COL331-OPERATING SYSTEMS

HOMEWORK-4

SHIKHAR GOEL
2016CH10089

Assignment: Spin locking

~~Assignment: spin locking~~

1.) On disabling interrupts ~~in~~ idew kernel panicks.

On examining %eip output, it is found ~~that~~ ~~has~~ in stack-trace that it calls myproc so in idew function, while ~~the~~ [illegible] the thread is sleeping it releases the spin-lock and . on waking up re-acquires the lock.

If an interrupt occurs before it sleeps then ide interrupt function gets called which again tries to acquire the lock. this will cause kernel to panic.

on disabling interrupts in filealloc, kernel does not panic

On receiving an interrupt during the file allocation, the command is taken over by interrupt handler which may context switch or ~~change~~ ~~given~~ give command back to kernel in different process. ~~Hence~~ ~~the~~ ~~lock~~ the kernel does not try to re-acquire the lock and hence it does not panick.

file_table_lock and ide_lock are different in this respect are ide-lock is acquired ~~as~~ till it runs in user space ~~stack~~ whereas file_table_lock is safely handled by kernel.

release() clears lk→pcs[0] and lk→cpu before ~~&~~ clearing lk→locked because if ~~lock is~~ lk→locked is cleared first then another process may try to acquire a lock overwrite the values of lk→pcs[0] and lk→cpu which can then get cleared by release().

Uniprocessor locking

first implementation does not work as the lock methods puts the program into an infinite loop for ~~to~~ acquiring a lock which is already locked. ~~the system cannot exit this loop and~~ without any mechanism to exit it without interrupts. Hence the original program which acquired the lock cannot proceed and unlock the lock.

second implementation works as it overcomes this problem.

2.) Assignment : sleep and wakeup .

Both consumer and producer cannot sleep on the same thread as this may create a deadlock. Suppose a scenario where all consumers are sleeping and command ~~with~~ is a with a producer P1. It wakes up another producer P2 after executing which in turn will wake up P1 if P1 and P2 both ~~are~~ occupy positions in ~~[illegible]~~ before consumers in ptable. This will never wakeup consumers and when ~~it~~ queue becomes full both producer will themselves sleep too creating a deadlock.

They should sleep on different channels as they can be woken exclusively after certain conditions are met.

An unrelated part of the code can call wakeup a consumer thread if it is ~~one~~ on the same channel ~~however the~~ but ~~there~~ there ~~does~~ has to be running process which should call it.

3.) Assignment : xv6 file system.

```
$ echo > a
log_write 34
log_write 34
log_write 59
```

→ the first statement allocates an inode for file a, writes to inode region, block 34

→ the second statement updates the value of inode for file a, writes to inode region, block 34.

→ the third statement adds an entry for a in the contents to the current working directory. (writes to data block region), block 59. is cwd block.

```
$ echo > a.
$ $ init : starting sh.
$ echo x > a.
log_write 58
log_write 644
log_write 644
log_write 34
log_write 644
log_write 34
```

→ the first statement allocates a block by writing to block bitmap region,, block 58

→ the second ~~second~~ statement write to the data block region to zero out it's content, data block 644

→ the third statement write 'x' to that data block region.

→ the fourth state write to inode region the location of block and size.

→ fifth and sixth statements are repeats of ~~8~~ third and fourth statements, asynchronous method in xv6 ~~in case of non [illegible]~~

```
$ rm a
log-write 59    # comes from write i
log-write 34    # comes from iupdate
log-write 58    # comes from bfree
log-write 34    # comes from iupdate
log-write 34    # comes from iupdate
```

→ first statement write zero to 'a''s directory entry record in parent's directory's data block in the data block region.

→ second statement updates parent's directory inode (size) in inode region

→ third statement frees ~~the~~ 'a' block in the block region

→ fourth ~~statement~~ and fifth statement updates 'a''s inode size and address. in the inode region.

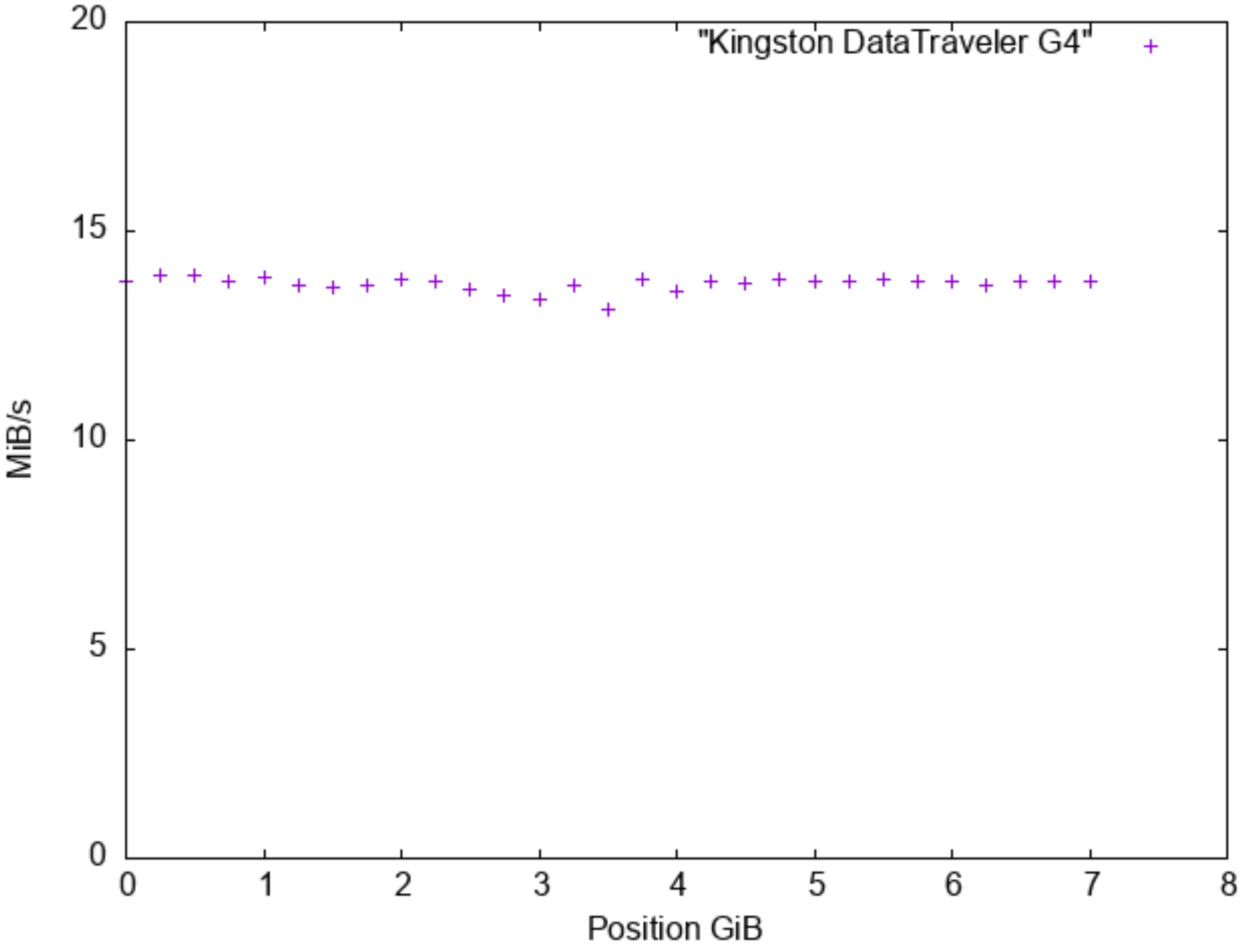
Assignment: 2CAV

The 2CAV ~~doe~~ tool fails to run on my macbook pro. However it runs a Kingston DataTraveler G4 8GB pendrive. It is observed the read speed is almost constant of about 13 MB/s.

The same ~~for~~ can be observed for other SSDs in general.

HDDs have constant angular velocity and hence storage space ~~is~~ at outer 'rings' are ~~most qu~~ more quickly accessed. H/W organises ~~into~~ it into ~~to~~ 'zones'.

For observation on pendrive, ~~sector of~~ ~~0.2~~ data offsets at 0.25 GB at noted for the whole sectors


"Kingston DataTraveler G4"
20
15
10
5
0
MiB/s
0
1
2
3
4
5
6
7
8
Position GiB