Prashant Singh Chandel
2016PH10566

Homework-4

Assignment: spinlocking

i) The kernel panicked because ide lock is a spin lock. And spinlocks in XV6 should always run with interrupts disabled, if not like in our case panic will result because the interrupt handler would want to acquire the lock resulting in a deadlock.

ii) The kernel didn't panic in this case of file_table lock because it is a sleep lock that doesn't need interrupts disabled and releases the lock before yielding. So a deadlock can't occur in this case. The difference in behaviour of file_table lock and ide lock is because they are diff types of locks. ide lock is spin lock while file_table lock is sleep lock.

iii) release() clears out lk->pcs[0] and lk->cpu while holding the lock to preserve atomicity and prevent interleaving so that these instructions don't change the value for while some other thread is holding the lock.

Uniprocessor locking

i) This won't work on a uniprocessor because interrupts have been disabled before while loop. This will result in the loop running endlessly and no other process will get a chance to execute.

ii) Yes, this will work on a uniprocessor as it will

give other processes a chance to execute by which can release or acquire the lock.

Assignment: sleep and wakeup

Yes both the operations sleeping on the same channel is correct because when producer or consumer finishes its operation it would like to wake up the consumer or producer respectively who are sleeping on the same channel only. If they sleep on diff channels then would be an issue because on wakeup(q) it ~~wakes up~~ by producer it wakes up all the ~~producers~~ consumers sleeping on that channel by going through the process table and one of them will be able to execute.

Unrelated part of the code can call wakeup on a channel and then the consumer would check if its condition is satisfied if it still isn't then it would go to sleep again.

Assignment: xv6 file system

```
echo >a
o/p  log_write 34
     log_write 34
     log_write 59
```

The first is allocating an inode on device driver by marking it allocated on disk.
The second is copying a modified in-memory inode to disk.
The third is writing data to inode.

echo x > a.

o/p 1 log_write 58. (bmap start is 58 so allocated the first block)
2 log_write 666
3 log_write 666
4 log_write 34
5 log_write 666
6 log_write 34

1 → allocating a zeroed block
2 → Zeroing the entire block
3 → writing data to inode.
4 → copy in memory inode to disk
5 → write data to inode
6 → Copy in memory inode to disk

Two call exist because first old block is allocated then a is written. This can be seen as after first inode → size is zero but after second it is 1.

rm a

1 log_write 59.
2 log_write 34
3 log_write 58.
4 log_write 34
5 log_write 34

1 - writing data to inode.
2 - Copy in memory inode to disk
3 - free a disk block.
4 → 5 → modified in memory inode to disk.

Assignment : ZCAV

Physical laptop disk : HP X360, i5 8th Gen
TOSHIBA MQ04AB100 1TB HDD
heads → 16
Cylinders → 120ns
physical sector size → 4096

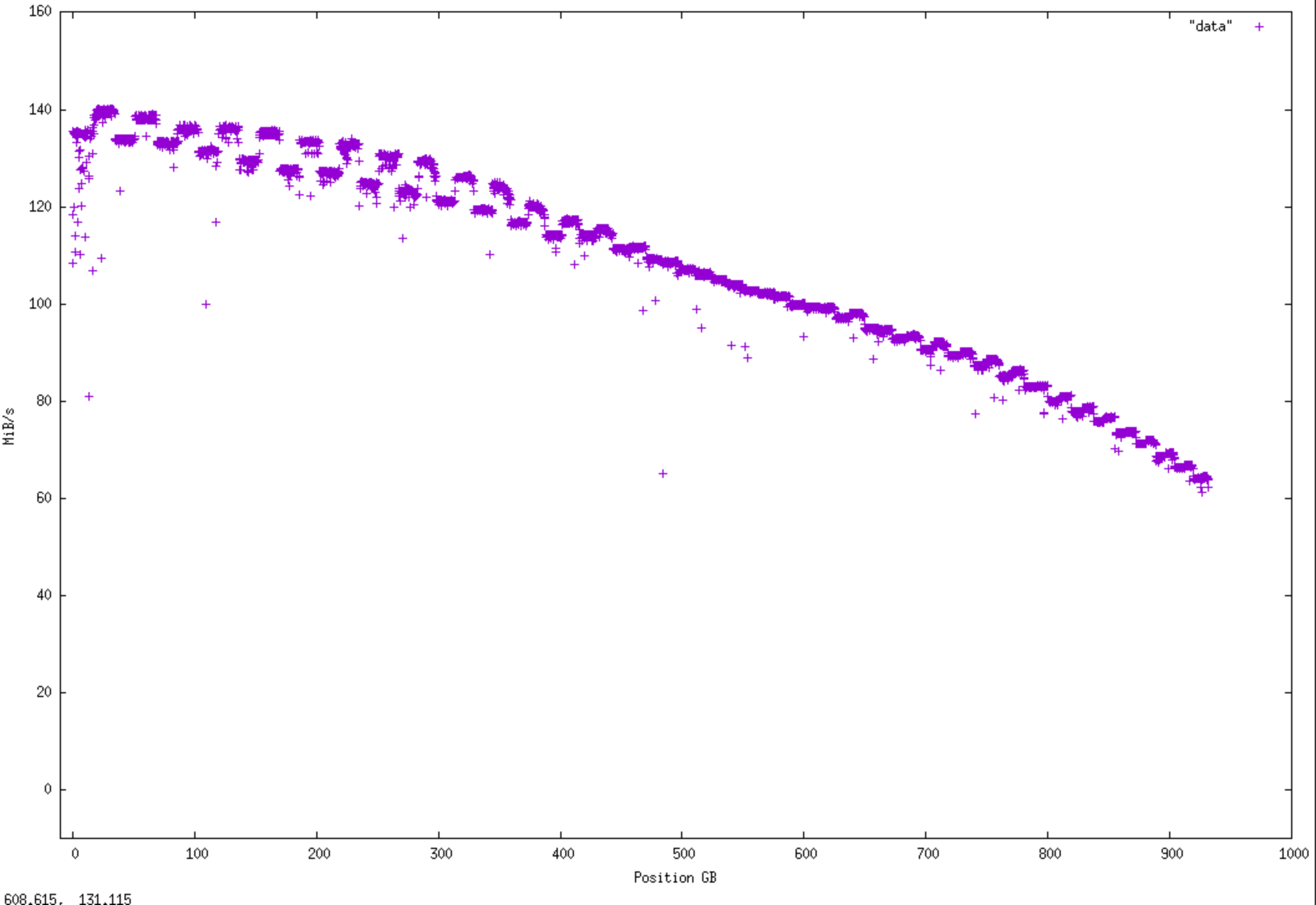
USB drive
Sandisk glide 16GB

Physical laptop disk → average throughput data → 97.5 MB/second.
9286B in 9511 seconds and 9825 seconds.

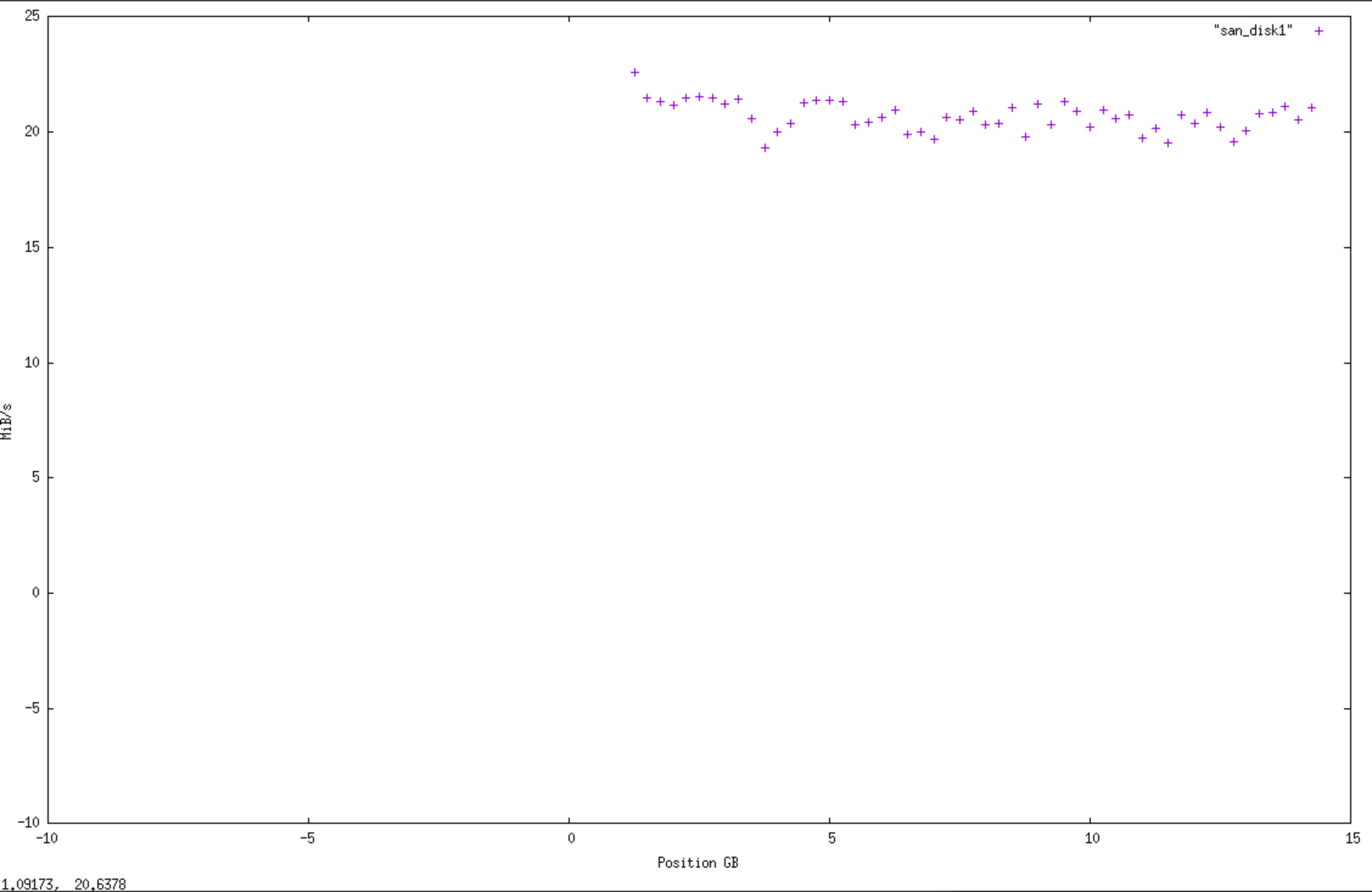
Laptop disk
The zones in hard disks are around 20. In this HDD more than that around 30 are observed ~~which~~ I.e. could be an error due to less number of loops scan.
The max is around 140 mb/s (throughput)
and min around 70 mb/s (throughput)
a drop of 50%.

USB Disk
It has no moving parts and zones. So an almost constant throughput is observed of around 20 MB/s.

The plots are attached the one with label "data", is for HDD and the second one for USB.



608.615, 131.115



1.09173, 20.6378