

Lecture 5-1

Synchronization Constructs

Introduction to OpenMP

Synchronization (or Coordination)

- Thread 들 사이의 실행 순서를 맞추는 것
- Thread 들이 서로 정보를 교환하는 행위

```
#pragma omp parallel for
for (int i = 0; i < n - 1; i++)
{
    double x_i = a + h * i;
    double x_j = a + h * (i + 1);
    double d = (f(x_i) + f(x_j)) / 2.0;

    sum += d*h;
}
```

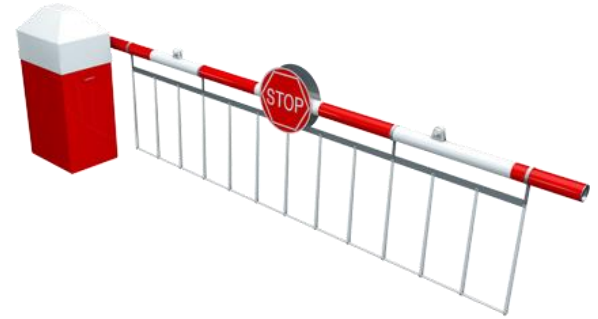
← Thread 1
← Thread 2
← Thread 3



Synchronization Types

- **Barrier**

- 모든 thread가 모일 때까지 기다렸다가 진행



- **Mutual Exclusion (상호배제)**

- Critical section
- 한번에 한 thread만 해당 영역에 진입



OpenMP Clauses for Synchronization

- 명시적인 Synchronization 지점/영역 지정 가능
- Barrier
- Critical
- Atomic
- Master

Barrier

#pragma omp barrier

- 모든 thread들이 모일 때까지,
다른 thread들의 진행을 막게 하는 지시어

```
#pragma omp parallel
{
    // work before barrier

    #pragma omp barrier

    // work after barrier
}
```

Barrier (example)

```
int tID = 0;
#pragma omp parallel private (tID)
{
    tID = omp_get_thread_num();

    if (tID % 2 == 0)
        Sleep(10);
    printf("[%d] before\n", tID);

    #pragma omp barrier

    printf("[%d] after\n", tID);
}
```

```
C:\WINDOWS\system32\cmd.exe
[1] before
[3] before
[5] before
[7] before
[0] before
[2] before
[4] before
[6] before
[6] after
[3] after
[4] after
[1] after
[5] after
[2] after
[0] after
[7] after
계속하려면 아무 키나 누르십시오 . . .
```

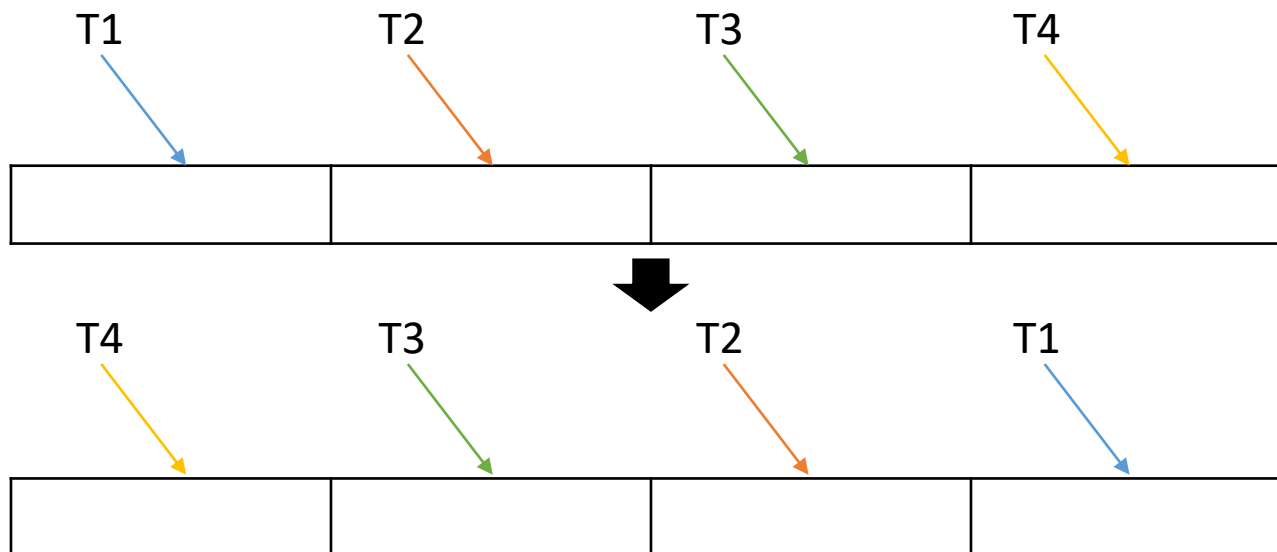
- Quick Lab.

Barrier

• 언제 쓸까?

- Barrier 전 후로, 접근 하는 data 영역이 변할 때
- 서로의 수행 결과를 참조해야 할 때
- 특정 thread만 일을 하면 되고, 나머지는 기다려야 할 때
 - E.g., master thread

• ...



Barrier (example)

```
#pragma omp parallel num_threads(4)
{
    int tID = omp_get_thread_num()
    a[tID] = tID * 10;

    #pragma omp barrier

    #pragma omp for
    for (int i = 0; i < 16 ; i++)
        b[i] = 2 * a[(i+1)%4];
}
```

C:\Windows\system32

a :	10	tID :	1
a :	0	tID :	0
a :	20	tID :	2
a :	30	tID :	3
b[0]	b :	20	
b[1]	b :	40	
b[2]	b :	60	
b[3]	b :	0	
b[4]	b :	20	
b[5]	b :	40	
b[6]	b :	60	
b[7]	b :	0	
b[8]	b :	20	
b[9]	b :	40	
b[10]	b :	60	
b[11]	b :	0	
b[12]	b :	20	
b[13]	b :	40	
b[14]	b :	60	
b[15]	b :	0	

Barrier (example)

```
#pragma omp parallel
{
    int tID = omp_get_thread_num();

    if (tID == 0) // master thread
    {
        // do something
    }
    #pragma omp barrier

    // do common work
}
```

Implicit Barrier in OpenMP

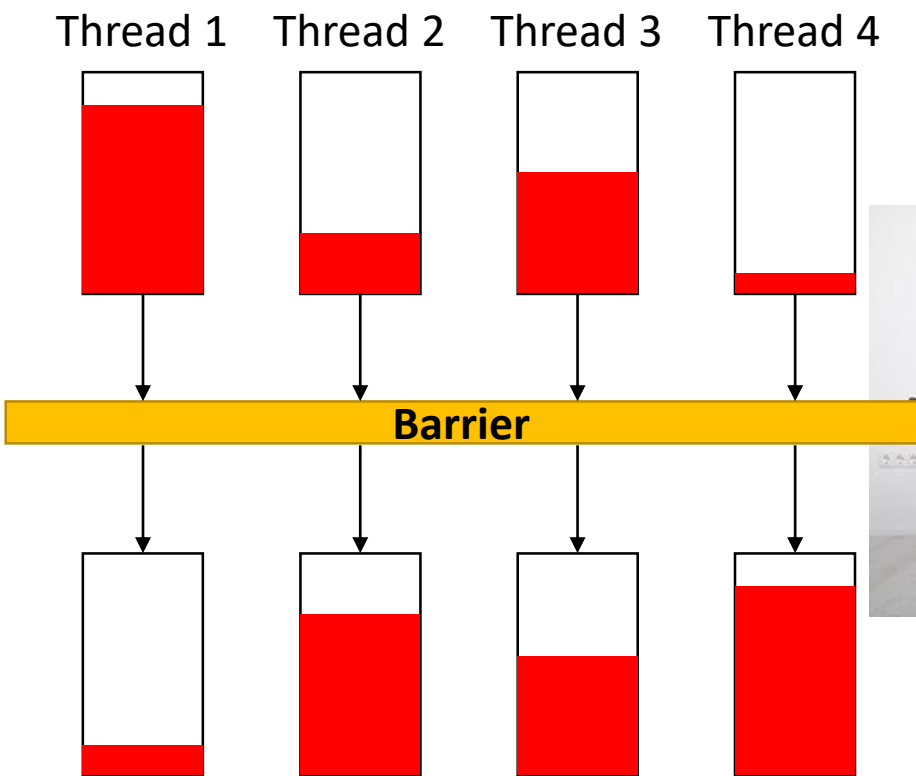
- Parallel Construct, Work-Sharing Constructs
마지막에 implicit barrier가 존재

```
#pragma omp parallel
{
    #pragma omp for
    for (int i = 0; i < SIZE; i++)
        a[i] = i;
    ← Implicit barrier

    #pragma omp for
    for (int i = 0; i < SIZE; i++)
        b[i] = 2 * a[i];
    ← Implicit barrier
}
```

```
#pragma omp sections [clause list]
{
    #pragma omp section
    {
        // do task A
    }
    #pragma omp section
    {
        // do task B
    }
    ← Implicit barrier
}
```

Workload Balancing Issue



- **Barrier가 병렬처리 성능의 bottleneck이 될 수 있음**
 - 불필요한 barrier를 최소화 해야 함
 - Barrier 위치 결정 시, Thread간 일의 양 분배에 신경 써야 함

Critical

```
#pragma omp critical [(name)]  
{ /* structured block*/ }
```

- **Critical section을 지정**
 - 한번에 한 thread만 진입
 - 다른 thread들은 대기
 - 대기 중인 thread들 중 다음에 누가 들어갈지는 비결정적
- **언제 사용 할까?**
 - 공유 데이터를 동시에 수정하는 것을 방지할 때

Critical (e.g., Trapezoidal Rule)

```
sum = 0;
#pragma omp parallel for num_threads(4)
for (int i = 0; i < n - 1; i++)
{
    double x_i = a + h * i;
    double x_j = a + h * (i + 1);
    double d = (f(x_i) + f(x_j)) / 2.0;

    #pragma omp critical
    {
        sum += d*h;
    }
}
```

```
#pragma omp parallel num_threads(4)
{
    int tid = omp_get_thread_num();

    #pragma omp for
    for (int i = 0; i < n-1; i++)
    {
        double x_i = a + h * i;
        double x_j = a + h * (i + 1);
        double d = (f(x_i) + f(x_j)) / 2.0;
        local[tid] += d*h;
    }
}

LOOP_I(4)
    sum += local[i];
```

- Quick Lab.
 - 두 버전을 구현 후, 결과 및 시간을 비교

Serialization Issue

- Critical section에서는 thread들의 연산이 serialize (직렬화) 됨
 - Bottleneck!
 - Parallel algorithm 성능에 치명적
- Scalable algorithm을 만들기 위해서는 Critical section을 최소화 해야 함



Atomic

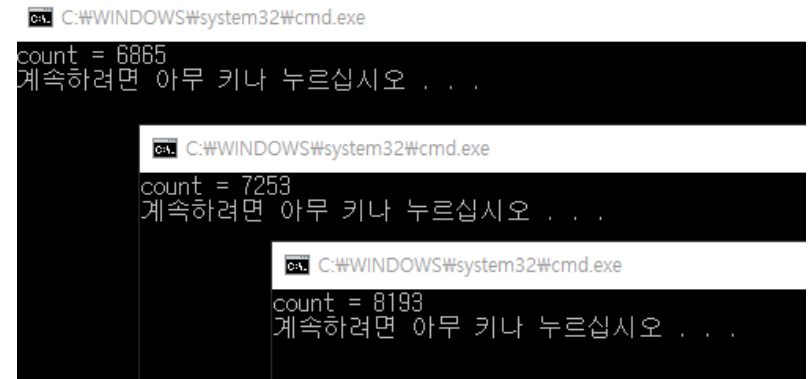
#pragma omp atomic

single assignment statement

- 공유 데이터를 한번에 한 thread만 수정하게 함
 - Critical section과 유사
- 바로 아래 statement에만 적용됨
 - 지원하는 연산자: +=, *=, -=, /=, &=, ^=, |=, <<=, >>=
- H/W가 atomic operation을 지원하면 효율적

Atomic (example)

```
int ic = 0;
#pragma omp parallel
{
    for (int i = 0; i < 1024; i++)
    {
        ic += 1;
    }
}
printf("count = %d\n", ic);
```



C:\WINDOWS\system32\cmd.exe
count = 6865
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
count = 7253
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
count = 8193
계속하려면 아무 키나 누르십시오 . . .

- Quick Lab.

Atomic (example)

```
int ic = 0;
#pragma omp parallel
{
    for (int i = 0; i < 1024; i++)
    {
        #pragma omp atomic
        ic += 1;
    }
}
printf("count = %d\n", ic);
```

- Quick Lab.

Atomic (e.g., Trapezoidal Rule)

```
sum = 0;
#pragma omp parallel for num_threads(4)
for (int i = 0; i < n - 1; i++)
{
    double x_i = a + h * i;
    double x_j = a + h * (i + 1);
    double d = (f(x_i) + f(x_j)) / 2.0;

    #pragma omp atomic
    sum += d*h;
}
```

- **Quick Lab.**

- Critical section 버전과 결과 및 속도 해보기

Atomic – 주의점

- 메모리 update만 atomic 하게 수행 됨

```
int ic = 0;
#pragma omp parallel
{
    for (int i = 0; i < 1024; i++)
    {
        #pragma omp atomic
        ic += bigFunc();
    }
}
printf("count = %d\n", ic);
```

*bigFunc() 수행에 있어 Mutual exclusion은 보장 되지 않음

Master

`#pragma omp master`

`{ /* structured block */ }`

- Master thread만 수행하는 영역 생성
- 진입 및 완료 시점에 implicit barrier가 없음
 - Barrier가 필요한 경우, 명시적으로 적어주어야 함

```
#pragma omp parallel
{
    #pragma omp master
    {
        // master's work
    }

    #pragma omp barrier
    // do common work
}
```

Synchronization – 주의점

- **Synchronization은 병렬처리의 주요 bottleneck!**
 - Thread들이 서로 진행을 방해 함
- **Synchronization은 가능한 최소화 해야 함**
 - Decompose jobs into independent task sets
 - Distribute tasks evenly (workload balancing)



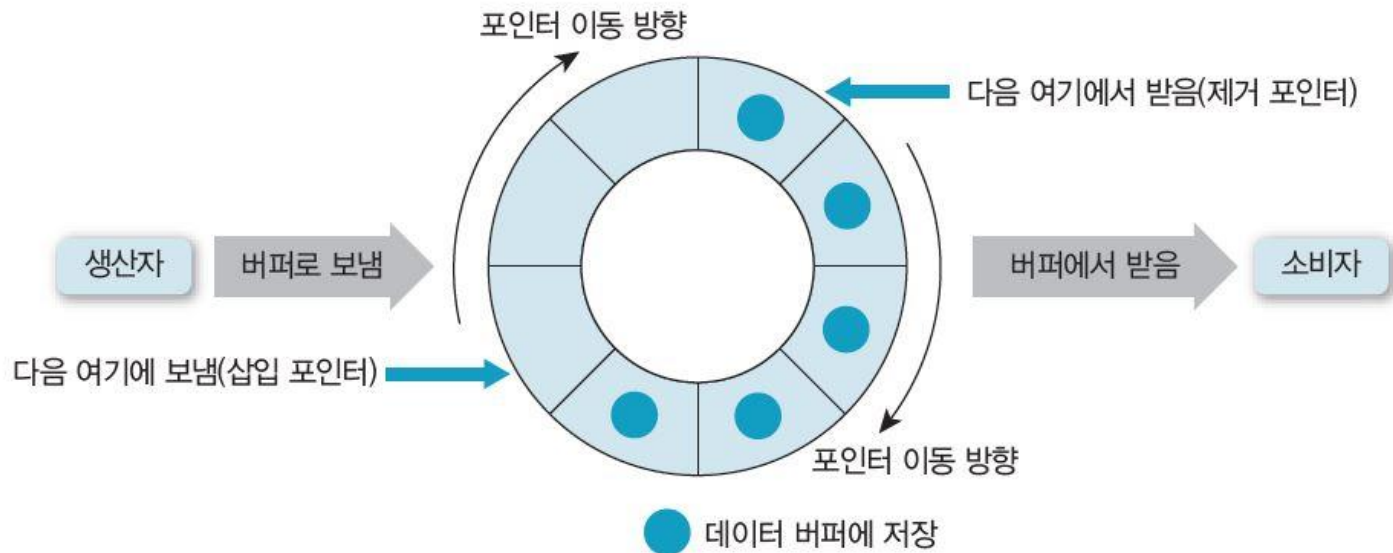
Lecture 5-2

Locks

Introduction to OpenMP

Producer-Consumer Problem

- With Synchronization Constructs?



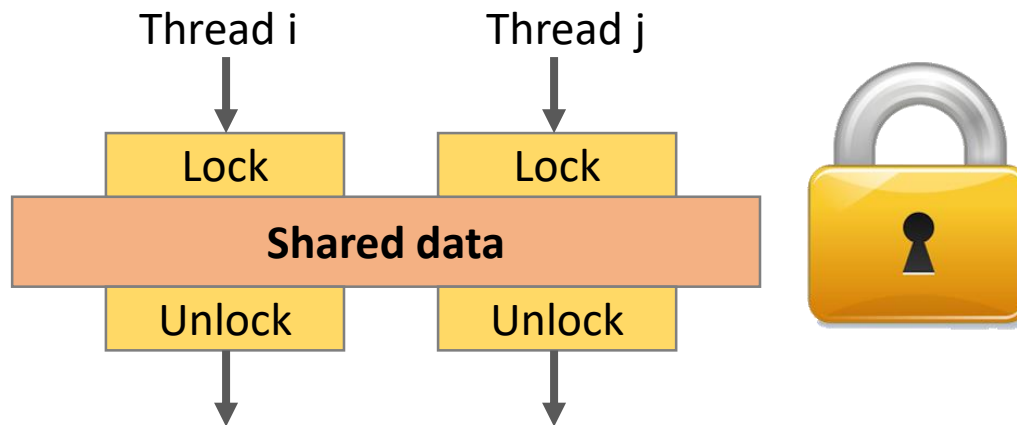
- What you need to solve the problem?

Synchronization Constructs

- 한계점

- 동일 코드 블록을 공유하는 경우만 사용 가능
- High-level synchronization

- Low-level synchronization 제어를 위해서는 **locking (or semaphore) mechanism**이 필요



OpenMP Func. For Synchronization

- Locking 기법을 특별한 변수 및 함수로 제공
- **omp_lock_t**
 - Special variable for locking
- **omp_init_lock** (omp_lock_t *lockVar)
- **omp_destroy_lock** (omp_lock_t *lockVar)
- **omp_set_lock** (omp_lock_t *lockVar)
- **omp_unset_lock** (omp_lock_t *lockVar)

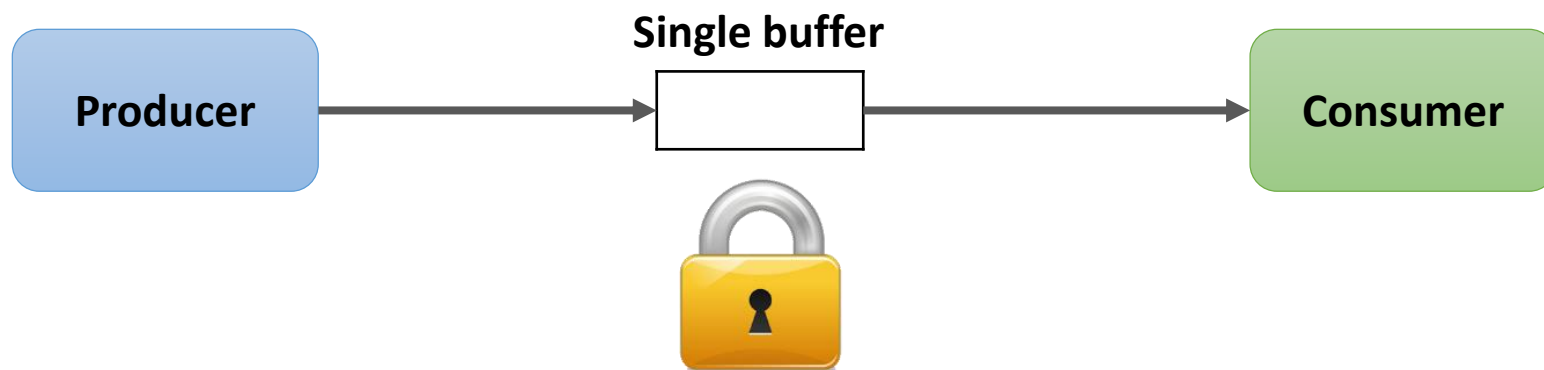
Locks (e.g., Trapezoidal Rule)

```
sum = 0;
omp_lock_t sumLock;
omp_init_lock(&sumLock);
#pragma omp parallel for num_threads(4)
for (int i = 0; i < n - 1; i++)
{
    double x_i = a + h * i;
    double x_j = a + h * (i + 1);
    double d = (f(x_i) + f(x_j)) / 2.0;

    omp_set_lock(&sumLock);
    sum += d*h;
    omp_unset_lock(&sumLock);
}
omp_destroy_lock(&sumLock);
```

Locks (example)

- **Producer-Consumer (buffer size = 1)**



- **Quick Lab.**

- 생산자는 500ms 마다 일 생성 총 10개
- 소비자는 일을 처리하는데 500ms이 걸림
- * Sleep(500)

Locks (example)

• Producer-Consumer (buffer size = 1)

```

void main(void)
{
    int buf = 0; // 0: empty, otherwise: full
    omp_lock_t lock;
    omp_init_lock(&lock);
    bool isFinish = false;
    #pragma omp parallel sections shared(isFinish, lock) num_threads(2)
    {
        #pragma omp section // Producer
        {
            int numProduce = 10;
            while (numProduce > 1 ) {
                omp_set_lock(&lock);
                if (buf == 0) {
                    buf = numProduce;
                    printf("Produce push %d\n", buf);
                    numProduce--;
                }
                omp_unset_lock(&lock);
                Sleep(500);
            }
            isFinish = true;
        }
    }
}

```

Locks (example)

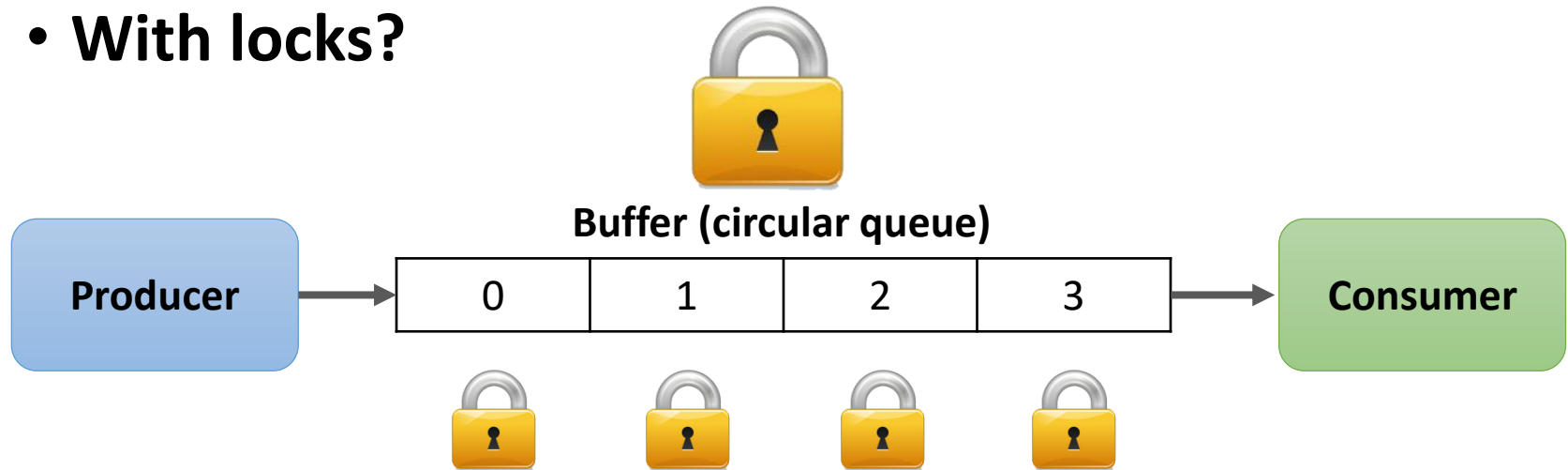
- **Producer-Consumer (buffer size = 1)**

```
#pragma omp section // Consumer
{
    int get = 0;
    while (!isFinish) {
        omp_set_lock(&lock);
        if (buf > 0){
            get = buf;
            buf = 0;
            printf("Consumer get %d\n", get);
        }
        omp_unset_lock(&lock);
        Sleep(500);
    }
}
omp_destroy_lock(&lock);

printf("Finished!\n");
}
```

Producer-Consumer Problem

- With locks?



Locks – 주의점

- **Lock (= Synchronization)**
 - Thread들의 동작을 serialize
 - 병렬처리 알고리즘의 주요 bottleneck 발생 지점
- Lock은 필요한 최소한의 영역에만 사용해야 함

Locks 관련 OpenMP 함수들

- `omp_test_lock`
- `omp_init_nest_lock`
- `omp_destroy_nest_lock`
- `omp_set_nest_lock`
- `omp_unset_nest_lock`
- `omp_test_nest_lock`
- <https://docs.microsoft.com/ko-kr/cpp/parallel/openmp/3-2-lock-functions>