05/22

Synchronization : Basics

Threads Memory Model

- Conceptual Model
  - → Multiple threads run within the context of a single process (여러 thread가 하나의 단일 프로세스 안에서 돌아감)
  - → 각 thread는 각각의 thread context를 가짐.
    - ↳ TID, stack, stack pointer, PC, condition codes, GP (Global Pointer) Register) 는 별도다!
  - → 모든 thread는 process context 를 공유.
    - ↳ Code, data, heap, shared library segments
    - ↳ file descriptor table, signal handler

Reality

→ 한 thread가 다른 thread에 read, write 가능

example program to illustrate sharing

→ Data segment

→ 각 thread의 stack

```
char **ptr;  /* global var */

int main()
{
  long i;
  pthread_t tid;
  char *msgs[2] = {
    "Hello from foo",
    "Hello from bar"
  };

  ptr = msgs;
  for (i = 0; i < 2; i++)
    Pthread_create(&tid, NULL, thread, (void *)i);
  Pthread_exit(NULL);
}
```

→ main thread의 stack

sharing.c

```
void *thread(void *vargp)
{
  long myid = (long)vargp;
  static int cnt = 0;

  printf("[%ld]: %s (cnt=%d)\n",
     myid, ptr[myid], ++cnt);
  return NULL;
}
```

전역변수 공간 (data) 영역에 할당
그렇지만 선언된 함수 안에서만 접근 가능

Mapping Variable Instances to Memory

- **Global variables**
  - → 함수밖에서 정의된 변수
  - → Virtual memory 상에서 오직 1개의 instance
- **Local variables**
  - → (static) 선언 없이 함수 내에서 정의된 변수
  - → 각 thread의 stack에는 각 local variable의 instance가 하나씩 존재
- **Local static variables**
  - → 함수 안에서 static으로 선언된 변수
  - → VM상에서 오직 1개의 instance 만 존재, Data 영역 내부

Shared Variable Analysis

	main thread	thread 0	thread 1
ptr	O	O	O
cnt	X	O	O
i.m	O	X	X
msgs.m	O	O	O
myid.p0	X	O	X
myid.p1	X	X	O

→ 전역변수.

→ ptr, cnt, msgs는 shared

→ i, myid는 그렇지 X

badcnt.c

```
/* Global shared variable */
volatile long cnt = 0; /* Counter */

int main(int argc, char **argv)
{
  long niters;
  pthread_t tid1, tid2;

  niters = atoi(argv[1]);
  Pthread_create(&tid1, NULL, thread, &niters);
  Pthread_create(&tid2, NULL, thread, &niters);
  Pthread_join(tid1, NULL);
  Pthread_join(tid2, NULL);

  /* Check result */
  if (cnt != (2 * niters))
    printf("BOOM! cnt=%ld\n", cnt);
  else
    printf("OK cnt=%ld\n", cnt);
  exit(0);
}
```

```
/* Thread routine */
void *thread(void *vargp)
{
  long i, niters = *((long *)vargp);

  for (i = 0; i < niters; i++)
    cnt++;

  return NULL;
}
```

cnt++는 atomic 하지 않다!!

Assembly Code for Counter loop

```
    movq   (%rdi), %rcx      → vargp → niters
    testq  %rcx,%rcx
    jle    .L2               → niters가 0이면 나가야해.   → Hi : Head
    movl   $0, %eax
.L3:
    movq   cnt(%rip),%rdx    → Li : Load cnt
    addq   $1, %rdx          → Ui : Update cnt
    movq   %rdx, cnt(%rip)   → Si : Store cnt
    addq   $1, %rax          → cnt++;
    cmpq   %rcx, %rax        → Ti : tail
    jne    .L3
.L2:
```

Concurrent Execution

→ 문제가 없는 상황

i (thread)	$instr_i$	$\%rdx_1$	$\%rdx_2$	cnt
1	H_1	-	-	0
1	L_1	0	-	0
1	U_1	1	-	0
1	S_1	1	-	1
2	H_2	-	-	1
2	L_2	-	1	1
2	U_2	-	2	1
2	S_2	-	2	2
2	T_2	-	2	2
1	T_1	1	-	2

→ thread 1의 critical section

→ thread 2의 critical section

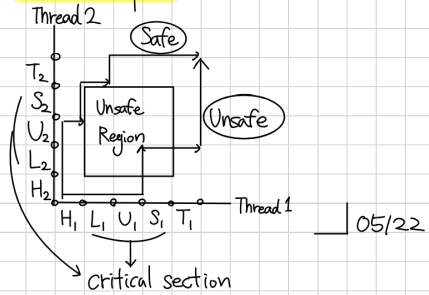
→ oops !

i (thread)	$instr_i$	$\%rdx_1$	$\%rdx_2$	cnt
1	H_1	-	-	0
1	L_1	0	-	0
1	U_1	1	-	0
2	H_2	-	-	0
2	L_2	-	0	0
1	S_1	1	-	1
1	T_1	1	-	1
2	U_2	-	1	1
2	S_2	-	1	1
2	T_2	-	1	1

i (thread)	$instr_i$	$\%rdx_1$	$\%rdx_2$	cnt
1	H_1			0
1	L_1	0		
2	H_2			
2	L_2		0	
2	U_2		1	
2	S_2		1	1
1	U_1	1		
1	S_1	1		1
1	T_1			1
2	T_2			1

→ 그렇다면 volatile로 선언?

Process Graphs



05/22

05/27

Enforcing Mutual Exclusion

→ critical section이 수행될 때 shared variable 에 대한 'Mutually Exclusive Access' 보장

Semaphores

→ 음수가 int 타입의 전역 변수이자, synchronization variable, P와 V 함수로 manipulate / "locking" mutex

- P(s) ⇒ s가 0이 아닌 양수이면 decrement하고 종료
  0 이면 thread가 suspend됨 → 다른 thread의 V 연산으로 signal 받아서 재실행
  caller에게 제어권 넘김.
- V(s) ⇒ s를 increment, 정확히 다른 하나를 restart?

∴ 둘 다 atomic함., s는 항상 >= 0

Binary Semaphore : 딱 하나만

Counting Semaphore : 여러 Thread 가능

```
sem_init (sem_t *s, 0, unsigned int val);   s를 val로 설정.
sem_wait (sem_t *s);   P(s)
sem_post (sem_t *s);   V(s)
```

goodcnt.c → 이론상) volatile 까지 해야하나?

```
volatile long cnt = 0;  /* Counter */
sem_t mutex;            /* Semaphore that protects cnt */

Sem_init(&mutex, 0, 1); /* mutex = 1 */
```

```
for (i = 0; i < niters; i++) {
    P(&mutex);
    cnt++;
    V(&mutex);
}
```

Why Mutexes Work