Homework 4

Rayala Harichandan
2016CS10320

1) Spin-Locking

c) When lk→locked is cleared, other threads will be able to aquire the lock and then, this process will be trying to clear lk→pcs[0] and lk→cpu and ~~the aquiring process will have these values instead of zeros.~~ also the aquiring process will be able to access lk→pcs[0] and lk→cpu and this causes race condition.

d) Uniprocessor locking.

i) The first implementation doesn't work in uniprocessor OS.

It disables the interrupts and spins on the lock. If the lock is not free, it keeps spinning. As interrupts are disabled, thread switching doesn't happen. So, the thread having the lock will not be able to release it. ~~So, it keeps spin~~

So, the process keeps spinning on lock. It never gets it.

ii) The second implementation works, as interrupts are being enabled after each loop and thread switching can happen.

3) XV6 File System. (Note: changed b→sector to b→blockno)

a) echo>a

Printed output : log_write 34
log_write 34
log_write 59

i) First write is to allocate inode to 'a'.

ii) Second write is to update size, status, addrs of inode of 'a'.

iii) Create 'a's directory entry record in the parent directory file.

The third disk write is to the parent directory to add directory record of a.

b) echo x >a

printed output: log_write 58
log_write 567
log_write 567
log_write 34
log_write 567
log_write 34

i) Allocate a block (write to bitmap)

ii) Zero out the block (write to block)

iii) Write 'x' to block

iv) Update a's inode (size and addrs[])

v) Write newline to block

vi) ~~ti~~ Update a's inode.

Block and inode are updated twice as first is for 'x' and second time for newline.

c) rm a

printed output: log_write 59
log_write 34
log_write 58
log_write 34
log_write 34

i) Zero out directory entry record of 'a' in parent directory.

ii) Unlink blocks from a's inode.

iii) Free up the blocks in bitmap.

iv) Change inode size to zero.

v) Mark inode as free.

Note:- To find what is happening in each write, I have added print statements in fs.c file.

2) Sleep and wakeup

If both sleeps on same channel, wakeup also wakes up other producers. If a producer aquires lock, as it is in while loop, it will check if q->ptr != 0 and if q->ptr != 0, it will sleep agian and when a consumer wakes, it will check and proceed.

So, sleeping on same channel has a little preformance issue but it is correct.

Waiting on different channels improves performance.