System Programming & OS 실습
6. Synchronization

이성현, 최민국

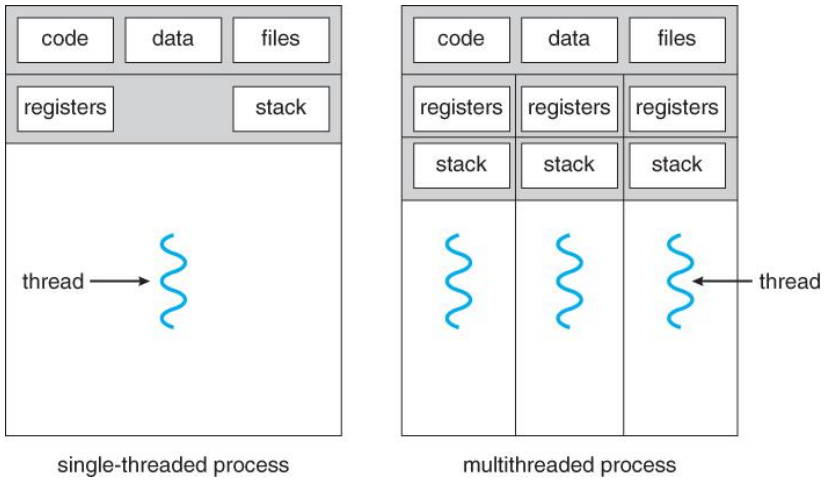
Dankook University

{leesh812, mgchoj}@dankook.ac.kr

Index

Thread

- Thread model
  - Share resources among threads
    - code, data, heap and files
  - Exclusively resources used by a thread
    - CPU abstraction and stack



(Source: A. Silberschatz, "Operating system Concept")

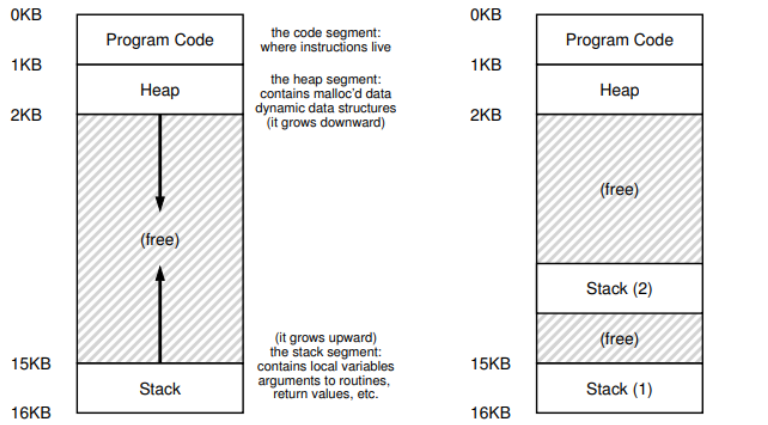


Figure 26.1: **Single-Threaded And Multi-Threaded Address Spaces**

Thread

- Benefit of Thread
  - Fast creation
  - Parallelism
  - Can overlap processing with waiting
  - Data sharing
- Thread management
  - Several stacks in an address space
  - Scheduling entity

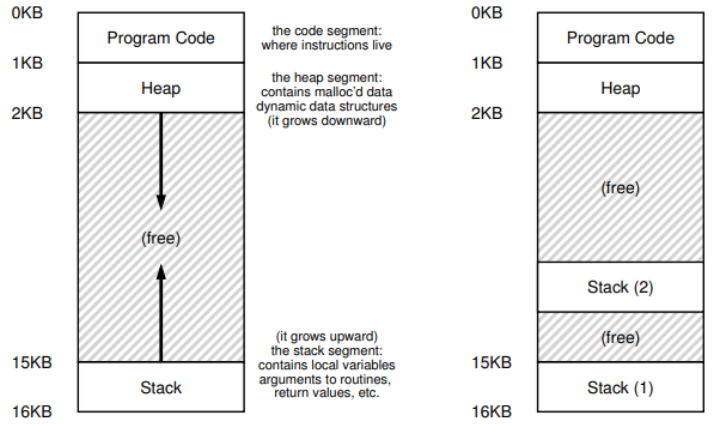
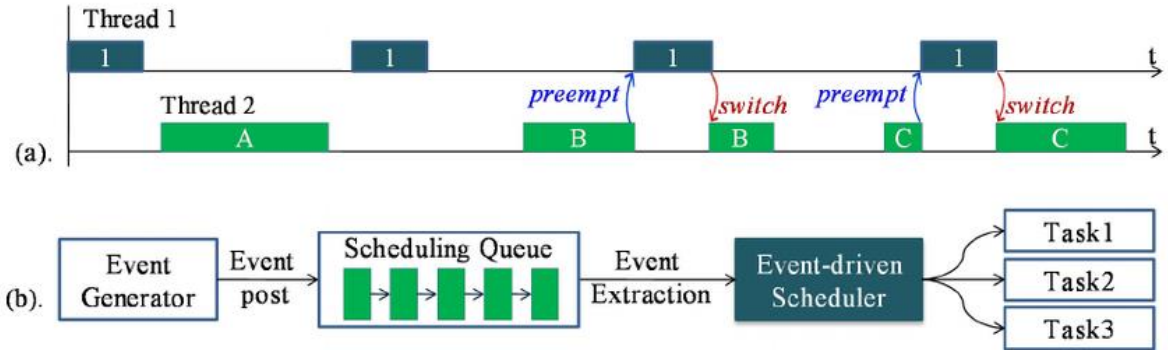


Figure 26.1: **Single-Threaded And Multi-Threaded Address Spaces**



Thread Scheduling

Thread

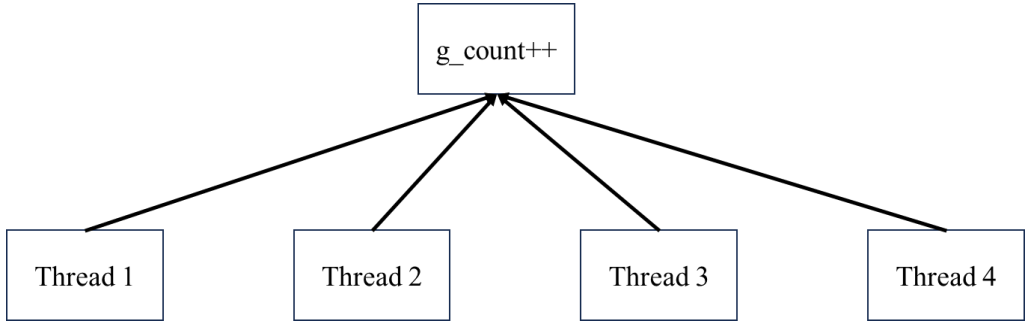
- #include <pthread.h>
- int pthread_create(pthread_t *restrict thread, const pthread_attr_t *restrict attr,
  void *(*start_routine)(void *), void *restrict arg);
  - similar to fork(), thread exits when the passed function reach the end.
  - arg1) thread structure to interact with this thread,
  - arg2) attribute of the thread such as priority and stack size, in most case it is NULL (use default)
  - arg3) function pointer for start routine
  - arg4) arguments

- int pthread_join(pthread_t thread, void **retval);
  - similar to wait(), for synchronization
  - arg1) thread structure, which is initialized by the thread creation routine
  - arg2) a pointer to the return value (NULL means “don’t care”)

Practice 1: Prepare

- Practice 1 command for prepare

  > mkdir thread_practice (디렉토리 생성)
  > cd thread_practice (디렉토리 이동)
  > vim thread.c (코드 작성)



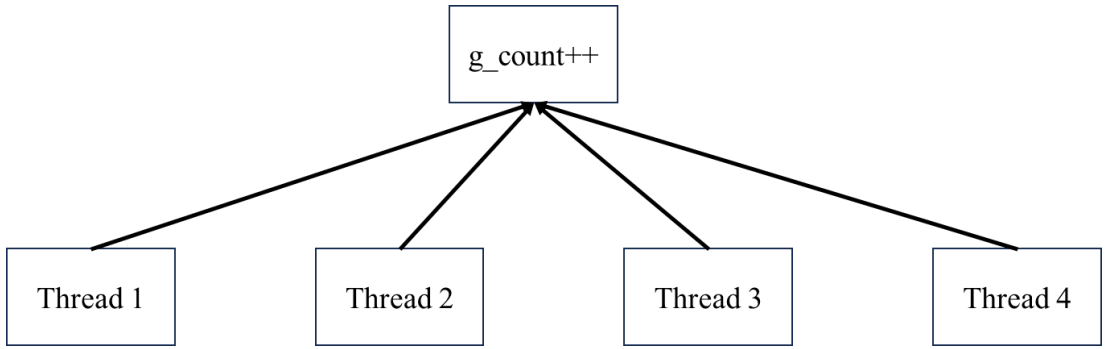
Practice 1: Code


```c
// thread.c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <assert.h>
#include <pthread.h>

int g_count = 0; // counter (critical section)
int g_nthd = 0; // num of threads
int g_worker_loop_cnt = 0;

static void *work(void* cnt); // thread routine

int main(int argc, char *argv[]){
    pthread_t *thd_arr; // thread array
    int thd_cnt; // thread count

    if (argc < 3){
        fprintf(stderr, "%s parameter : nthread, worker_loop_cnt\n", argv[0]);
        exit(-1);
    }

    // alloc memory for thread
    thd_arr = malloc(sizeof(pthread_t) * g_nthd);

    for(thd_cnt=0; thd_cnt < g_nthd; thd_cnt++){
        // create thread
        assert(pthread_create(&thd_arr[thd_cnt], NULL,
                work, (void*) thd_cnt) == 0);
    }

    for(thd_cnt=0; thd_cnt < g_nthd; thd_cnt++){
        // join thread
        assert(pthread_join(thd_arr[thd_cnt], NULL) == 0);
    }
    printf("Complete\n");

}

static void *work(void* cnt){
    int thd_cnt = (int)cnt;
    int i;

    for(i = 0; i < g_worker_loop_cnt; i++)
        g_count++;

    printf("Thread number %d: %d \n", thd_cnt, g_count);
    return NULL;
}
```

Pratice 1: Run

- Practice 1 command2
  - > gcc thread.c -lpthread -o thread.out (컴파일)
  - > ./thread.out 4 1000 (실행1)
  - > ./thread.out 4 10000 (실행2)

```
embedded@embedded:~/thread_test$ ./thread.out 4 10000
Thread number 0: 10000
Thread number 2: 20000
Thread number 1: 30000
Thread number 3: 40000
Complete
embedded@embedded:~/thread_test$ ./thread.out 4 100000
Thread number 0: 99991
Thread number 2: 218531
Thread number 3: 279583
Thread number 1: 379583
Complete
```



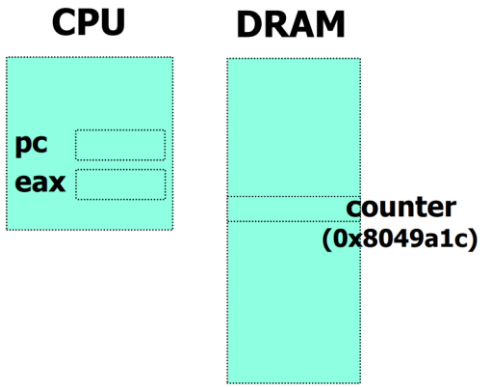
Practice 1: Result

- High level viewpoint

```
17        for (i = 0; i < 1e7; i++) {
18            counter = counter + 1;
19        }
```

- CPU level viewpoint

```
mov 0x8049a1c, %eax
add $0x1, %eax
mov %eax, 0x8049a1c
```



- Scheduling viewpoint

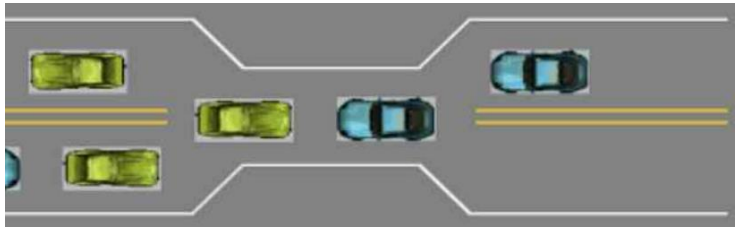
OS	Thread 1	Thread 2	PC	eax	counter
			(after instruction)		
	before critical section		100	0	50
	`mov 8049a1c,%eax`		105	**50**	50
	`add $0x1,%eax`		108	**51**	50
interrupt					
save T1					
restore T2			100	0	50
		`mov 8049a1c,%eax`	105	**50**	50
		`add $0x1,%eax`	108	**51**	50
		`mov %eax,8049a1c`	113	51	**51**
interrupt					
save T2					
restore T1			108	51	51
	`mov %eax,8049a1c`		113	51	**51**

Figure 26.7: **The Problem: Up Close and Personal**

Thread Problem

- Reason
  - Numerous threads access shared data(critical section) at the same time
    → race condition
  - Uncontrolled scheduling
    → Results are different at each execution depending on scheduling order

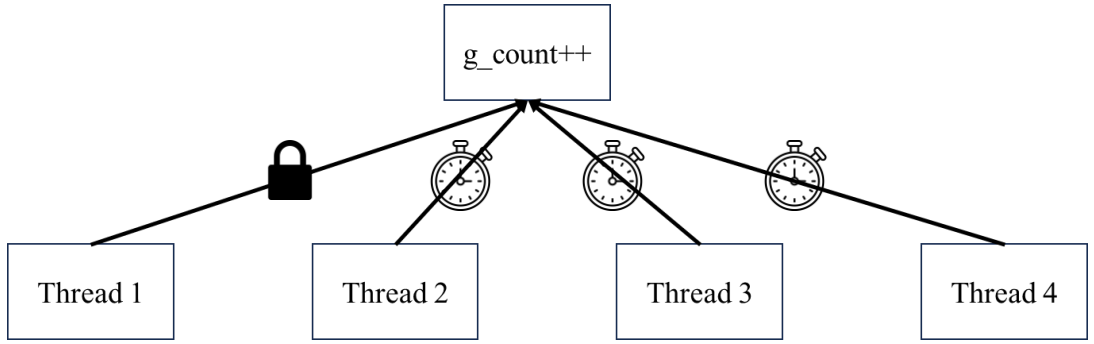
- Solution
  - Controlled scheduling: Do all or nothing (indivisible) → atomicity
  - The code that can result in the race condition → critical section
  - Allow only one thread in the critical section → mutual exclusion

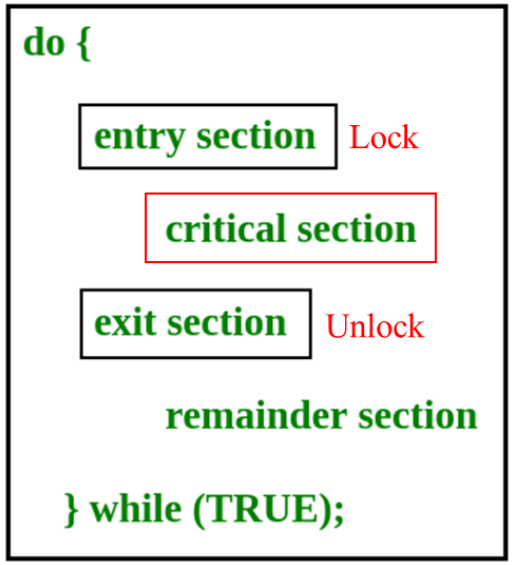


Thread Problem

- Mutual exclusion API (mutex_***)
  - #include <pthread.h>

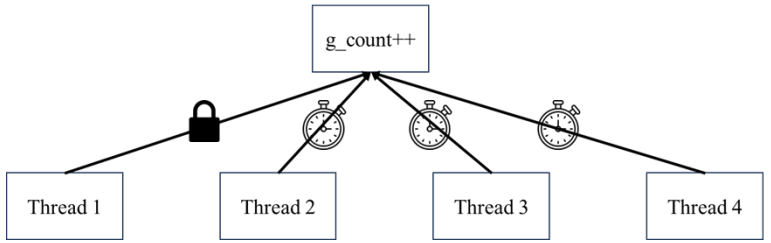
  - pthread_mutex_t lock;
  - int pthread_mutex_init(pthread_mutex_t *restrict mutex,
  - const pthread_mutexattr_t *restrict attr);
  - int pthread_mutex_lock(pthread_mutex_t *mutex);
  - int pthread_mutex_unlock(pthread_mutex_t *mutex);





Practice 2: Prepare

- Practice 2 command for prepare
  > cp thread.c thread_lock.c (파일 복사)
  > vim thread_lock.c (코드 작성)




```
do {
    entry section
        critical section
    exit section
        remainder section
} while (TRUE);
```

Practice 2: Code

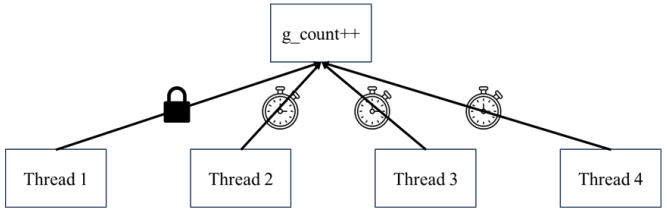

```c
// thread_lock.c
// ...
pthread_mutex_t lock;

int main(int argc, char *argv[]){
    // ...
    thd_arr = malloc(sizeof(pthread_t) * g_nthd);

    pthread_mutex_init(&lock, NULL);

    for(thd_cnt=0; thd_cnt < g_nthd; thd_cnt++){
        // create thread
        assert(pthread_create(&thd_arr[thd_cnt], NULL,
               work, (void*) thd_cnt) == 0);
    }
    // ...
}
```

```c
static void *work(void* cnt){
    int thd_cnt = (int)cnt;
    int i;

    for(i = 0; i < g_worker_loop_cnt; i++){
        pthread_mutex_lock(&lock);
        g_count++;
        pthread_mutex_unlock(&lock);
    }
    printf("Thread number %d: %d \n", thd_cnt, g_count);
    return NULL;
}
```

Practice 2: Result

- Practice 1 command2
  - > gcc -o thread_lock.out thread_lock.c -lpthread (컴파일)
  - > ./thread.out 4 10000 (실행1)
  - > ./thread_lock.out 4 10000 (실행2)

```
embedded@embedded:~/thread_test$ ./thread.out 4 100000
Thread number 0: 99991
Thread number 2: 218531
Thread number 3: 279583
Thread number 1: 379583
Complete
embedded@embedded:~/thread_test$ ./thread_lock.out 4 100000
Thread number 1: 235328
Thread number 2: 379740
Thread number 3: 380224
Thread number 0: 400000
Complete
```




```
do {
    entry section
        critical section
    exit section
        remainder section
} while (TRUE);
```