Computer Networks (Lab)

Multi-threading-based Server (& Client)

2023. 5. 19

Young Deok Park (박영덕)

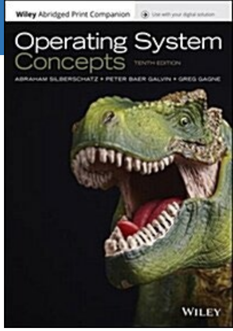
Let’s Implement More Practical Client/Server

- Multiple clients can be served at a time 😆

- How?
  - ✓ Multi-process-based server
  - I/O Multiplexing-based server
  - **Multi-threading-based approach (Today!)**
    - We will do not focus on echo client/server anymore!!
      - ✓ We will develop practical chatting program Today

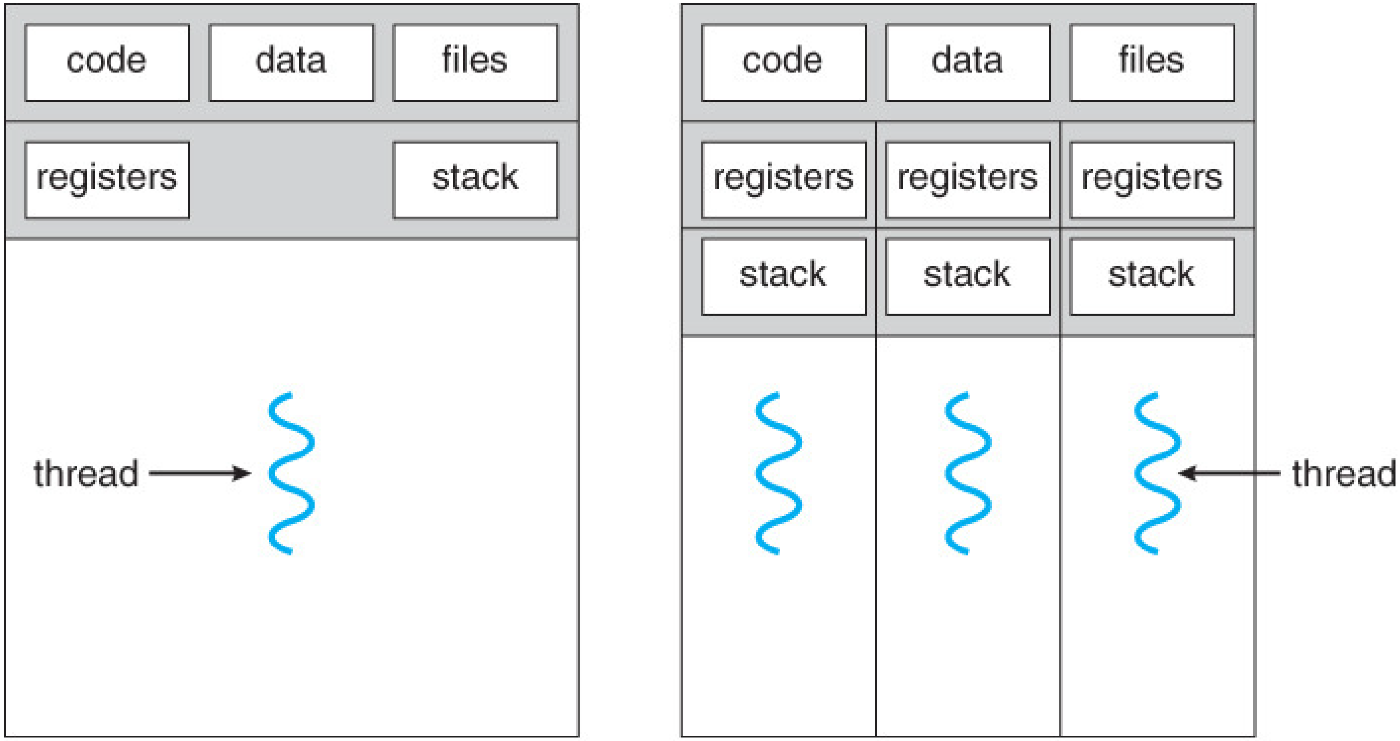
Multi-process-based Server

- Process-creation method is time consuming and resource intensive
- Inter process communication (IPC) is required



Multi-threading-based Approach (1/2)

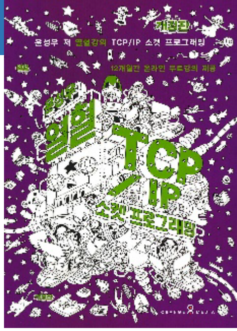
- Thread
  - Basic unit of CPU utilization
  - Sometimes called a “lightweight process”



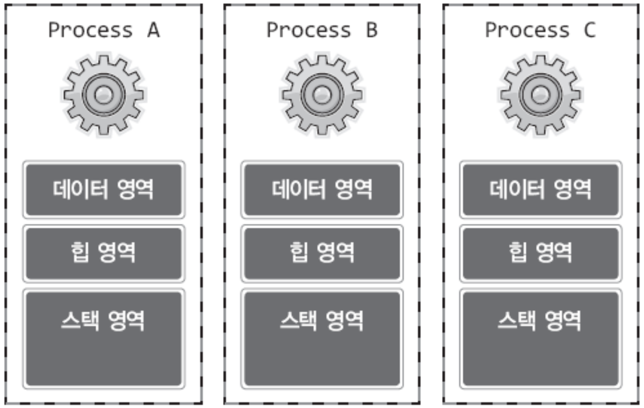
single-threaded process

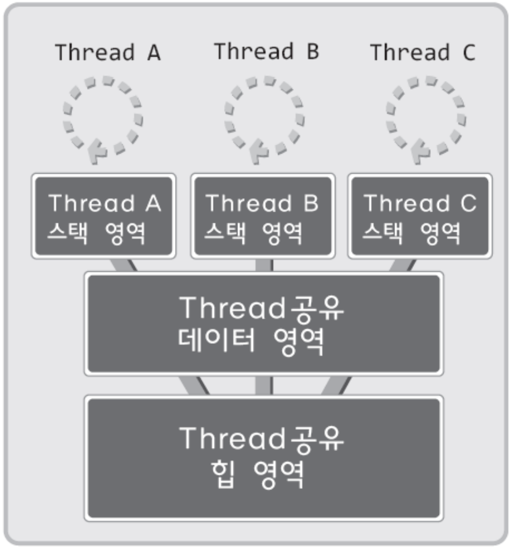
multithreaded process

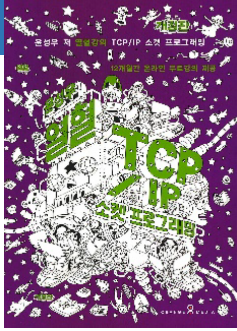
Multi-threading-based Approach (2/2)



- Thread
  - Basic unit of CPU utilization
  - Sometimes called a “lightweight process”

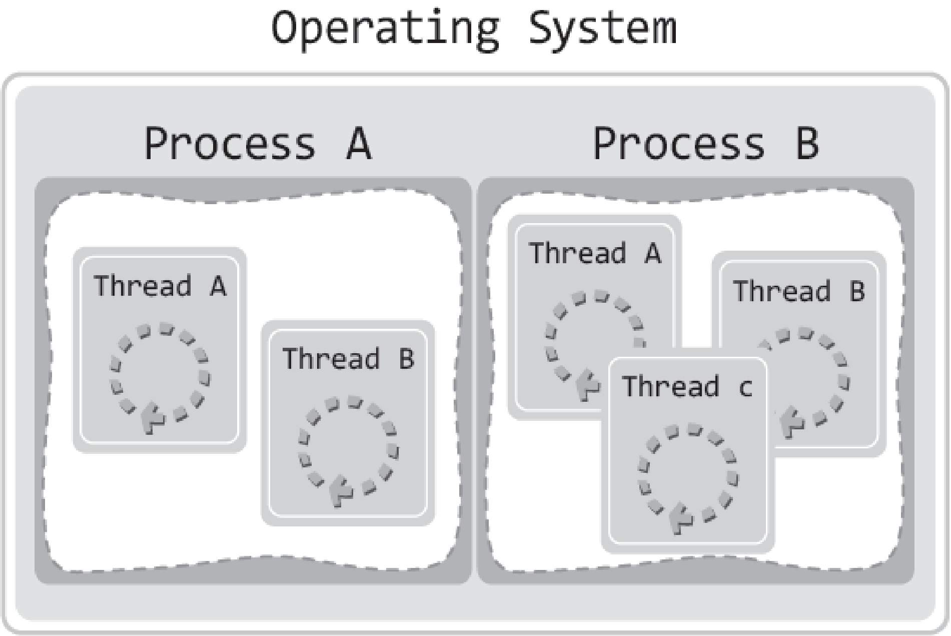






Multi-threading-based Approach (2/2)

- Thread
  - Basic unit of CPU utilization
  - Sometimes called a “lightweight process”



Thread Creation

```
#include <pthread.h>

int  pthread_create (pthread_t * restrict thread, const  ptread_attr_t * restrict attr, void * (*start_routine)
(void *), void * restrict arg);
```

thread 생성된 쓰레드의 ID가 저장될 변수의 주소값

attr 생성할 쓰레드의 특성 정보를 전달 (NULL 전달 시 기본 특성의 쓰레드 생성)

start routine 쓰레드의 main 함수 역할을 하는 함수 주소 전달 (별도의 실행흐름이 시작되는 함수주소 전달)

arg 쓰레드의 main 함수로 전달될 인자값

※ Return value
Success : 0
Error : !=0

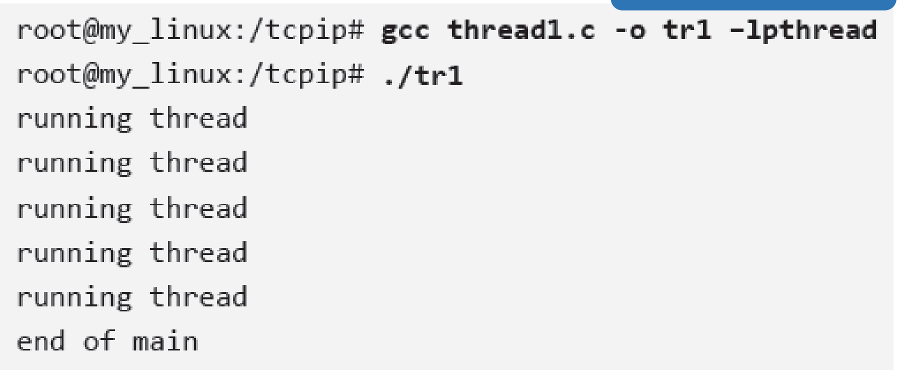
Thread Creation - Example

thread1.c

```
int main(int argc, char *argv[])
{
    pthread_t t_id;
    int thread_param=5;

    if(pthread_create(&t_id, NULL, thread_main, (void*)&thread_param)!=0)
    {
        puts("pthread_create() error");
        return -1;
    };
    sleep(10); puts("end of main");
    return 0;
}
```

```
void* thread_main(void *arg)
{
    int i;
    int cnt=*((int*)arg);
    for(i=0; i<cnt; i++)
    {
        sleep(1); puts("running thread");
    }
    return NULL;
}
```

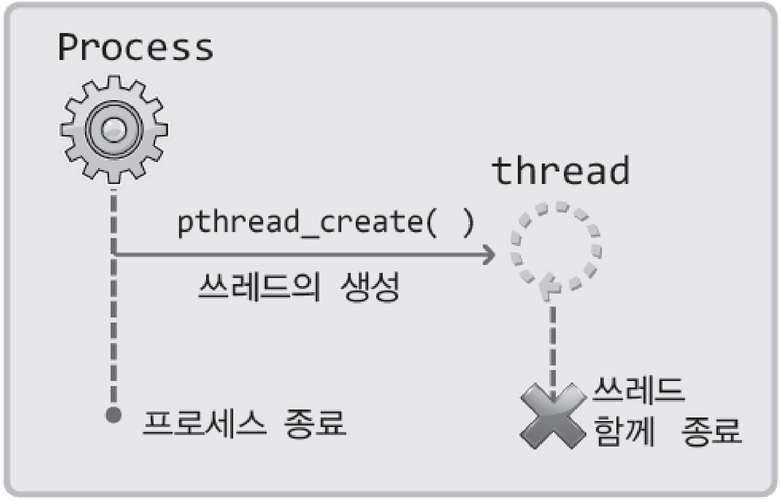
실행결과

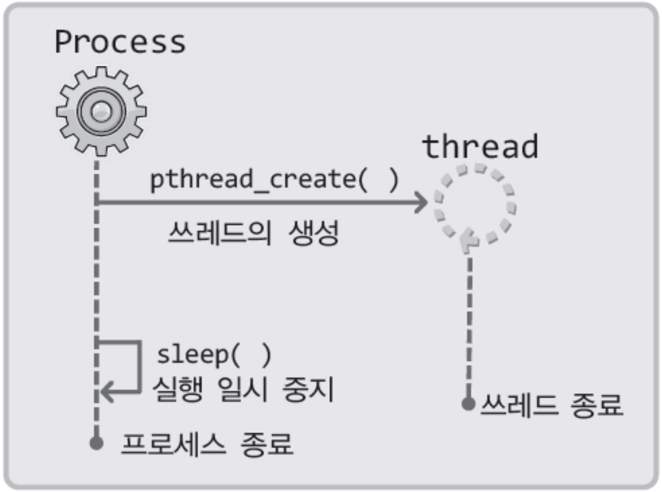


```
root@my_linux:/tcpip# gcc thread1.c -o tr1 -lpthread
root@my_linux:/tcpip# ./tr1
running thread
running thread
running thread
running thread
running thread
end of main
```

Yeungnam University

Thread Termination - Example





프로세스가 종료되면 해당 프로세스 내에서 생성된 쓰레드도 함께 종료

sleep() 함수를 통해 프로그램의 흐름을 관리하는 데에는 한계가 있음

Thread Termination – pthread_join

```
#include <pthread.h>

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

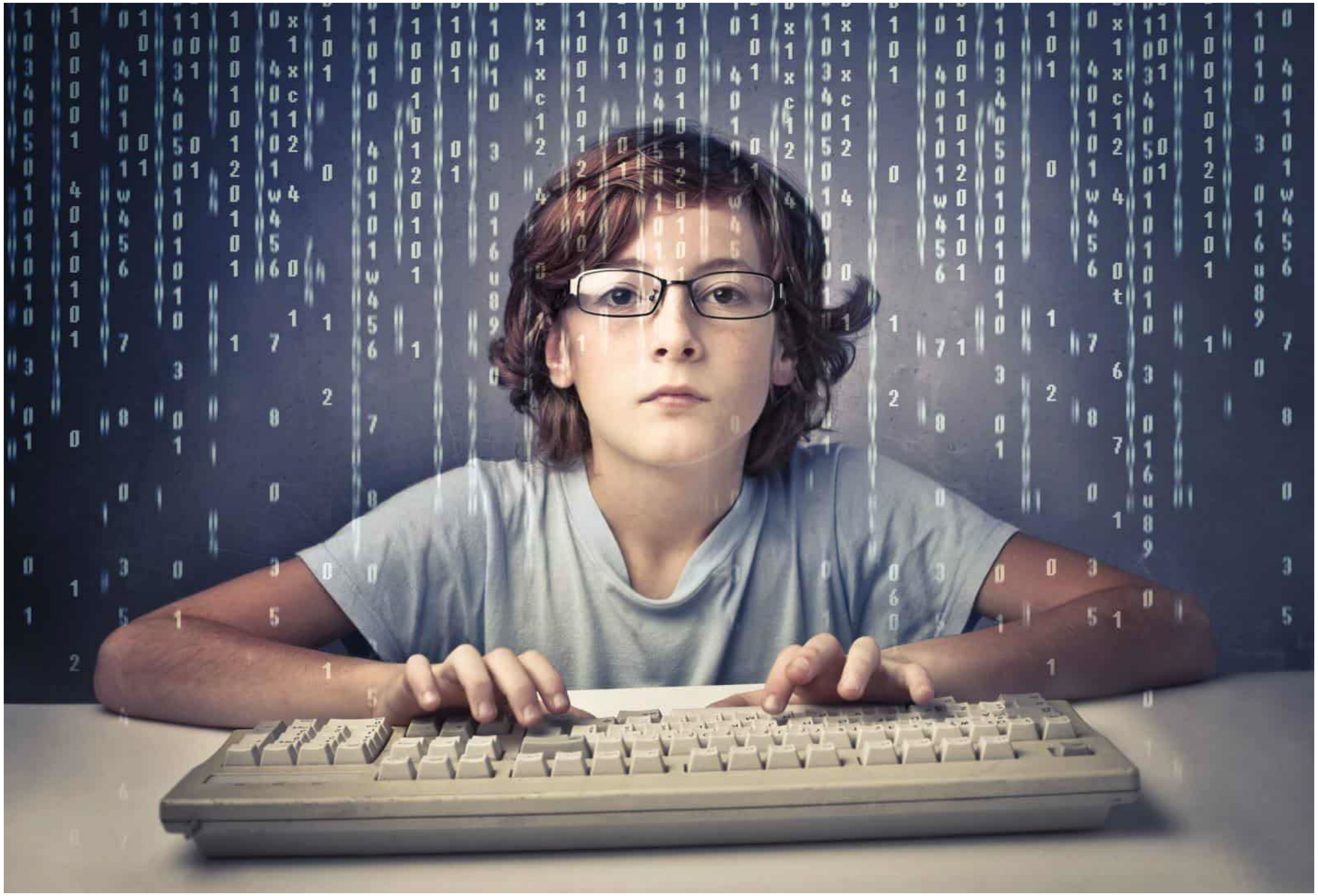
- **thread** 해당 ID의 쓰레드가 종료될때까지 blocking
- **thread** 쓰레드의 main 함수가 반환하는 값이 저장될 포인터 변수의 주소 값

※ Return value
Success : 0
Error : !=0

- pthread_join()은 blocking 함수
- pthread_join()시 종료된 쓰레드는 자원을 자동으로 반납

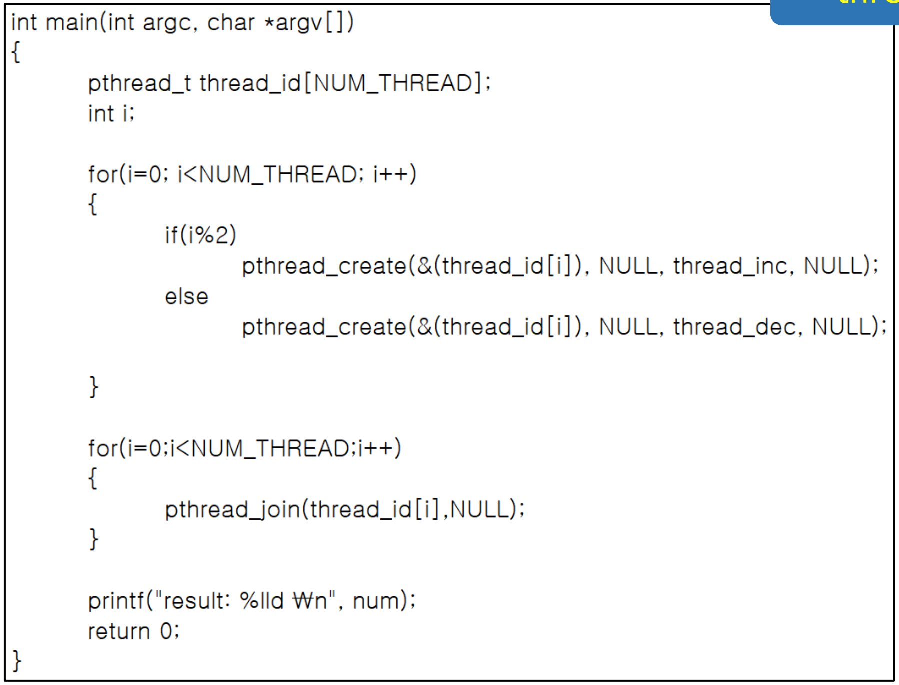
- Let’s modify “thread1.c” using pthread_join()!!

Let’s Investigate “chat_serv.c” & “*chat_clnt.c*”

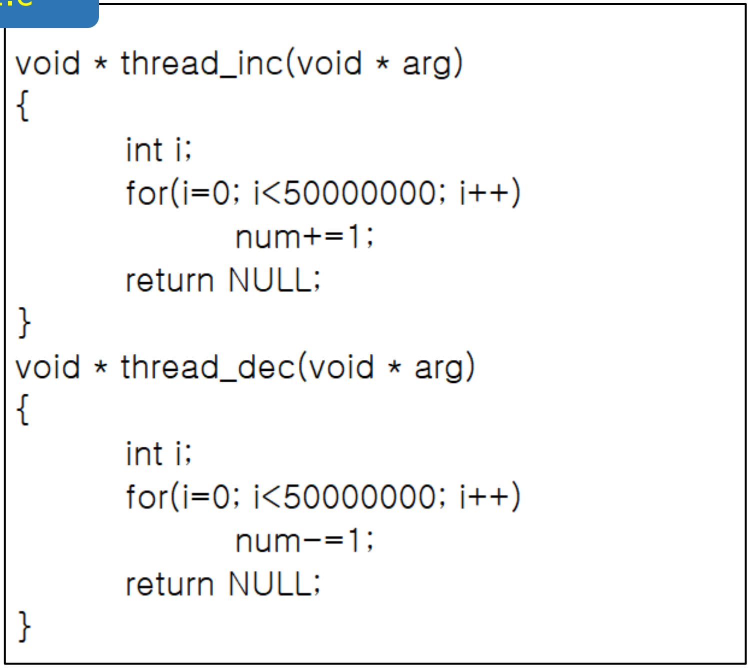


Thread & Critical Section

thread2.c



```
int main(int argc, char *argv[])
{
        pthread_t thread_id[NUM_THREAD];
        int i;

        for(i=0; i<NUM_THREAD; i++)
        {
                if(i%2)
                        pthread_create(&(thread_id[i]), NULL, thread_inc, NULL);
                else
                        pthread_create(&(thread_id[i]), NULL, thread_dec, NULL);

        }

        for(i=0;i<NUM_THREAD;i++)
        {
                pthread_join(thread_id[i],NULL);
        }

        printf("result: %lld ₩n", num);
        return 0;
}
```



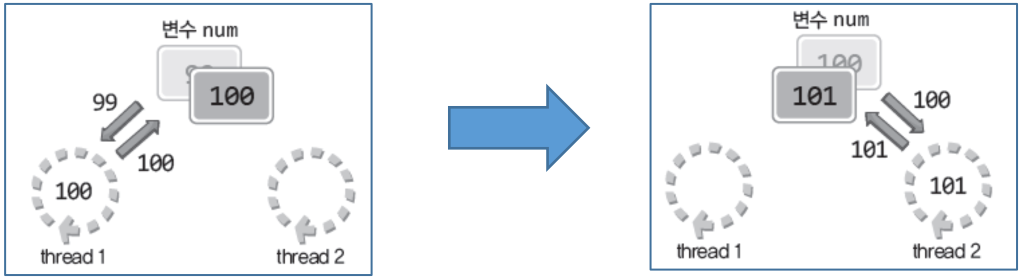
```
void * thread_inc(void * arg)
{
        int i;
        for(i=0; i<50000000; i++)
                num+=1;
        return NULL;
}
void * thread_dec(void * arg)
{
        int i;
        for(i=0; i<50000000; i++)
                num-=1;
        return NULL;
}
```

- Execute “thread2.c”
  - What is the expected output value?
  - What is the output value?

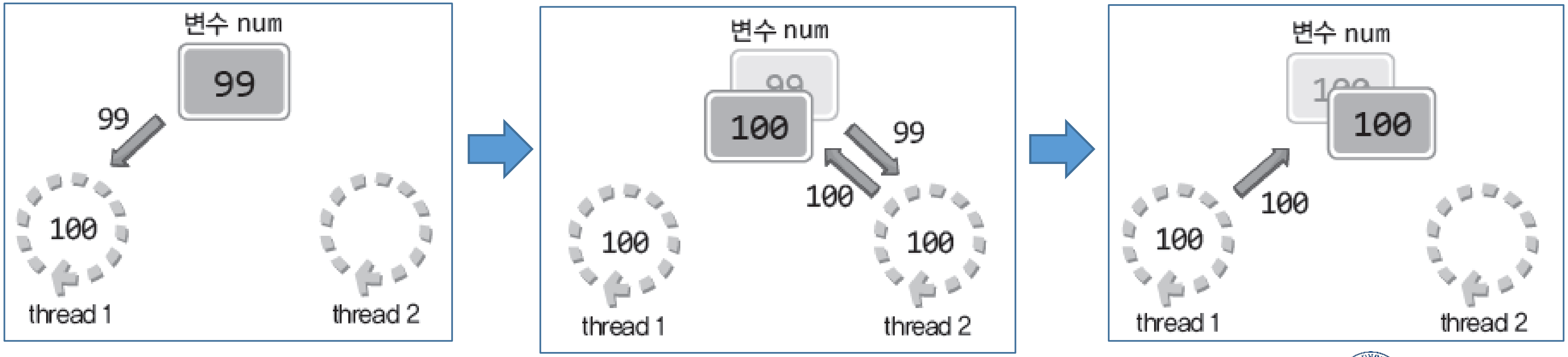
- Execute “thread2.c” again!
  - What is the output value?

Thread & Critical Section

- Our expectation

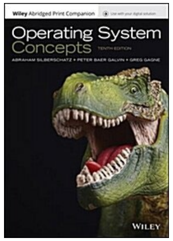


- OS may not work as expected

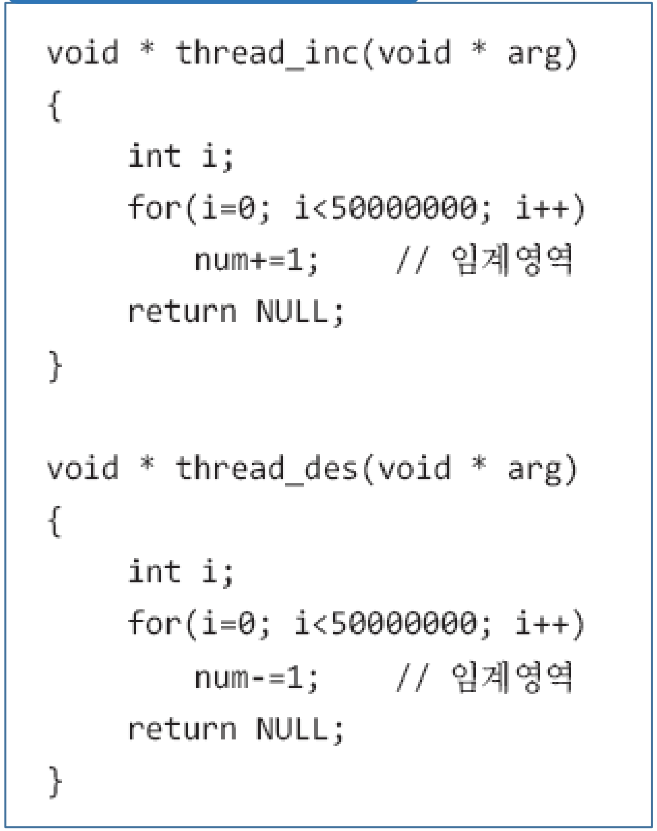


Thread & Critical Section

- Critical Section
  - “A code segment that accesses shared data”



num은 전역변수



```
void * thread_inc(void * arg)
{
    int i;
    for(i=0; i<50000000; i++)
        num+=1;    // 임계영역
    return NULL;
}

void * thread_des(void * arg)
{
    int i;
    for(i=0; i<50000000; i++)
        num-=1;    // 임계영역
    return NULL;
}
```

Synchronization

- Mutual exclusion
  - Prevents two or more threads (processes in OS course may be) from simultaneously execute critical sections

- mutex()
  - we will use it

- Semaphore
  - You can learn in the operating system course in detail

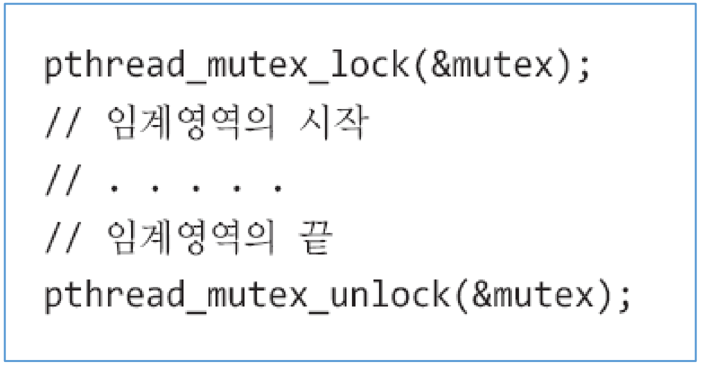
mutex()

```
#include <pthread.h>

int  pthread_mutex_init (pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);

int pthread_mutex_lock (pthread_mutex_t *mutex);

int pthread_mutex_unlock (pthread_mutex_t * mutex);

int pthread_mutex_destroy (pthread_mutex_t *mutex);
```

※ **Return value**
Success : 0
Error : !=0

mutex	mutex object address value
attr	Attribute of the created mutex object (generally NULL)

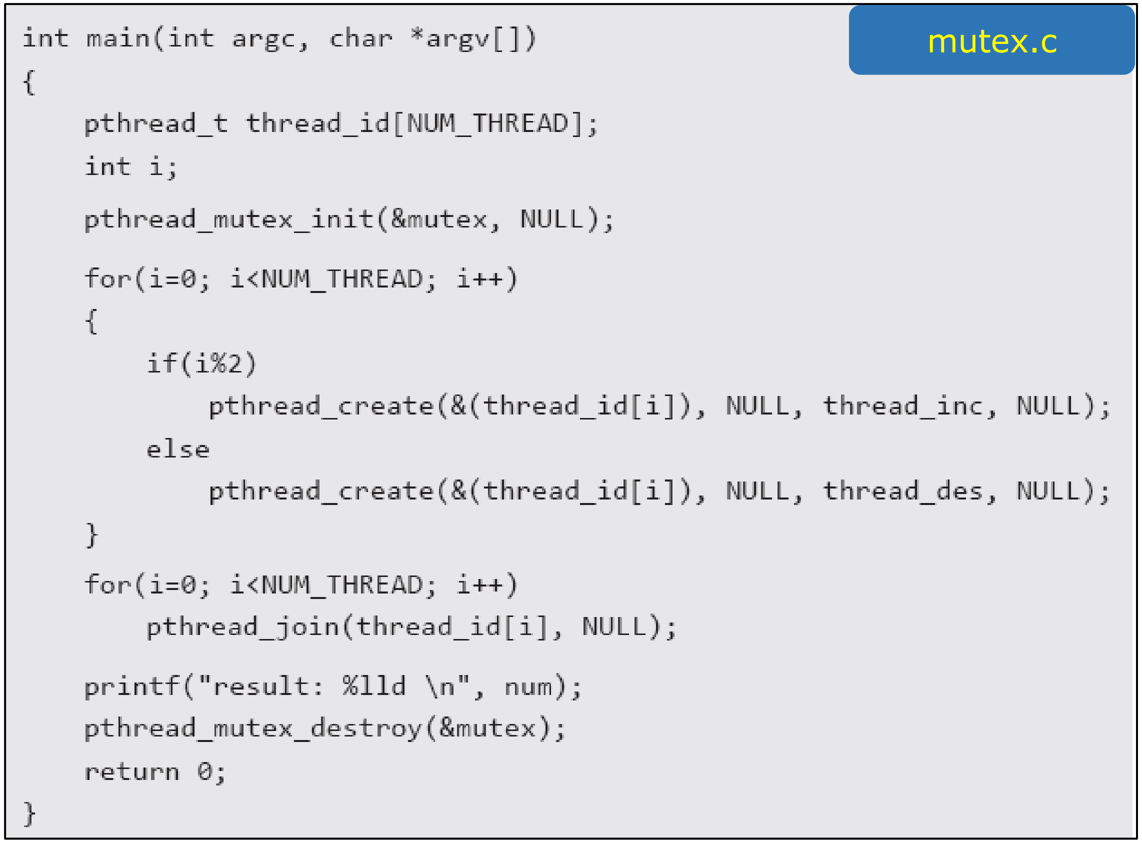
- Usage



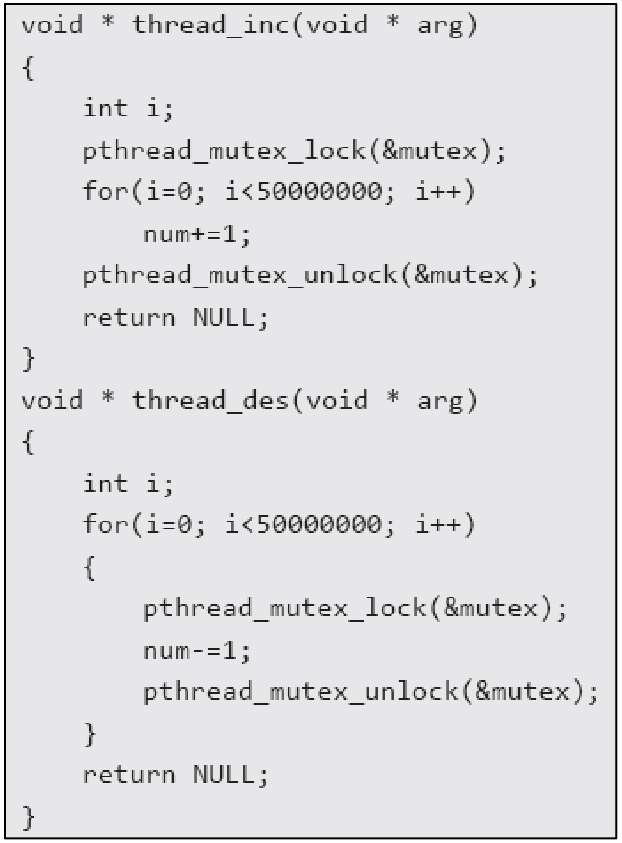
```
pthread_mutex_lock(&mutex);
// 임계영역의 시작
// . . . . .
// 임계영역의 끝
pthread_mutex_unlock(&mutex);
```

mutex() Example

mutex.c

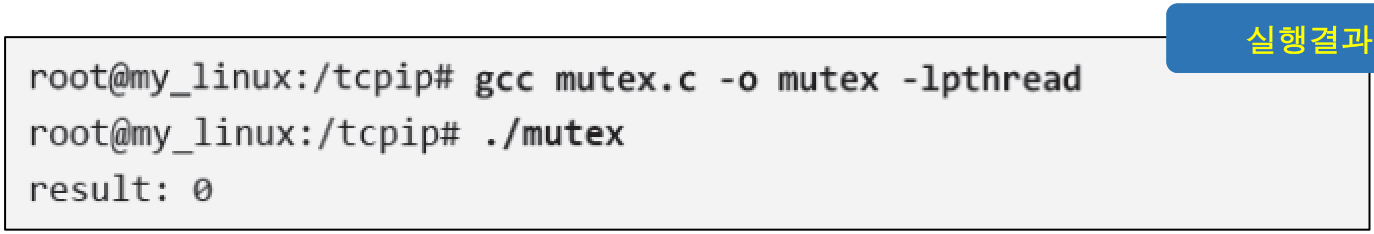


```
int main(int argc, char *argv[])
{
    pthread_t thread_id[NUM_THREAD];
    int i;
    pthread_mutex_init(&mutex, NULL);

    for(i=0; i<NUM_THREAD; i++)
    {
        if(i%2)
            pthread_create(&(thread_id[i]), NULL, thread_inc, NULL);
        else
            pthread_create(&(thread_id[i]), NULL, thread_des, NULL);
    }
    for(i=0; i<NUM_THREAD; i++)
        pthread_join(thread_id[i], NULL);

    printf("result: %lld \n", num);
    pthread_mutex_destroy(&mutex);
    return 0;
}
```



```
void * thread_inc(void * arg)
{
    int i;
    pthread_mutex_lock(&mutex);
    for(i=0; i<50000000; i++)
        num+=1;
    pthread_mutex_unlock(&mutex);
    return NULL;
}
void * thread_des(void * arg)
{
    int i;
    for(i=0; i<50000000; i++)
    {
        pthread_mutex_lock(&mutex);
        num-=1;
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}
```

실행결과



```
root@my_linux:/tcpip# gcc mutex.c -o mutex -lpthread
root@my_linux:/tcpip# ./mutex
result: 0
```

Critical Section (Additional)

- Thread-safe function
  - 둘 이상의 쓰레드가 동시에 호출해도 문제가 발생하지 않는 함수

- Thread-unsafe function
  - 둘 이상의 쓰레드가 동시에 호출하면 문제가 발생할 여지가 있는 함수

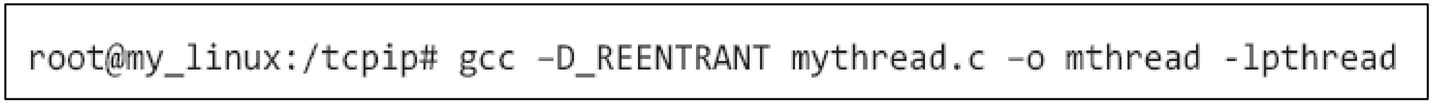
- Example

```
struct hostent * gethostbyname (const char * hostname);  //Thread-unsafe function
```

```
struct hostent * gethostbyname_r (const char * hostname, struct hostent * result,
char *buffer, intbuflen, int *h_errnop);                    //Thread-safe function
```

- We should use “-D_REENTRANT”

※ 사용자 정의 함수에는 적용되지 않음



```
root@my_linux:/tcpip# gcc -D_REENTRANT mythread.c -o mthread -lpthread
```

pthread_detach()

```
#include <pthread.h>

int  pthread_detach (pthread_t thread);
```

thread Detach할 쓰레드 ID 정보

※ Return value
Success : 0
Error : !=0

- 참고
  - pthread_join(): blocking function
  - pthread_detach(): non-blocking function

  → 상황에 맞게 택일하여야 함

Let’s Investigate “chat_serv.*c*” & “*chat_clnt.c*” Again!