COL 331 Homework 4

N TARUN SAI GANESH
2016EE10438

Spin locking:

Q) What would happen if the following code snippet is executed?

```
struct spinlock lk;
init lock(&lk, "test lock");
acquire(&lk);
acquire(&lk);
```

Ans) Inside acquire() fn, it checks whether the lock is already held by this CPU or not. So, there will be a Panic.

Q) For ide_lock, why did the kernel panic when interrupts are enabled?

Ans) If we set interrupts (using sti()) after acquiring ~~the~~ lock, then interrupts can come in the middle of critical section & can leave the system in an inconsistent state. For example, when interrupt occurs and the new process also acquired the same lock, then this will violate the condition that only one thread can hold a lock at a time.

Q) Why do file_table_lock and ide_lock have different behaviour?

Ans) Kernel did not panic in the file_table_lock case. The difference in behaviour might be due to difference in the lengths of critical sections in the two cases.

In the ide_lock case, the critical section takes much more time to complete because it waits for the disk request to complete. Hence, there is a high chance of an interrupt occuring in between (when interrupts are enabled).

The critical section in the file table case is much smaller. Hence, there is a very less chance that can interrupt will occur in between. the critical section.

Hence, the kernel has paniked in ide lock but not for file table lock

N TARUN SAI GANESH
2016EE10438

Q) Why does release() clear lk->pcs[0] and lk->cpu before and not after?

Ans) This is because, if we first set lk-> locked=0, there is a chance that another process can acquire the lock. This leaves the lock in an inconsistent state wherein the lock is held by one CPU but details stored in the lock are that of the previous CPU. Thus the info we get when we call 'getcallerpcs()' fn would be incorrect. Thus, first the cpu info should be cleared before setting lock to 0.

Uniprocessor locking:

Q) Does this implementation of locks work on a uniprocessor?

```
lock(L){
  cli();
  while (L==0) continue;
  L=0;
  sti();
}
```

Ans) No this will not work because in case another has already held the lock, then this thread will keep on spinning forever. And since interrupts are disabled, there is no way the other thread gets to run and release the lock.

Q) Does this implementation of locks work on uniprocessor?

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

Ans) This implementation will work because after each iteration, interrupts are enabled again. So, there is a chance of this thread preempting and the thread which has held the lock can get to run. Thus, there would be no deadlock.

N TARUN SAI GANESH
2016EE10438

Sleep and wakeup:

Q) Can both producer (pcqwrite) and consumer (pcqread) sleep on the same channel?

Ans) yes, this is correct and both producer and consumer can sleep on the same channel. But sleeping on a single channel is inefficient because wakeup(q) would wake up all the producer and consumer threads sleeping unnecessarily. ~~There~~ one of these threads would acquire the lock and all the remaining threads would sleep again. This is an overhead. But implementation wise, this is correct. Unrelated part of code which has access to queue can wake up the producers/consumers sleeping on the queue (q).

XV6 file system:

```
$ echo > a
output: log_write 34
        log-write 34
        log-write 59
```

From observation, we can infer the following:

Block no 34 → inode of file a.
59 → data block of parent (directory).
58 → Bitmap block of file a.
571 → data block of file a.

For creating file, the following operations occur:
- Creating inode (34) and setting its values such as type, size, nlink etc.
- Add entry for 'a' in parent directory (59) of file a.

```
$ echo x > a
output: log_write 58
        log_write 571
        log_write 571
        log_write 34
        log_write 571
        log_write 34
```

For writing to a file, following operations occur:
- allocate a block and write to bitmap (58)
- zero out block contents and write "x" to block. (571)
- update a's INODE (34)

```
$ rm a
output:
  logwrite 59
  log-write 34
  log-write 58
  log-write 34
  log-write 34
```

For removing the file, operations occurring are:
- writing zero to a's parent directory record (59)
- update a's INODE values and set it to free (34)
- update free bit in the bitmap for a's datablocks (58)

N TARUN SAI GANESH
2016EE10438

ZCAV:

Laptop model: DELL PRECISION 7510.

File system of HDD : NTFS

Published disk characteristics — model: HGST HTS721010A9E630

Capacity = 1 TB

RPM = 7200

6 GB/s SATA interface

Avg read speed obtained from ZCAV experiment ≃ 110 Mbps

Maximum Read Speed obtained = 143 Mbps

Minimum Read speed obtained = 70 Mbps

[Please find the plot attached]

USB drive model: STRONTIUM POLLEX 32GB

File system = FAT32

Read rate (published) = 25mbps

Avg read speed obtained from ZCAV experiment ≃ 22 Mbps

maximum read speed obtained = 23.26 Mbps

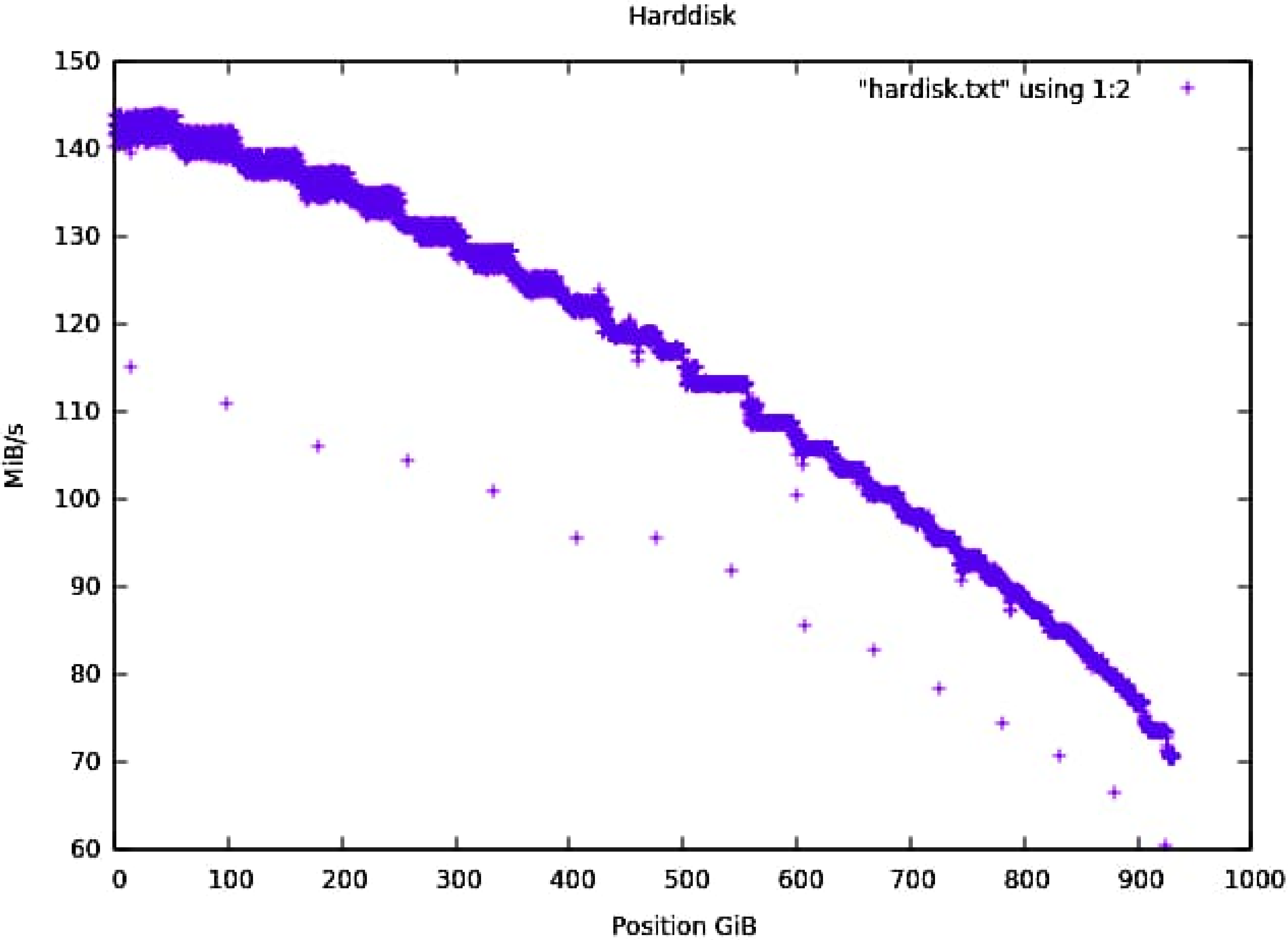
minimum read speed obtained = 20.3 Mbps

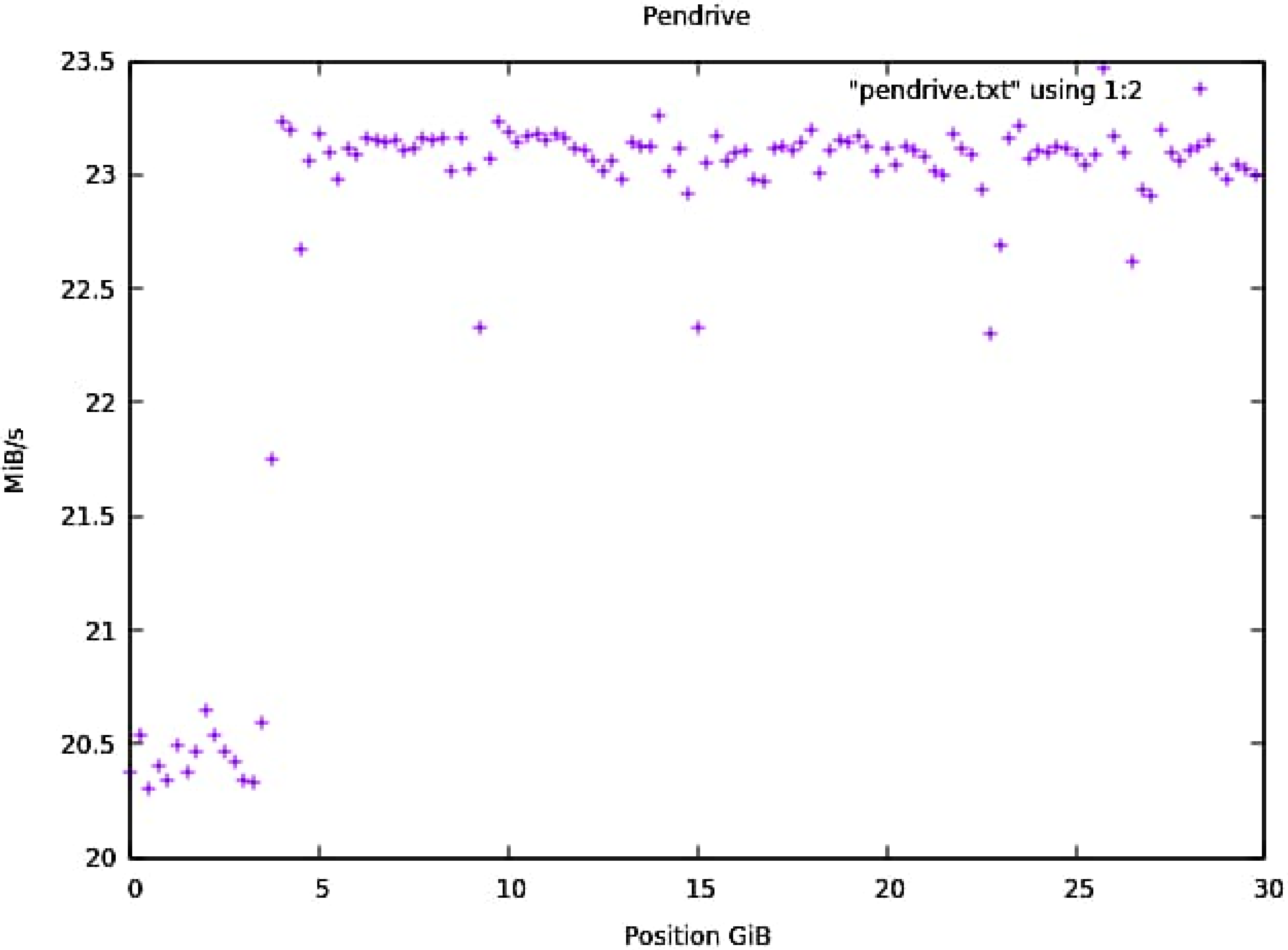
[Please find the plot attached]

For hardisk: Number of Zones found
= no of steps in the plot ≃ 25

From the graph, it is clear that lower block nos have the highest rate, indicating they are mapped on the outermost tracks and higher block nos are on the inner tracks.

For USB: Rate is almost uniform because there no tracks present There are no rotating parts.


Harddisk
"hardisk.txt" using 1:2
MiB/s
150
140
130
120
110
100
90
80
70
60
0
100
200
300
400
500
600
700
800
900
1000
Position GiB


Pendrive
"pendrive.txt" using 1:2
MiB/s
23.5
23
22.5
22
21.5
21
20.5
20
0
5
10
15
20
25
30
Position GiB