Project 4
Hello, File System!

May 25, 2021

SNU Operating Systems

Overview

- Geo-tagged file system (based on <u>only ext2</u>)
  - Attach a GPS tag to each regular file
  - GPS tag should be set whenever a file is created / modified
- Access control with the GPS location
  - Files are only accessible from the location where they were most recently created/modified

Overview

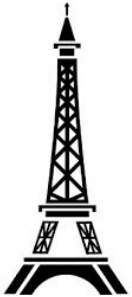
Building 301

Create hi.txt

hi.txt

Say hi to 301 students

<Geo-tag>
Building 301



Paris

Building 302

Assume that 301 & 302 are geometrically close

Overview

Building 301

Building 302

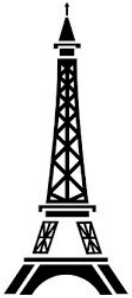
Append hi.txt

hi.txt

Say hi to 301 students
Say hi to 302 students

<Geo-tag>
Building 302

GPS location modified!



Paris

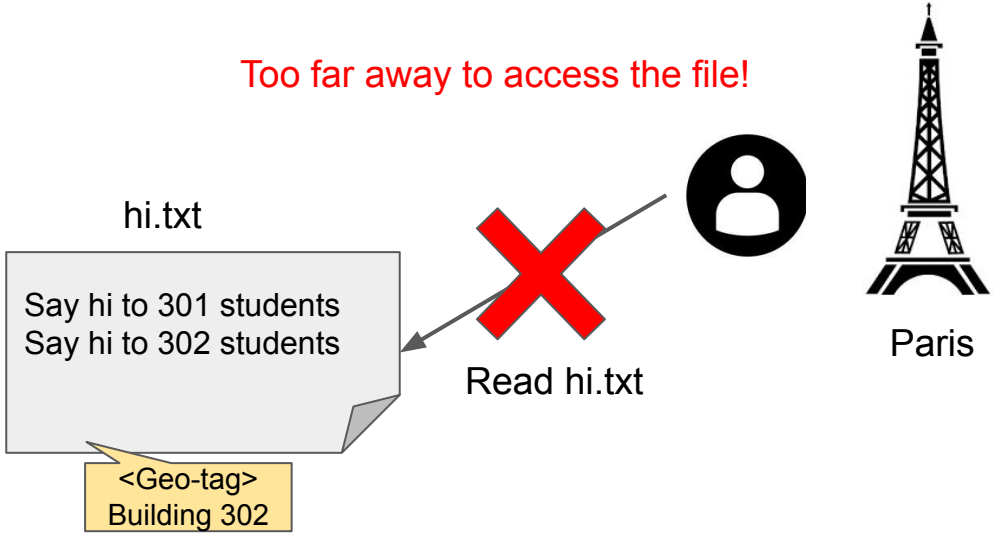
Assume that 301 & 302 are geometrically close

Overview

Building 301

Building 302

Assume that 301 & 302 are geometrically close



☢ The `get_gps_location` syscall succeeds regardless of the current location

Four Steps of Project 4

① Syscall to set the device’s current GPS location

② Add geo-tag support to the Linux inode structure

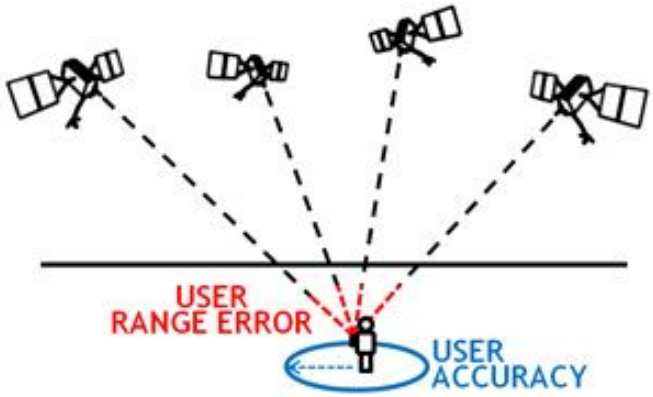
③ User-space testing for step ②

④ Access control based on GPS location

1. Set GPS location

```
struct gps_location {
  int lat_integer;
  int lat_fractional;
  int lng_integer;
  int lng_fractional;
  int accuracy;
};
```

- Struct `gps_location` has five fields
  - Latitude, Longitude, Accuracy
  - `*_fractional` accurate to sixth decimal places
- Implement system call `sys_set_gps_location`
  - Set current location of the device in the kernel
  - Recall what you did in project 2!



2. Add geo-tag to inode structure

- What to do with inode?
  - Whenever a file is created/modified in our custom ext2 file system, the current GPS location in the kernel should be recorded in the inode.
- What is “inode”?
  - Data structure that manages file metadata
  - Owner, date/time, permission control, file type, …
  - Related operations are defined internally as `inode_operations`
- Its implementation in the kernel is very similar with `struct sched_class`

Implementation of sched_class

```
// kernel/sched/rt.c
const struct sched_class rt_sched_class = {
        .next                   = &fair_sched_class,
        .enqueue_task           = enqueue_task_rt,
        .dequeue_task           = dequeue_task_rt,
        .yield_task             = yield_task_rt,

        .check_preempt_curr     = check_preempt_curr_rt,

        .pick_next_task         = pick_next_task_rt,
        .put_prev_task          = put_prev_task_rt,

#ifdef CONFIG_SMP
        .select_task_rq         = select_task_rq_rt,

        .set_cpus_allowed       = set_cpus_allowed_rt,
        .rq_online              = rq_online_rt,
        .rq_offline             = rq_offline_rt,
        .pre_schedule           = pre_schedule_rt,
        .post_schedule          = post_schedule_rt,
        .task_woken             = task_woken_rt,
        .switched_from          = switched_from_rt,
#endif


        .set_curr_task          = set_curr_task_rt,
        .task_tick              = task_tick_rt,

        .get_rr_interval        = get_rr_interval_rt,

        .prio_changed           = prio_changed_rt,
        .switched_to            = switched_to_rt,
};
```

Interface

Implementation

Multiple implementation sets

```
const struct sched_class rt_sched_class = {

const struct sched_class fair_sched_class = {

const struct sched_class idle_sched_class = {
```

Pointing from each task_struct

```
struct task_struct {
        volatile long state;    /* -1 unrunnabl
        void *stack;
        atomic_t usage;
        unsigned int flags;     /* per process
        unsigned int ptrace;

#ifdef CONFIG_SMP
        struct llist_node wake_entry;
        int on_cpu;
#endif
        int on_rq;

        int prio, static_prio, normal_prio;
        unsigned int rt_priority;
        const struct sched_class *sched_class;
        struct sched_entity se;
        struct sched_rt_entity rt;
```

Similar for inode_operations

```
const struct inode_operations ext4_file_inode_operations = {
        .setattr        = ext4_setattr,
        .getattr        = ext4_getattr,
        .setxattr       = generic_setxattr,
        .getxattr       = generic_getxattr,
        .listxattr      = ext4_listxattr,
        .removexattr    = generic_removexattr,
        .get_acl        = ext4_get_acl,
        .fiemap         = ext4_fiemap,
};
```

Interface

Implementation

```
struct inode {
        umode_t                 i_mode;
        unsigned short          i_opflags;
        kuid_t                  i_uid;
        kgid_t                  i_gid;
        unsigned int            i_flags;

#ifdef CONFIG_FS_POSIX_ACL
        struct posix_acl        *i_acl;
        struct posix_acl        *i_default_acl;
#endif

        const struct inode_operations   *i_op;
        struct super_block      *i_sb;
```

```
const struct inode_operations ext3_dir_inode_operations = {
const struct inode_operations ext4_file_inode_operations = {
const struct inode_operations ext4_special_inode_operations = {
```

2. Add geo-tag to inode: Modify inode operations

- Add the following two function pointer fields to the `inode_operations` structure in `include/linux/fs.h`
  - `int (*set_gps_location)(struct inode *);`
  - `int (*get_gps_location)(struct inode *, struct gps_location *);`
- Implement & register those functions with ext2 file system `inode_operations`
- Call the *setter* properly whenever the file is created or modified

2. Update GPS at inode: Modify inode structure

- Modify two inode structures in `fs/ext2/ext2.h`
  - `ext2_inode`: on-disk representation
  - `ext2_inode_info`: in-memory representation
- You have to add 5 attributes to each structure
- Modify some functions that transform between those two structures
- Be careful with physical(on-disk) representation
  - You should consider endianness, order of the fields, …

3. User-space testing

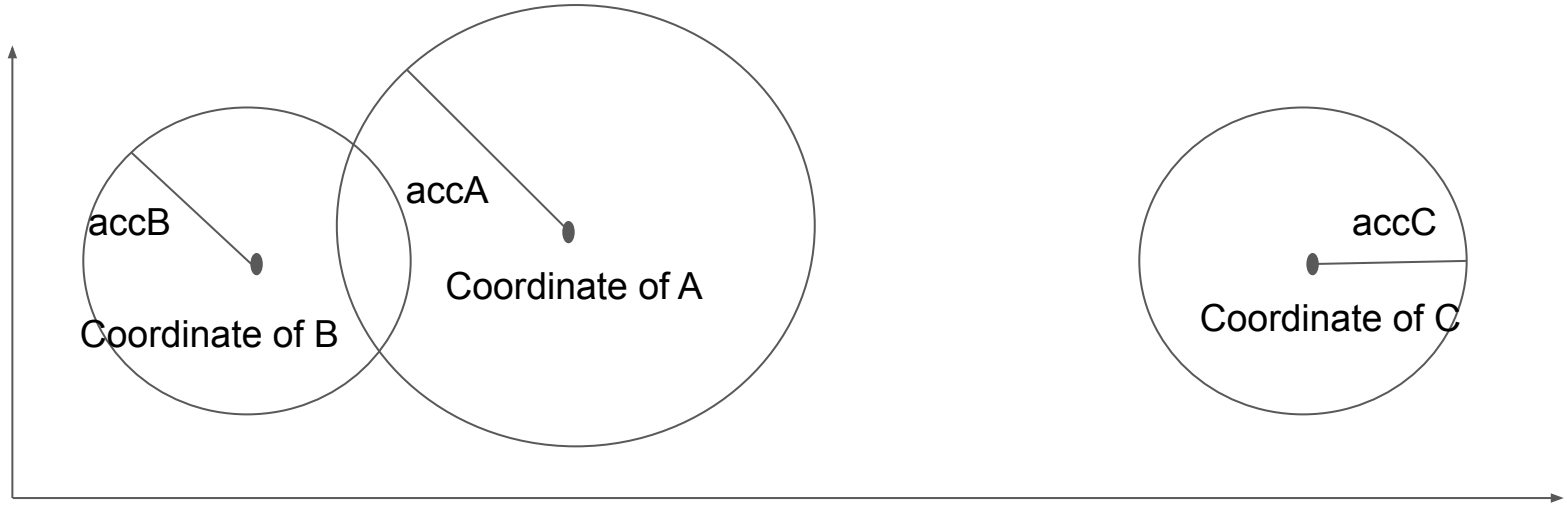
- Testing the GPS location setter/getter
  - By default, the root file system inside QEMU is btrfs
  - We will use `e2fsprogs` to test our custom ext2 file system!
- Create ext2 file system with `mke2fs`.
  - You need to modify the ext2 inode structure inside `e2fsprogs` to make `mke2fs` match your modified ext2 file system in the kernel
  - `e2fsprogs/lib/ext2fs/ext2_fs.h`

4. Location-based access control

- Files of the modified ext2 can be only accessible from the location close to where they are recently created/modified.
- When user tries to access to geo-tagged file,
  - Check based on existing access control mechanism first
  - If it passes the existing one, then check geometric proximity condition → this work as an extra condition!
- TIP: there is an ***inode operation*** *used for* ***general access control****.* You can use it.

4. Location-based access control

- ‘Accuracy’ is used to determine whether the current location is *geometrically close* to file location
  - (A, B) is close
  - (A, C), (B, C) is not close



4. Location-based access control

- Remember: there are no float/double types in the kernel
- You should calculate the distance between two coordinates with your own algorithm.
- Document any assumptions or approximations on README.md

Be careful!

- The current device location is a shared mutable state, so you should <u>properly synchronize</u> access to it.
- Never access the user-space memory address directly.
  - Remind what we did in Project 1 :)
- For parameters in `struct gps_location` in the `set_gps_location` system call, make sure they are in their valid range.
  - Do NOT include 180.xxx for longitude!

About submission (IMPORTANT!)

- **No Design Review!**
- Three things to submit
  - `test/gpsupdate.c` : updates kernel GPS location with sys_set_gps_locatin
  - `test/file_loc.c`: get GPS location recorded in geo-tagged ext2 file
  - `proj4.fs` : should contain at least 1 directory & 2 files with different GPS coordinates

About submission (IMPORTANT!)

- Don't be late!
  - TAs will clone all repositories exactly at the deadline
- Submit code
  - Your team’s private project 4 repo (swsnu/project4-hello-file-system-team-n), master branch
  - README: description of your implementation, how to build,and lessons learned
- Submit slides and demo (n is your team number)
  - Email: osspr2021@gmail.com
  - Title: [Project 4] Team n
  - Attachments: `team-n-slides.{ppt,pdf}, team-n-demo.{mp4,avi,…}`
    - One slide file, one demo video!

Q & A