멀티쓰레드 기반의 서버 구현

인하공업전문대학 컴퓨터정보과
최효현 교수

주요사항

- 쓰레드의 개념
- 임계 영역 이해
- 동기화 필요성 이해
- Mutex 사용법 숙지
- 세마포어 사용법 숙지

쓰레드(Thread)의 이론적 이해

1. 경량화 된 프로세스

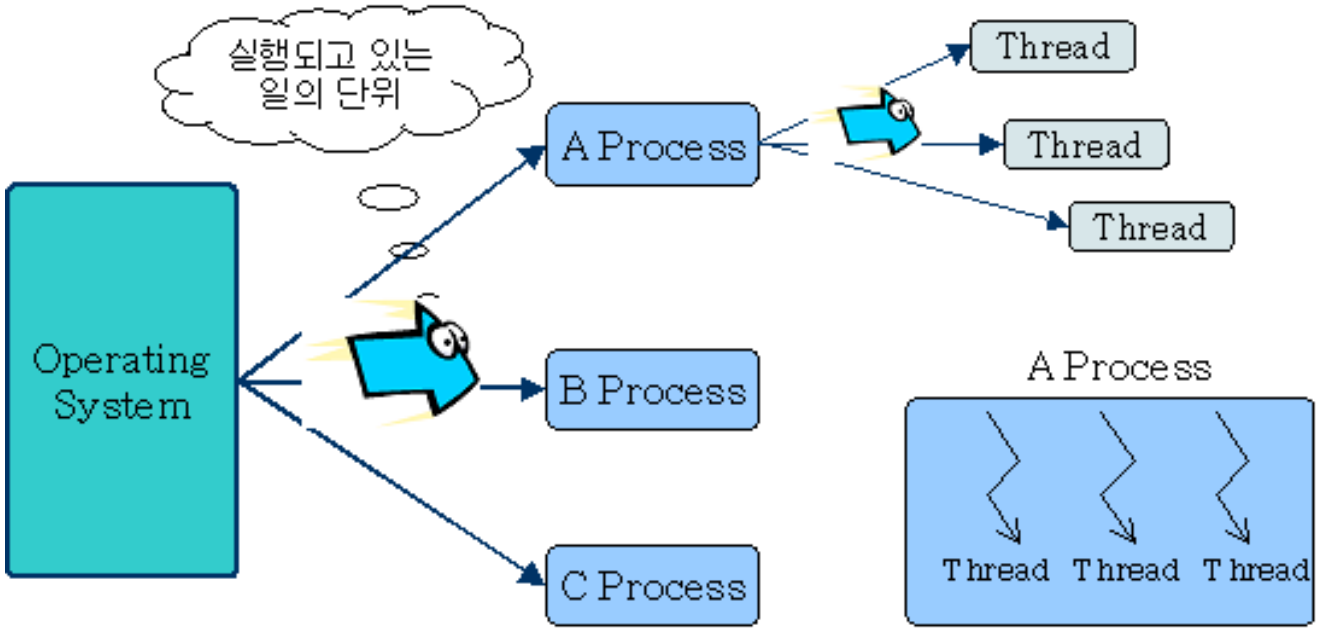
- 프로세스와 마찬가지로 동시 실행이 가능하다.
- 프로세스의 단점을 극복하기 위해 등장.

2. 프로세스와의 차이점

- 스택을 제외한 나머지 메모리 영역을 공유
- 프로세스 보다 간단한 컨텍스트 스위칭
- 일부 메모리를 공유하므로 쓰레드간 통신이 편리
- 생성시간 비교시 쓰레드가 20배-100배 정도 빠름

쓰레드(Thread)의 이론적 이해

3. 프로세스와 쓰레드



[그림 17-1]

쓰레드의 생성 및 실행

- **POSIX 쓰레드 생성**

  - POSIX 쓰레드 라이브러리에 있는 함수를 사용하여 생성함

```
#include <pthread.h>

int pthread_create (pthread_t * thread,
                    pthread_attr_t * attr,
                    void*(*start_routine)(void*),
                    void * arg
);                                          (성공시 0을 리턴)
```

- thread: 생성된 **쓰레드의 ID를 저장할 변수**의 포인터를 전달함
- attr : 생성하고자하는 쓰레드의 속성을 설정, 일반적으로 **NULL**을 전달함
- start_routine : 함수 포인터를 요구함, 쓰레드 생성 후 **실행할 함수 루틴** 설정
- arg: 쓰레드에 의해 호출되는 함수(*start_routine)에 전달할 **인자값**

※ **POSIX(Portable Operating System Interface)**의 약자로써 운영체제의 공통적인 규약을 규정하고 있는 표준.

쓰레드의 생성 및 실행

1. 프로그램 예제

- thread1.c
- 컴파일시 옵션

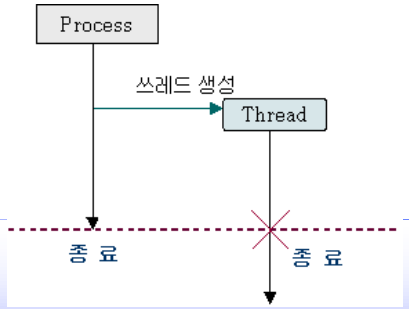
$> gcc -D_REENTRANT thread1.c –o thread -lpthread

불안전한 쓰레드 함수 호출문제를 해결

쓰레드 라이브러리에 링크 걸어줌

2. 실행결과 문제점

- 메시지가 **3**번 출력되어야 하는데, **1**번만 출력 **->** 너무 빨리 쓰레드가 소멸됨
- **3**번 메시지를 출력하기 위해서는 **main** 함수에서 **6**초 이상 지연이 필요
- 생성된 쓰레드에 의해 실행되는 **thread-function** 함수보다 **main** 함수가 더 일찍 종료됨으로 생기는 원인
  - 해결**: main** 함수를 **10**초 정도 대기시킴



쓰레드의 생성 및 실행

3. 개선방법

- 실행시간 예측하지 않고 해결 -> pthread_join() 이용

```
#include <pthread.h>

int pthread_join(pthread_t th, void **thread_return);
```

(성공시 0을 리턴)

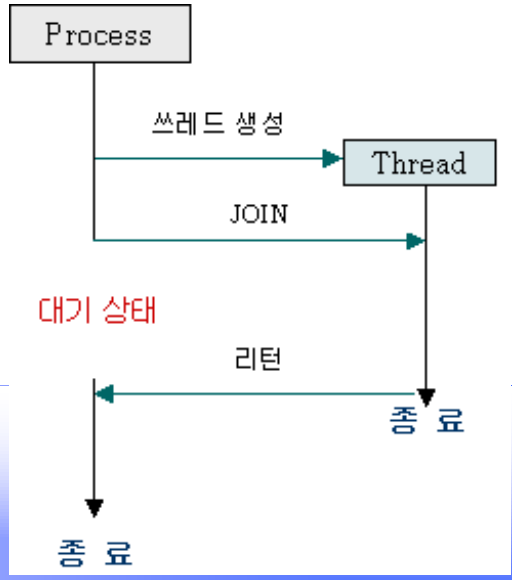
- th : th 인자로 들어오는 ID의 쓰레드가 종료할때까지 실행을 지연시킴
- thread_return: 쓰레드가 종료시 반환하는값에 접근할 수 있는 포인터

- pthread_join 함수는 인자로 전달되는 ID에 해당하는 쓰레드가 종료될때까지 대기 상태에 들어가 위해 호출하는 함수

쓰레드의 생성 및 실행

4. 프로그램실행
   - thread2.c

5. 실행과정
   - 프로세스가 thread에게 JOIN 메시지를 전달하면서 일단 대기상태에 들어감.
   - JOIN 메시지를 받은 쓰레드가 종료되게 되면 프로세스는 다시 이어서 실행



쓰레드의 문제점과 임계영역(Critical Section)

1. 임계영역
   - 두개 이상의 쓰레드에 의해서 동시에 실행되면 안 되는 영역.

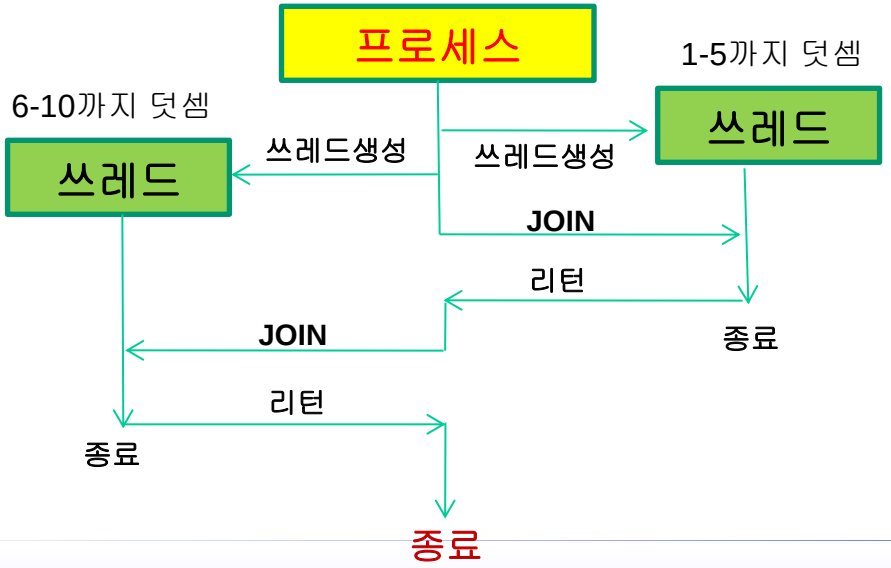
2. 쓰레드 안전한 함수(Thread safe function)
   - 임계영역에서 호출이 가능한 함수.
   - 예) gethostbyname 대 gethostbyname_r

3. 컴파일시 안전함수 사용문제 해결방법
   - 헤더화일을 포함하기전에 "_REENTRANT" 매크로가 정의되어 있는경우 gethostbyname 호출문장이 자동적으로 gethostbyname_r 함수 호출로 변경된다.

```
$> gcc  -D_REENTRANT thread1.c –o thread -lpthread
```

쓰레드의 문제점과 임계영역

동시에 실행되는 쓰레드 프로그램 예제
- Boss/Worker 쓰레드 모델
  - 1-10까지 더하는 문제
  - thread3.c

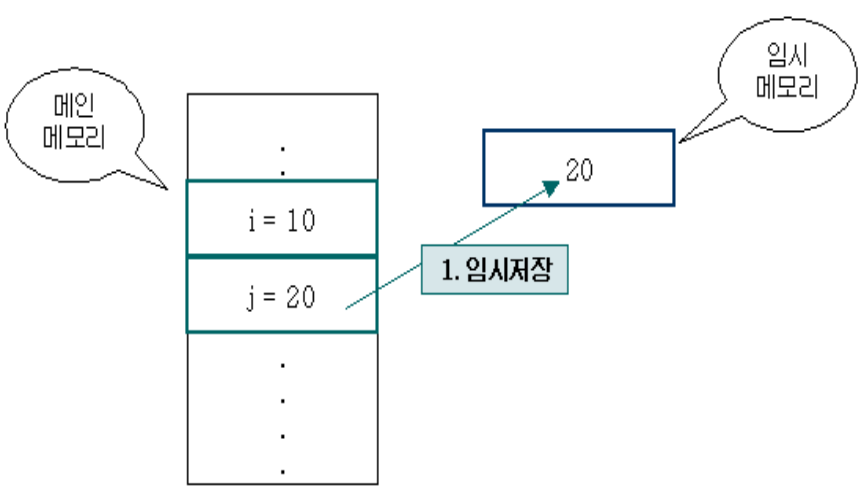


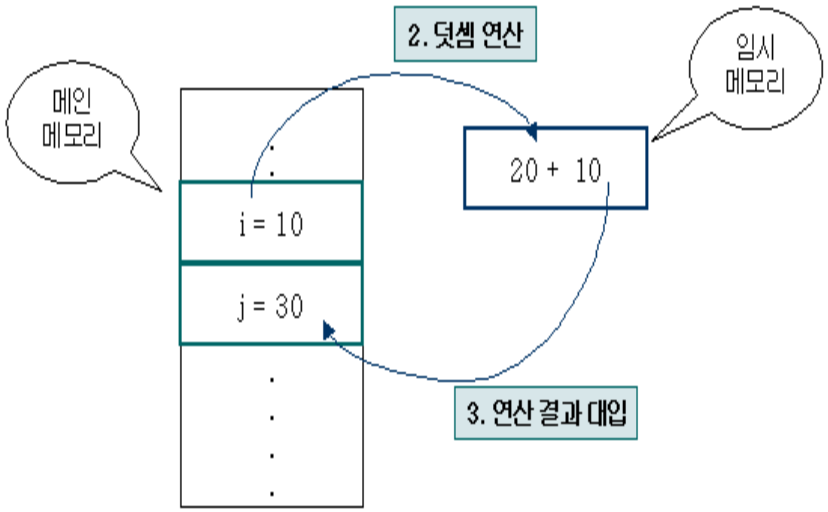
쓰레드의 문제점과 임계영역

- 임계영역(Critical Section) 처리 문제
- 덧셈 계산원리
  - 쓰레드로 계산시 문제가 발생할 소지가 있음

쓰레드의 문제점과 임계영역

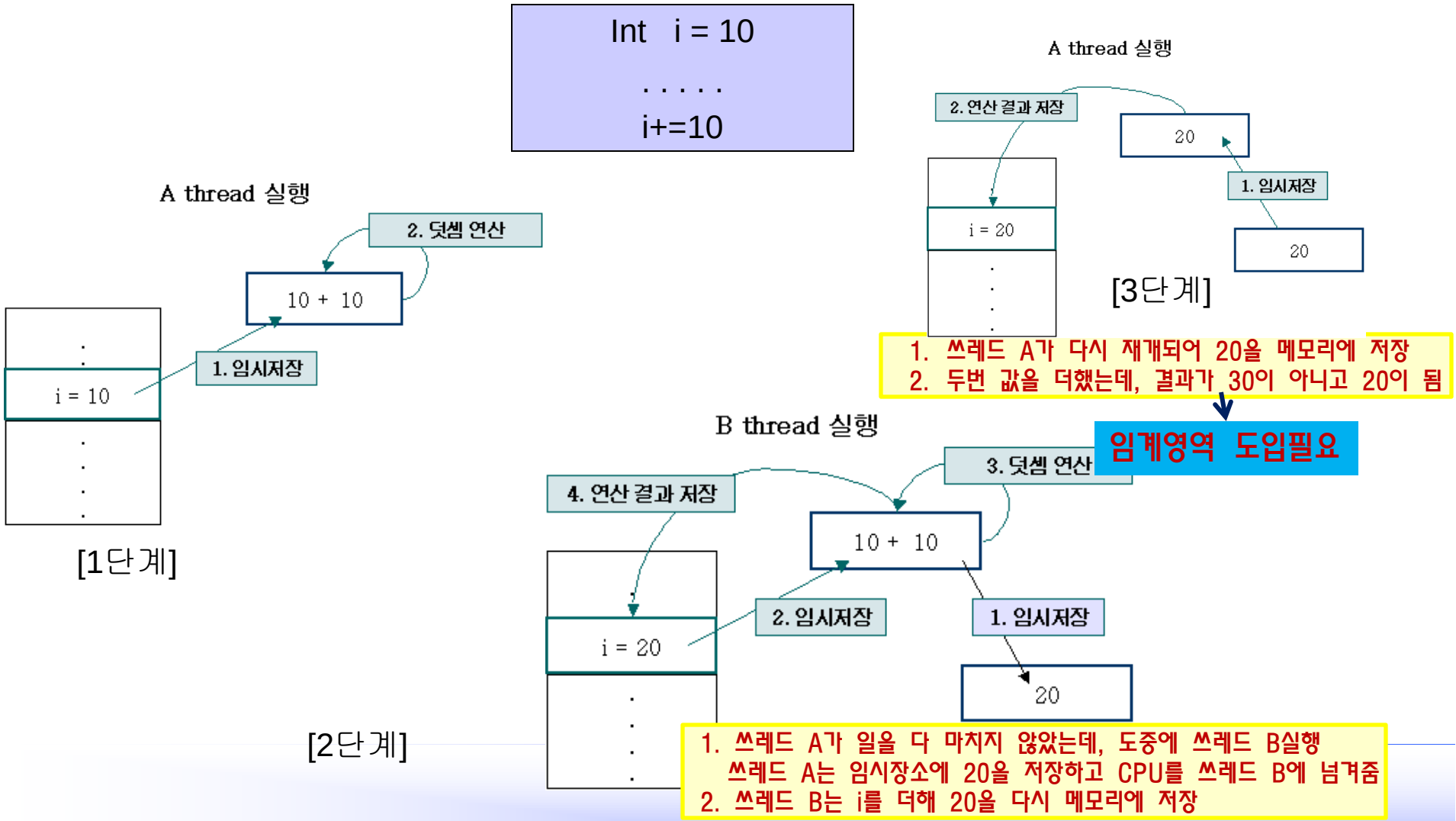
```
int i = 10
int j = 20
j+=i
```





j가 다른 메모리영역에서 계산되므로 문제발생

쓰레드의 문제점과 임계영역



쓰레드 동기화(Synchronization)

1. 임계영역
   - 둘 이상의 쓰레드에 의해서 공유되는 메모리 공간에 접근하는 코드영역.

2. 동기화.
   - 첫째 : 공유된 메모리에 둘 이상의 쓰레드가 동시 접근하는 것을 막는 행위
   - 둘째 : 둘 이상의 쓰레드 실행순서를 컨트롤(control)하는 행위.

3. 대표적인 동기화 기법.
   - 뮤텍스(mutex), 세마포어(semaphore)

뮤텍스(MUTEX)

1. 무엇을 의미하는 것인가?

- pthread_mutex_t 타입의 변수를 가리켜 뮤텍스라고 표현한다.
- 일종의 문고리에 해당한다.

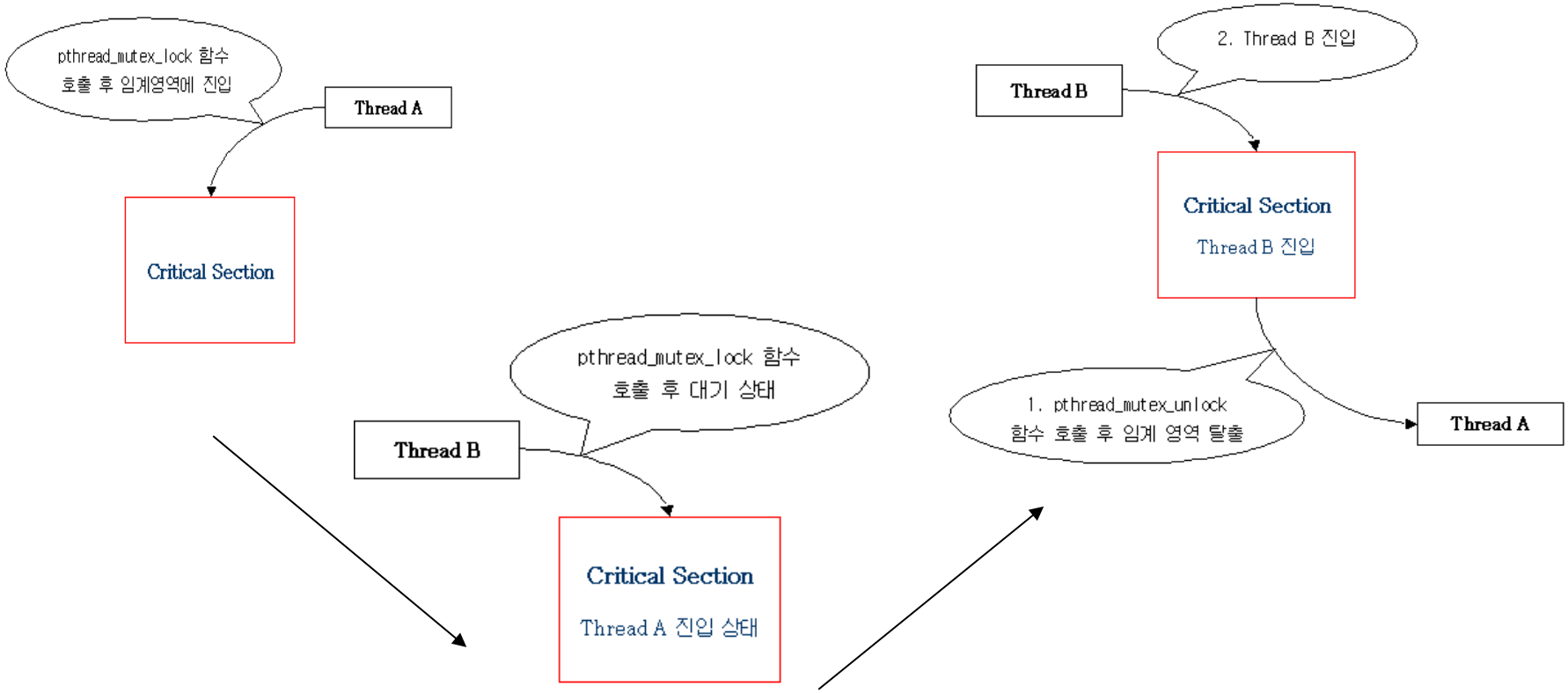
2. 기본 원리는 무엇인가?

- 임계영역에 들어 갈 때 뮤텍스(문고리)를 잠그고 들어간다.
- 임계영역에서 빠져 나올 때 뮤텍스를 풀고 나간다.

3. 뮤텍스를 조작하는 함수들

- 뮤텍스 초기화 함수 : pthread_mutex_init
- 뮤텍스 소멸 함수 : pthread_mutex_destroy
- 뮤텍스 잠그는 함수 : pthread_mutex_lock
- 뮤텍스 풀어주는 함수 : pthread_mutex_unlock

뮤텍스의 동기화 원리


pthread_mutex_lock 함수
호출 후 임계영역에 진입
Thread A
Critical Section
pthread_mutex_lock 함수
호출 후 대기 상태
Thread B
Critical Section
Thread A 진입 상태
2. Thread B 진입
Thread B
Critical Section
Thread B 진입
1. pthread_mutex_unlock
함수 호출 후 임계 영역 탈출
Thread A

예제 확인

1. 프로그램 예제

- mutex.c

쓰레드가 호출하는 함수 루틴내에서 전역으로
선언된 변수 및 배열에 접근하는 코드가 존재시
=> 그곳이 CS영역이 됨

임계영역에 쓰레드의 동시접근을 제어하기위해
CS영역 시작위치에서 : pthread_mutex_lock
CS가 끝나는위치에서 : pthread_mutex_unlock

세마포어

1. 무엇을 의미하는 것인가?

- sem_t 타입의 변수를 가리켜 흔히 세마포어라 표현한다.

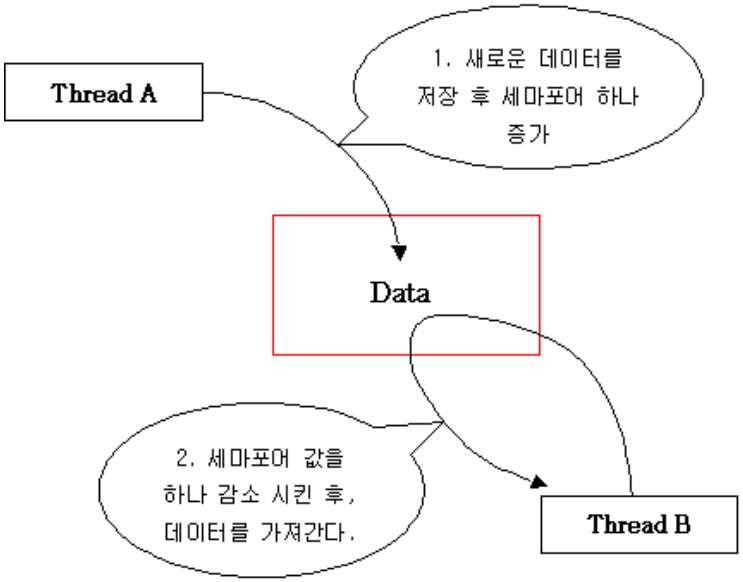
2. 기본 원리는 무엇인가?

- 세마포어는 정수를 지닌다. 정수의 값이 0이면 실행 불가를 의미한다.
- 세마포어가 1이상이 되면 실행 가능을 의미한다.
- 세마포어는 0미만이 될 수 없지만 1이상은 될 수 있다
- 0과 1의 상태만을 지니는 세마포어를 가리켜 바이너리 세마포어라 한다.

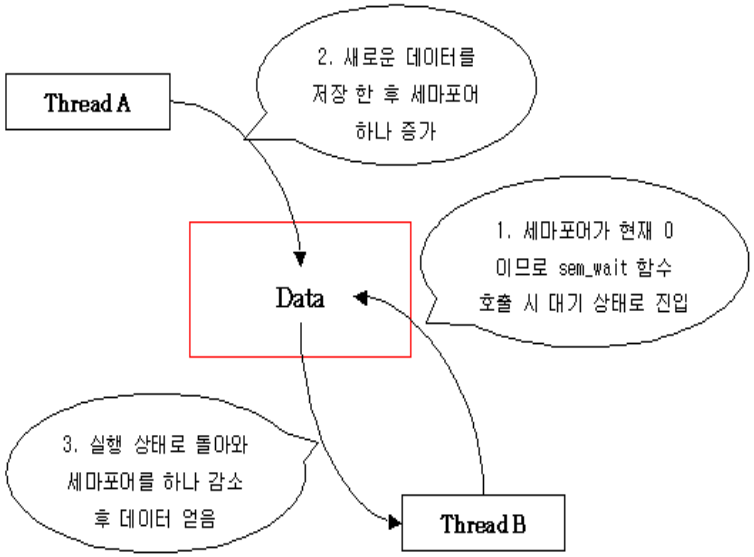
3. 세마포어를 조작하는 함수들

- 세마포어 초기화 함수 : sem_init
- 세마포어 소멸 함수 : sem_destroy
- 세마포어 감소 함수 : sem_wait
- 세마포어 증가 함수 : sem_post

세마포어 동기화 원리



[그림 17-17]



[그림 17-18]

쓰레드 기반 다중 접속 서버구현하기

1. 채팅프로그램 – 쓰레드 기반으로 구현하기
   - chat_serv.c
   - chat_clnt.c

2. 서버에 접속한 사람들간에 메시지 주고받기
3. 동기화 기법 확인

Q&A