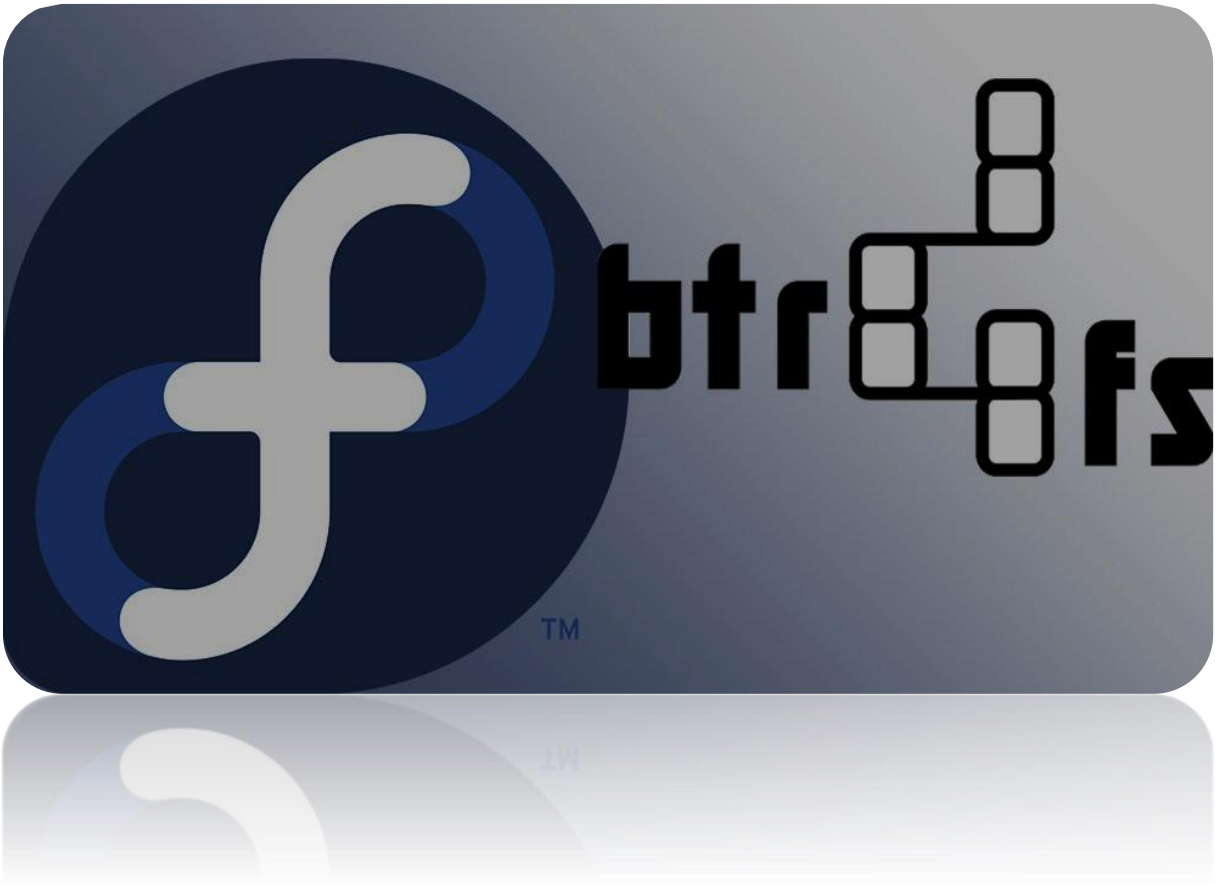
Linnéuniversitetet
Kalmar Växjö

20HT – 1DV512 – Operating Systems
Individual Assignment 2



i

Student: Sami Mwanje

ID/mail: mm223kk@student.lnu.se

Assignment date: 2020-12-06

Hand in date: 2021-06-18

Sami Mwanje
mm223kk@student.lnu.se

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?** There are three main file systems in Linux. BTRFS, XFS and Ext(n). The extended file system (Ext) was first released in 1992. The currently most deployed filesystem on Linux is called is ext4. The older version ext3 was redeveloped with new features and is today replaced with the developed version "Ext4". The ext3 filesystem itself was developed from the ext2 with features as "journaling". Journaling means that every file transaction is recorded to the journal before its written to the storage device. The ext2 now have a journal log which states if a transaction was completed or not, therefore the disk recovery time after a false shutdown can vary between ext2 and ext3.The Fourth extended file system (ext4) is developed from the ext3 with improved file size limitations, performance fixes and much more. The ext4 has a maximum file size up to 16 TB and can hold volumes as large as 1EB. The filesystem XFS was started in 1993 on Linux, compared to ext2 it already contains journalism, and just like BTRFS it uses B-trees. This XFS file system was the first one to use a journal in Linux and was also one of the most memory scalable file system. XFS has a maximum file size up to 9 EB and the largest volume that can be created can be up to 18 EB. There are some other file systems that supported Linux such as LFS, NILFS, WAFL, ReiserFS and ZFS. ZFS that is developed by sun microsystems can have a maximum file size up to 16 EB. WAFL, write anywhere filesystem layout can only have a maximum file size up to 16 TB, but the volume can extend up to 100 TB depending on the platform. All these file systems can today support volumes with sizes over 100 TB, expect to ReiserFS and Ext4 that have the same size limits, which should not be considered small.

2. **Does BTRFS share any ideas and design goals with ZFS? If yes, which ones?** The ZFS file system was developed by sun microsystems in 2001, and first released 2006 in the Solaris operating system. The development goal of this file system was to replace UFS (Unix file system). The UFS file system had some minor issues that developers did not like. It could go from smaller size limits to data corruption without storage device

detecting it and some other features that were considered missing in the UFS. The ZFS come with the use of checksums. The internal checksums exist in blocks, data, and metadata. They are always stored with the pointer of the block that is being check summed. BTRFS was designed to fit all Linux running systems while ZFS was mostly designed for machines running a Solaris server. Checksums that are used to detect error are both available on BTRFS and ZFS, but the storage of these checksums is not the same. ZFS keeps the checksums on block-pointers, while the B-tree structed BTRFS keeps them on a separate tree. RAID is available both on ZFS and BTRFS. ZFS uses a RAID called “RAID-Z”, while BTRFS uses a more standard RAID though it can only support mirroring, striping and “mirror + striping”. The copy-on-write (COW) method is used on both filesystems. More about this mechanism later.

As we can see they both have a familiar goal, and that is a file system with today’s mostly needed futures. Though the goal may be the same they can vary in the way of receiving this goal. For example, snapshots used by ZFS, and clones used by BTRFS.

3. **What is the "extent" concept used for storage by BTRFS? What are its pros and cons compared to the alternative approaches?** Each directory in Linux uses a data structure called B-tree to store to store the index of the file name in each directory. Sometimes there can be very large directories .These large directories have a top level with a pointer. This pointer points to the physical area of the disk. Summed up, an extent maps a file from its logical memory area to the physical memory area on the disk. Big files that are mapped to small number of large extents will be working efficiently when it comes to disk operations. As a result, extent can work as memory space used to handle drawbacks that occur when files grow to very large sizes. In the book by Abraham, it is called “a chunk of contiguous space”. Sometimes the extents can be too large which can cause internal fragmentation. Extents vary in size and are allocated and deallocated while running. This can cause external fragmentation. The BTRFS filesystem uses checksums, which are blocks of data derived from other blocks with the main purpose to detect errors. The risk that extents may grow and obtain a large size is big, therefore the checksums and other blocks must be kept in a separate tree.

4. **Which structures does BTRFS use to support larger storage devices and map between physical and logical storage space?** The BTRFS filesystem uses the Log Structed Merge Trees (LSM-Trees) structure by 0'Neil. As every search tree the LSM tree holds a pair of children on each parent. The LSM opens many possibilities for developers to makes us of its tree-structed structure. The tree is mostly useful for systems that require a high update rate. This can today for example be Facebook, twitter, google, yahoo and so on. The files that are going through the LSM-tree usually goes through two levels starting will level zero. Each level contains so called segments. Which can be seen as arrays containing files. Each file contains a key and a value, this is called a key-value. Each key-value inside the segment-array is sorted by its key. The system will collect more and more segments over time, there for a level of segments will be full and that files must get cleaned up. This cleanup is called compaction. Compaction combines older segments and creates a new level of segments that is a sum of the combined ones. The older the key-value are the larger is the risk that it will get compacted from the current segment to a segment on a newer level. The data written to the segments is stored inside the physical memory until the LSM-tree becomes too large. When the LSM-tree is too large the writes are flushed and then stored sorted by key on the hard drive.

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?** Copy-on-write allows a parent processes and its child to share the same pages (address space). If either one of these process writes to a shred page, a copy of the shared page will be created. This avoids the old pages to be overwritten and can be powerful when it comes to system crashes and so on. On a BTRFS filesystem the user is able to disable the copy-on-write. This can be done with the command: **"$ mv /path/to/dir /path/to/dir_old**

   $ mkdir /path/to/dir

   $ chattr +C /path/to/dir

   $ cp -a /path/to/dir_old/* /path/to/dir

   $ rm -rf /path/to/dir_old" and will disable copy-on-write for all existing files in a directory. There are some other ways to disable CoW depending on a specific file or

directory.[1] For example the so called “nocow” option can be set to cancel CoW for data blocks in a file system. This can be a good option to avoid fragmentation that can occur with copy-on-write.

The copy-on-write operation enables the system to make very fast copies with creating a file. A newly file created file has the same pointer as its parent until it is modified. Therefore, the user will never notice that they are sharing the data of a specific file. This avoids the system to make unnecessary duplicates of multiple files that have no been modified, and therefore the parent call of fork() will be much faster. One drawback can be the management of the overhead tables (maps). This needs to be modified every time a new file is created, so the system knows which files that share the same pages, and which have been modified and not. If the original file (parent) may be deleted the data of which file that has been changed will disappear. The copy-on-write approach have an impact on the performance when a file is modified. This occur though fragmentation due to a modified copy is always written on a newly and fresh disk area. As stated earlier, fragmentation can be avoided by disabling copy-on-write for various blocks.

6. **Do changes to the file system immediately get written to the disk, when using BTRFS? How does this proceed?** There are a lot of things going on in the background of BTRFS when a user modifies a file or a directory. Firstly, the file system will update both the page and the extent of the file/directory. These changes will not be directly stored on the disk. The modifications are first stored in the memory until a 30 second time out, or enough pages have been modified. When this occur, a new checkpoint is written and is pointed at by a superblock. A checkpoint is where updates are written to after they have been accumulated in memory. A superblock is located at a fixed location on the disk and points to a “tree of tree roots”. These “tree of tree roots” are numerous of B-trees and sums together up the files system. When the superblock points to the new checkpoint the disk gets modified. When writing to a file the i-node, file-extents, checksums, and back-references are modified. All these modifications are noted in checkpoints. With all this happening the copy-on-write also occurs if need,

[1] https://wiki.archlinux.org/title/Btrfs#Copy-on-Write_(CoW)

which means that files that are modified with a parent gets a separate memory space on the disk, and copies only get at pointer to the parent's disk space.

7. **Did Rodeh et al. discover large differences in performance between BTRFS and ZFS in their comparisons? How/why?** Rodeh et al. Did not compare the performance of the ZFS filesystem with BTFRS. Rodeh et al. declares that the ZFS filesystem cannot be compared to the BTRFS because of the significant differences it has compared to BTRFS. He also states that it should not be observed as a native Linux filesystem.

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?** In the benchmarks Rodeh et al. used three different storage devices. SSD, hard disk, and multiple disks (RAID10). The benchmarks were performed on the so-called mature filesystems. BTRFS, XFS and Ext4.

   The first benchmark was performed on the hard disks using a so called "Linux Kernel make". The comparisons were surrounding throughput, seek count, and IOPS ((input/output)/s performance). BTRFS had an average lower seek count than XFS and Ext4. The throughput was slightly higher on the Ext4 while BTRFS only had a throughput higher than XFS. BTRFS had a significant spike at the beginning of it seek time, which Rodeh et al. explains as an effect of the copy-on-write. When it came to the IOPS the overall winner here was ext4 while with 54.27 IOPS while BTRFS came on the second place with 44.02 IOPS. In general, the filesystems in this benchmark could be seen as slightly equal with ext4 as the winner.

   Filebench was used to run some more complex workloads to see the performance of the filesystems. These are so called "Macrobenchmarks". The benchmarks were done on 16GB RAM machine with an Intel CoreTM i7-2600 3.40GHz CPU. The CPU had eight cores and the system was limited to 2GB. In the workloads a SATA II, 3Gbit/S, 7200 RPM hard drive, and a 270 MB/s / 205 MB/s (R/W) SSD were used. What a filebench benchmark does is that it locates a filesystem tree and executes the chosen workload. The performance is at the same time measured and reported in as operations per second (ops/s) and CPU per operation (CPU/op). An operation can be seen as a read of a whole file or appending an amount of data to a file. Each disk type

(SDD/HDD) was benchmarked five times for each workload. The inner workloads where web, file, mail and OTLP. Rodeh et al. used the median of these three as a blueprint to the results. The measures were median ops/s, relerr(ratio) and median CPU/op.

On the hard disk the measured values of mail and web showed BTRFS that had the lowest median ops/s. The OLTP values showed that BTRFS had a median value relative to the others and on file BTRFS was dominating with 432 ops/s and XFS on the last place with 231 ops/s. Relerr(ratio) was quite the same on each measure, a little notice on the OLTP measure was that BTRFS had a ratio of 0.01 with cow and 0.00 with nocow. The nocow option that was used in the OLTP disables the shadowing and the checksums for data pages. Another notice was on web where BTRFS showed a ratio value of 0.04 while the other two remained 0. For file ext4 had a ratio of 0.07 that was significantly larger than XFS, and BTRFS which remained 0. Lastly the results of CPU/op showed BTFRFS have the highest us(microseconds) on all measures expect to OLTP where it shared the second place with its own nocow.

The same measurements were done on the SSD where BTRFS dominated every CPU/op measure in microseconds. A large notation here is on the OLTP where BTRFS with nocow disabled got a value that was around 17400us larger than the others (17897us). The ratio showed once again noticeable changes on OLTP with and without nocow. With a value of 0.50 without nocow and 0.02 with cow. The other filesystem maintained around 0.01-0.01. Another noticeable change in ratio was on web where the result was 0.09 compared to the other two that remained 0. On mail Ext4 had a significant spike (0.11) while XFS and BTRFS remained low (0.03 – 0.02). Lastly the median ops/s showed BTRFS with the largest value (3524), Ext4 was not so far from it with 3465 and XFS on the last place with 3151. On mail BTRTS came on the last place (4007) relative to the other that had values twice and three times larger. Ext4 (11701) and XFS (7816).

On OLTP BTRFS had a significantly low value of 426 without nocow. With nocow the value was 7104 which was much closer to the other filesystems 7349 and 9068. The web measurements showed BTRFS on the last place with 14945 ops/s. Ext4 and XFS

had values around 17500. Rodeh et al. stated that BTRFS requires more CPU/cycles. When it came to mail fsync was used which makes BTRFS suffer due to the slow fsync implementation. Fsync performance was the main issue for the BTRFS and when it came to heavy workloads the COW option could be disabled in order of avoiding performance issues. Rodeh et al. stated that the BTRFS is a relatively young Linux filesystem that is currently competing for becoming the main filesystem for Linux. It might become a challenge task due to that the filesystem must work well on a various amount of hardware's, but it absolutely not impossible.

Task 2:

9.
```
C:\Program Files\Java\jdk-16.0.1\bin>
```
First located the "jdk-bin" folder in order to execute **jcmd.exe** and **jstack**.

```
C:\Program Files\Java\jdk-16.0.1\bin>jcmd -l
20528 Eclipse
21436 jdk.jcmd/sun.tools.jcmd.JCmd -l
23548 mm223kk_assignment_3.MultithreadedServiceMain
```
Using "**jcmd –l**" to list all running java processor and their PID.

```
C:\Program Files\Java\jdk-16.0.1\bin>jstack 23548
```
And then using "jstack PID". To run the jstack thread dump. The output is on the screenshots bellow.

I chose to implement the threads using the thread pool class "**ExecutorService**". I loaded the class by using:
"**ExecutorService executor = Executors.newFixedThreadPool(4)"**. The number four indicates how many threads I want to use. I chose this method because I saw it as it most logic one to used based on what the task was about. Since I was building "**task classes**" I could easily attain these "tasks" to threads by using "**executor.submit(new Task);"**. This made the program easy to code and now the only thing that was missing was handling the finale results of the simulation.

To handle the termination of threads of simulation finish I first of all I had to create a timer for the simulation. Here I used the standard "**System.currentTimeMillis()"** returns the current time in Milliseconds. By monitoring the time when simulation starts I could there on take use of it to check if 15 seconds had elapsed. While this check was going on the main program was stuck in a while loop. When there was less than a second left until the 15 seconds, I made sure that the program broke out from the while loop and jumped to a method called "**shutdown();**".

Sami Mwanje
mm223kk@student.lnu.se

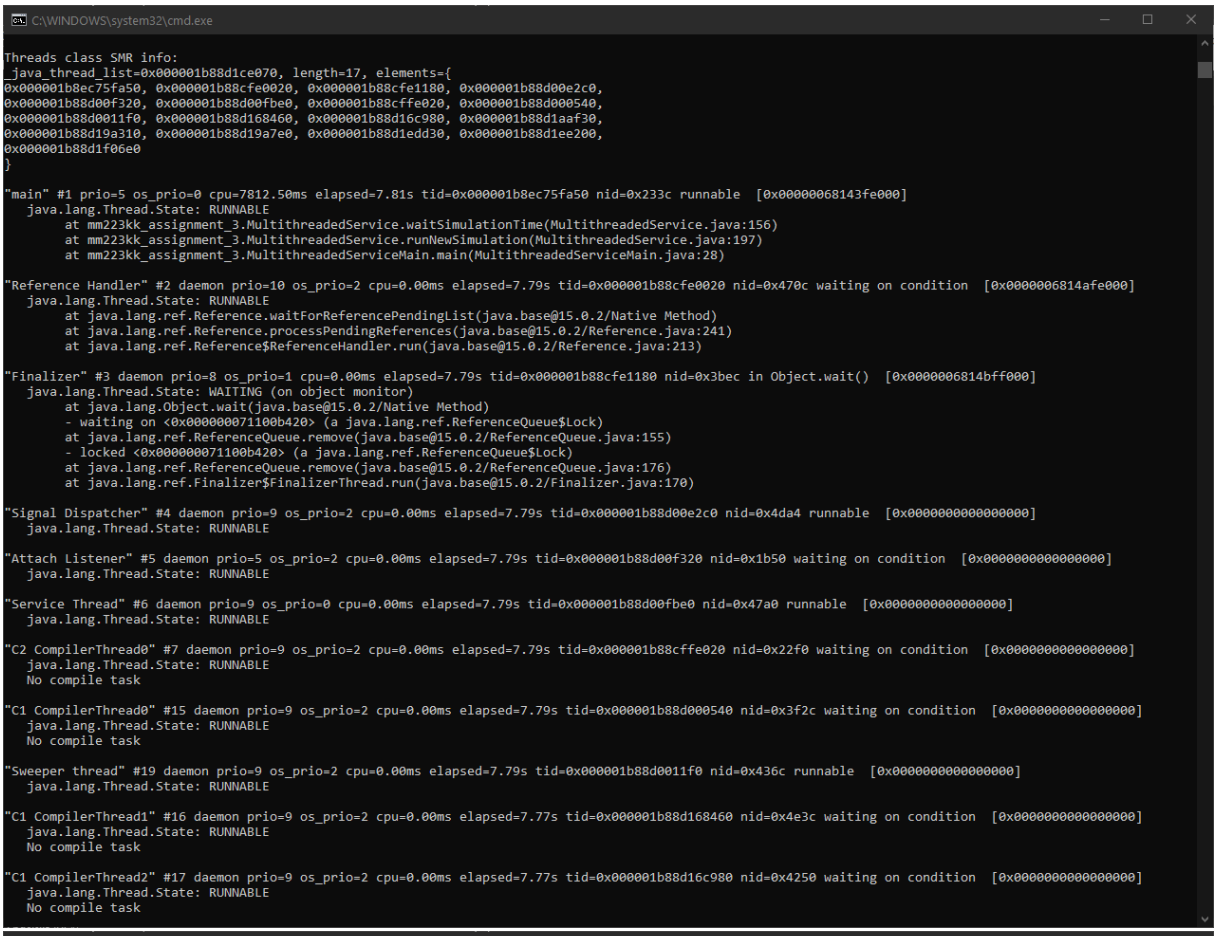
In the "**shutdownExe();**" method I used the lines "**executor.shutdown()"** and "**executor.shutdownNow()". "executor.shutdown()"** is used to signal the thread that no new tasks can be signed to them. When this is called the waiting tasks will never attach to threads. "**executor.shutdownNow()"** is used to stop all tasks the are currently executing tasks and halts the processing of waiting tasks. It is said that it should also return a list of the waiting process, but I did not use this list for my tasks.

Instead, I mostly relied on the "**System.currentTimeMillis()"** to make the calculations for me. Every task had an instance called "**startTime = 0**". This instance was only larger than 0 once a thread has started to "**run()"** the task. So, if the "**startTime"** was 0 when the threads got interrupted, it meant that the task was waiting and had never started running. This "**startTime"** could also be used to make sure that the program only finishes its execution when "**currentTime – startTime >= burstTime**". If the program was done with its execution the instance "**endTime**" noted that time stamp. "**endTime"** also noted the time stamp when the program was interrupted by a dying thread. This could therefore be used to find out if the program were done execution or got interrupted and so on.

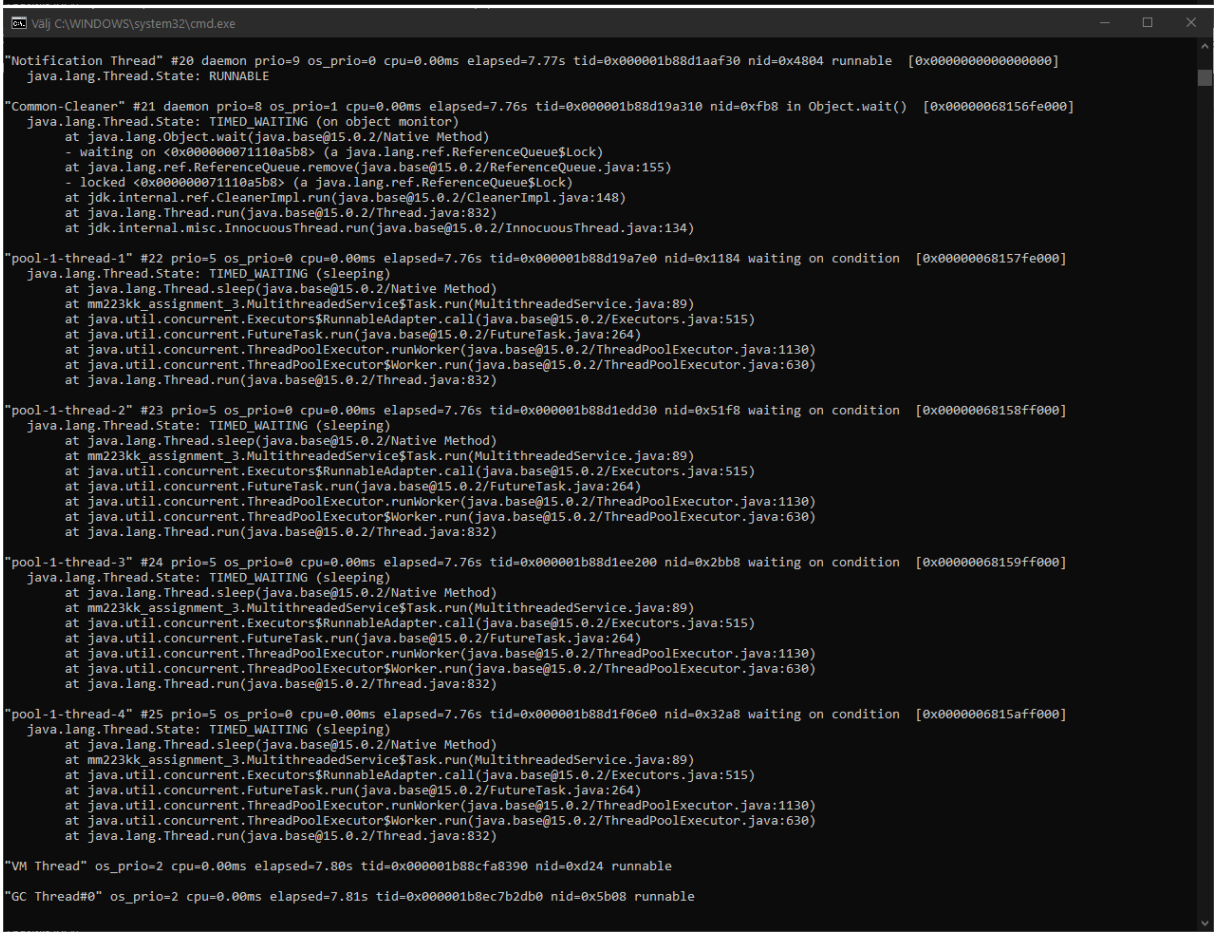
By observing the output of the simulation, I could see that around **9-12** tasks got completed before the program got interrupted. Around **4** tasks got interrupted on each simulation and the **15-30** tasks left never had the chance to run. You can easily notice how a task with a large burst time at the beginning can starve other tasks with lower burst times when it gets loaded to the thread first. Some tasks had a burst time between 1-3 but never got executed because of starvation.

The **jstack** output provided information about number of threads, type, the time they had elapsed for, their cpu priority, state, which classes that were using them and so on. Since I used "**Executors.newFixedThreadPool(4)"** I could see this in the **jstack** output. The output noted that there were 4 threads that were being used to run the program, and that these were so called "pool threads". Even information about threads outside the java program could be seen on the **jstack** output. The other threads that are mentioned in the **jstack** are in some way important in order for the JDK to function and the code to compile. Also, the threads for the "main class" itself are mentioned. Thinking of it I may in some way affected this output by splitting the original code into two classes. A Main class and a thread service class. I am not sure, but this may in some way have an effect on the output.

Sami Mwanje
mm223kk@student.lnu.se

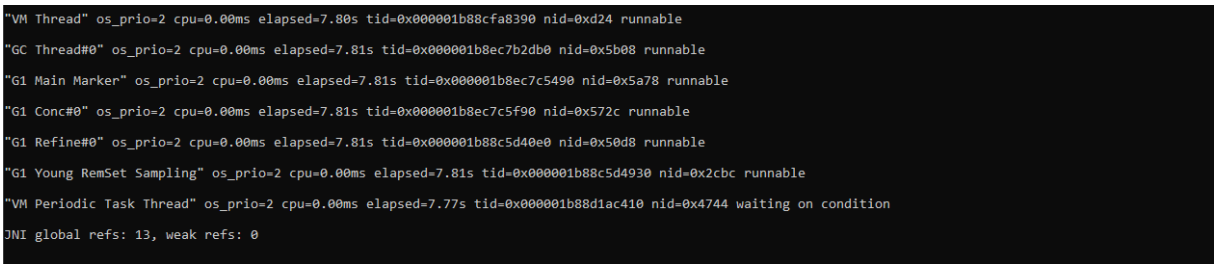


```
C:\WINDOWS\system32\cmd.exe

Threads class SMR info:
_java_thread_list=0x000001b88d1ce070, length=17, elements={
0x000001b8ec75fa50, 0x000001b88cfe0020, 0x000001b88cfe1180, 0x000001b88d00e2c0,
0x000001b88d00f320, 0x000001b88d00fbe0, 0x000001b88cffe020, 0x000001b88d000540,
0x000001b88d0011f0, 0x000001b88d168460, 0x000001b88d16c980, 0x000001b88d1aaf30,
0x000001b88d19a310, 0x000001b88d19a7e0, 0x000001b88d1edd30, 0x000001b88d1ee200,
0x000001b88d1f06e0
}

"main" #1 prio=5 os_prio=0 cpu=7812.50ms elapsed=7.81s tid=0x000001b8ec75fa50 nid=0x233c runnable  [0x00000068143fe000]
   java.lang.Thread.State: RUNNABLE
        at mm223kk_assignment_3.MultithreadedService.waitSimulationTime(MultithreadedService.java:156)
        at mm223kk_assignment_3.MultithreadedService.runNewSimulation(MultithreadedService.java:197)
        at mm223kk_assignment_3.MultithreadedServiceMain.main(MultithreadedServiceMain.java:28)

"Reference Handler" #2 daemon prio=10 os_prio=2 cpu=0.00ms elapsed=7.79s tid=0x000001b88cfe0020 nid=0x470c waiting on condition  [0x0000006814afe000]
   java.lang.Thread.State: RUNNABLE
        at java.lang.ref.Reference.waitForReferencePendingList(java.base@15.0.2/Native Method)
        at java.lang.ref.Reference.processPendingReferences(java.base@15.0.2/Reference.java:241)
        at java.lang.ref.Reference$ReferenceHandler.run(java.base@15.0.2/Reference.java:213)

"Finalizer" #3 daemon prio=8 os_prio=1 cpu=0.00ms elapsed=7.79s tid=0x000001b88cfe1180 nid=0x3bec in Object.wait()  [0x0000006814bff000]
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(java.base@15.0.2/Native Method)
        - waiting on <0x0000000071100b420> (a java.lang.ref.ReferenceQueue$Lock)
        at java.lang.ref.ReferenceQueue.remove(java.base@15.0.2/ReferenceQueue.java:155)
        - locked <0x0000000071100b420> (a java.lang.ref.ReferenceQueue$Lock)
        at java.lang.ref.ReferenceQueue.remove(java.base@15.0.2/ReferenceQueue.java:176)
        at java.lang.ref.Finalizer$FinalizerThread.run(java.base@15.0.2/Finalizer.java:170)

"Signal Dispatcher" #4 daemon prio=9 os_prio=2 cpu=0.00ms elapsed=7.79s tid=0x000001b88d00e2c0 nid=0x4da4 runnable  [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"Attach Listener" #5 daemon prio=5 os_prio=2 cpu=0.00ms elapsed=7.79s tid=0x000001b88d00f320 nid=0x1b50 waiting on condition  [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"Service Thread" #6 daemon prio=9 os_prio=0 cpu=0.00ms elapsed=7.79s tid=0x000001b88d00fbe0 nid=0x47a0 runnable  [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"C2 CompilerThread0" #7 daemon prio=9 os_prio=2 cpu=0.00ms elapsed=7.79s tid=0x000001b88cffe020 nid=0x22f0 waiting on condition  [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE
   No compile task

"C1 CompilerThread0" #15 daemon prio=9 os_prio=2 cpu=0.00ms elapsed=7.79s tid=0x000001b88d000540 nid=0x3f2c waiting on condition  [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE
   No compile task

"Sweeper thread" #19 daemon prio=9 os_prio=2 cpu=0.00ms elapsed=7.79s tid=0x000001b88d0011f0 nid=0x436c runnable  [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"C1 CompilerThread1" #16 daemon prio=9 os_prio=2 cpu=0.00ms elapsed=7.77s tid=0x000001b88d168460 nid=0x4e3c waiting on condition  [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE
   No compile task

"C1 CompilerThread2" #17 daemon prio=9 os_prio=2 cpu=0.00ms elapsed=7.77s tid=0x000001b88d16c980 nid=0x4250 waiting on condition  [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE
   No compile task
```



```
Välj C:\WINDOWS\system32\cmd.exe

"Notification Thread" #20 daemon prio=9 os_prio=0 cpu=0.00ms elapsed=7.77s tid=0x000001b88d1aaf30 nid=0x4804 runnable  [0x0000000000000000]
   java.lang.Thread.State: RUNNABLE

"Common-Cleaner" #21 daemon prio=8 os_prio=1 cpu=0.00ms elapsed=7.76s tid=0x000001b88d19a310 nid=0xfb8 in Object.wait()  [0x00000068156fe000]
   java.lang.Thread.State: TIMED_WAITING (on object monitor)
        at java.lang.Object.wait(java.base@15.0.2/Native Method)
        - waiting on <0x000000071110a5b8> (a java.lang.ref.ReferenceQueue$Lock)
        at java.lang.ref.ReferenceQueue.remove(java.base@15.0.2/ReferenceQueue.java:155)
        - locked <0x000000071110a5b8> (a java.lang.ref.ReferenceQueue$Lock)
        at jdk.internal.ref.CleanerImpl.run(java.base@15.0.2/CleanerImpl.java:148)
        at java.lang.Thread.run(java.base@15.0.2/Thread.java:832)
        at jdk.internal.misc.InnocuousThread.run(java.base@15.0.2/InnocuousThread.java:134)

"pool-1-thread-1" #22 prio=5 os_prio=0 cpu=0.00ms elapsed=7.76s tid=0x000001b88d19a7e0 nid=0x1184 waiting on condition  [0x00000068157fe000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
        at java.lang.Thread.sleep(java.base@15.0.2/Native Method)
        at mm223kk_assignment_3.MultithreadedService$Task.run(MultithreadedService.java:89)
        at java.util.concurrent.Executors$RunnableAdapter.call(java.base@15.0.2/Executors.java:515)
        at java.util.concurrent.FutureTask.run(java.base@15.0.2/FutureTask.java:264)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(java.base@15.0.2/ThreadPoolExecutor.java:1130)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(java.base@15.0.2/ThreadPoolExecutor.java:630)
        at java.lang.Thread.run(java.base@15.0.2/Thread.java:832)

"pool-1-thread-2" #23 prio=5 os_prio=0 cpu=0.00ms elapsed=7.76s tid=0x000001b88d1edd30 nid=0x51f8 waiting on condition  [0x00000068158ff000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
        at java.lang.Thread.sleep(java.base@15.0.2/Native Method)
        at mm223kk_assignment_3.MultithreadedService$Task.run(MultithreadedService.java:89)
        at java.util.concurrent.Executors$RunnableAdapter.call(java.base@15.0.2/Executors.java:515)
        at java.util.concurrent.FutureTask.run(java.base@15.0.2/FutureTask.java:264)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(java.base@15.0.2/ThreadPoolExecutor.java:1130)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(java.base@15.0.2/ThreadPoolExecutor.java:630)
        at java.lang.Thread.run(java.base@15.0.2/Thread.java:832)

"pool-1-thread-3" #24 prio=5 os_prio=0 cpu=0.00ms elapsed=7.76s tid=0x000001b88d1ee200 nid=0x2bb8 waiting on condition  [0x00000068159ff000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
        at java.lang.Thread.sleep(java.base@15.0.2/Native Method)
        at mm223kk_assignment_3.MultithreadedService$Task.run(MultithreadedService.java:89)
        at java.util.concurrent.Executors$RunnableAdapter.call(java.base@15.0.2/Executors.java:515)
        at java.util.concurrent.FutureTask.run(java.base@15.0.2/FutureTask.java:264)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(java.base@15.0.2/ThreadPoolExecutor.java:1130)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(java.base@15.0.2/ThreadPoolExecutor.java:630)
        at java.lang.Thread.run(java.base@15.0.2/Thread.java:832)

"pool-1-thread-4" #25 prio=5 os_prio=0 cpu=0.00ms elapsed=7.76s tid=0x000001b88d1f06e0 nid=0x32a8 waiting on condition  [0x0000006815aff000]
   java.lang.Thread.State: TIMED_WAITING (sleeping)
        at java.lang.Thread.sleep(java.base@15.0.2/Native Method)
        at mm223kk_assignment_3.MultithreadedService$Task.run(MultithreadedService.java:89)
        at java.util.concurrent.Executors$RunnableAdapter.call(java.base@15.0.2/Executors.java:515)
        at java.util.concurrent.FutureTask.run(java.base@15.0.2/FutureTask.java:264)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(java.base@15.0.2/ThreadPoolExecutor.java:1130)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(java.base@15.0.2/ThreadPoolExecutor.java:630)
        at java.lang.Thread.run(java.base@15.0.2/Thread.java:832)

"VM Thread" os_prio=2 cpu=0.00ms elapsed=7.80s tid=0x000001b88cfa8390 nid=0xd24 runnable

"GC Thread#0" os_prio=2 cpu=0.00ms elapsed=7.81s tid=0x000001b8ec7b2db0 nid=0x5b08 runnable
```

Sami Mwanje
mm223kk@student.lnu.se



Output:

```
Running simulation #0
Simulation time passed: 15.0s
Simulation results:
----------------------
Completed tasks:
Task ID: 0, burst time: 2.256s , start time: 0.004s , finished time: 2.331s
Task ID: 1, burst time: 1.078s , start time: 0.004s , finished time: 1.123s
Task ID: 2, burst time: 3.911s , start time: 0.004s , finished time: 3.943s
Task ID: 3, burst time: 9.63s , start time: 0.004s , finished time: 9.682s
Task ID: 4, burst time: 7.499s , start time: 1.123s , finished time: 8.675s
Task ID: 5, burst time: 3.979s , start time: 2.331s , finished time: 6.359s
Task ID: 6, burst time: 3.04s , start time: 3.943s , finished time: 7.063s
Task ID: 7, burst time: 4.876s , start time: 6.359s , finished time: 11.293s
Task ID: 8, burst time: 1.852s , start time: 7.063s , finished time: 8.977s
Task ID: 9, burst time: 2.002s , start time: 8.675s , finished time: 10.689s
Task ID: 10, burst time: 5.621s , start time: 8.977s , finished time: 14.614s
Task ID: 11, burst time: 3.109s , start time: 9.682s , finished time: 12.802s
Task ID: 12, burst time: 4.209s , start time: 10.689s , finished time: 14.916s
Interrupted tasks:
Task ID: 13, burst time: 7.915, start time: 11.293s , interrupted time: 15.0s
Task ID: 14, burst time: 7.219, start time: 12.802s , interrupted time: 15.0s
Task ID: 15, burst time: 9.223, start time: 14.614s , interrupted time: 15.0s
Task ID: 16, burst time: 2.584, start time: 14.916s , interrupted time: 15.0s
Waiting tasks:
Task ID: 17, burst time: 9.918s , start time: waiting
Task ID: 18, burst time: 6.031s , start time: waiting
Task ID: 19, burst time: 7.252s , start time: waiting
Task ID: 20, burst time: 4.843s , start time: waiting
Task ID: 21, burst time: 9.433s , start time: waiting
Task ID: 22, burst time: 3.519s , start time: waiting
Task ID: 23, burst time: 5.478s , start time: waiting
Task ID: 24, burst time: 7.716s , start time: waiting
Task ID: 25, burst time: 5.34s , start time: waiting
Task ID: 26, burst time: 8.044s , start time: waiting
Task ID: 27, burst time: 5.36s , start time: waiting
Task ID: 28, burst time: 4.409s , start time: waiting
Task ID: 29, burst time: 5.61s , start time: waiting


Running simulation #1
Simulation time passed: 15.0s
Simulation results:
----------------------
Completed tasks:
Task ID: 0, burst time: 8.918s , start time: 0.0s , finished time: 8.962s
Task ID: 1, burst time: 2.545s , start time: 0.0s , finished time: 2.618s
Task ID: 2, burst time: 2.588s , start time: 0.0s , finished time: 2.618s
Task ID: 3, burst time: 1.895s , start time: 0.0s , finished time: 1.913s
```

Sami Mwanje
mm223kk@student.lnu.se

```
Task ID: 4, burst time: 4.626s , start time: 1.913s , finished time: 6.545s
Task ID: 5, burst time: 1.294s , start time: 2.618s , finished time: 3.927s
Task ID: 6, burst time: 6.968s , start time: 2.618s , finished time: 9.667s
Task ID: 7, burst time: 7.497s , start time: 3.927s , finished time: 11.479s
Task ID: 8, burst time: 4.176s , start time: 6.545s , finished time: 10.774s
Task ID: 9, burst time: 4.741s , start time: 8.962s , finished time: 13.794s
Interrupted tasks:
Task ID: 10, burst time: 9.112, start time: 9.667s , interrupted time: 15.0s
Task ID: 11, burst time: 6.685, start time: 10.774s , interrupted time: 15.0s
Task ID: 12, burst time: 3.772, start time: 11.479s , interrupted time: 15.0s
Task ID: 13, burst time: 2.742, start time: 13.794s , interrupted time: 15.0s
Waiting tasks:
Task ID: 14, burst time: 6.508s , start time: waiting
Task ID: 15, burst time: 4.756s , start time: waiting
Task ID: 16, burst time: 1.157s , start time: waiting
Task ID: 17, burst time: 5.887s , start time: waiting
Task ID: 18, burst time: 9.865s , start time: waiting
Task ID: 19, burst time: 7.574s , start time: waiting
Task ID: 20, burst time: 4.178s , start time: waiting
Task ID: 21, burst time: 8.514s , start time: waiting
Task ID: 22, burst time: 3.305s , start time: waiting
Task ID: 23, burst time: 9.691s , start time: waiting
Task ID: 24, burst time: 1.19s , start time: waiting
Task ID: 25, burst time: 3.598s , start time: waiting
Task ID: 26, burst time: 2.281s , start time: waiting
Task ID: 27, burst time: 8.304s , start time: waiting
Task ID: 28, burst time: 9.951s , start time: waiting
Task ID: 29, burst time: 8.881s , start time: waiting


Running simulation #2
Simulation time passed: 15.0s
Simulation results:
----------------------
Completed tasks:
Task ID: 0, burst time: 5.994s , start time: 0.001s , finished time: 6.043s
Task ID: 1, burst time: 7.179s , start time: 0.001s , finished time: 7.251s
Task ID: 2, burst time: 1.947s , start time: 0.001s , finished time: 2.015s
Task ID: 3, burst time: 9.571s , start time: 0.001s , finished time: 9.667s
Task ID: 4, burst time: 3.234s , start time: 2.015s , finished time: 5.338s
Task ID: 5, burst time: 3.489s , start time: 5.338s , finished time: 8.861s
Task ID: 6, burst time: 6.819s , start time: 6.043s , finished time: 12.889s
Task ID: 9, burst time: 4.882s , start time: 9.667s , finished time: 14.601s
Interrupted tasks:
Task ID: 7, burst time: 9.832, start time: 7.251s , interrupted time: 15.0s
Task ID: 8, burst time: 7.97, start time: 8.861s , interrupted time: 15.0s
Task ID: 10, burst time: 5.149, start time: 12.889s , interrupted time: 15.0s
Task ID: 11, burst time: 2.307, start time: 14.601s , interrupted time: 15.0s
Waiting tasks:
Task ID: 12, burst time: 2.796s , start time: waiting
Task ID: 13, burst time: 6.229s , start time: waiting
Task ID: 14, burst time: 3.111s , start time: waiting
Task ID: 15, burst time: 6.498s , start time: waiting
Task ID: 16, burst time: 1.916s , start time: waiting
Task ID: 17, burst time: 7.446s , start time: waiting
Task ID: 18, burst time: 1.512s , start time: waiting
Task ID: 19, burst time: 3.912s , start time: waiting
Task ID: 20, burst time: 1.135s , start time: waiting
Task ID: 21, burst time: 8.006s , start time: waiting
```

Sami Mwanje
mm223kk@student.lnu.se

```
Task ID: 22, burst time: 7.948s , start time: waiting
Task ID: 23, burst time: 1.444s , start time: waiting
Task ID: 24, burst time: 1.971s , start time: waiting
Task ID: 25, burst time: 4.537s , start time: waiting
Task ID: 26, burst time: 7.173s , start time: waiting
Task ID: 27, burst time: 2.865s , start time: waiting
Task ID: 28, burst time: 7.118s , start time: waiting
Task ID: 29, burst time: 9.168s , start time: waiting


----------------------
Exiting...
```

[i] https://marijuanapy.com/linux-btrfs-file-system-should-become-the-default-in-fedora-33/

Sami Mwanje
mm223kk@student.lnu.se