Introduction to Embedded Software

Practice #8

Locking Protocol and Atomic Operation

Dongkun Shin
Embedded Software Laboratory
Sungkyunkwan University
http://nyx.skku.ac.kr/

Objective

- Understanding the concept of lock.
  - mutex, semaphore and spin lock.
- Implement the sample code which uses mutex and spin lock.
  - Check the different result
    - ✓ with lock
    - ✓ without lock
  - Measuring time when using mutex or spinlock.

Synchronization Techniques

Race Condition & Critical Section

- **Race Condition**
  - Result may change according to implementation order

- **Critical Region**
  - Section of code that must be completely executed before another kernel control path can enter it

Thread1	Thread2
Read value i (7) Increase value i (8) Save value i (8)	
	Read value i (8) Increase value i (9)
	Save value i (9)

Thread1	Thread2
Read value i (7)	
	Read value i (7)
Increase value i (8)	
	Increase value i (8)
Save value i (8)	
	Save value i (8)

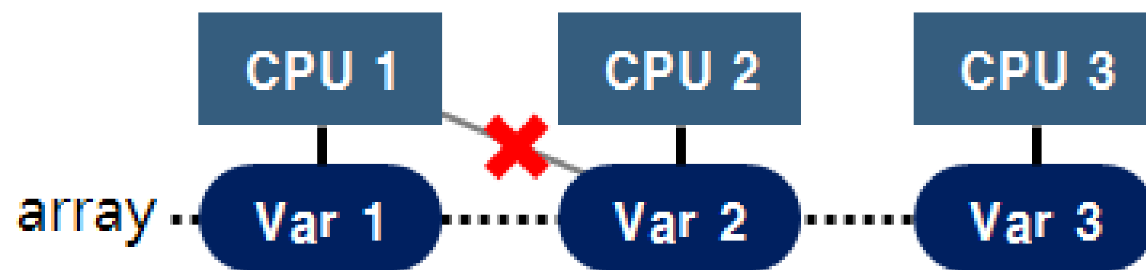
Synchronization

- **Synchronization is required to protect the critical region**
- **Locking protocol is used to synchronize the critical section in kernel**

Technique	Description	Scope
Per-CPU variables	Duplicate a data structure among the CPUs	All CPUs
Atomic operation	Atomic read-modify-write instruction to a counter	All CPUs
Memory barrier	Avoid instruction reordering	Local CPU or All CPUs
Spin lock	Lock with busy wait	All CPUs
Semaphore	Lock with blocking wait (sleep)	All CPUs
Seqlocks	Lock based on an access counter	All CPUs
Local interrupt disabling	Forbid interrupt handling on a single CPU	Local CPU
Local softirq disabling	Forbid deferrable function handling on a single CPU	Local CPU
Read-copy-update (RCU)	Lock-free access to shared data structures through pointers	All CPUs

Per-CPU Variables

- **One element per each CPU in the system**



- **Usage**

```
DEFINE_PER_CPU(type, name) : per CPU array call with a specific type and name
per_cpu (var, cpu)             : get the value of var allocated in per-cpu section
per_cpu_ptr (ptr, cpu): get the point of var allocated in  per-cpu section
get_cpu_var (var)     : operate __get_cpu_var(var)
__get_cpu_var (var)          :  get the value of var allocated in per_cpu of current cpu
```

Atomic Operation

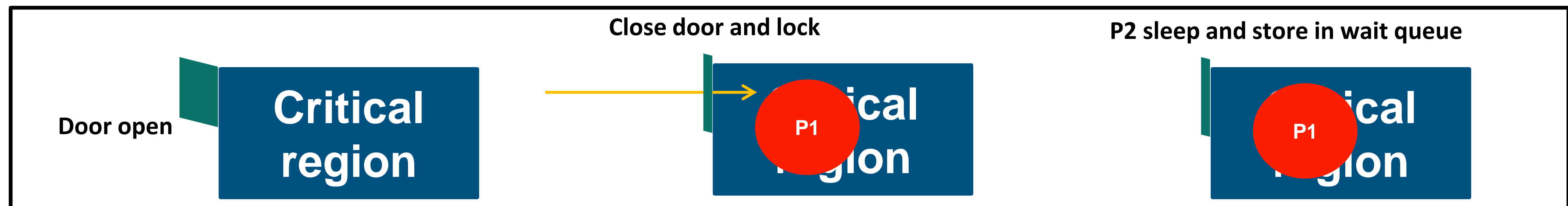
- **Every operation must be executed in a single instruction**
- **"Read-Modify-Write" → access a memory twice**
- **Usage**

atomic_read(v): return 'v'	**atomic_inc(v): Add 1 to 'v'**
atomic_set(v, i): set 'v' to i	**atomic_add_return(i, v): Add i to 'v' and return 'v'**
atomic_add(i, v): add i to 'v'	
atomic_sub(i, v): subtract i from 'v'	
atomic_sub_and_test(i, v): subtract i from 'v', if result is 0 return 1	

Semaphore

- **Only for process that can sleep**

- **Struct semaphore**
  - Count (atomic_t) [>0 : free, =0 : busy, no waiting process, <0 : busy, waiting process]
  - Wait: Address of wait queue list
  - Sleeps: Flag whether processes are sleeping

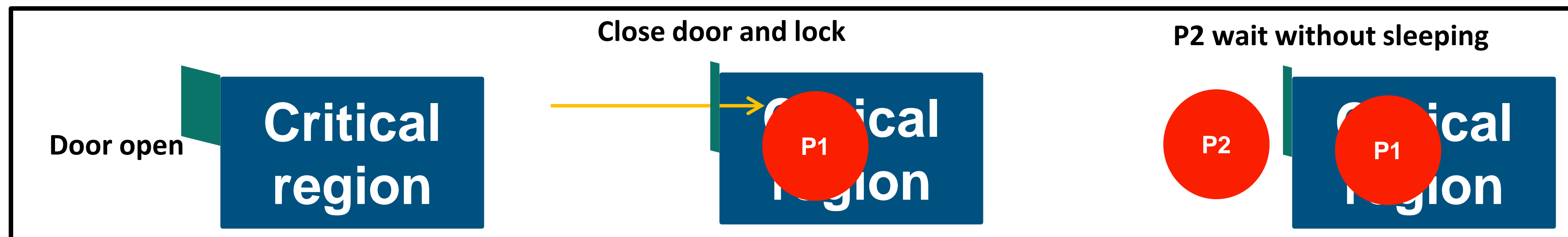


Mutex

- **Mutex have more limitation than semaphore**
  - Mutex count 1
  - Getting and releasing lock → in same context
  - Interrupt handler → can't use mutex
- **Better to use mutex than semaphore (much simpler)**
  - Semaphore can become mutex, but not vice versa
  - Semaphore cannot be owned, but mutex is owned
  - Mutex can be released by its owner, but semaphore can be released by one without ownership

Spin Lock

- While loop until kernel control path get "Lock" (busy waiting)
- Waste time → use when releasing CPU costly
- Preemption disable!



Macro	Description
spin_lock_init()	Set the spin lock to 1 (unlocked)
spin_lock()	Cycle until spin lock become 1, then set it to 0
spin_unlock()	Set the spin lock to 1
spin_unlock_wait()	Wait until the spin lock becomes 1
spin_is_locked()	Return 0 if the spin lock is set to 1

Pthread Mutex

- **Use pthread_mutex_t**
  - Get mutex lock with pthread_mutex_lock() before critical section
  - Release using pthread_mutex_unlock() after

```
int ncount;    // 쓰레드간 공유되는 자원
pthread_mutex_t  mutex = PTHREAD_MUTEX_INITIALIZER; // 쓰레드 초기화
```

```
void* do_loop(void *data)
{
    int i;
    for (i = 0; i < 10; i++)
    {
        pthread_mutex_lock(&mutex); // 잠금을 생성한다.
        printf("loop1 : %d\n", ncount);
        ncount ++;
        if(i == 10) return;
        pthread_mutex_unlock(&mutex); // 잠금을 해제한다.
        sleep(1);
    }
}
```

Exercise

- **Compare the performance of locking**
  - Spin lock, mutex
  - Measure the time with gettimeofday()
- **Initialize mutex**
  - pthread_mutex_t mutex= PTHREAD_MUTEX_INITIALIZER;
- **Create Mutli-thread**

```
sinban@eslab07:~$ ./a.out
thread created : 0
thread created : 1
thread created : 2
thread created : 3
thread created : 4
0 global value: 1
3 global value: 2
3 global value: 4
3 global value: 5
3 global value: 6
3 global value: 7
1 global value: 8
2 global value: 9
4 global value: 11
1 global value: 10
4 global value: 13
4 global value: 15
4 global value: 16
4 global value: 17
0 global value: 3
0 global value: 18
0 global value: 19
0 global value: 20
2 global value: 12
2 global value: 21
2 global value: 22
1 global value: 14
1 global value: 24
2 global value: 23
1 global value: 25
Completely join with thread 1205421824 status= 19
Completely join with thread 1205421824 status= 19
ERROR! return code from pthread_join() is 3, thread 1197029120
Completely join with thread 1205421824 status= 19
```

```
sinban@eslab07:~$ ./a.out
thread created : 0
thread created : 1
thread created : 2
thread created : 3
thread created : 4
0 global value: 1
0 global value: 2
0 global value: 3
0 global value: 4
0 global value: 5
3 global value: 6
3 global value: 7
3 global value: 8
3 global value: 9
3 global value: 10
Completely join with thread -2032572672 status= 0
2 global value: 11
2 global value: 12
2 global value: 13
2 global value: 14
2 global value: 15
1 global value: 16
1 global value: 17
1 global value: 18
1 global value: 19
1 global value: 20
4 global value: 21
4 global value: 22
4 global value: 23
4 global value: 24
4 global value: 25
Completely join with thread -2032572672 status= 0
ERROR! return code from pthread_join() is 3, thread -2040965376
Completely join with thread -2032572672 status= 0
ERROR! return code from pthread_join() is 3, thread -2040965376
ERROR! return code from pthread_join() is 3, thread -2049358080
Completely join with thread -2032572672 status= 0
ERROR! return code from pthread_join() is 3, thread -2040965376
ERROR! return code from pthread_join() is 3, thread -2049358080
ERROR! return code from pthread_join() is 3, thread -2057750784
Completely join with thread -2032572672 status= 0
ERROR! return code from pthread_join() is 3, thread -2040965376
ERROR! return code from pthread_join() is 3, thread -2049358080
ERROR! return code from pthread_join() is 3, thread -2057750784
ERROR! return code from pthread_join() is 3, thread -2066143488
sinban@eslab07:~$
```

Result

- **Simple add operation**

mutex

```
sinban@eslab07:~$
sinban@eslab07:~$ ./a.out
Usage: [command] <# of threads> <# of loop>
sinban@eslab07:~$
sinban@eslab07:~$ ./a.out 10 1000
10 pthreads, and 1000 loops
10 1000
0:5610
sinban@eslab07:~$ ./a.out 10 10000
10 pthreads, and 10000 loops
10 10000
0:39214
sinban@eslab07:~$ ./a.out 10 100000
10 pthreads, and 100000 loops
10 100000
0:309992
sinban@eslab07:~$ ./a.out 10 1000000
10 pthreads, and 1000000 loops
10 1000000
3:711292
sinban@eslab07:~$
```

spinlock

```
sinban@eslab07:~$
sinban@eslab07:~$ ./a.out 10 1000
10 pthreads, and 1000 loops
10 1000
0:2133
sinban@eslab07:~$ ./a.out 10 10000
10 pthreads, and 10000 loops
10 10000
0:36919
sinban@eslab07:~$
sinban@eslab07:~$ ./a.out 10 100000
10 pthreads, and 100000 loops
10 100000
0:408445
sinban@eslab07:~$ ./a.out 10 1000000
10 pthreads, and 1000000 loops
10 1000000
3:531413
sinban@eslab07:~$
```

s:us