

공유자원.

- 하나를 두고 여러이 사용.
- 데이터 중개 RW \Rightarrow 문제 발생.

• P_1 P_2 .

Read: 10.

Read: 10.

$t = 5$.

Write: 15.

$t = 10$

Write 20.

$\Rightarrow \text{Result} = 20.$

Race Condition (경쟁 상태).

- 2개 이상 작업, 동시에 공유자원 사용 시. "데이터가 오염되었다"
- 실행 순서가 비결정적임: 실행 순서에 따라 달라짐.

Synchronization (동기화).

- 공유자원에 대해서 독점적 접근.
- 동시 접근 x, 차례대로 접근.

}
 \Rightarrow 상호 배제. 상호배제의 대상: 임계 영역.
 Mutual exclusion Critical Section.
 Mutex.

Critical Section.

- 공유자원의 존재 위치. \rightarrow
- 공유자원에 접근하는 프로그램 영역.

Mutual exclusion. (Mutex).

- 프로세스가 임계구역에 들어자면, 다른 프로세스는 임계구역 밖에서 대기.

Mutual Exclusion.

• 임계 영역에는 1개 프로세스만 진입.

• enterCS().

→ 락 만들고 열쇠를 만들어서 Lock Unlock.

• exitCS().

• (Non-Critical Code.)

enterCS().

(Critical Code).

exitCS().

(Non-Critical code).

* 이미 있으면 열릴때까지 대기.
↓

• 만족해야 하는 조건.

• 상호 배제.

• 환경 대기. (무한정 대기 X). 락 풀지 않고 진행이 CS 밖으로 간 경우.

• 진행 용량성. 다른 프로세스 진행 방해 X.

• 구현 방법.

SW ($\leq 1\%$).

HW (99%).

1. 인터럽트 금지.

2. Atomic Instruction 사용.

• 구현 / 인터럽트 금지.

• 인터럽트 금지 \Rightarrow 스케줄링 커다.

\Rightarrow Context Switching 금지.

• cli : Clear Interrupt flag.

: Critical Space.

• sti : set interrupt flag.

• 모든 인터럽트 무시 \Rightarrow 시스템 효율성 \downarrow

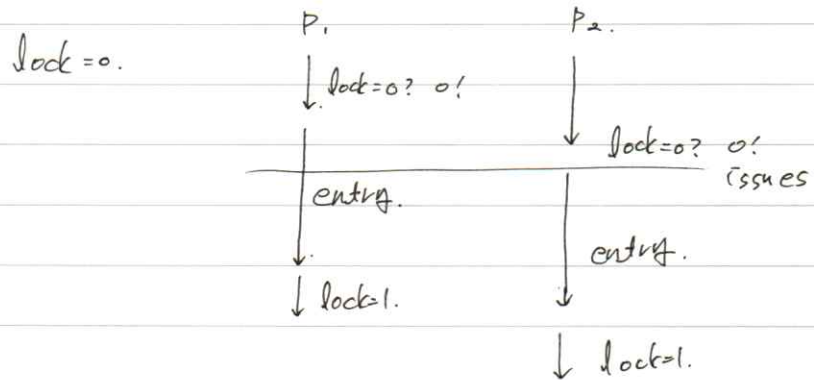
* 해결책: 전제 무시.

\Rightarrow 진행 용량성 \downarrow . \Rightarrow 해결책이 안.

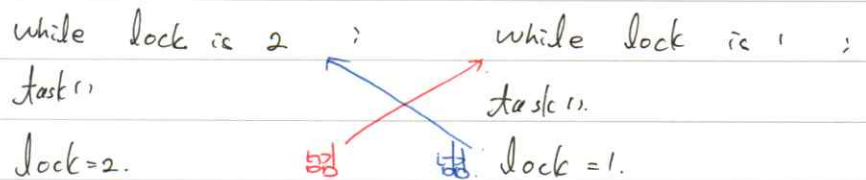
• System-wide Lock.

Mutual exclusion.

• Lock 플래그 사용 문제점. → 문제 이슈.

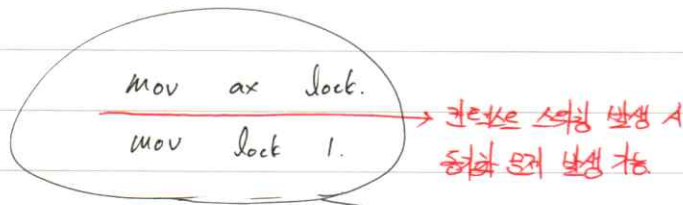


+1 flag 두개... 4원조 받기.



Atomic Instruction.

... 하나의 명령을 처리한다.



→ 합쳐서 처리! ⇒ tssl (=test and set lock).
tssl ax, lock.

동시성 제어

lock 방식: mutex

spin-lock.

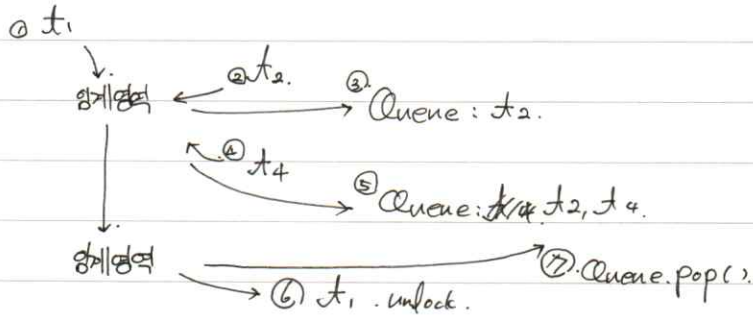
상호배제되도록 만들어진 lock 활용.

wait signal: semaphore.

n개 자원 사용에 대해 밀려서 들어올 수 있는 (n:m).

Mutex.

sleepWait.



* Mutex 실행 시 < atomic inst. 실행 >

⇒ 최적화

* 자원이 매우 적을 때 mutex queue → sleep → Awake 때문에 느리다...



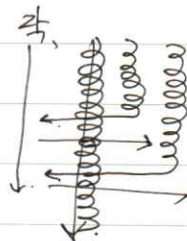
Spinlock.

BusyWait.



* lock이 풀릴 때까지 계속 체크 (≈ while, poll).

* 싱글코어에서 비효율적, 멀티코어에서 적합.

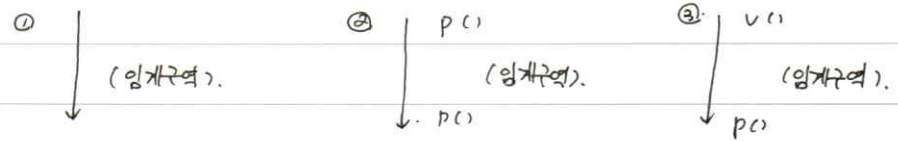
다중처리에 발생 가능. (정확성을 유실함)



→ 여러 스레드의 경쟁.

# Mutex vs spinlock.	
락 상태 ↑.	작업들 X \Rightarrow 다른 스레드에 양보.
다른 CPU.	다른 CPU에서 스핀락 해체 X. 가아 상태 발생 가능.
멀티코어 CPU.	잠재고 가아 컨디션 스리핑 X.
사용자 인터페이스	커보코드. { 커보코드, 인터럽트 서비스 루틴: 별리 실행 인터럽트 서비스 루틴, 잠깐 수 없음.
# 세마포어.	"수신료" 관리에 유용함.
	· 자원: n개.
	대기: 자원 할당 못받은 스레드의 대기.
	Counter 변수: 사용 가능한 자원의 개수.
P/V 연산.	{ P wait. 자원 요청 시 실행 V signal. 자원 반환 시 실행.
· 구현.	busy wait sleep wait.
	P(s). P(s).
	while s < 0.;
	s--
	if s < 0 then sleep().
	V(s).
	s++.
	if s < 0 then awake().
# 이진 세마포어	· 세마포어가 관리하는 자원이 하나인 경우 (\approx 뮤텁스). \Rightarrow 
# Mutex vs semaphore.	
동작화 대상 1개	1개 ↑.
	
자원 소유 책임.	자원 소유 불.
소유 스레드 해제 가능.	소유 스레드를 해제 가능.

1/0/2021 (+ 10/10) 24/8



Monitor.

- Δ 에 따라 \cdot 가변 \mathbb{R}^n 상에서 Δ 에 따라 변.
 - Δ 에 따라 ... \mathbb{R}^n 상에서 Δ 에 따라 변.
- P, U 는 \mathbb{R}^n 상에서 Δ 에 따라 변.

Priority Inversion. $\Delta t_{\text{idle}} = \Delta t_{\text{idle}} + \Delta t_{\text{idle}} + \Delta t_{\text{idle}} + \Delta t_{\text{idle}} + \Delta t_{\text{idle}}$

ਮਾਧੁਸੂਤ ਦਿਆ.

- Real Time System의 2가지 발생

- T_3 가 lock을 지니고 있기 때문에 대기 상태.
 T_2 가 먼저 그 lock을 지니고 있음.
 $\Rightarrow T_2 \rightarrow T_3 \dots$ 하지만 T_3 의 lock을 $\uparrow \Rightarrow$ 역전.

* 해설해.

- உதாரணம்: லாக் செய்து அதை வாங்கி ↑.

→ $\frac{A}{B} \uparrow \downarrow$ 일. (분자를 바꿀 때 곱할... 나눌 몫이...
구분할 몫은 다(나)지).

- 유니버설 상속: 낮은 순위의 스레드 종류가 될수록 ...
높은 순위의 유형이
낮은 순위의 상속을 높은 순위의 상속 실행.

442 - 542 - 571.

- 생선과 : enqueue()

~~Satz~~: dequene (.)

⇒ 생각과 습관이 함께 실행된다면? { 비어있는데 믿음
과 참신이 심

3차 문제.

수배제

Empty head.

fuel load.

- 해결: 두개의 시디를

Read only

Write Only.

→ $\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2}$?

→ 4가지 기능의 개수?

- Consumer

while (P(R). ←)

Producer

while $\left(\begin{array}{c} P(w) \\ \vee (P) \end{array} \right)$