Homework-4

2016CS10348
Himanshu

Assignment : Spin-locking

Turn in: explain in a few sentences why the kernel panicked. You may find it useful to look up the stack trace (the sequence of %eip values printed by panic) in the kernel.asm listing.

In this code, Sti() function enables interrupt in ide which effectively means that interrupt(such as time-interrupt) can come after acquire-lock.
Also, Ide is running in kernel mode.
This causes panic due to some interrupts(eg time-interrupt) while running lock.
r = lock−>locked && lock−>cpu == mycpu(); (at line 1656, of holding() in spinlock.c)
As a result, some times ‘r’ became non-zero(due to interrupts) and give panic. In other times interrupt don’t come so, it doesn't give me panic.

Turn in: explain in a few sentences why the kernel didn't panic. Why do file_table_lock and ide_lock have different behavior in this respect?

File_table_lock is in user mode whereas ide_lock is in kernel mode. Usermode donot need to disable and enable interrupts. Kernel’s interrupt handlers are not expected to touch user data, so disabling interrupts is not required.
As a result, even after disabling interrupt by using sti(), File_table_lock don’t get any panic.

Turn in: Why does release() clear lk->pcs[0] and lk->cpu *before* clearing lk->locked? Why not wait until after?

As soon as you release the lock, other processes can access the cpu-variable and pcs-variable. So programmer is using same lock(lk->locked) to protect internal structures lk->cpu, lk->pcs[0]. That is why we can’t use them after lk->locked().

Uniprocessor Locking

lock(L) {

```
 cli();   //disable preemption
  while (L==0) continue;
  L = 0;
  sti();   //enable preemption
}

unlock(L) {
  L = 1;
}
```

Does this implementation of locks work on a uniprocessor? If not, why not?

No,This code will move to infinite while-loop when one process has already acquired lock and other process is in while(L==0). It won't work as it will stuck in while loop because interrupts are disabled.

Lets assume there are 2 processes P1 and P2 running on uniprocessor and want to use same lock(L). Let say P1 acquired lock and currently implementing the critical section. Now an timer interrupt comes, and it switches to process P2. Now P2 is trying to acquire lock(L). P2 will never be able to come out of while(L==0) because interrupts are disabled(any interrupt cant come) at this time(by cli()) and it is uniprocess means that P1 can't able to unlock(L).

As a result, two processes simultaneously enter the critical section at the same time.

lock(L) {

```
 int acquired = 0;
  while (!acquired) {
    cli();
    if (L == 1) {
        acquired = 1;
        L = 0;
    }
    sti();
   }
}

unlock(L) {
  L = 1;
}
```

Does this implementation of locks work on a uniprocessor? If not, why not?

No, it won't work as two processes can simultaneously enter the critical section at the same time.
For example, 2 process P1, P2 want to acquire same lock(L). Lets assume that currently P1 process is running. Just after entring while(!acquired), time interrupt came and it shift to process P2 (this happen before P1 run cli). Now P2 also want to acquire same lock and has successfully got lock and now start running critical section. So now, value of acquire is 1 and L is 0. Now, During critical section of P2, time-interrupt switches again switches to P1. It will come out of while-loop without enabling L and start running Critical section by P1. As a result, critical section of P1, P2 are simultaneously being implemented. So, it is incorrect implementation.

Assignment : sleep and wakeup

Turn in: Both producer (pcqwrite) and consumer (pcqread) are sleeping on the same channel q. Is this correct? Why or why not? Should they sleep on different channels? For example, what happens if the producer calls wakeup(q)? Can some unrelated part of the code call wakeup a consumer thread?

This question is the special case where the size of the queue is one.
Both producer and consumer are sleeping on the same channel. It don't have any problem because only one out of both sleep will be able to come out of while-loop.
As this is a special case the size of the queue is one. It is correct to use sleep on one channel.

If producer calls wakeup(q), then all sleep of both write and read will start working. By q->ptr = p, we already set the value of ptr to non-zero, then the write function will again go to sleep mode because while-loop is non-zero. In this only read-function will be able to execute.

No unrelated part will not be able to wake-up a consumer thread because unrelated code will stuck in sleep-loop.

Assignment : xv6 file system

$ echo > a
Turn in: Report the printed output and explain what is being written in each disk write. What is the third disk write (to sector 29)? You may want to insert cprintf() statements in xv6 code to see where the writes are coming from.

log_write 34 //(for ialloc)
log_write 34 //(for iupdate)
log_write 59 //(for writei)

ialloc(): allocate an inode
In this, first allocate an inode on devic. Mark it allocated by giving it type. Returns an unlocked but allocated and referenced inode.

iupdate(): update the size/status of the inode
It copy a modified in-memory inode to disk. It is called after every change to an ip->xxx field that lives on disk, since i-node cache is write through.

writei(): write data to inode.

Third disk write means
Writei-function is called to write a new directory entry into directory dp. add a directory-entry record to the parent directory's data blocks.

$ echo x > a
Turn in: Report the printed output and explain what is being written in each disk write. Why do you see writes to block 4 (and to 426) twice? You may want to insert cprintf() statements in xv6 code to see where the writes are coming from.

log_write 58 //(for balloc)
log_write 660 //(for bzero)
log_write 660 //(for writei)
log_write 34 //(for iupdate)
log_write 660 //(for writei)
log_write 34 //(for iupdate)

balloc(): Allocate a zeroed disk block.(write to block bitmap region)

bzero(): Zero out block contents

writei(): write to data block region

iupdate(): writei-function internally calls iupdate(). It copy a modified in-memory inode to disk. It is called after every change to an ip->xxx field that lives on disk, since i-node cache is write through.

writei(): write "x" to block (write to data block region)

iupdate(): update "a"'s inode (size, addrs)
It copy a modified in-memory inode to disk. It is called after every change to an ip->xxx field that lives on disk, since i-node cache is write through.

$ rm a
Turn in: Report the printed output and explain what is being written in each disk write.

log_write 59 (for writei)
log_write 34 (for iupdate)
log_write 58 (for bfree)
log_write 34 (for iupdate)
log_write 34 (for iupdate)

writei(): write zero to "a"'s directory-entry record in parent directory's data blocks (write to data block region)
.
iupdate(): update parent directory's inode (size)
(It copy a modified in-memory inode to disk. It is called after every change to an ip->xxx field that lives on disk, since i-node cache is write through.)

bfree(): Free a disk block

iupdate(): update "a"'s inode to mark it free
(called during Truncate inode. It is called when inode has no links to it and has no in-memory reference to it. It copy a modified in-memory inode to disk.)

iupdate(): update free block bitmap to mark data blocks free
(Drop a reference to an in-memory inode. Inode has no links and it is finally freed)

Assignment : ZCAV

Turn in
Run ZCAV using the following command for the following types of disks:

- A disk on a physical laptop. If you have a laptop, report its make and model, and published disk characteristics (seek, latency, throughput). Then measure it using ZCAV and report. Also, report any interesting conclustions (if any).

If you do not have access to a laptop, please borrow it from one of your friends or one of the TAs, and clearly specify whose laptop you borrowed.

Make and Model

Product Name: Mac-6F01561E16C75D06

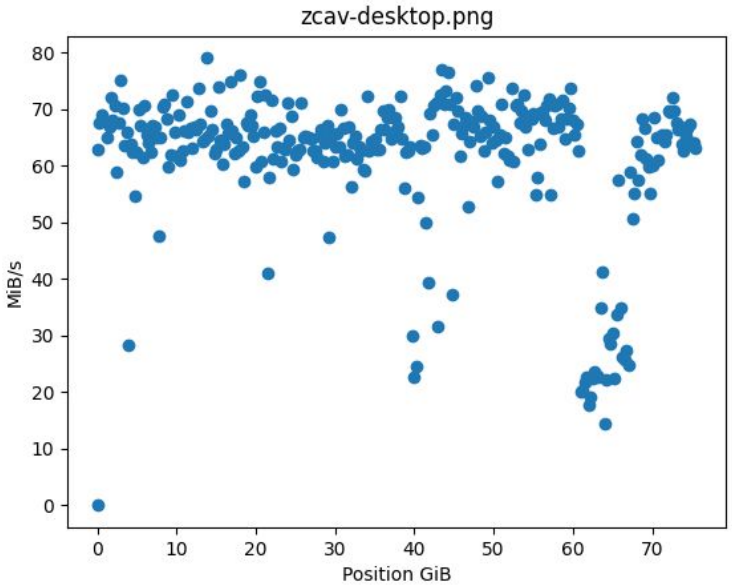
Version: MacBookPro9,2

Serial Number: C0763230292F1YLHD.

Storage: **2.5GHz,** 500GB 5400-rpm hard drive

Memory: 4GB of 1600MHz DDR3 memory

ZCAV: Read 75 gigs in 1387 seconds, 55 megabytes per second.



Observation Mac has SSD as device storage while normal windows or Ubuntu laptops have magnetic disk i.e. HDD. Zcav is specially made to check zone velocities while SSDs have no moving parts as it uses flash memory. The results are appropriate for SSD.

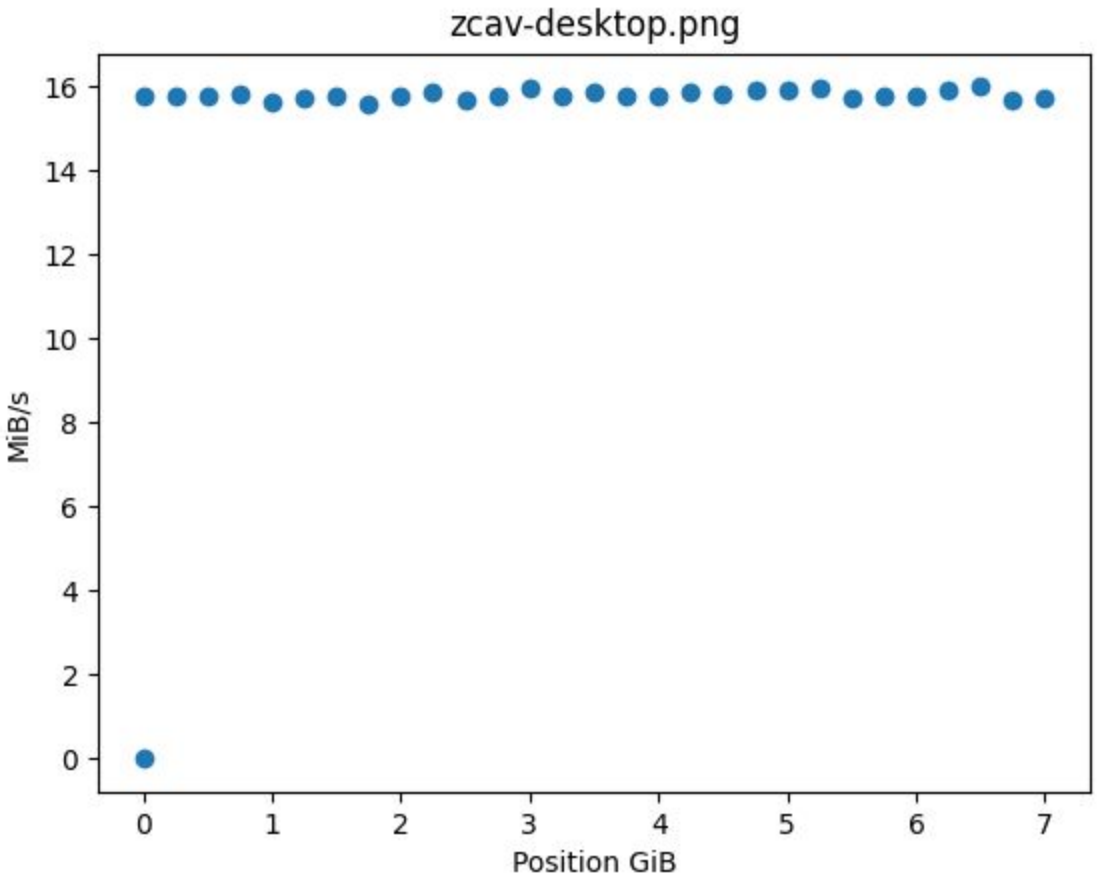
So, no zone is formed.

- A USB (pen) drive. Report the make, model, and published characteristics of the USB drive and compare with your obtained results. If you do not have a USB drive, please borrow one.

Product Name: Hp 8GB pendrive

Serial Number: V165W

ZCAV: Read 7 gigs in 469 seconds, 15 megabytes per second.



Observation: Speed of pendrive is constant across all memory access. This is because it don’t store in rotating-disk form.
No zone is formed.