

## **Linnaeus University**

Faculty of Technology – Department of Computer Science

### **1DV512 – Operating Systems Individual Assignment 3**

**Assignment Number: 3**

**Student Name: Mohammadali Rashidfarokhi**

**Student Email: mr223jp@student.lnu.se**

**Course Code: 1DV512**

**Semester: Autumn 2020**



# Operating Systems Individual Assignment3

## **Task 1: BTRFS and Related Systems**

### **1. Did some of the earlier file systems supported by Linux before BTRFS focus on support for larger volumes of stored data? If yes, which ones?**

Indeed, some of the earlier file systems are supported by Linux before BTRFS has aimed to focus on support for large volumes of saved information. Two well-known file system which will correspond to the above idea are EXT4 and XFS. Regarding EXT4 which is known as the fourth extended filesystem, is considered a mainly backward-compatible extension to the previous general intention Linux filesystem. However, Ex3 was manufactured to address filesystem and file size limitations. It also has helped to improve performance.

These days, EXT4 will be referred to as the default Linux filesystem for some common Linux allocations.

Regarding the XFS, it is referred to as a filesystem that has been initially developed by SGI. The creation process has started in 1993, for the IRIX operating system. By the year 2000, it was ported to Linux and became available on GNU/Linux distributions. The ideal of designing XFS was to achieve high scalability in terms of IO threads, the number of disks along filesystem size.

### **2. Does BTRFS share any ideas and design goals with ZFS? If yes, which ones?**

Although there are many conflictions between ZFS and BTRFS, there are sharing some common ideas. Consequently, both ZFS and BTRFS are based on COW. They will take advantage of COW for causing support for snapshots and clones. Moving on, ZFS and BTRFS will use checksum that are considered 256 bits long on both metadata and data to propose integrity and support multiple checksumming algorithms. When it comes to addressing the scaling requirements of modern systems, both ZFS and BTRFS are designed based on a 128-bit architecture. What I mean by this is that the block addresses are 128 bits long. As a result, it will add support for extremely large files, in terms of million terabytes (TB) or zettabytes (ZB). An interesting point about both file systems is that they have multiple device support integrated through the usage of software redundant Array of low-cost disks (RAID). The concept of RAID is a way of distributing information over multiple hard disks, for achieving better performance and redundancy. Regarding the existence of the multiple RAID, most of them are supported by ZFS and BTRFS.

Lastly, both file systems will support data compression for increasing throughput.

### **3. What is the "extent" concept used for storage by BTRFS? What are its pros and cons compared to the alternative approaches?**

When it comes to the large files, they are saved to the extent. What I mean by this is that larger files are contiguous on-disk areas that keep user information without extra headers or formatting. An extent-item records a generation number recording when the extent was developed, a pair to record the area on disk, and a pair specifically for the recording the file area. Moving on, an extent is considered a mapping from a logical area in a file to a physical area on a disk. In case of being saved (a file) in a small number of large extents, a common operation such as a full file read will be efficiently linked to a few disk processes.

Apart from the mapping, BTRFS will bring about extent-based file storage with a maximum file size of 50 TB and at the same time a maximum file system with a size of 50 TB. Consequently, the entire information and metadata are copy-on-write. That is, blocks of information will not change on the disk. On the other hand, BTRFS will only copy the blocks and then writes out the copies to a different location.

Regarding the BTRFS, one of the main pros is that extents are used instead of blocks. This will remove the demand for special block sizes. On the contrary, if the system is left with an extremely fragmented free space, writes to disk are still going to be inefficient.

Also, some filesystems use fixed-size blocks instead of extents. By using an extended representation, will result in better space efficiency. On the other hand, this will result in complexity.

### **4. Which structures does BTRFS use to support larger storage devices and map between physical and logical storage space?**

When it comes to the file structure of the b-tree, it will be used to save the greater file. The greater files are saved in extents. They are known as contiguous on disk areas that are responsible for keeping the user data. Therefore, no need to worry about extra headers or formatting. The usability of an extent-item is to records a generation number recording when the extent was manufactured, a [disk block, length] pair to record the region on disk, and a pair [logical file offset, length] to preserve the file area. What I mean by this is that, apart from holding the data and block numbers, extents will record the extents on disk enabling insertion in the center of an extent without reading that information previously. In case of storing a file in a small number of large extents, an operation (common) such as a full file will be well mapped to a few disk operations. As a result, the concept of attractiveness of extents will become clear.

## **5. Does BTRFS support copy-on-write? If yes, is it possible to disable it? Could such an approach have impact on performance and fragmentation?**

BTRFS – (B-tree file system) is considered a copy on the write file system for Linux. Copy on write which can be shortened to COW is an optimization technique for maintaining a copy of a collection of information, taking care of resources when multiple tasks are using the same data.

The idea of examining a file system is to bring about more efficient space management and better data integrity features to Linux.

BTRFS is known for implementing advanced features whilst maintaining wrong tolerance, and reliability. The files system has been under development since 2007 and its codebase has not touched a level of wisdom for not changing its disk format. All in all, BTRFS will support copy-on-write.

Regarding disabling the BTRFS, it is possible to disable it. What I mean by this is that the `nocow` option has the ability to be set on an entire file system or a particular file. That is, it cancels copy-on-write for data blocks if there is not a snapshot. However, COW still applies to metadata blocks. It is logical to use `NOCOW` for workloads where COW would be costly. Also, disabling COW will result in the same ballpark as the other filesystems. In addition, OLTP had one run with very low ops/s, which caused a relative standard error of 50%.

When it comes to the impact, the normal BTRFS policy would write the blocks to disk in update order. Therefore, it will result in an extremely bad performance in the sequential scan. Also, for heavy database workloads, the `nodatacow` mount option will be possible per file in the kernel.

## **6. Do changes to the file system immediately get written to the disk, when using BTRFS? How does this proceed?**

The updates in the BTRFS process will be collected in memory. Also, it will be written to disk at sync time. As a result, it will allow batching optimizations like delayed allocation and collocating data from the same file. Due to the reason that the sync timer usually lasts 30 seconds, and memory buffer space is not without any boundaries. Moving on, blocks of information will not be changed on disk. That is, BTRFS will just make a copy of the blocks before writing down the copied file to another location. If the first location of BTRFS has not been updated, it can eventually result in the removal of the risks which can cause information corruption in case of power failure.

Moving on, the policy of a usual BTRFS will write the blocks to disk in update order. Consequently, it will result in an extreme performance in the sequential scan. Also, the

copy-on-write in BTRFS will enable some of the options of the file system that are known as replication, backup, and saving of data.

Lastly, a snapshot is known as the second version of the complete BTRFS subvolume that has been used for a specific point in time. An interesting fact about subvolume trees is that they can be snapshotted and cloned. Consequently, they are ref-counted. However, the rest of the trees will keep metadata per disk range, and they will under no circumstances be snapshotted. In other words, reference counting is meaningless to them.

## **7. Did Rodeh et al. discover large differences in performance between BTRFS and ZFS in their comparisons? How/why?**

A key point about the ZFS is that it has ports to Linux. However, it is not considered a native Linux filesystem. Therefore, it will prevent a real apple to apple comparison.

Although there are some similarities between ZFS and BTRFS which I have already mentioned, there are some undeniable differences between them as well.

### **❖ Goals**

ZFS is planned for an enterprise-class Solaris server. On the other hand, BTRFS is planned to run on entire things that are accessible on Linux.

### **❖ Filesystem check fsck.**

ZFS does not have access to the fsck utility. That is, the RAID along with the checksum algorithms are not suitable enough, the ZFS user can expect to retrieve from an external backup. However, Linux can be installed on many desktops and smartphones that are not supported by enterprise-grade backups. Therefore, Linux users can expect FSCK utilities. Since writing an adequate recovery tool for BTRFS is sophisticated, there will be no fixed location for metadata.

### **❖ Blocks vs extents.**

ZFS will take advantage of blocks that power two (4KB, 8KB, 128KB). A single object has an unchangeable block size. The merit which RAID-Z has is that it works perfectly with unchangeable block sizes. On the other hand, due to the existence of various block sizes, running out of a size. Therefore, a procedure called ganging in which larger blocks will be developed from smaller blocks. When it comes to the BTRFS, extents will be used rather than blocks. Automatically, the need got special block sizes will be removed. However, if the system is left with a highly fragmented free space, writes to the disk will still be considered inadequate.

#### ❖ Checksums.

ZFS and BTRFS can calculate the checksums. However, ZFS saves them in the block-pointers. On the other hand, BTRFS saves data checksums in a different tree. As a result, for BTRFS this action is vital because extents could be very large, and the desire for validating each page separately is high.

#### ❖ Snapshots vs clones.

ZFS will use birth times to develop snapshots. On the other hand, BTRFS uses the reference-counting mechanism instead. Although the outcome is s close to each other, the BTRFS mechanism is considered more general. Therefore, it will support clones as initial class citizens.

#### ❖ RAID.

ZFS will use RAID-Z and it can support several RAID levels involving single/dual parity. BTRFS can use something closer to a standard RAID layout. However, the supporting option is only available mirroring and striping.

#### ❖ Back-References.

ZFS does not in need of backup from back-references, while BTRFS does. ZFS is expecting to be able to run in a big server with many disks, and it can be thought of that redundancy through RAID-Z is implemented. What it means is that ZFS can recover inadequate blocks through reconstruction. Consequently, BTRFS can not be thought of as underlying RAID and multiple devices. Therefore, it is required to agree to the terms with a bad block by other tools.

#### ❖ Deduplication.

ZFS can support the deduplication, but it will result in an undeniable memory cost. On the contrary, BTRFS does not have this ability currently. Based on the memory requirements, this can be referred to as the only fit for high-end servers.

## **8. Which workloads did Rodeh et al. use for their benchmark tests with FileBench? How did BTRFS compare to the more mature file systems in these tests?**

Rodeh has used 4 workloads for his benchmark tests along with FileBench.

- 1. The Web workload**
- 2. The File workload**
- 3. The mail workload**
- 4. The OLTP workload**

Every single workload was run five times against the HDD and five times against the SSD. When it comes to the execution of each experiment, it has lasted ten minutes. Consequently, it has been reported on the median op/s, median CPU/op, and the relative standard fault. As a result, the outcome was fairly consistent. A problem has been encountered which was related to the OLTP. As a result, occasionally reported very low ops/s.

The web workload imitates a Web server that is responsible for serving specific files to HTTP end-users. It will take advantage of 100 threads that have the power to read all the files sequentially, assuming that they were web pages. However, the threads in a 1:10 ratio, attached to a common file, simulation a weblog.

The mail workload can simulate an electronic mail server. Also, it develops a flat directory structure containing many small files. As a result, it will manufacture 100 threads that can emulate e-mail operations. What it means is that it can read mail, compare, and eliminate them. The result is the translation and breaking down into many small file and metadata actions.

The file workload can imitate a server that will host home directories of multiple consumers. A thread can emulate a user. Consequently, it can be thought of for giving access to special files (Only the files and directories in the unique mentioned home directory).

The concept of every single thread is performing a sequence of developing, attaching, reading, writing, and stat actions.

The OLTP workload also can emulate a database that can operate online transaction processing.

It has a strongly used IO model used by Oracle TM 9i for its foundation. Moving on, it had developed 200 data reader threads, 10 data writer threads, and a single log thread. The readers can proceed with small random reads. That is, the writer can perform tiny random writes. The log thread can write 256KB log writes.

To sum up, the above workload was no means exercises all the different methods someone can use a file system. The idea of mentioning them was to show that they are representative of the methods that most of the file systems are used. BTRFS was in the

same ballpark as its more established. However, a few low fsync performances were problematic and it has been overcome by addressing it while writing.

Workload	Avg. file size	#files	Footprint	IO size (R/W)	# threads	R/W ratio
web	32KB	350000	11GB	1MB/16KB	100	10:1
file	256KB	50000	12.5GB	1MB/16KB	50	1:2
mail	16KB	800000	12.5GB	1MB/16KB	100	1:1
OLTP	1GB	10	10GB	2KB/2KB	200+10	20:1

Fig. 32. Summary of workload parameters.

Workload	Filesystem	median ops/s	relerr(ratio)	median cpu/op
file	btrfs	432	0.00	579us
	ext4	396	0.07	356us
	xfs	231	0.00	488us
mail	btrfs	132	0.01	903us
	ext4	613	0.02	464us
	xfs	195	0.02	619us
oltp	btrfs	229	0.01	27066us
	btrfs_nocow	252	0.00	24469us
	ext4	209	0.00	30821us
	xfs	234	0.02	21327us
web	btrfs	370	0.04	796us
	ext4	859	0.00	215us
	xfs	400	0.00	274us

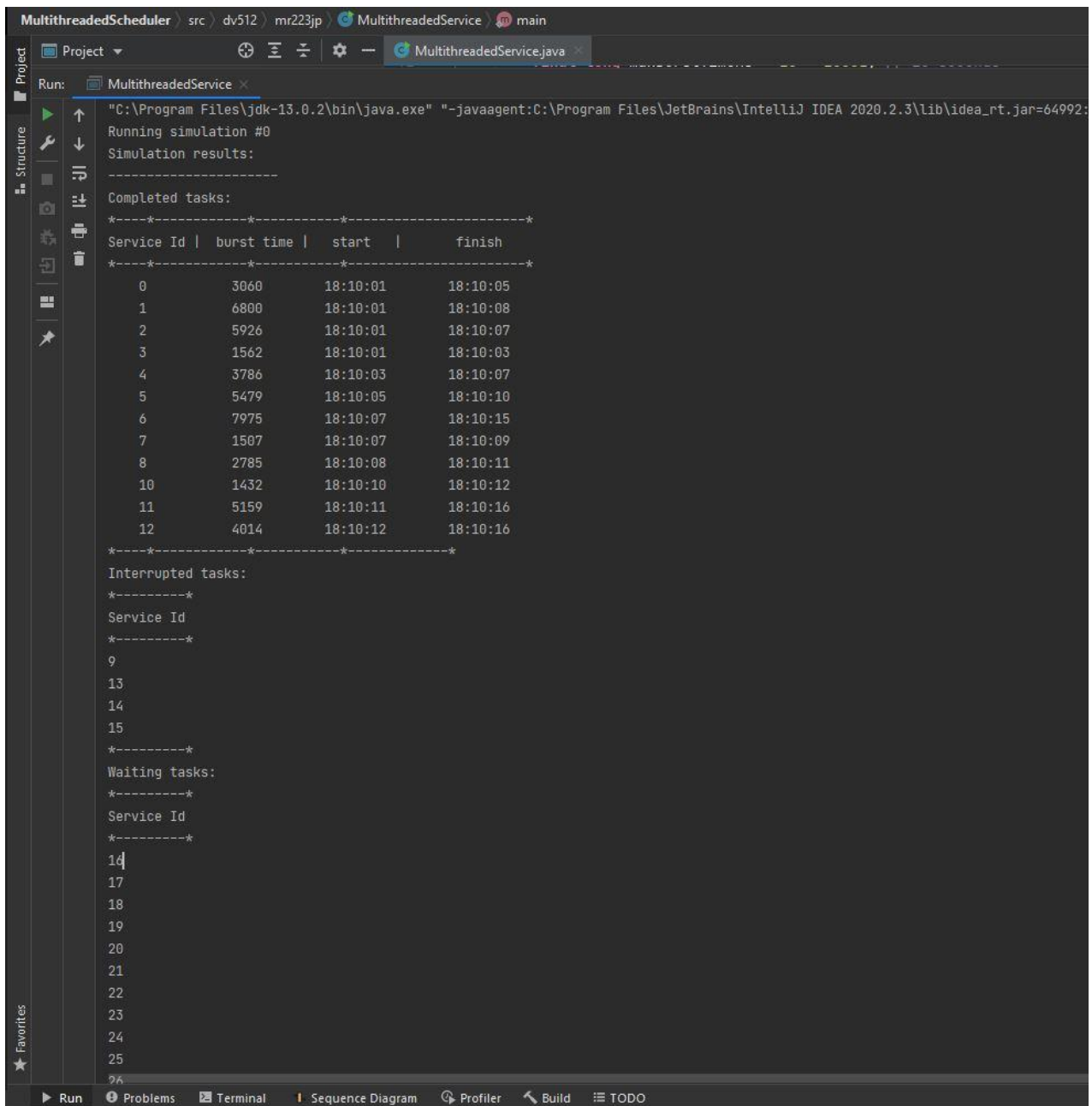
Fig. 33. Hard disk results.

Workload	Filesystem	median ops/s	relerr(ratio)	median cpu/op
file	btrfs	3524	0.00	834us
	ext4	3465	0.00	381us
	xfs	3151	0.00	535us
mail	btrfs	4007	0.02	929us
	ext4	11701	0.11	665us
	xfs	7616	0.03	765us
oltp	btrfs	426	0.50	17897us
	btrfs_nocow	7104	0.02	421us
	ext4	7349	0.00	385us
	xfs	9068	0.01	285us
web	btrfs	14945	0.09	549us
	ext4	17585	0.00	294us
	xfs	17828	0.00	295us

Fig. 34. SSD results.



## Task 2: Developing a Multithreaded Service in Java



```
MultithreadedScheduler > src > dv512 > mr223jp > MultithreadedService > main
Run: MultithreadedService
"C:\Program Files\jdk-13.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\idea_rt.jar=64992:..."
Running simulation #0
Simulation results:
-----
Completed tasks:
*-----*
Service Id | burst time | start | finish
*-----*
0          | 3060       | 18:10:01 | 18:10:05
1          | 6800       | 18:10:01 | 18:10:08
2          | 5926       | 18:10:01 | 18:10:07
3          | 1562       | 18:10:01 | 18:10:03
4          | 3786       | 18:10:03 | 18:10:07
5          | 5479       | 18:10:05 | 18:10:10
6          | 7975       | 18:10:07 | 18:10:15
7          | 1507       | 18:10:07 | 18:10:09
8          | 2785       | 18:10:08 | 18:10:11
10         | 1432       | 18:10:10 | 18:10:12
11         | 5159       | 18:10:11 | 18:10:16
12         | 4014       | 18:10:12 | 18:10:16
*-----*

Interrupted tasks:
*-----*
Service Id
*-----*
9
13
14
15
*-----*

Waiting tasks:
*-----*
Service Id
*-----*
14
17
18
19
20
21
22
23
24
25
26
```

```

*-----*
Interrupted tasks:
*-----*
Service Id
*-----*
9
13
14
15
*-----*
Waiting tasks:
*-----*
Service Id
*-----*
16
17
18
19
20
21
22
23
24
25
26
27
28
29
*-----*

```

MultithreadedService x

↑

↓

⋮

⋮

⋮

⋮

⋮

Running simulation #1

Simulation results:

-----

Completed tasks:

\*-----\*

Service Id	burst time	start	finish
0	9839	18:10:16	18:10:26
1	4890	18:10:16	18:10:21
2	10000	18:10:16	18:10:26
3	6335	18:10:16	18:10:23
4	1298	18:10:21	18:10:23
5	5668	18:10:23	18:10:28
7	3010	18:10:26	18:10:29

\*-----\*

Interrupted tasks:

\*-----\*

Service Id

\*-----\*

6

8

9

10

\*-----\*

Waiting tasks:

\*-----\*

Service Id

\*-----\*

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

```

*-----*
Interrupted tasks:
*-----*
Service Id
*-----*
6
8
9
10
*-----*
Waiting tasks:
*-----*
Service Id
*-----*
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
*-----*

```

```

MultithreadedService x
Running simulation #2
Simulation results:
-----
Completed tasks:
*-----*
Service Id | burst time | start | finish
*-----*
0          7574    18:10:31    18:10:39
1          8102    18:10:31    18:10:40
2          6348    18:10:31    18:10:38
3          3361    18:10:31    18:10:35
4          5011    18:10:35    18:10:40
5          6307    18:10:38    18:10:44
6          5158    18:10:39    18:10:44
7          1164    18:10:40    18:10:41
8          4739    18:10:40    18:10:45
*-----*
Interrupted tasks:
*-----*
Service Id
*-----*
9
10
11
12
*-----*
Waiting tasks:
*-----*
Service Id
*-----*
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

```

```
*-----*
Interrupted tasks:
*-----*
Service Id
*-----*
9
10
11
12
*-----*
Waiting tasks:
*-----*
Service Id
*-----*
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
*-----*

-----
Exiting...

Process finished with exit code 0
|
```

- What strategy have you chosen for implementing tasks and their multithreaded execution? Have you considered any alternatives? How have you handled the termination of threads on simulation finish (when the simulation time runs out)?

Regarding the process, it is considered a dynamic program execution process. What I mean by this is that it has passed over the absolute process. To be more specific, it comes from the code loading and execution until the completion. When it comes to a multi-process operating system, it has the capability of executing several programs at the same time. Due to the reason that the CPU contains a time-sharing mechanism, every single process will gain the ability to get its CPU time slice.

Moving on, due to the reason that the CPU runs very fast, the entire program will look like they are executing at the same time. Regarding multi-threading, it is a process that can make multiple threads in the execution process. A noticeable point about these threads is that they can be present at the same time, execute at the same time.

In java, there are two ways to multithread a thread based on the Oracle. Firstly, the inheritance of a thread class. Secondly, its realization of Runnable. By creating an instance of thread in an application, that application is supposed to deliver the code that is going to be executed in that thread.

There are two ways for accomplishing this:

1. Delivering a runnable object.
2. Subclass Thread. (Oracle.com,2021).

Regarding the Runnable, it has been used in my implementation in the inner class Service.java. To be more specific, Public class Service implements Runnable. Moving on, it is considered a class where it provides the getter and setter for the runnable service. Noticeably, the interface Runnable will implement only one run().

The functionality of the Runnable implementation is only describing the task that is supposed to be performed. What I mean by this is that it will not make any inherent threading capabilities. Therefore, it is required to explicitly attach tasks to the thread.

As it has been mentioned already, the tasks are required to be attached. As a result, the rest (Sleep) action will be controlled by a do-while loop based on the burst time.

- What does ExecutorService do and why it was used in this implementation?

The ExecutorService(interface) is considered to be a generally used subclass. Therefore, it will deliver a tactic to control the life cycle, the ability to send back an upcoming object, and a method that can trace the run state of one or more asynchronous tasks.

When it comes to the executors, they can make several thread pools along with the methods for changing Runnable to Callable. Consequently, the thread pool which will be taken advantage of by this object will save a specific number of threads. Regarding the top load of the thread executing, the time of the task is not when the task is executed. However, following the process of thread creation, after the thread finishes the task, it will not terminate instantly, but the thread will stay active. In case of emerging another task, the live thread will continue working.

The shutdown() method will result in starting once and closing sequentially when all the executions are done. That is, running the previously sent tasks without accepting new ones.

Regarding the awaitTermination() method, it contains two parameters that are timeout period and unit. This method can result in the thread waiting for the timeout time. As soon as the timeout time has been reached, it will check whether the ExecutorService is closed or not. As a result, it will return true if the executorService is closed. Otherwise, false will be returned. This method is used with the shutdown method.

In simple words the tasks will run until  $\text{System.currentTimeMillis()} < \text{System.currentTimeMillis()} + \text{the burst time}$ . When it is done the Boolean done will be "true" which will help us to know which tasks are completed and which are Interrupted. When the while loop is true the Boolean waiting will be "true" which means that this task is "Waiting".

Regarding the calculation of the begin and finish time, it will be done in the run() method in Service.java. The start time will be set when the execution is started and the end time when it is done.

- Based on the simulations that you have witnessed, are all the tasks completed during the simulation time? How many are reported incomplete/interrupted and waiting/non-scheduled?

Based on the provided figures above, in each running simulation, there were some tasks that have been interrupted. Therefore, not all the tasks have been completed.

Running simulation #0: 9,13,14, and 15.

Running simulation #1: 6,8,9, and 10.

Running simulation #2: 9,10,11, and12.

As a result, each simulation had 4 tasks that were reported incomplete. Although the number of the interrupted tasks was not very high, the number of non-scheduled tasks was high.

Running simulation #0: 16,17,18,19,20,21,22,23,24,25,26,27,28, and 29.

Running simulation #1: 11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28, and 29.

Running simulation #2: 13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28, and 29.

Lastly, when it comes to the completed tasks, the majority of the simulation holds the record of finishing 7 to 12 tasks.

- What information has the output of *jstack* provided to you with regard to the threads in your Java program implementation?

The following figures will demonstrate the information that has been obtained from the Jstack. To begin, in figure 1, I have used the command `jcmd -l` to obtain and portrait all the processes that is executing.

```
C:\Users\Ali\IdeaProjects\MultithreadedScheduler>jcmd -l
12724 dv512.mr223jp.MultithreadedService
18584 jdk.jcmd/sun.tools.jcmd.JCmd -l
1868 org.jetbrains.jps.cmdline.Launcher C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\jdom.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\gson-2.8.6.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\nanoxml-2.2.3.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\commons-lang3-3.10.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\maven-repository-metadata-3.6.1.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\platform-api.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\jps-model.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\plugins\java\lib\maven-resolver-connector-basic-1.3.3.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\slf4j-api-1.7.25.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\intellij-deps-fastutil-8.4.1-4.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\maven-resolver-util-1.3.3.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\plexus-interpolation-1.25.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\jps.cmdline.Launcher C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\jdom.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\gson-2.8.6.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\nanoxml-2.2.3.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\commons-lang3-3.10.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\maven-repository-metadata-3.6.1.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\platform-api.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\jps-model.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\plugins\java\lib\maven-resolver-connector-basic-1.3.3.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\slf4j-api-1.7.25.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\intellij-deps-fastutil-8.4.1-4.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\maven-resolver-util-1.3.3.jar;C:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\plexus-interpolation-1.25.jar;C:\Program Files\JetBrains\IntelliJ IDEA
```

Figure 1

```
Terminal: Local x +
C:\Users\Ali\IdeaProjects\MultithreadedScheduler>jstack -l 12724
2021-06-10 20:15:59
Full thread dump Java HotSpot(TM) 64-Bit Server VM (13.0.2+8 mixed mode, sharing):

Threads class SMR info:
  _java_thread_list=0x000001322479a3d0, length=19, elements={
    0x0000013204528000, 0x0000013223b7c800, 0x0000013223b7d800, 0x0000013223b96000,
    0x0000013223b96800, 0x0000013223b9b800, 0x0000013223bac800, 0x0000013223bb5800,
    0x0000013223bc2800, 0x0000013224685000, 0x0000013224685800, 0x0000013224701800,
    0x0000013224702800, 0x0000013224705800, 0x0000013224707800, 0x00000132249e6800,
    0x00000132247bc800, 0x0000013223bd4000, 0x0000013223bd5000
  }

"main" #1 prio=5 os_prio=0 cpu=218.75ms elapsed=19.66s tid=0x0000013204528000 nid=0x2d9c waiting on condition [0x0000012647ff000]
  java.lang.Thread.State: TIMED_WAITING (parking)
    at jdk.internal.misc.Unsafe.park(java.base@13.0.2/Native Method)
    - parking to wait for <0x0000000089fe7250> (a java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject)
    at java.util.concurrent.locks.LockSupport.parkNanos(java.base@13.0.2/LockSupport.java:235)
    at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.awaitNanos(java.base@13.0.2/AbstractQueuedSynchronizer.java:2123)
    at java.util.concurrent.ThreadPoolExecutor.awaitTermination(java.base@13.0.2/ThreadPoolExecutor.java:1454)
    at dv512.mr223jp.MultithreadedService.runNewSimulation(MultithreadedService.java:184)
    at dv512.mr223jp.MultithreadedService.main(MultithreadedService.java:97)

    Locked ownable synchronizers:
      - None

"Reference Handler" #2 daemon prio=10 os_prio=2 cpu=0.08ms elapsed=19.64s tid=0x0000013223b7c800 nid=0x4728 waiting on condition [0x000001264eff000]
  java.lang.Thread.State: RUNNABLE
    at java.lang.ref.Reference.waitForReferencePendingList(java.base@13.0.2/Native Method)
    at java.lang.ref.Reference.processPendingReferences(java.base@13.0.2/Reference.java:241)
    at java.lang.ref.Reference$ReferenceHandler.run(java.base@13.0.2/Reference.java:213)

    Locked ownable synchronizers:
      - None

"Finalizer" #3 daemon prio=8 os_prio=1 cpu=0.08ms elapsed=19.64s tid=0x0000013223b7d800 nid=0x441c in Object.wait() [0x000001264ffe000]
  java.lang.Thread.State: WAITING (on object monitor)
    at java.lang.Object.wait(java.base@13.0.2/Native Method)
    - waiting on <0x000000008a10ac8> (a java.lang.ref.ReferenceQueue$Lock)


```

Figure 2

Regarding figure 2, after using the jcmd -l, the process id will be obtained. Consequently, the process id in my case based on figure 1 is 12724. Moving on, in order to portrait the thread dump, a command line which is jstack -l "process id" will be used.

By taking a look at the output of jstack that has provided me it can be observed that it will both provide the number of the thread and the number of the pool along with their specific information.

Apart from the fact that the Jstack has provided the data about the quantity of the pool and thread, it will present the data of both stack and heap.

```
java.lang.Thread.State: WAITING (on object monitor)
  at java.lang.Object.wait(java.base@13.0.2/Native Method)
  - waiting on <0x000000008a10aec8> (a java.lang.ref.ReferenceQueue$Lock)
  at java.lang.ref.ReferenceQueue.remove(java.base@13.0.2/ReferenceQueue.java:155)
  - locked <0x000000008a10aec8> (a java.lang.ref.ReferenceQueue$Lock)
  at java.lang.ref.ReferenceQueue.remove(java.base@13.0.2/ReferenceQueue.java:176)
  at java.lang.ref.Finalizer$FinalizerThread.run(java.base@13.0.2/Finalizer.java:170)

Locked ownable synchronizers:
  - None

"Signal Dispatcher" #4 daemon prio=9 os_prio=2 cpu=0.00ms elapsed=19.62s tid=0x0000013223b96000 nid=0x3894 runnable [0x0000000000000000]
  java.lang.Thread.State: RUNNABLE

  Locked ownable synchronizers:
    - None

"Attach Listener" #5 daemon prio=5 os_prio=2 cpu=15.63ms elapsed=19.62s tid=0x0000013223b96800 nid=0x27f8 waiting on condition [0x0000000000000000]
  java.lang.Thread.State: RUNNABLE

  Locked ownable synchronizers:
    - None

"C2 CompilerThread0" #6 daemon prio=9 os_prio=2 cpu=234.38ms elapsed=19.62s tid=0x0000013223b9b800 nid=0x45d0 waiting on condition [0x0000000000000000]
  java.lang.Thread.State: RUNNABLE
  No compile task

  Locked ownable synchronizers:
    - None

"C1 CompilerThread0" #9 daemon prio=9 os_prio=2 cpu=203.13ms elapsed=19.62s tid=0x0000013223bac800 nid=0x4d58 waiting on condition [0x0000000000000000]
  java.lang.Thread.State: RUNNABLE
  No compile task

  Locked ownable synchronizers:
    - None

"Sweeper thread" #10 daemon prio=9 os_prio=2 cpu=0.00ms elapsed=19.61s tid=0x0000013223bb5800 nid=0x44ac runnable [0x0000000000000000]
  java.lang.Thread.State: RUNNABLE
```



```

Locked ownable synchronizers:
- None

"Common-Cleaner" #11 daemon prio=8 os_prio=1 cpu=0.00ms elapsed=19.58s tid=0x0000013223b62800 nid=0x4b8c in Object.wait() [0x00000012655ff000]
java.lang.Thread.State: TIMED_WAITING (on object monitor)
  at java.lang.Object.wait(java.base@13.0.2/Native Method)
  - waiting on <0x000000008a1f9748> (a java.lang.ref.ReferenceQueue$Lock)
  at java.lang.ref.ReferenceQueue.remove(java.base@13.0.2/ReferenceQueue.java:155)
  - locked <0x000000008a1f9748> (a java.lang.ref.ReferenceQueue$Lock)
  at jdk.internal.ref.CleanerImpl.run(java.base@13.0.2/CleanerImpl.java:148)
  at java.lang.Thread.run(java.base@13.0.2/Thread.java:830)
  at jdk.internal.misc.InnocuousThread.run(java.base@13.0.2/InnocuousThread.java:134)

Locked ownable synchronizers:
- None

"Monitor Ctrl-Break" #12 daemon prio=5 os_prio=0 cpu=15.63ms elapsed=19.49s tid=0x0000013224683000 nid=0x3694 runnable [0x00000012657fe000]
java.lang.Thread.State: RUNNABLE
  at sun.nio.ch.SocketDispatcher.read0(java.base@13.0.2/Native Method)
  at sun.nio.ch.SocketDispatcher.read(java.base@13.0.2/SocketDispatcher.java:46)
  at sun.nio.ch.NioSocketImpl.tryRead(java.base@13.0.2/NioSocketImpl.java:262)
  at sun.nio.ch.NioSocketImpl.implRead(java.base@13.0.2/NioSocketImpl.java:313)
  at sun.nio.ch.NioSocketImpl.read(java.base@13.0.2/NioSocketImpl.java:351)
  at sun.nio.ch.NioSocketImpl$1.read(java.base@13.0.2/NioSocketImpl.java:802)
  at java.net.Socket$SocketInputStream.read(java.base@13.0.2/Socket.java:937)
  at sun.nio.cs.StreamDecoder.readBytes(java.base@13.0.2/StreamDecoder.java:297)
  at sun.nio.cs.StreamDecoder.implRead(java.base@13.0.2/StreamDecoder.java:339)
  at sun.nio.cs.StreamDecoder.read(java.base@13.0.2/StreamDecoder.java:188)
  - locked <0x0000000089f120e0> (a java.io.InputStreamReader)
  at java.io.InputStreamReader.read(java.base@13.0.2/InputStreamReader.java:185)
  at java.io.BufferedReader.fill(java.base@13.0.2/BufferedReader.java:161)
  at java.io.BufferedReader.readLine(java.base@13.0.2/BufferedReader.java:326)
  - locked <0x0000000089f120e0> (a java.io.InputStreamReader)
  at java.io.BufferedReader.readLine(java.base@13.0.2/BufferedReader.java:392)
  at com.intellij.rt.execution.application.AppMainV2$1.run(AppMainV2.java:47)

Locked ownable synchronizers:
- <0x0000000089f0a8f8> (a java.util.concurrent.locks.ReentrantLock$NonfairSync)

```

```

"Service Thread" #13 daemon prio=9 os_prio=0 cpu=0.00ms elapsed=19.49s tid=0x0000013224685800 nid=0x43a4 runnable [0x0000000000000000]
java.lang.Thread.State: RUNNABLE

Locked ownable synchronizers:
- None

"pool-1-thread-1" #14 prio=5 os_prio=0 cpu=31.25ms elapsed=19.46s tid=0x0000013224701800 nid=0xd28 waiting on condition [0x0000001265aff000]
java.lang.Thread.State: TIMED_WAITING (sleeping)
  at java.lang.Thread.sleep(java.base@13.0.2/Native Method)
  at java.lang.Thread.sleep(java.base@13.0.2/Thread.java:335)
  at java.util.concurrent.TimeUnit.sleep(java.base@13.0.2/TimeUnit.java:446)
  at dv512.mr223jp.MultithreadedService$Service.run(MultithreadedService.java:349)
  at java.util.concurrent.ThreadPoolExecutor.runWorker(java.base@13.0.2/ThreadPoolExecutor.java:1128)
  at java.util.concurrent.ThreadPoolExecutor$Worker.run(java.base@13.0.2/ThreadPoolExecutor.java:628)
  at java.lang.Thread.run(java.base@13.0.2/Thread.java:830)

Locked ownable synchronizers:
- <0x0000000089f8c570> (a java.util.concurrent.ThreadPoolExecutor$Worker)

"pool-1-thread-2" #15 prio=5 os_prio=0 cpu=62.50ms elapsed=19.46s tid=0x0000013224702800 nid=0x3cc4 waiting on condition [0x0000001265bfe000]
java.lang.Thread.State: TIMED_WAITING (sleeping)
  at java.lang.Thread.sleep(java.base@13.0.2/Native Method)
  at java.lang.Thread.sleep(java.base@13.0.2/Thread.java:335)
  at java.util.concurrent.TimeUnit.sleep(java.base@13.0.2/TimeUnit.java:446)
  at dv512.mr223jp.MultithreadedService$Service.run(MultithreadedService.java:349)
  at java.util.concurrent.ThreadPoolExecutor.runWorker(java.base@13.0.2/ThreadPoolExecutor.java:1128)
  at java.util.concurrent.ThreadPoolExecutor$Worker.run(java.base@13.0.2/ThreadPoolExecutor.java:628)
  at java.lang.Thread.run(java.base@13.0.2/Thread.java:830)

Locked ownable synchronizers:
- <0x0000000089f8cbd0> (a java.util.concurrent.ThreadPoolExecutor$Worker)

"pool-1-thread-3" #16 prio=5 os_prio=0 cpu=78.13ms elapsed=19.46s tid=0x0000013224705800 nid=0x4e14 waiting on condition [0x0000001265cff000]
java.lang.Thread.State: TIMED_WAITING (sleeping)
  at java.lang.Thread.sleep(java.base@13.0.2/Native Method)
  at java.lang.Thread.sleep(java.base@13.0.2/Thread.java:335)
  at java.util.concurrent.TimeUnit.sleep(java.base@13.0.2/TimeUnit.java:446)
  at dv512.mr223jp.MultithreadedService$Service.run(MultithreadedService.java:349)
  at java.util.concurrent.ThreadPoolExecutor.runWorker(java.base@13.0.2/ThreadPoolExecutor.java:1128)

```



```

Terminal: Local x +
  at java.util.concurrent.ThreadPoolExecutor$Worker.run(java.base@13.0.2/ThreadPoolExecutor.java:628)
  at java.lang.Thread.run(java.base@13.0.2/Thread.java:830)

Locked ownable synchronizers:
  - <0x0000000089f8ce70> (a java.util.concurrent.ThreadPoolExecutor$Worker)

"pool-1-thread-4" #17 prio=5 os_prio=0 cpu=46.88ms elapsed=19.46s tid=0x0000013224707800 nid=0x3f38 waiting on condition [0x0000001265dff000]
  java.lang.Thread.State: TIMED_WAITING (sleeping)
    at java.lang.Thread.sleep(java.base@13.0.2/Native Method)
    at java.lang.Thread.sleep(java.base@13.0.2/Thread.java:335)
    at java.util.concurrent.TimeUnit.sleep(java.base@13.0.2/TimeUnit.java:446)
    at dv512.mr223jp.MultithreadedService$Service.run(MultithreadedService.java:349)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(java.base@13.0.2/ThreadPoolExecutor.java:1128)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(java.base@13.0.2/ThreadPoolExecutor.java:628)
    at java.lang.Thread.run(java.base@13.0.2/Thread.java:830)

Locked ownable synchronizers:
  - <0x0000000089f8d140> (a java.util.concurrent.ThreadPoolExecutor$Worker)

"pool-2-thread-1" #18 prio=5 os_prio=0 cpu=0.00ms elapsed=4.44s tid=0x00000132249e6800 nid=0x4d84 waiting on condition [0x0000001265ffe000]
  java.lang.Thread.State: TIMED_WAITING (sleeping)
    at java.lang.Thread.sleep(java.base@13.0.2/Native Method)
    at java.lang.Thread.sleep(java.base@13.0.2/Thread.java:335)
    at java.util.concurrent.TimeUnit.sleep(java.base@13.0.2/TimeUnit.java:446)
    at dv512.mr223jp.MultithreadedService$Service.run(MultithreadedService.java:349)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(java.base@13.0.2/ThreadPoolExecutor.java:1128)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(java.base@13.0.2/ThreadPoolExecutor.java:628)
    at java.lang.Thread.run(java.base@13.0.2/Thread.java:830)

Locked ownable synchronizers:
  - <0x0000000089f79a8> (a java.util.concurrent.ThreadPoolExecutor$Worker)

"pool-2-thread-2" #19 prio=5 os_prio=0 cpu=0.00ms elapsed=4.44s tid=0x00000132247bc800 nid=0x3d40 waiting on condition [0x0000001265efe000]
  java.lang.Thread.State: TIMED_WAITING (sleeping)
    at java.lang.Thread.sleep(java.base@13.0.2/Native Method)
    at java.lang.Thread.sleep(java.base@13.0.2/Thread.java:335)
    at java.util.concurrent.TimeUnit.sleep(java.base@13.0.2/TimeUnit.java:446)
    at dv512.mr223jp.MultithreadedService$Service.run(MultithreadedService.java:349)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(java.base@13.0.2/ThreadPoolExecutor.java:1128)

"pool-2-thread-4" #21 prio=5 os_prio=0 cpu=0.00ms elapsed=4.44s tid=0x0000013223bd5000 nid=0x2644 waiting on condition [0x00000012660ff000]
  java.lang.Thread.State: TIMED_WAITING (sleeping)
    at java.lang.Thread.sleep(java.base@13.0.2/Native Method)
    at java.lang.Thread.sleep(java.base@13.0.2/Thread.java:335)
    at java.util.concurrent.TimeUnit.sleep(java.base@13.0.2/TimeUnit.java:446)
    at dv512.mr223jp.MultithreadedService$Service.run(MultithreadedService.java:349)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(java.base@13.0.2/ThreadPoolExecutor.java:1128)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(java.base@13.0.2/ThreadPoolExecutor.java:628)
    at java.lang.Thread.run(java.base@13.0.2/Thread.java:830)

Locked ownable synchronizers:
  - <0x0000000089fe81c8> (a java.util.concurrent.ThreadPoolExecutor$Worker)

"VM Thread" os_prio=2 cpu=15.63ms elapsed=19.64s tid=0x0000013223b79800 nid=0x4e4c runnable
"GC Thread#0" os_prio=2 cpu=15.63ms elapsed=19.65s tid=0x000001320456b800 nid=0xa14 runnable
"G1 Main Marker" os_prio=2 cpu=0.00ms elapsed=19.65s tid=0x000001320457d000 nid=0x1ca4 runnable
"G1 Conc#0" os_prio=2 cpu=0.00ms elapsed=19.65s tid=0x000001320457e800 nid=0x3a64 runnable
"G1 Refine#0" os_prio=2 cpu=0.00ms elapsed=19.65s tid=0x00000132239f6800 nid=0x4bec runnable
"G1 Young RemSet Sampling" os_prio=2 cpu=0.00ms elapsed=19.65s tid=0x00000132239f9000 nid=0x4b0c runnable
"VM Periodic Task Thread" os_prio=2 cpu=0.00ms elapsed=19.49s tid=0x00000132246c7000 nid=0x4ef8 waiting on condition

JNI global refs: 15, weak refs: 0

C:\Users\Ali\IdeaProjects\MultithreadedScheduler>

```

## Reference

- ✓ <https://sakisk.me/files/copy-on-write-based-file-systems.pdf>
- ✓ <https://www.linuxlinks.com/btrfs/>
- ✓ [https://mymoodle.lnu.se/pluginfile.php/6370534/mod\\_assign/introattachment/0/Rodeh%20et%20al.%20-%20BTRFS%20-%20The%20Linux%20B-Tree%20Filesystem%20%282013%29.pdf?forcedownload=1](https://mymoodle.lnu.se/pluginfile.php/6370534/mod_assign/introattachment/0/Rodeh%20et%20al.%20-%20BTRFS%20-%20The%20Linux%20B-Tree%20Filesystem%20%282013%29.pdf?forcedownload=1)