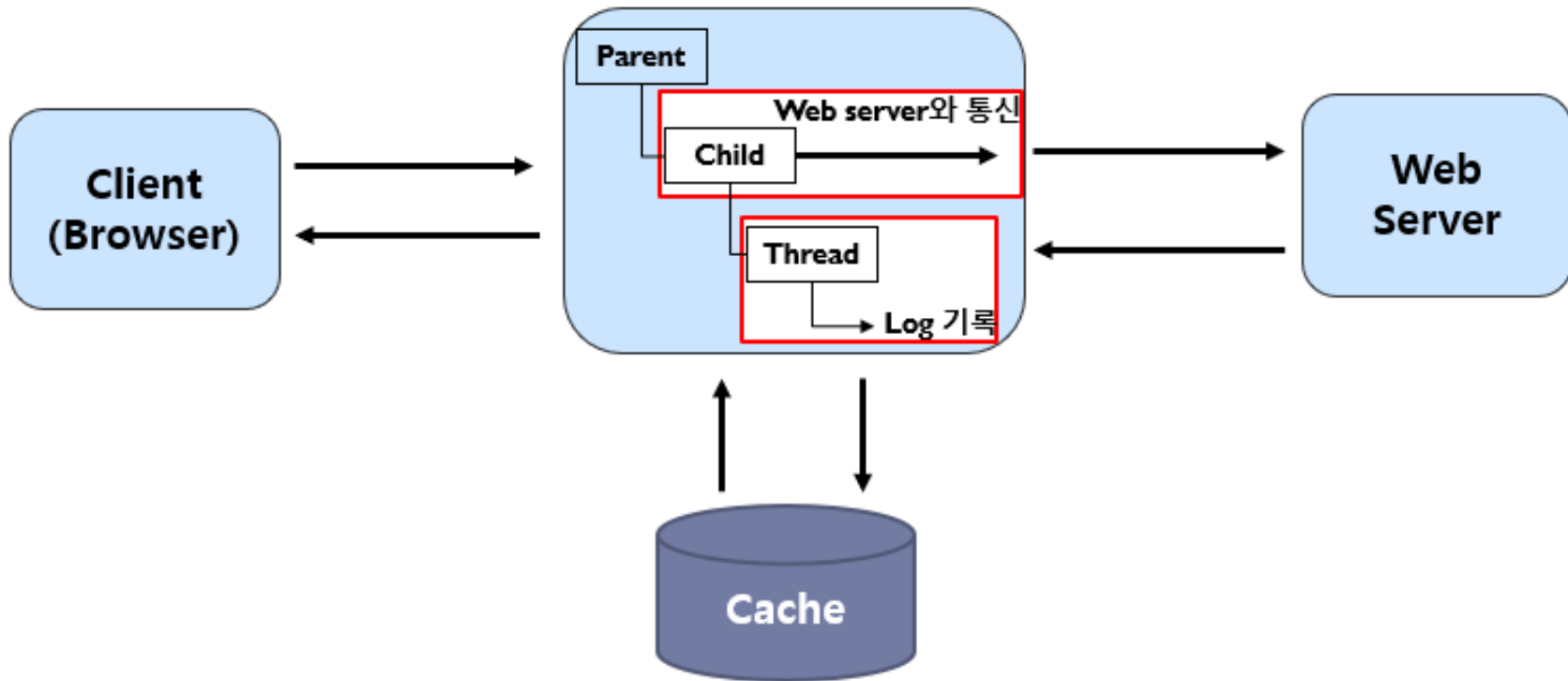
3st Assignment's Descriptions

- **Assignment 3-1**
 - Synchronize the shared resource
- **Assignment 3-2**
 - Logging using threads.

Proxy Server의 동작(1/2)



Proxy Server의 동작(2/2)

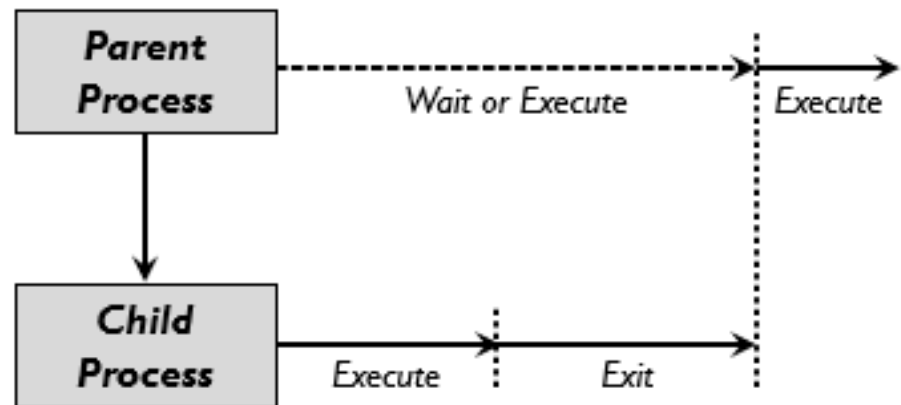
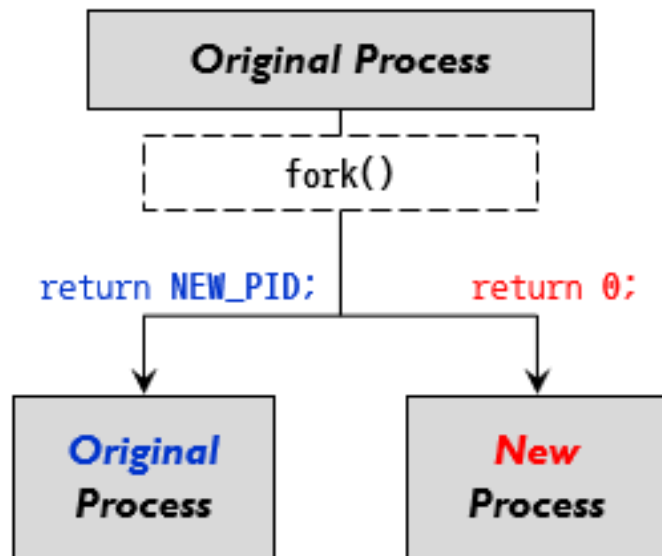


- **Child:** log 기록을 위한 thread를 생성하고 웹 서버와 통신
- **Thread:** log 기록 동작을 thread에서 수행

Process Creation API (1/2)

- **fork()**

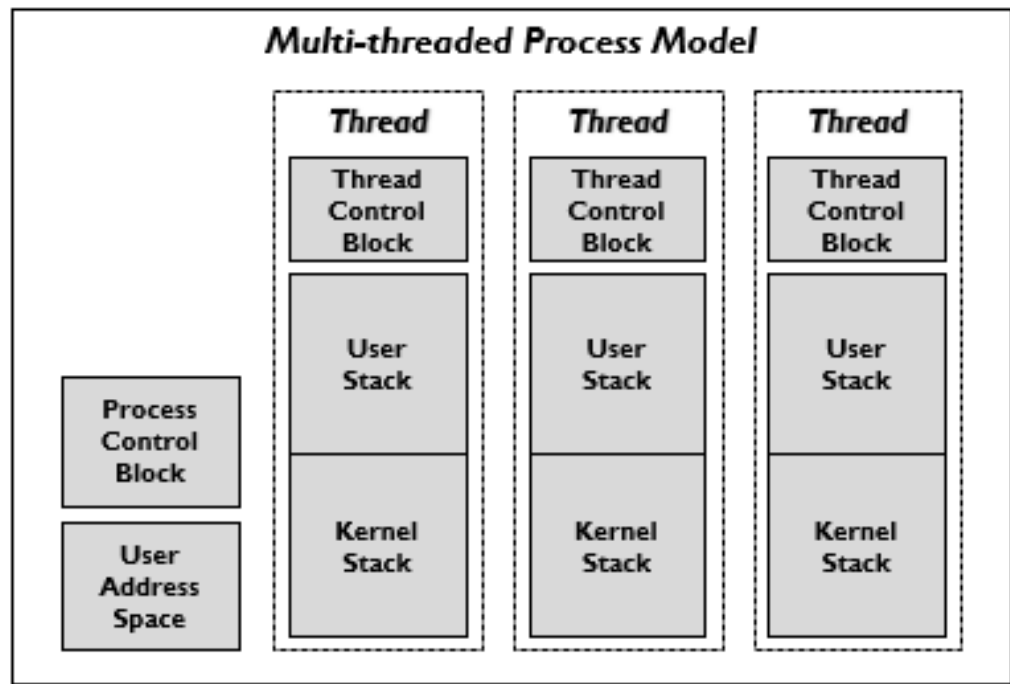
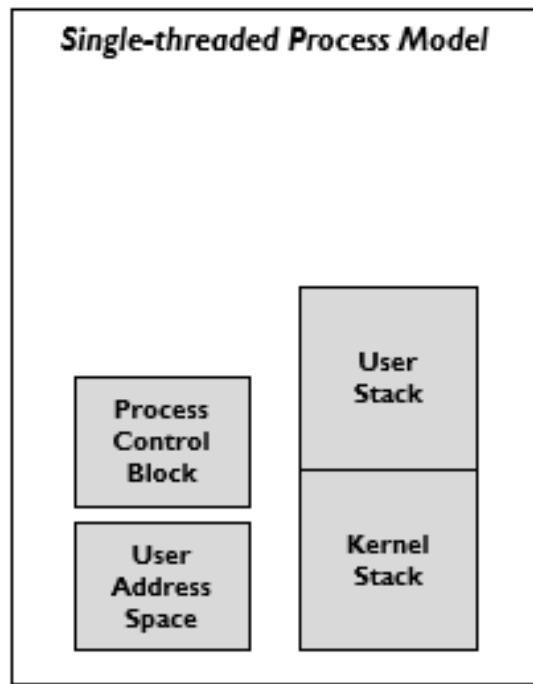
- 새로운 프로세스는 부모 프로세스로부터 생성
 - 생성된 프로세스 : 자식 프로세스 (child process)
 - fork()를 호출한 프로세스 : 부모 프로세스 (parent process)
 - 이 시점에서 두 프로세스가 동시 작업 수행



Process Creation API(2/2)

▪ Thread

- 특정 Process 내에서 실행되는 하나의 흐름을 나타내는 단위
- 독립된 Program Counter를 갖는 단위
- 독립된 Register Set과 Stack을 가짐
- 비 동기적인(Asynchronous) 두 개의 작업이 서로 독립적으로 진행 가능
 - 처리를 위해 조건 변수나 Semaphore와 같은 방법을 사용함



POSIX Thread

- **POSIX**

- 이식 가능 운영 체제 인터페이스(Portable Operating System Interface)
- 서로 다른 UNIX OS의 공통 API를 정리하여 이식성이 높은 유닉스 응용 프로그램을 개발하기 위한 목적으로 IEEE가 책정한 애플리케이션 인터페이스 규격

- **POSIX Thread**

함수 명	설명
pthread_create	새로운 Thread를 생성함
pthread_detach	Thread가 자원을 해제하도록 설정
pthread_equal	두 Thread의 ID 비교
pthread_exit	Process는 유지하면서 지정된 Thread 종료
pthread_kill	해당 Thread에게 Signal를 보냄
pthread_join	임의의 Thread가 다른 Thread의 종료를 기다림
pthread_self	자신의 Thread ID를 얻음

- 컴파일시 **-lpthread** 혹은 **-pthread** 옵션 추가
 - e.g. \$ **gcc -pthread** thread_test.c

POSIX Thread: Creation

- Thread는 `pthread_t` 타입의 thread ID로 처리
- POSIX thread는 사용자가 지정한 특정 함수를 호출함으로써 시작
 - 이 thread 시작 function은 `void*` 형의 인자를 하나 취한다

- **사용 함수: `pthread_create()`**

```
#include <pthread.h>
```

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start)(void*), void *arg);
```

- `pthread_t *thread` : Thread ID
- `const pthread_attr_t *attr` : Thread 속성 지정. 기본값은 NULL.
- `void *(*start)(void*)` : 특정 함수(**start**) 를 호출함으로써 thread가 시작
- `void *arg` : start 함수의 인자

POSIX Thread: Termination

- Process는 유지 하면서 pthread_exit() 함수를 호출하여 thread 자신을 종료
- 단순히 thread를 종료 하는 역할만 수행
 - 단, thread의 resource가 완전히 정리되지 않음

```
#include <pthread.h>

int pthread_exit(void *retval);
```

- void ***retval** : Return value가 저장. 사용하지 않으면, NULL

POSIX Thread: Detach and Join

- **Detach: 분리**
 - Process와 thread가 분리되면서 종료 시 자신이 사용했던 자원을 바로 반납
- **Join: 결합**
 - 생성된 thread가 pthread_join()을 호출한 thread에게 반환 값을 전달하고 종료
- 즉, thread를 종료 할 때 분리 혹은 결합이 필요

POSIX Thread: Detach

- 결합 가능(joinable)한 상태의 thread
 - 분리되지 않은 thread
 - 종료되더라도 자원이 해제되지 않음
- **pthread_detach()**
 - Thread 종료 시 자원을 반납하도록 지정된 thread를 분리(detach) 상태로 만든다.

```
#include <pthread.h>

int pthread_detach (pthread_t thread);
```

- pthread_t **thread** : 분리시킬 쓰레드 식별자
- Return value
 - 성공 시: 0
 - 실패 시: 0이 아닌 오류 코드

POSIX Thread: Join

- 다른 thread가 `thread_join()`을 반드시 호출해야 함
 - Thread의 Memory Resource가 완전히 정리되지 않음
- 지정된 thread가 종료될 때까지 호출 thread의 수행을 중단

```
#include <pthread.h>
```

```
int pthread_join (pthread_t thread, void **value_ptr);
```

- `waitpid()`의 역할과 유사
- `pthread_t thread` : 기다릴 thread의 식별자
- `void **value_ptr` : thread의 종료코드가 저장될 장소.
- Return value
 - 성공 시: 0
 - 실패 시: 0이 아닌 오류 코드

실습1. thread() – Example

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

void *thr_fn(void* buf)
{
    printf("thread[%s] is created.\n", (char*)buf);
}

int main()
{
    char buf[255];
    int status;
    int err;
    void *tret;

    pthread_t tid;

    printf("%s\n", " ");
    while(fgets(buf, 255, stdin) != NULL){
        buf[strlen(buf) - 1] = 0;

        err = pthread_create(&tid, NULL, thr_fn, (void*)buf);

        if(err != 0){
            printf("pthread_create() error.\n");
            return 0;
        }

        pthread_join(tid, &tret);

        printf("%s\n", " ");
    }

    return 0;
}
```

```
sslab@sslab-VirtualBox:~$ gcc -pthread thread.c
sslab@sslab-VirtualBox:~$ ./a.out
% a
thread[a] is created.
% b
thread[b] is created.
% c
thread[c] is created.
% <Ctrl + D>
```

2022년 1학기 시스템프로그래밍실습

Proxy #3-2

System Software Laboratory
College of Software and Convergence
Kwangwoon Univ.

Proxy #3-2

- Requirements

- 3-1 과제에 log file을 기록하는 thread 추가

- 즉, Thread는 3-1의 critical section 내에서 log를 기록하는 thread 추가

- Terminal에 메시지 추가

- 3-1에서 구현했던 메시지는 그대로 유지
 - critical section 접근부만 TID 관련 정보 추가
 - 어떤 자식 프로세스가 스레드를 생성하였는가?
ex) *PID# 19564 create the *TID# 11.
 - 스레드가 종료될 시
ex) *TID# 11 is exited.

- 캐시 파일의 락 경쟁 문제는 고려하지 않음

- Terminal 화면

```
root@sslab-desktop:/proxy# ./proxy
*PID# 3996 is waiting for the semaphore.
*PID# 3996 is in the critical zone.
*PID# 3996 create the *TID# 3997419264.
*TID# 3997419264 is exited.
*PID# 3996 exited the critical section.
*PID# 4119 is waiting for the semaphore.
*PID# 4119 is in the critical zone.
*PID# 4119 create the *TID# 3997419264.
*TID# 3997419264 is exited.
*PID# 4119 exited the critical section.
```