1. Documentation of the Linux Kernel Programming Project

Authors:

1. Krzysztof Kostrzewa, 380029
2. Rico Stanosek, 433500

2. List of features implemented and functional

2.1. Helper functions for eviction policies

In `eviction_policy/eviction_policy.h/.c` you will find helper functions that are used in the implementations of the eviction policies. These functions are:

1. A default eviction policy that serves as a fallback if no other eviction policy is specified. It does nothing.
2. `register_eviction_policy` and `unregister_eviction_policy` to register and unregister eviction policies.
3. `set_eviction_policy` to set which eviction policy is to be used.
4. `traverse_dir` is used to recursively traverse directory. It is written very flexible so we can use it in different eviction policies by adjusting the data that is passed to it.
5. `ouichefs_remove_file` is our implementation of deleting a file based on whether we have the `dentry` available or not.
6. `ouichefs_file_in_use` is used to check whether a file is used. We know that this approach by iterating over every process, grabbing their open files and checking their inode against our inode is highly inefficient. However, our attempts to use the `i_count` field of `struct inode` failed as there were many side effects increasing and also not increasing the `i_count` field in a way we needed it to.

2.2. Eviction policies

In both eviction policies you will find the following functions:

1. `clean_dir` gets called when a directory is full and we need to free up space as another file is to be created.
2. `clean_partition` gets called when the partition exceeds a given threshold.
3. `leaf_action` performs some action on a file (leaf). Depending on the eviction policy it compares whether the file is to be deleted or not.
4. A data structure to track which file is to be deleted

2.2.1. Least Recently Used (LRU)

In `wich_lru.c` you will find the implementation of the LRU eviction policy. The ease of just modifying the `is_older` function to depend on either `i_atime`, `i_mtime` or `i_ctime` makes it very flexible and enabled us to implement three eviction strategies with minimal effort.

The default eviction policy is LRU (based on `i_ctime`). You can dynamically change the policy when inserting the module.

```
insmod wich_lru.ko mode=1 # i_atime
insmod wich_lru.ko mode=2 # i_mtime
insmod wich_lru.ko mode=3 # i_ctime, default
```

`clean_partition` collects basic information about the files in the partition and starts the traversal of the partition to find the file to be deleted. It uses `traverse_dir` to traverse the directories and to pass the information to `leaf_action` of the current eviction policy to determine whether the current node is e.g. the least-recently used.

Based on the inode that gets returned, we lastly perform `null` checks and prevent the deletion of the root directory. If these checks pass, we call `ouichefs_remove_file` to delete the file.

In `clean_dir` we can iterate over every file in the given directory and compare them using `is_older` to find the file to delete. We run into an error if the current folder only contains directories.

If we find a file to delete, we call `ouichefs_remove_file` to delete the file.

2.2.2. Size-based eviction

Most of the parts of this eviction policy are similar to the LRU eviction policy. As we do not want to repeat ourselves, we want to highlight the differences.

In `wich_size.c` you will find the implementation of the size-based eviction policy.

Instead of comparing dates in `leaf_action`, we compare the sizes of files to find the largest one.

`clean_partition` works exactly the same as in the LRU eviction policy, except we pass a different function to `traverse_dir` to compare the sizes of the files and another data structure to keep track of the largest file. Afterwards the same checks are performed to prevent the deletion of the root directory and to delete the file.

Again, `clean_dir` works exactly the same as in the LRU eviction policy, except we search for the biggest file instead of the oldest before deleting.

2.3. Printing policy

You will also find a eviction policy `wich_print.c` that will not evict anything. This eviction policy is used to print file and inode information that were very handy during the development of the eviction policies in a tree like fashion.

It gets triggered like every other policy.

2.4. Automatic eviction

The two scenarios in which the eviction process is triggered are:

1. When a directory is full and we need to free up space as another file is to be created. You can see our implementation in `ouichefs_create` in `inode.c`.
2. When the partition exceeds a given threshold. The threshold is set to 80% and defined as `PERCENT_BLOCKS_FREE` in `eviction_policy.h`. The triggering of the eviction process is implemented in `ouichefs_write_end` in `file.c`.

2.5. Base setup

To compile our project you should be able to use the `Makefile` in the root directory. `make install` will additionally move the `ko` files and our `scripts` directory to the specified path in the virtual machine.

```
make
make install
```

To use the eviction policies, you need to traverse to the shared directory and insert the `ouichefs` module and have a image mounted.

```
insmod ouichefs.ko
./scripts/mount.sh test.img
```

Afterwards you can insert the eviction policies of your choice.

```
insmod wich_size.ko
insmod wich_lru.ko
```

2.6. Changing eviction policies

You can retrieve the available eviction policies by using the following command:

```
cat /proc/ouiche/eviction

Following eviction policies are available:
default (does nothing)
wich_size
wich_lru          [ACTIVE]
```

By default the last inserted policy is active. You can change the active policy with:

```
echo -n "wich_size" > /proc/ouiche/eviction

ouichefs:evictions_proc_write: Received policy name: wich_size
set eviction policy to 'wich_size'
```

2.7. Triggering manual eviction

To manually trigger the eviction process, you need the target partition. You can retrieve the available partitions of ouichefs with:

```
cat /proc/ouiche/partitions

Following partitions use ouiche_fs:
0:/dev/loop1
```

You can then manually trigger the eviction process with:

```
echo -n "/dev/loop1" > /proc/ouiche/evict
```

2.8. Removing the modules

You can remove the module with:

```
./scripts/umount.sh test.img
rmmod wich_size.ko
rmmod wich_lru.ko
rmmod ouichefs.ko
```

3. List of features implemented but not fully functional/not implemented

According to the requirements, we have implemented every feature that was required. We performed extensive testing and are confident that our implementation is functional.

4. List of bugs

Todo: Krzysztof