Entry: 2013CS50286
Anket Prakash Hirulkar

2020

Homework-4

Wk 11 • Day 070-296

March 10
Tuesday

- Assignment: Spin-locking

09

Q: with sti() after acquire and cli() before released the kernel panicked.

10 Sol⇒ There was a clear message on shell showing "panick"

11 after the above changes were made.

12 CLI (clear interrupt flag) disables any upcoming interrupts.

01

Here there is race condition in ide.c because of

02 which the kernel panicks.
The atomicity is not maintained

03 The kernel panicks on second acquire().

04 Q: With changes in file.c [sti() after acquire() & cli() before two releases] the kernel does not

05 panick.

06⇒ In file_table_lock, the race condition does not occur.

07 There is atomicity in instructions achieved.

first release cleared lk→pcs[0] and lk→cpu before clearing lk-locked.

08 Since the interrupts were disabled with cli(), second release ran only after first release completed its execution.
After that second release ran into execution.

APRIL MAY JUNE

# • Uniprocessor Locking:

09

① Lock(L){
10 cli(); // disable preemption
while (L==0) continue; (A)
11 L=0
sti(); # enable preemption
12
unLock(L){
01 L=1; (B)
}

02 ⇒ This implementation will not work:

03 consider thread 1 was at step A, where it had to check if L!=0 (unlocked

04

Assuming L was already 0, thread 1 continued.
05 But at the same time, thread 2 was at step B so it set L=1, by the time
06 thread 1 moved ahead to set L=0.

07 then thread 2 made an error of judgement

Due to this discrepancy (potential discrepancy), this lock will not work.

Since both threads execute L=0 & L=1 at the same time - thread 2 would unlock the lock nullifying effect of thread 1

(2) lock (L) {
int acquired = 0;
while (!acquired) { A
cli(); B
if (L == 1) {
acquired = 1;
L = 0
}
sti();
}
}

unlock (L) {
L = 1
}

⇒ Yes, this will work.

In this case, after thread 1 check the condition at A and moves to step B, its enables preemption, so it will not let any other thread unlock the lock.

Because of this cli() command, race condition does not occur.

Also if we recall, this locking discipline ensures race-freedom because following condition holds (as mentioned in lecture on locking)
(1) We associate a lock with shared region (i.e. L)
(2) Before assesing shared region, ensure corresponding associated lock is held.
This conditions are fulfilled here.

APRIL
MAY
JUNE

- Assignment: Sleep and wakeup

Sol^n: This implementation is correct.

It is okay, if both producer (pcq write) and consumer (pcqread) are sleeping on the same channel q.

Even if the producer calls wakeup(q);
- consumer does not have authority to acquire the lock before producer released it. And producer releases lock only after wakeup call.

Since this is Spinlock, consumer will be kept waiting unless producer releases the lock.

So this implementation is good.

- Assignment: XV6 file system

09

- After adding the line

10 ~~cprintf(f)~~

cprintf("log_write %d \n", b→sector/blockno);

11

① Output of echo>a

12 ⇒ log_write 34

log_write 34

01 log_write 59

02 (I didn't get sector 29 as mentioned in homework. Instead I got sector 59).

03

04 ② Output of echo X>9

⇒ log_write 58

05 log_write 571

log_write 571

06 log_write 34

log_write 571

07 log_write 34

③ Output of rm a

⇒ log_write 59

log_write 34

log_write 58

log_write 34

log_write 34

Explaination for this are on next page.

⊙ Explanation for echo 79.

09

⇒ In file creation, this steps happen

10 (1) allocate an inode (write to inode region)

(2) Update the size/status of the inode

11 (write to inode region)

(3) Add a directory-entry record to parent

12 directory's data blocks

(write to data block region)

01

∴ in log-write 34

02 log-write 34

log-write 59

03

34 is inode region

04 59 is data block region

05 ⊙ Exp. for echo x > a

06 The step happened are as follows.

1. Allocate a block (write to block

07 bitmap region)

2. Zero out block content (data block region)

3. Write "x" to block (to block data region)

4. update "a"s inode (size, addrs)

∴ In this case, log-write 58

Indicates that 58 is block bitmap region.

~~[redacted]~~ "X" was written to data block region 571

two entries of log-write 34 = one for size update

one for addr update

09 (•) Explanation for rm a

10 ⇒ following steps occur:

11 ~~1- Allocate a block~~

1- Write zero to "a"s directory-entry record
12 in parent directory's data blocks (write to data block region)

01 2- Update parent directory's inode (size)

3- Update "a"s inode to mark it free

02 4- Update free block bitmap to mark data blocks free.

03

∴ In output of rm a

04

In log-write 58, "a"s block bitmap region was
05 freed.

06 In log-write 34, "a"s inode was marked free.

07 In log write 59, zeroes were written to data block region.

• Assignment ZCAV (plot attached for disk on physical laptop)