11 Feb 2025 - Operating Systems - II - Week 06

Peterson's algorithm : problem : instructions get reordered during compilation.

$\searrow$ critical section problem

instructions $\downarrow$ modifying global variables

→ o algorithm . cpp → which lines in which section

→ happens even in single core systems → preemption, round robin

How to solve this ?

0> Hardware

* uniprocessors: disable interrupts for a short while to not get preemption
* multiprocessors → similar
* not feasible → interrupts can be important

Project

times might have been longer ← { printf → internally uses mutual exclusion & locks

cont → x

1> Memory barriers

→ Memory models : strongly ordered, weakly ordered.
→ internally ensures reading from cache instead of memory
→ not popular : slow, architecture dependent

2) Hardware instructions

* apropos ?
in terminal

→ C / C++

→ test_and_set → provided by hardware
→ available in all hardware : names change, all have a wrapper in C

→ executed atomically (without any interruption)

→ returns the original val of passed param

→ cpp

Why not atomic variables?
→ comes with a cost (what ?)

<u>Critical Section Problem</u>

→ mutual exclusion ✓

→ progress ✗

→ bound ✗

(CAS)
→ compare_and_swap : more powerful and expensive

→ expects int*

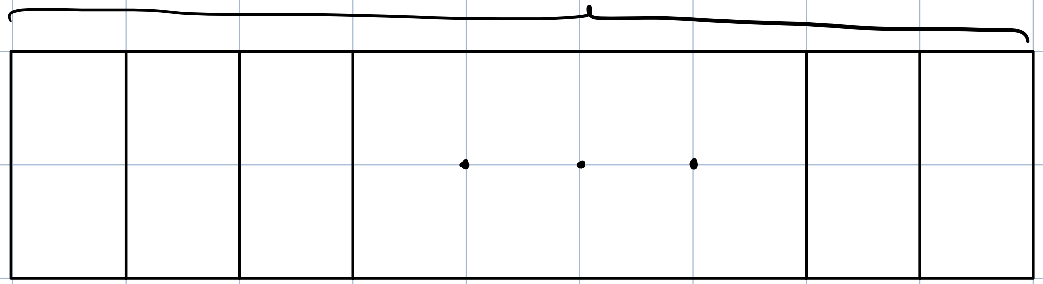
→ consensus

→ drawback : bounded x

12 Feb 2024

Bounded :

waiting



→ disadvantage
cannot be created immediately
→ create, destroy, fork, join

key = 1

lock = 0

NSM
IIT Hyderabad

```
1.  while true
    1.  Set waiting [i] = true;            // like flag
    2.  key = 1
    3.  while (waiting [i] && key == 1)
            1.  key = compare_and_swap (& lock, 0, 1)
    5.  waiting [i] = false

    6.  Critical section
/* exit : To be fair : help others */
    7.  set j = (i+1)% n                   /* next in loop */
    8.  while((j != i) && !waiting [j])    /* if process not
            1.  j = (j+1)% n                  waiting, check next
                                              until yo
```

$$\begin{tikzcd} \text{lock } 0 \arrow[r, "\text{change}", "\text{lock}"'] & 0 \; \} \text{ winner} \\ 1 \arrow[r] & 1 \; \} \text{ waiters} \end{tikzcd}$$

has to be atomic

Altruism promotes shared growth

```
9.    if ( j = = i )
          lock = 0;
      else
          waiting[j] = false;

10.   Remainder section
```

→ Bounded but not fair

40	50	60
40 requests ☹ wait more	C.S exit	60 gets CPU ☺

For fairness → need common clock

4 : 00 — P_{40}

4 : 10 — P_{60}

$\boxed{P_{50}}$

store in FIFO queue
↳ define notion of first when multiple threads are enqueing stuff.

$\boxed{??}$

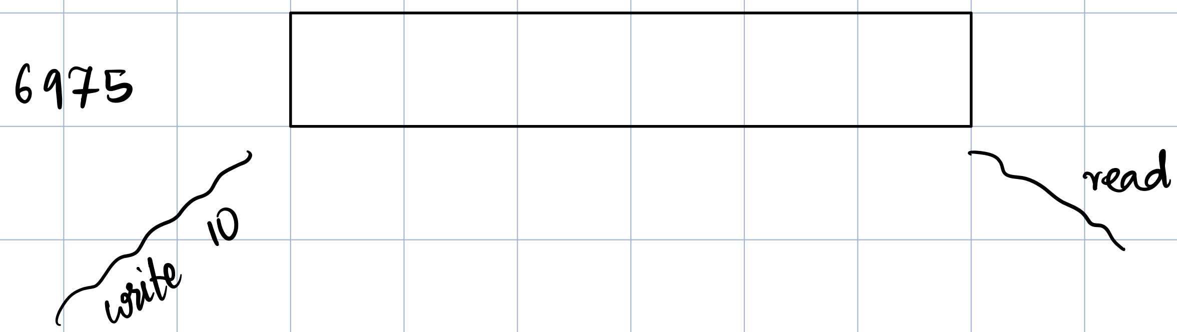
CAS
↳ hardware, determinism

Queue → object
↓
software

Atomic Variables

→ increment without interruption

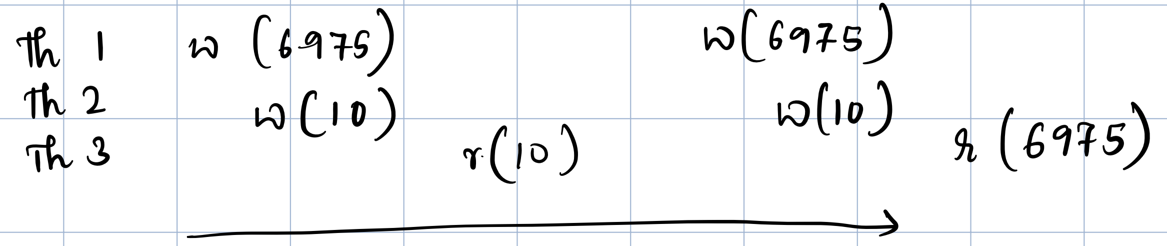
C++ → template ⇝ load and store



⇝ either 6975 or 10 not anything else

does not have to be same multiple times

read without interruption



Make $\underline{\underline{\text{struct}}}$ $\underline{\underline{\text{atomic}}}$?

↳ KBs of data not just 64 bits

$(x_1, y_1) \longrightarrow (x_2, y_2)$ $(5, 28)$ ✗

$(5, 10) \longrightarrow (17, 28)$ $(17, 10)$ ✗

$\downarrow$ atomic $\downarrow$ atomic $\not\Rightarrow$ overall struct is atomic

Solution : ?

increment ( ) $\longrightarrow$ atomic but not bounded waiting