# TDT4225 Assigment 2

Exercise 1 – I/O Programming

Code:

import os

import array

import time

BLOCKSIZE = 8192

NBLOCKS = 131072

# Define a array of 8192 bits

BLOCK = array.array('b', [1 for x in range(BLOCKSIZE - 64)])

def pretty(ms, size):

'''

This method pretty prints the output

'''

megabyte = size \* 1024

throughput = megabyte / (ms / 1000)

throughput = '{0:.2f}'.format(throughput)

ms = '{0:.2f}'.format(ms)

print('{} GB\t{} MB/s\t{} ms'.format(size, throughput, ms))

print('{}\t{}\t{}'.format('EXT4', 'Throughput', 'Time'))

for size in [1, 2, 4, 8, 16, 32]:

# Open the file in binary form

with open('test{}'.format(size), 'wb') as f:

start = time.clock()

for i in range(NBLOCKS \* size):

f.write(BLOCK)

f.flush()

os.system('sync')

#os.fsync(f)

stop = time.clock()

interval = stop - start

pretty(interval\*1000, size)

The file is also uploaded to its learning.

## Result:

Device: iMac

OS: Mac OS X 10.10

Filesystem: HFS+

Disk: SSD

HFS+ Throughput Time

1 GB 679.58 MB/s 1506.82 ms

2 GB 681.96 MB/s 3003.11 ms

4 GB 683.43 MB/s 5993.29 ms

8 GB 655.67 MB/s 12494.10 ms

16 GB 663.56 MB/s 24691.05 ms

32 GB 658.44 MB/s 49765.79 ms

Device: Lenovo X220i

OS: Debian 8 Jessie

Filesystem: EXT4

Disk: HDD

EXT4 Throughput Time

1 GB 553.62 MB/s 1849.63 ms

2 GB 466.81 MB/s 4387.19 ms

4 GB 422.20 MB/s 9701.66 ms

8 GB 411.10 MB/s 19926.89 ms

16 GB 377.90 MB/s 43355.18 ms

32 GB 368.84 MB/s 88840.23 m

Device: VM

OS: Debian 8 Jessie

Filesystem: EXT4

Disk: HDD 10k RPM in Raid 50

EXT4 Throughput Time

1 GB 533.68 MB/s 1918.76 ms

2 GB 530.99 MB/s 3856.94 ms

4 GB 521.46 MB/s 7854.93 ms

This server only has 10GB of space, with 2GB occupied.

I think the implementation is correct, and I have verified the size of each file to be the correct size. I personally think that it may seem a little to fast, but it may be that the program is only writing 1s.

Exercise 2 – Various question on file systems

a)

-

b)

c)

d)

e)

f)

RAID is a technique used to split data over multiple storage devices. It can both increase performance or redundancy and sometimes both.

RAID 0

Data is spliced evenly across the drives in the RAID set. There is no redundancy and the performance is usually increased. If one drive fails, all data is lost.

RAID 1

Mirrors all data to every drive in the RAID set. There is full redundancy but the performance may suffer. RAID1 is usually fast at reads as it can be done by either drive. Write speed is usually equal to the write rate of the slowest disk.

RAID 2

Stripes the data at the bit level. The disks are generally syncronised in a way that makes the associated data reside on the same place on each disk. RAID 2 cannot generally service multiple requests simultaneously, but can hit extremely high transfer rates as data from disks that belong together can be read at the same time.

RAID 3

Uses a dedicated parity disk for error correction. The data is striping on the byte-level.

As in RAID 2, this implementation collects data and synchronize the hard drive so that long sequential reads and writes can go very fast.

RAID 4

Uses a dedicated parity disk. Uses block-level striping.

RAID 5

Mostly replaces RAID 3 and 4.

It uses block-level striping, but the parity is distributed over all the disks. Upon failure, the reads can be calculated based on the parity so no data is lost.

RAID 6

Extends RAID 5 with another parity block

RAID 10 (1 + 0)

Combination of RAID 1 and RAID 0

RAID 50 (5 + 0)

Combination of RAID 5 and RAID 0

RAID 60 (6 + 0)

Combination of RAID 6 and RAID 0