

Kenichi Yasukata & Gauthier Gain

INFO0940
OPERATING SYSTEMS

Project #3

Academic year 2019-2020



YOUR THIRD PROJECT

Hacking the Linux kernel

1. You will implement a kernel-space Key-Value store;
2. The main objective of this assignment is a practice for adding system calls in the Linux kernel;
3. We provide different material to help you (test program, script, ...);
4. It will be a bit more difficult than before. Sorry :(

WHAT'S KEY-VALUE STORE (KVS) ?

- KVS is a type of data storage applications such as relational database management systems;
- KVS is specialized for storing pairs of Key and Value;
- Example implementations
 - Memcached
 - Redis



HOW DOES A KVS WORK?

Primary 3 operations: *Insert*, *Search*, and *Delete*

- *Insert*: register a Key-Value pair;
- *Search*: find a Value from a Key;
- *Delete*: remove an entry that has a specified Key.

HOW DOES A KVS WORK?

Let's make a KVS storage that has pairs of web-browsers and vendors. First, let's insert the records using the *insert* operation:

- *Insert("Internet Explorer", "Microsoft")*
- *Insert("Edge", "Microsoft")*
- *Insert("Firefox", "Mozilla")*
- *Insert("Chrome", "Google")*

HOW DOES A KVS WORK?

After the insert operations, we will have the pairs of Key and Value in the storage:



HOW DOES A KVS WORK?

Now, we want to know which company provides "Internet Explorer" (a great browser).

We can know it by using the *search* operation:

Search("Internet Explorer")

The *search* operation will find "Microsoft" in the storage.

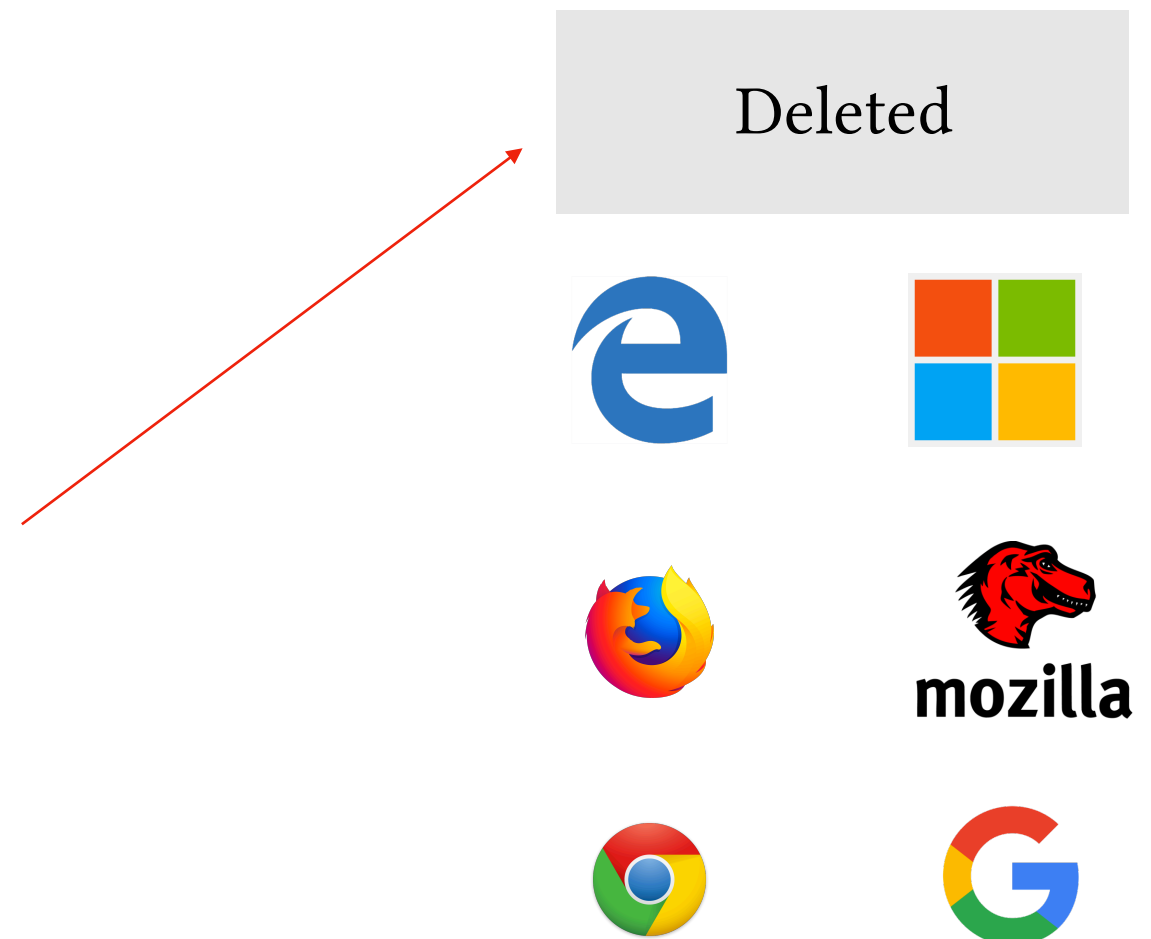


HOW DOES A KVS WORK?

Oh, by the way, Microsoft has stopped to develop Internet Explorer.
If we do not need such old information, we can remove it using the *delete* operation:

Delete("Internet Explorer")

The *delete* operation removes the entry on the storage



KVS WITHIN KERNEL

- We will implement the KVS functionality inside the Linux kernel;
- Precisely, we will implement 3 system calls for the *insert*, *search*, and *delete* operations;
- The kernel-space KVS will store the pairs of Key and Value inside the kernel address space.

THE 3 SYSTEM CALLS

Precisely, we will implement 3 system calls for the *insert*, *search*, and *delete* operations;

Kernel C Code

```
long kvs_insert(const char *key, size_t keylen,  
               const char *val, size_t vallen);  
  
long kvs_search(const char *key, size_t keylen,  
               char *val, size_t max_vallen);  
  
long kvs_delete(const char *key, size_t keylen);
```

THE *INSERT* SYSCALL

```
long kvs_insert(const char *key, size_t keylen,  
               const char *val, size_t vallen);
```

- `const char *key`: a pointer to the key string;
- `size_t keylen`: the length of the key;
- `const char *val`: a pointer to the value string;
- `size_t vallen`: the length of the value string;

THE *SEARCH* SYSCALL

```
long kvs_search(const char *key, size_t keylen,  
               char *val, size_t max_vallen);
```

- `const char *key`: a pointer to the key string;
- `size_t keylen`: the length of the key;
- `const char *val`: a pointer to a buffer where the return value will be stored by the kernel;
- `size_t max_vallen`: the length of the "*val*" buffer;

THE *DELETE* SYSCALL

```
long kvs_delete(const char *key, size_t keylen);
```

- `const char *key`: a pointer to the key string;
- `size_t keylen`: the length of the key;

EXAMPLE

```
kvs_insert("Internet Explorer", 17, "Microsoft", 9);  
kvs_insert("Edge", 4, "Microsoft", 9);
```

```
char buf[16];  
kvs_search("Edge", 4, buf, sizeof(buf));
```

Supposedly, the kernel KVS stores "Microsoft" in `buf`.

```
kvs_delete("Internet Explorer", 17);
```

KVS WITHIN KERNEL

Demonstration

HINTS

For this project, you need to do some research but here are some hints to help you:

- You need to add 3 syscalls... think that we only consider **x86_32bits** architecture;
- You can investigate within the Linux source code and analyse existing system calls such as `read`, `write`, `fork`, ...
- Don't forget to check user inputs (see lab 5);
- Use `printf` to debug and `goto` for errors handling (see lab 5);
- Check if your kernel was upgraded (`uname -a`).

SUBMISSION

- Submit **all patches** that have diffs from Linux-4.15 (the first commit);
- Patches should be made by `git format-patch` (refer to the slides/tutorial page to know how to use it);
- The patch files should be saved in a directory named `patch`, and it should be compressed into an archive called `patch.tar.gz`

TESTING ENVIRONMENT (1)

- We offer the test setup that actually runs on the submission platform (but it is limited concerning tests).
- **Benefits:**
 - You can test whether your patches are applicable or not;
 - You will be able to have better understanding about expected behaviour.
- Privately, I guess this would be a good starting point!

TESTING ENVIRONMENT (2)

- How to use it?
 - First download the archive at the following link: http://www.montefiore.ulg.ac.be/~yasukata/lecture/asset/INFO0940_2020/project3/info0940_project3_test.tar.gz
 - Then put this archive on your reference VM (for testing):
 - **Don't use this in other places!!!**
 - After enter the following command to extract:
 - `tar xvf info0940_project3_test.tar.gz`
 - Finally, you will observe 4 entities:
 - `companion script0.sh script1.sh test.c`

TESTING ENVIRONMENT (3)

The content of this archive:

- `companion: Linux-4.15` source;
- `script0.sh`:
 - Applies patch and compile the kernel;
 - Initially, patch adaptation part is commented out in order not to delete your modifications.
- `script1.sh`: run tests;
- `test.c`: the test C program that calls your syscalls;

USING TESTING ENVIRONMENT

(1)

- Prepare your `patch.tar.gz` (to be submitted), and put it in “`info0940_project3_test`” folder;
- Remove comment out in `script0.sh`:
 - Please pay attention to the consequences of this;
 - Your patch will be applied to files in companion;
 - The commented out part deletes patch files first for cleaning up
- If you do not remove the comment out, `script0.sh` just compiles and installs the kernel

USING TESTING ENVIRONMENT

(2)

```
#
# Please remove this comment out after your prepare your patch.tar.gz
#
##/bin/rm -rf patch &> /dev/null
##if [ ! -e patch.tar.gz ]; then
##  mark 20 "Could not find expected tar.gz file, please upload the file named patch.tar.gz"
##  exit 1
##fi
#
#tar zxvf patch.tar.gz &> /dev/null
#if [ $? -ne 0 ]; then
#  mark 20 "System Error 2, Please report to TAs"
#  exit 2
#fi
#
#if [ ! -e patch ]; then
#  mark 20 "Could not find the directory named patch"
#  exit 3
#fi
#
cd companion &> /dev/null
if [ $? -ne 0 ]; then
  mark 20 "System Error 3, Please report to TAs"
  exit 4
fi

#
# Please remove this comment out after your prepare your patch.tar.gz
#
#git config --global user.name "student" &> /dev/null
#git config --local user.email "someone@student.uliege.be" &> /dev/null
#echo -n "Applying patches ..."
#git am ../patch/* &> patch.log
#if [ $? -ne 0 ]; then
#  git reset --hard HEAD &> /dev/null
#  /bin/rm .config
#  git am ../patch/* &> patch.log
#  if [ $? -ne 0 ]; then
#    mark 20 "Failed, please check the log"
#    cat patch.log
#    exit 5
#  fi
#fi
#
#else
#  echo "OK"
#fi
```



```
#
# Please remove this comment out after your prepare your patch.tar.gz
#
/bin/rm -rf patch &> /dev/null
if [ ! -e patch.tar.gz ]; then
  mark 20 "Could not find expected tar.gz file, please upload the file named patch.tar.gz"
  exit 1
fi

tar zxvf patch.tar.gz &> /dev/null
if [ $? -ne 0 ]; then
  mark 20 "System Error 2, Please report to TAs"
  exit 2
fi

if [ ! -e patch ]; then
  mark 20 "Could not find the directory named patch"
  exit 3
fi

cd companion &> /dev/null
if [ $? -ne 0 ]; then
  mark 20 "System Error 3, Please report to TAs"
  exit 4
fi

#
# Please remove this comment out after your prepare your patch.tar.gz
#
#git config --global user.name "student" &> /dev/null
#git config --local user.email "someone@student.uliege.be" &> /dev/null
#echo -n "Applying patches ..."
#git am ../patch/* &> patch.log
#if [ $? -ne 0 ]; then
#  git reset --hard HEAD &> /dev/null
#  /bin/rm .config
#  git am ../patch/* &> patch.log
#  if [ $? -ne 0 ]; then
#    mark 20 "Failed, please check the log"
#    cat patch.log
#    exit 5
#  fi
#fi
#
#else
#  echo "OK"
#fi
```

USING TESTING ENVIRONMENT (3)

- Afterwards, run `./script0.sh`;
- Supposedly, the script automatically compiles and installs new kernel in companion;
- After `./script0.sh` completes, reboot the reference VM;
- After reboot, run `/script1.sh`;
- This script executes the actual test program.

USING TESTING ENVIRONMENT (SUMMARY)

- First, run `./script0.sh`
 - This will compile and install the kernel
- Second, please enter `"sudo reboot"`
 - This will reboot
- Third, please run `./script1.sh`
 - This will run the test
- After you have your submission file `patch.tar.gz`, put it in `info0940_project3_test`, and remove comment outs in `script0.sh`, and repeat from the first step.

USING TESTING ENVIRONMENT

- TAs encourage you to confirm your patch works correctly in our reference VM and the offered environment.
- **Don't use the submission platform for debug:**
 - Panic kernels occupy test runners of submission platform for a long time;
 - If you submit a lot of patches that crash the kernel, you and your friends may need to wait for really long time until starting the test...
- **We limited the number of submissions**

KVS WITHIN KERNEL: REQUIREMENTS

Others:

- Group of **two** that you will **keep** the whole semester
- Submit a tar file on the submission platform
- More information on the statements (available this evening).

You should be good programmers: We want clean code (good style), without error and warning. Within kernel programming you can use goto.

Don't forget: We detect **plagiarism** so don't try...

Plagiarism = **0 for the course!**

KVS WITHIN KERNEL: REQUIREMENTS

Deadline: 15th April 2020

Happy Coding!