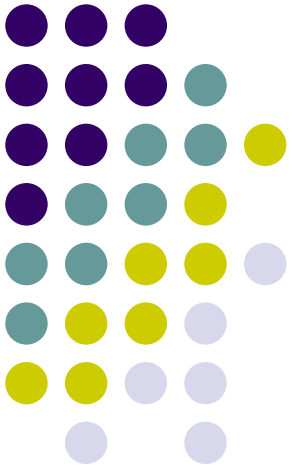
뮤텍,/세모포어 기반의 멀티쓰레드 동기화 기법

편집 김혜영

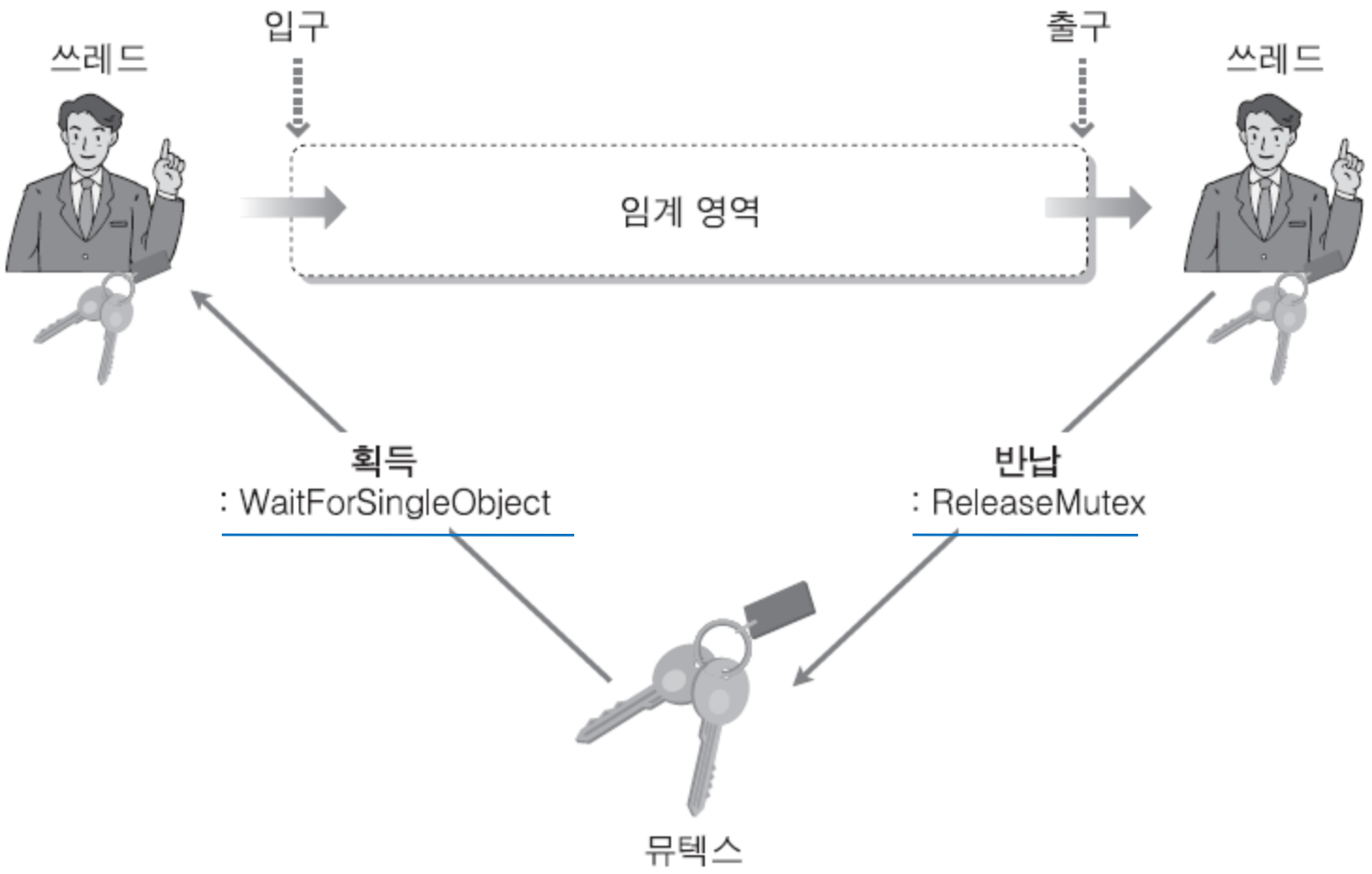


뮤텍스(mutex)의 생성

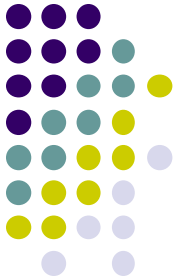
```
HANDLE  CreateMutex (
    LPSECURITY_ATTRIBUTES lpMutexAttributes,    // 보안 설정
    BOOL bInitialOwner,   // 소유자 지정 (TRUE, FALSE)
    LPCTSTR lpName        // 뮤텍스 이름 지정
);
```

If the function fails, the return value is NULL.

뮤텍스(mutex)기반의 동기화



스레드 조작 – 스레드 종료 대기



- WaitForSingleObject() 함수
  - 특정 스레드가 종료할 때까지 대기

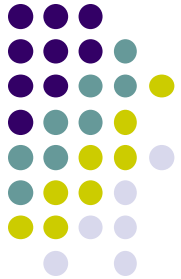
```
DWORD WaitForSingleObject (
    HANDLE hHandle,
    DWORD dwMilliseconds
) ;
```

성공: **WAIT_OBJECT_0** 또는 **WAIT_TIMEOUT**,
실패: **WAIT_FAILED**

- WaitForSingleObject() 함수 사용 예

```
HANDLE hThread = CreateThread(...);
WaitForSingleObject(hThread, INFINITE);
```

스레드 조작 – 스레드 종료 대기

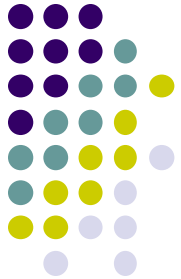


- WaitForMultipleObjects() 함수
  - 두 개 이상의 스레드가 종료할 때까지 대기

```
DWORD WaitForMultipleObjects (
   DWORD nCount,
   const HANDLE* lpHandles,
   BOOL bWaitAll,
   DWORD dwMilliseconds
) ;
```

성공: WAIT_OBJECT_0 ~ WAIT_OBJECT_0 + nCount-1
또는 WAIT_TIMEOUT,
실패: WAIT_FAILED

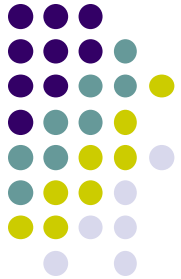
스레드 조작 – 스레드 종료 대기



- WaitForMultipleObjects() 함수 사용 예 ②

```
// 두 스레드 중 하나의 종료를 기다릴 경우
HANDLE hThread[2];
HANDLE hThread[0] = CreateThread(...);
HANDLE hThread[1] = CreateThread(...);
DWORD retval = WaitForMultipleObjects(2, hThread, FALSE, INFINITE);
switch(retval){
case WAIT_OBJECT_0:          // hThread[0] 종료
    break;
case WAIT_OBJECT_0+1:  // hThread[1] 종료
    break;
case WAIT_FAILED:            // 오류 발생
    break;
}
```

스레드 조작 – 실행 중지와 재실행



- 실행 중지 함수 ①

```
DWORD SuspendThread (
    HANDLE hThread // 스레드 핸들
) ;
```

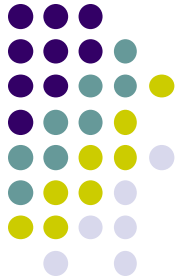
성공: 중지 횟수, 실패: -1

- 재실행 함수

```
DWORD ResumeThread (
    HANDLE hThread // 스레드 핸들
) ;
```

성공: 중지 횟수, 실패: -1

스레드 조작 – 실행 중지와 재실행



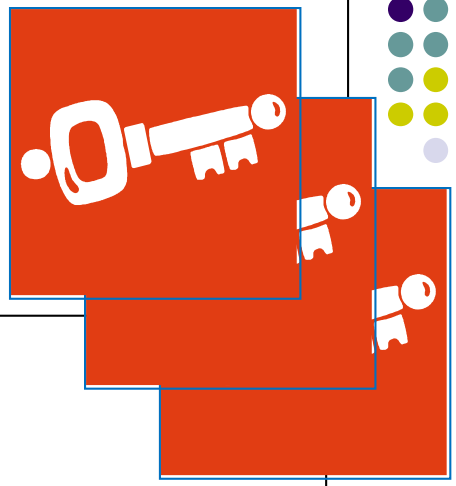
- 실행 중지 함수 ②
  - 스레드가 실행을 멈추고 일정 시간동안 대기

```
void Sleep (
    DWORD dwMilliseconds // 밀리초(ms)
) ;
```

뮤텍스 기반 동기화 예제

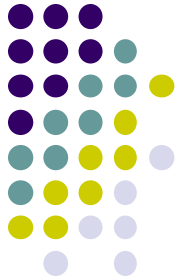
CriticalSectionSyncMutex.cpp

세마포어(Semaphore)의 생성



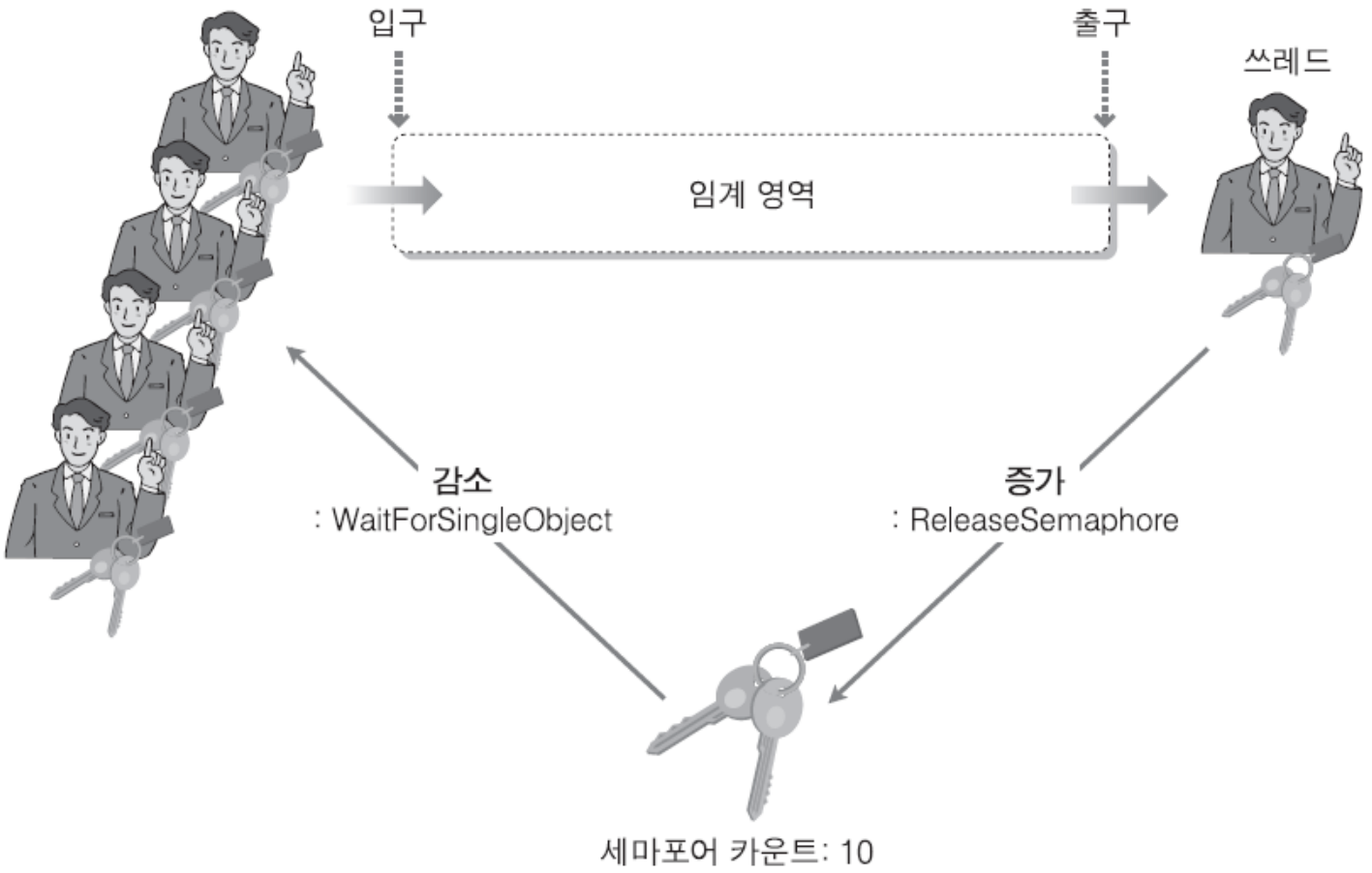
```
HANDLE  CreateSemaphore (
    LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
    LONG lInitialCount,
    LONG lMaximumCount,
    LPCTSTR lpName
);
```

If the function fails, the return value is NULL.



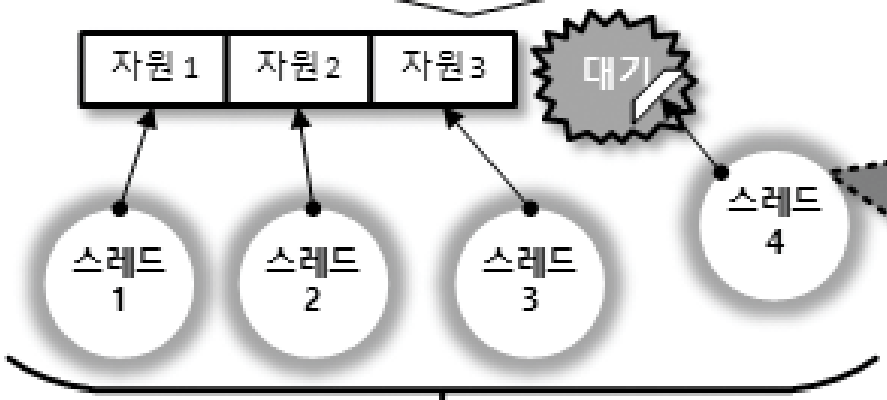
```
BOOL ReleaseSemaphore(
        HANDLE hSemaphore,
        LONG LRelaseCount,
        LPLONG lpPreviousCount
)
return 0: fail
```

세마포어(Semaphore) 기반의 동기화



① 메인 스레드 등에서 세마포를 생성한다.
초기 자원수는 3, 최대치 역시 3으로 설정한다.

```
HANDLE hSema = CreateSemaphore(NULL, 3, 3, NULL);
```



③
- 이미 자원계수가 0이므로 사용 가능한 자원이 없음을 의미한다.
- 스레드 4는 WaitForXXX 내에서 대기 상태로 진입한다.
- 이 상태는 스레드 1~3중 하나가 ReleaseSemaphore를 호출하여 자원계수를 증가시키기 전까지 지속된다.

스레드마다 우선 대기 함수를 호출해서 자원의 사용이 가능해질 때까지 기다린다.

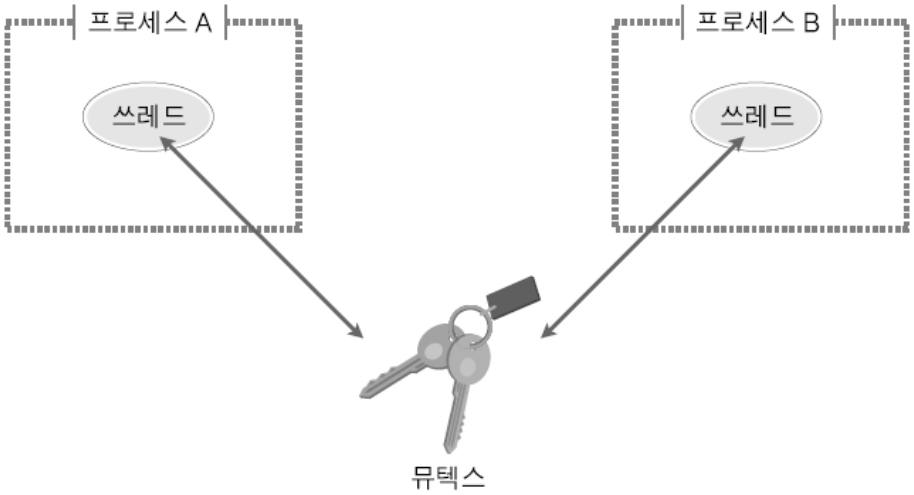
```
WaitForSingleObject(hSema, ...);
```

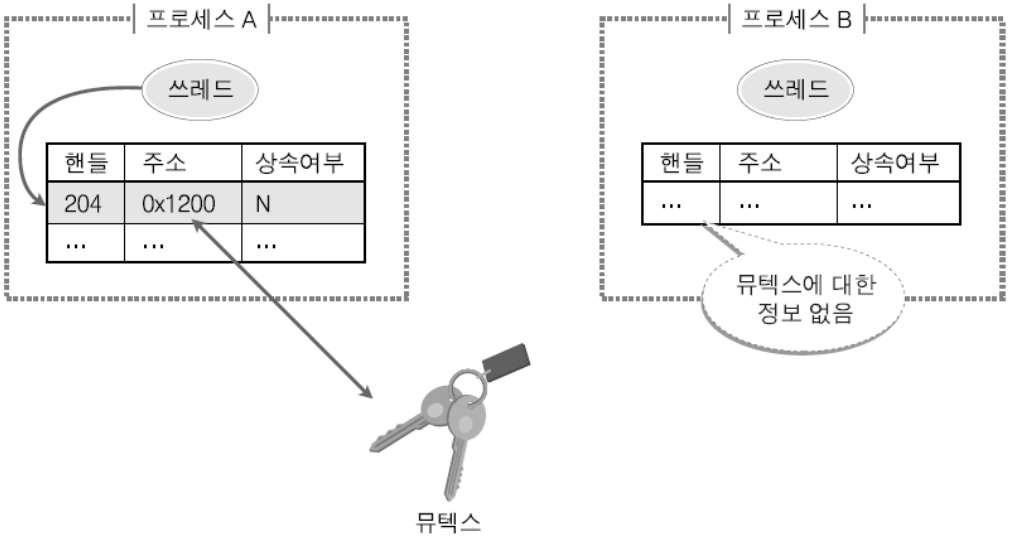
② 스레드 1~3까지는 남은 자원계수가 0보다 크기 때문에 대기 함수는 바로 리턴되고 남은 자원계수는 1씩 감소된다.

세마포어 기반 동기화 예제

MyongDongKyoja.cpp

이름있는 뮤텍스 기반의 프로세스 동기화

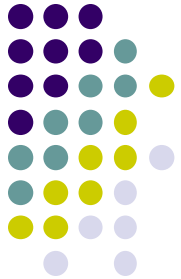




이름있는 뮤텍스 동기화 예제

NamedMutex.cpp

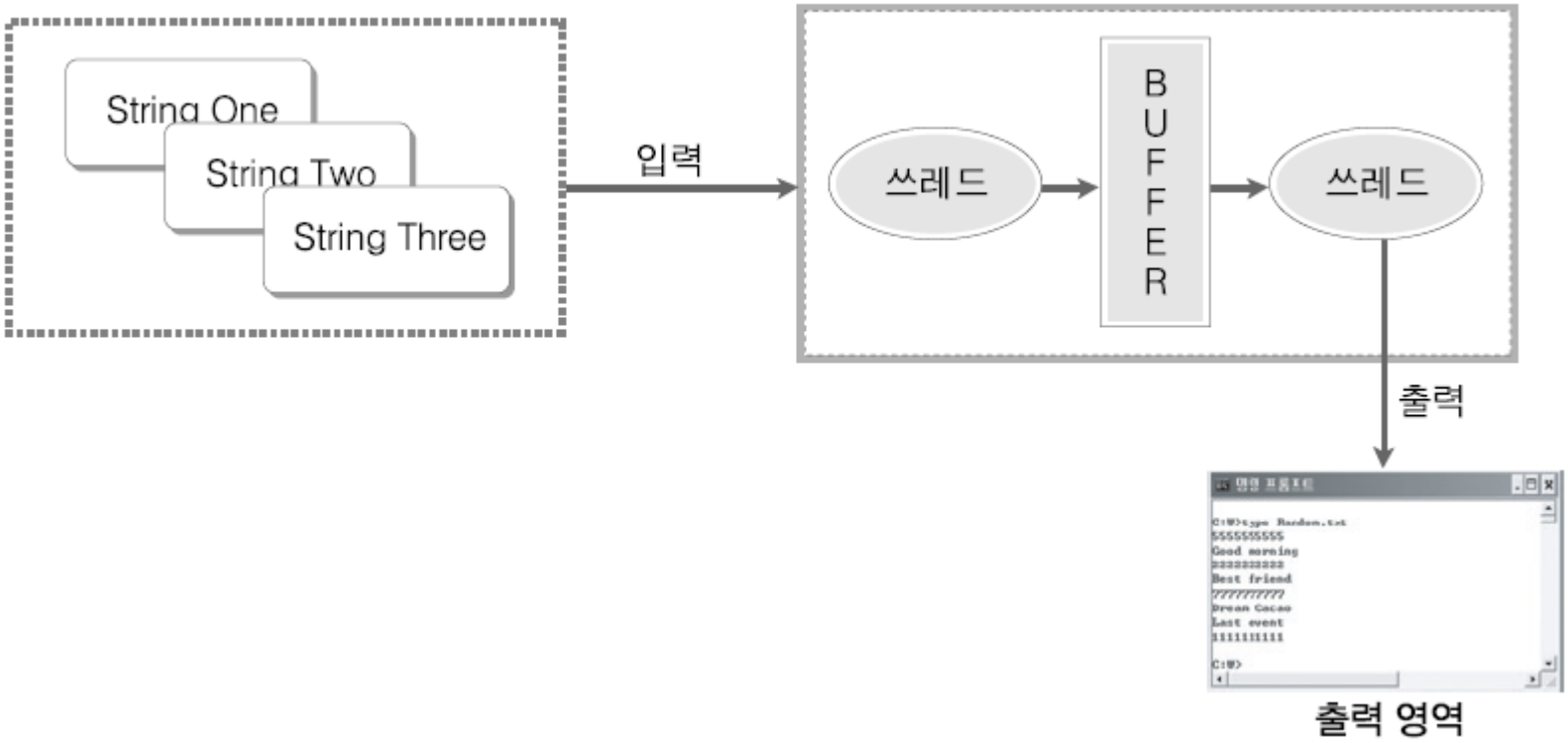
뮤텍스의 소유와 WAIT_ABANDONED



- 소유의 관점에서 세마포어와 뮤텍스의 차이점은?

MUTEX_WAIT_ABANDONED.cpp

생산자/소비자 모델



실습 2

12개의 숫자를 입력받은 값들을 출력하고, 그 합계를 구하는 프로그램을 3개의 스레드를 사용하여 작성하시오.

즉, 뮤텍스를 사용하여 입력받는 부분과 합을 구하는 부분에 대한 동기화를 수행하여, 입력 후 합을 구하도록 프로그램하고, 세 개의 스레드 작업에 의해 합계를 구하는 부분은 크리티컬 섹션을 사용하여 스레드의 동기화를 수행하시오.