ZoneIDAProc

Tzung-Bi Shih
<penvirus@gmail.com>

Motivation

- Course assignment of Operating System
    - everything is a file (descriptor)
- QA engineer often checks process internal states via checking debug logs
    - the log.. trustworthy?

Related Works

- Debugger (process trace-able utility)
  - variable monitoring / tampering
  - code instrumentation

=> debug symbols are required
=> accessing interface is domain-specific

Problem Statement

We wish to deliver defect-less software to customers.

To verify behavior of our program is correct, QA engineer often triggers state transition inside the process and checks new state is as expected. However, most internal states are available only in debug logs which may not trustworthy enough.

We will use Instrumentation-based Dynamic Accessing Proc to export an interface for accessing the internal states easily.

Design

exporting interface

- Aggregation for relevant states
  - structured addressing
- Manipulation on specified state
  - fine-grained access

Example:
- endpoint
  - ip
  - port
  - name

=> something like Linux proc[1]
=> directory, read-only file, read-write file

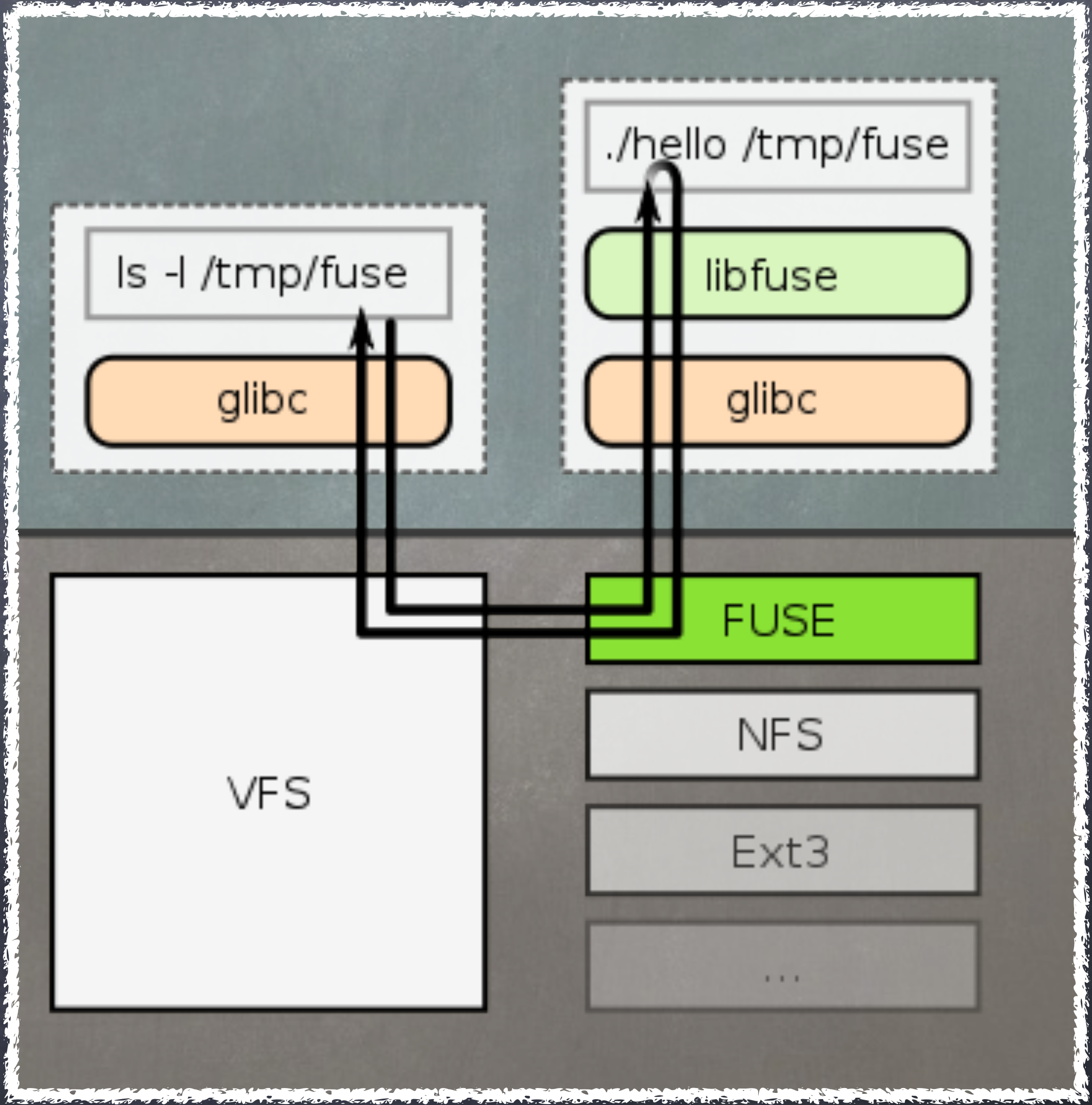
Design
accessing internal state

- Unawareness of aimed process
  - process trace
- Freshness of internal states
  - on-demand access
  - dedicated (spy) thread

Implementation

Linux proc-like interface

- Virtual File System[2]
- Filesystem in Userspace[3]



Implementation
code instrumentation[4]

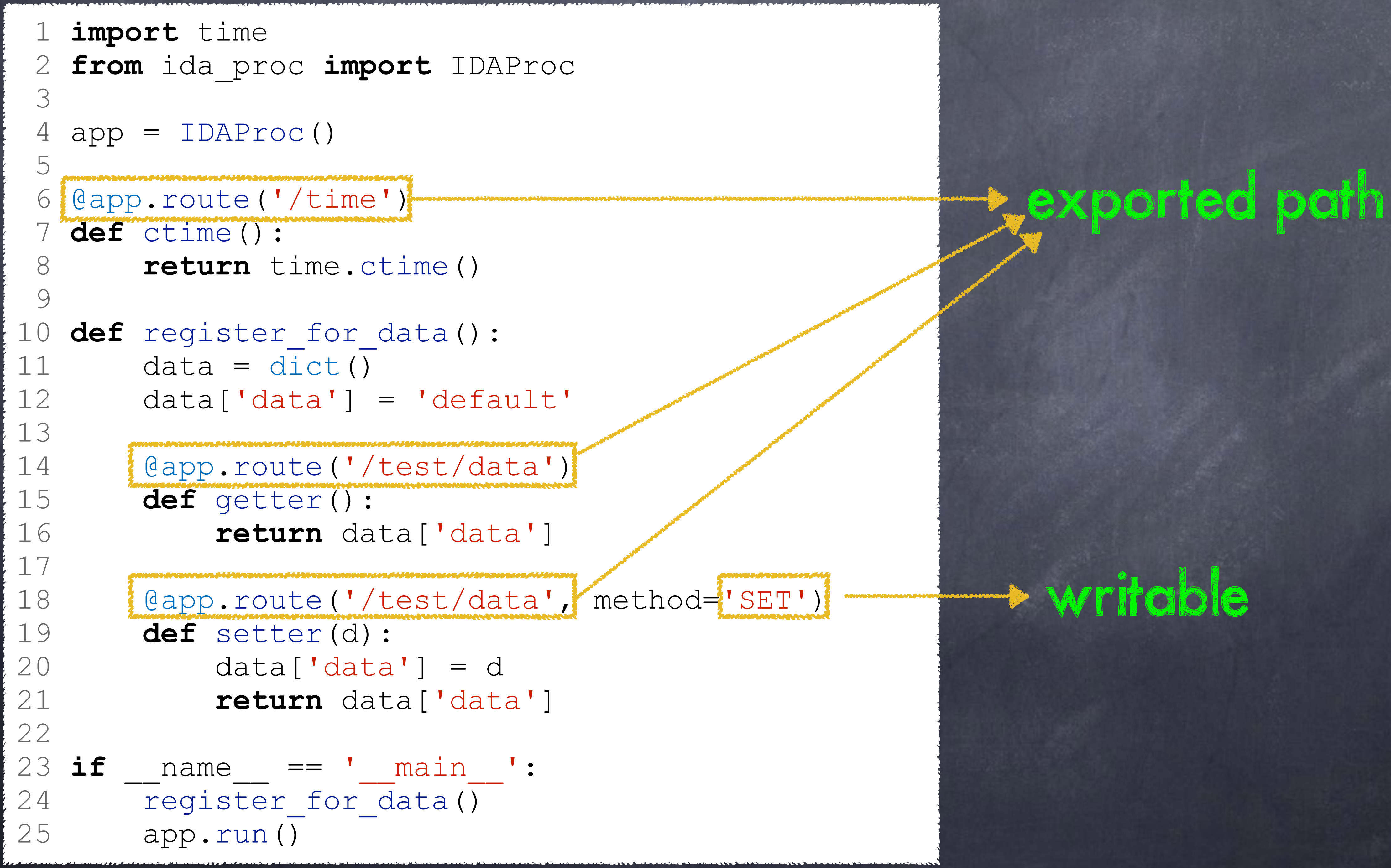
- Easy version
  - gdb
- Difficult version
  - “ptrace(2)”[5][6]

- LSM Yama[7]
  - CAP_SYS_PTRACE
  - PTRACE_TRACEME
  - …

```
$ sudo setcap cap_sys_ptrace+eip ./gdb
```

Example[8]

basic read/write



```python
import time
from ida_proc import IDAProc

app = IDAProc()

@app.route('/time')
def ctime():
    return time.ctime()

def register_for_data():
    data = dict()
    data['data'] = 'default'

    @app.route('/test/data')
    def getter():
        return data['data']

    @app.route('/test/data', method='SET')
    def setter(d):
        data['data'] = d
        return data['data']

if __name__ == '__main__':
    register_for_data()
    app.run()
```

Example

spy thread

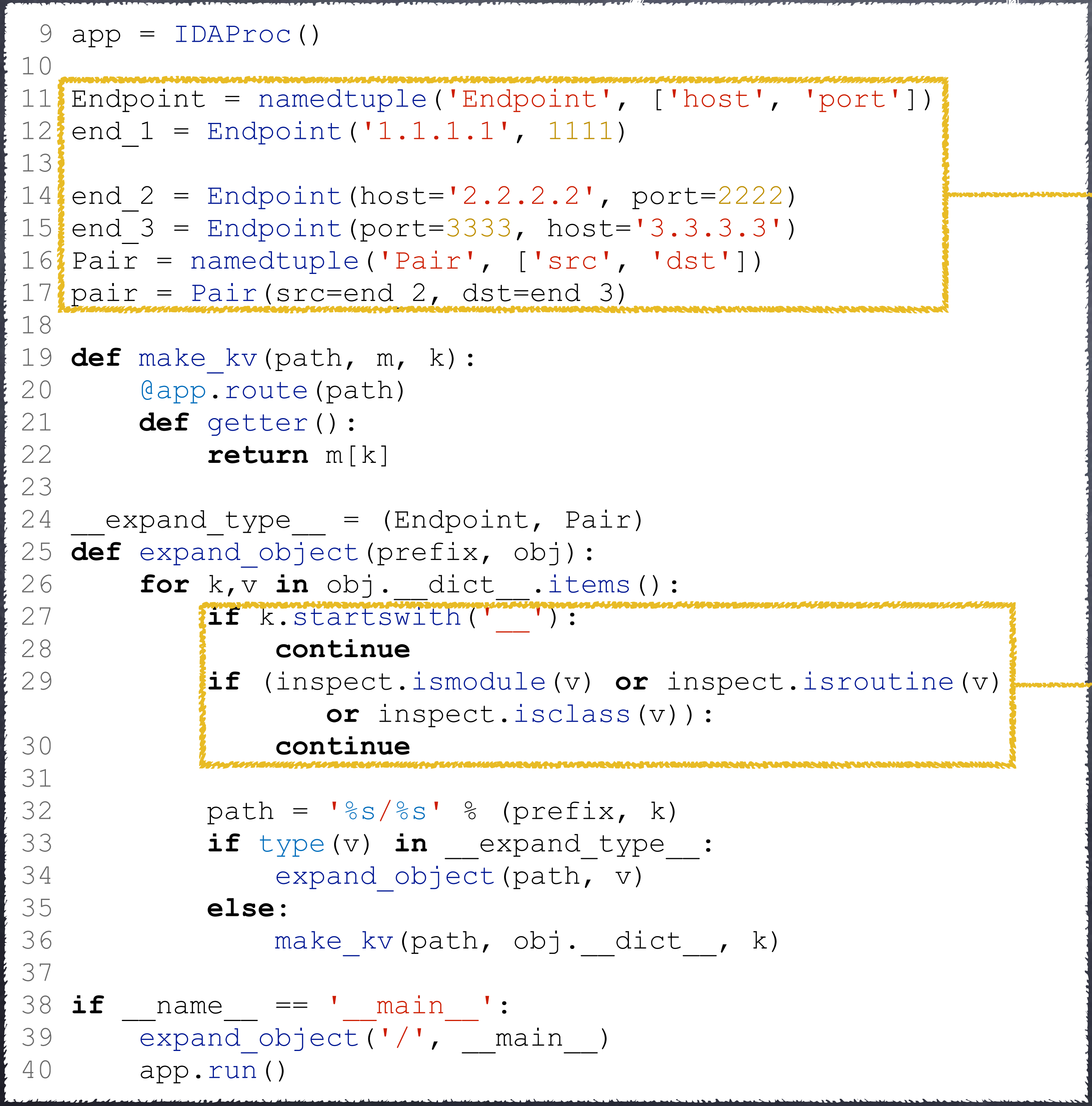
```python
data = dict()
data['data'] = 'default'

def main():
    while True:
        print "[%s]        data['data'] = %s" % (time.ctime(), data['data'])
        sleep(1)

def proc():
    app = IDAProc()

    @app.route('data')
    def getter():
        return data['data']

    @app.route('data', method='SET')
    def setter(d):
        data['data'] = d
        return data['data']

    def fusermount():
        p = subprocess.Popen(['/bin/fusermount', '-u', app.get_mount_point()],
                                 close_fds=True, shell=False)
        p.communicate()
    atexit.register(fusermount)

    app.run()

if __name__ == '__main__':
    t = threading.Thread(target=proc)
    t.daemon = True
    t.start()
    spawn(main).join()
```

main thread

the spy thread has no idea about when will the main thread be terminated

Example

symbol explorer



```python
app = IDAProc()

Endpoint = namedtuple('Endpoint', ['host', 'port'])
end_1 = Endpoint('1.1.1.1', 1111)

end_2 = Endpoint(host='2.2.2.2', port=2222)
end_3 = Endpoint(port=3333, host='3.3.3.3')
Pair = namedtuple('Pair', ['src', 'dst'])
pair = Pair(src=end_2, dst=end_3)

def make_kv(path, m, k):
    @app.route(path)
    def getter():
        return m[k]

__expand_type__ = (Endpoint, Pair)
def expand_object(prefix, obj):
    for k,v in obj.__dict__.items():
        if k.startswith('__'):
            continue
        if (inspect.ismodule(v) or inspect.isroutine(v)
               or inspect.isclass(v)):
            continue

        path = '%s/%s' % (prefix, k)
        if type(v) in __expand_type__:
            expand_object(path, v)
        else:
            make_kv(path, obj.__dict__, k)

if __name__ == '__main__':
    expand_object('/', __main__)
    app.run()
```

some test data

skip uninterested

Example

all-in-one: target program

```python
import time
from collections import namedtuple

Endpoint = namedtuple('Endpoint', ['host', 'port'])
end_1 = Endpoint('1.1.1.1', 1111)

end_2 = Endpoint(host='2.2.2.2', port=2222)
end_3 = Endpoint(port=3333, host='3.3.3.3')
Pair = namedtuple('Pair', ['src', 'dst'])
pair = Pair(src=end_2, dst=end_3)

data = 'default'

while True:
    current = time.ctime()
    print '[%s]        data = %s' % (current, data)
    time.sleep(1)
```

Example

all-in-one: intruder

```python
def instrument_code(pid, filename):
    cmd = list()
    cmd.append('./gdb')
...ignored...
    cmd.append('--pid')
    cmd.append('%s' % pid)
    cmd.append('\'--eval-command=call dlopen("/tmp/pycode_instrumentation.so", 2)\'')
    cmd.append('\'--eval-command=call instrument_file("%s")\'' % filename)
...ignored...

if __name__ == '__main__':
...ignored...
    pid = int(sys.argv[1])
    filename = '/tmp/zone_ida_instrumentation.py'
    code = '''
...ignored...
'''

    with open(filename, 'w') as f:
        f.write(code)
    instrument_code(pid, filename)

    os.remove(filename)
```

execute code within the target process' memory

Example

all-in-one: pycode_instrumentation

```c
int instrument_file(const char *filename)
{
...ignored...
    if(!_Py_IsInitialized()){
        printf("Py_IsInitialized returned false.\n");
        goto error;
    }

    PyInterpreterState *head = _PyInterpreterState_Head();
    if(head == NULL)  {
        printf("Interpreter is not initialized\n");
        goto error;
    }

    PyGILState_STATE pyGILState = _PyGILState_Ensure();
    fp = fopen(filename, "r");
    if(fp == NULL)  {
        printf("file %s doesn't exist", filename);
        goto error;
    }
    _PyRun_SimpleFile(fp, "Instrumentation");
    _PyGILState_Release(pyGILState);

    if(fp)
        fclose(fp);
    return 1;
...ignored...
}
```

key point

Conclusion

- Proc could be an alternative configuration interface
    - persistent configuration file is still needed
- Share states between main thread and spy thread
    - beware of race condition

References

[1]: http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html
[2]: http://en.wikipedia.org/wiki/Virtual_file_system
[3]: http://fuse.sourceforge.net/
[4]: http://stackoverflow.com/questions/8755211/what-is-meant-by-the-term-instrumentation
[5]: http://www.linuxjournal.com/article/6100
[6]: http://www.linuxjournal.com/node/6210
[7]: https://www.kernel.org/doc/Documentation/security/Yama.txt
[8]: https://github.com/penvirus/ZoneIDAProc