Embedded Python in Practice

D. S. Ljungmark
Modio AB

~~Embedded Python in Practice~~
Low Performance Python
Python for Tiny Data using Python

D. S. Ljungmark
Modio AB

We do embedded

Robust systems that

- live on the net
- do things with things
- the machines you don't see

MODIO
Secure & efficient energy information

“Embedded”

Computers you don’t see.

Things you don’t interact with.

SCADA

(**S**ystems **C**ontrol **A**nd **D**ata **A**cquisition)

Not Realtime Controllers

- Gather
- Log
- Upload
- Reconfigure

MODIO
Secure & efficient energy information

SCADA: In Practice

Serial connections
of different sorts

Embedded: In Practice

The code is not very creative

Embedded: In Practice

Optimize for Maintainability

Linux

It’s all Linux systems

Sometimes with *unorthodox* userspace

The Pleasures of Python

Easy to go low level

- Bit banging
- Structs
- C integration

The Pleasures of Python

Rapid development

- Libraries
- Cross platform
- Tooling (tests, etc)

MODIO
Secure & efficient energy information

The Pleasures of Python

Solid

- Reliable
- Maintainable
    - If you do it “right”
- Fast enough

MODIO
Secure & efficient energy information

But there are problems

And sometimes solutions

The Problems

Constraints:

- Memory
- Disk
- Licensing
- Clocks
- Performance

The Problems

Behaviour:

- Recovery
- Debugging
- Reliability

The Problems

Infrastructure:

- Deploying
- Updating
- Launching
- Running

Constraints

- Memory
- Disk
- Licensing
- Clocks
- Performance

MODIO
Secure & efficient energy information

Constraints: Memory

The famous OOM killer

MODIO
Secure & efficient energy information

Constraints: Memory

Avoid multiple CPython instances
(even when they should be separate)

Constraints: Memory

```
for step in itertools.izip(
                    second.mainloop(),
                    first.mainloop()):
  step()
```

MODIO
Secure & efficient energy information

Constraints: Memory

Manually calling GC

```
while True:
  do_stuff()
  gc.collect()
  time.sleep(1)
```

MODIO
Secure & efficient energy information

Constraints: Memory

Manual memory management

- lifetime
- allocation in loops

MODIO
Secure & efficient energy information

Constraints: Memory

```
oldgarb = len(gc.garbage)
while True:
  do_stuff()
  newgarb = len(gc.garbage)
  assert oldgarb is newgarb
```

MODIO
Secure & efficient energy information

Constraints: Disk

Less storage than RAM

Constraints: Disk

tmpfs is RAM

What happens if you run out of RAM?

Constraints: Disk

binary .egg use a lot of space (temp files, unpacking)

Constraints: Disk

Ship .egg directories on compressed filesystem

Constraints: Disk

Ship .pyo files only
(ugly, but works)

Constraints: Disk

Or disable cache files:

```
export PYTHONDONTWRITEBYTECODE=”bah”
```

MODIO
Secure & efficient energy information

Constraints: Disk

SQLite lock contention

Constraints: Disk

```
def _execute(self, *args, **kwargs):
    for attempt in range(self.timeout + 1):
        try:
            with self._connection:
                return self._cursor.execute(*args, **kwargs)
        except sqlite3.OperationalError as e:
            if attempt == self.timeout:
                raise
            if e.args[0] == "database is locked":
                time.sleep(1)
            else:
                raise
```

Constraints: Licensing

We write GPL code

Constraints: Licensing

GPL requires awareness

- Need to save exact version

Constraints: Licensing

Can’t trust the Cheese Shop

MODIO
Secure & efficient energy information

Constraints: Licensing

Licences:

- Differ between pypi & code

Constraints: Licensing

Licences:

- Change between versions

Constraints: Licensing

Licence summary:

- GPL is good
- BSD is good
- MIT is good

Be careful and ever vigilant

MODIO
Secure & efficient energy information

Constraints: (RTC) Clocks

IÄ! IÄ! Cthulhu!

MODIO
Secure & efficient energy information

Constraints: (RTC) Clocks

Time warps due to clock sync

Constraints: Clocks

There are ways to “cheat”

- Jump to last time we logged
- Go online
- Fix clock and store difference

Constraints: Clocks

Time is relative

And ~~sometimes~~ wrong

And **mostly** wrong

Constraints: Clocks

This causes issues

- "Do the time warp"
- Timestamps
- Racing
- Locking

Constraints: Clocks

All code has to be aware

- readings
- loops
- lockfiles

Constraints: Clocks

.pyc files newer than .py files

after updates

Constraints: Clocks

It’s better now (py3)

- `time.monotonic()`

Constraints: Clocks

Life without an RTC?

Painful, no matter what.

Constraints: Performance

Usually good enough

MODIO Secure & efficient energy information

Constraints: Performance

But without an FPU?

MODIO
Secure & efficient energy information

Constraints: Performance

Fake it.

(hint: Linux does it for you)

Constraints: Performance

But still: Don’t do math

Constraints: Performance

Most crypto doesn't use floating point

Constraints: Performance

FPU-less hardware is slow

Constraints: Performance

This makes race conditions…

….fun

MODIO
Secure & efficient energy information

Constraints: Performance

1. start A
2. start B
3. A loads modules… (IO)
4. B starts faster…
5. B needs A…. (fail, crash)
6. A started

Constraints: Performance

You need infrastructure

`“I am functional”`

Constraints: Performance

You need infrastructure

Not usually present

Constraints: Performance

Slow IO + Slow CPU

file collisions

Constraints: Performance

```
with open(fnam, “r”) as f:
   # Race happens here
   unlink(fnam) # OSError
   data = f.read()
```

Constraints: Performance

Wanted: Proper support for “atomic open & unlink”

Constraints: Performance

```
with open(fnam, “ru”) as f:
   os.path.exists(fnam)
   # Should not exist
```

MODIO
Secure & efficient energy information

Constraints: Performance

Also Wanted: Proper support for “open & lock”

MODIO Secure & efficient energy information

Constraints: Performance

```
with open(fnam, “w”) as f,
     fcntl.flock(f) as lock:
  # Can this be atomic?
```

Behaviour

- Recovery
- Debugging
- Reliability

Behaviour: Recovery

"Crash only software"

https://lwn.net/Articles/191059

Behaviour: Recovery

No normal termination

MODIO
Secure & efficient energy information

Behaviour: Recovery

No long lived processes

```
~~while True:~~
while periodical():
```

(Too complex to show)

MODIO
Secure & efficient energy information

Behaviour: Debugging

- Logging Exceptions to disk
  - `sys.excepthook`
- ~~Uploading Exceptions~~

Behaviour: Debugging

Exception hooks

```
def delayed_exc(delay):
    def dec(orig_hook):
        def exc_hook(type_, value, trace):
            orig_hook(type_, value, trace)
            if type_ is not KeyboardInterrupt:
                time.sleep(delay)
        return exc_hook
    return dec
```

Behaviour: Debugging

Stacktrace on -SIGUSR1

```
def stacktrace_on_sigusr1(logfile):
    def stacktrace(sig, frame):
        with open(logfile, "a") as output:
            with redirect_stderr(output):
                dumpstacks(sig, frame)
        return
    return signal.signal(signal.SIGUSR1,
                                stacktrace)
```

Behaviour: Reliability

Delayed exceptions

“It’s fine to crash, later”

Behaviour: Reliability

Delayed Exceptions

```
error = None
while mainloop():
    if error:
        raise error
  for thing in things():
      try:
          mangle(thing)
      except Exception as e:
          error = e
```

MODIO
Secure & efficient energy information

Behaviour: Reliability

What I want to write

```
while mainloop():
    with delayed_exceptions():
        for thing in things():
            mangle(thing)
```

Behaviour: Debugging

(I want type hints)

```
def fooncetion(bar, baz):
    assert isinstance(bar, Barclass)
    assert isinstance(baz, Bazclass)
```

MODIO
Secure & efficient energy information

Infrastructure

- Deploying
- Updating
- Launching
- Running

MODIO
Secure & efficient energy information

Infrastructure: Code

We built our own for:

- Deploy & Update
- Manual cleanout
  - Including .pyc / .pyo __pycache__

Infrastructure: Code

We built our own

It mostly looks like ‘`rpm`’

Infrastructure

- Launching
- Running

Infrastructure: Launching

- Ordering
- Wait for finish (signaling)
- Timeout

Infrastructure: Running

- Is it running?
- Has it hung?
- Reliably restart
- Locks & lockfiles?

Infrastructure: systemd

Systemd fixes:

- Launching
- Event based
- Signalling “I’m ok”

MODIO
Secure & efficient energy information

systemd: signalling

```python
def systemd_ready(addr, sock):
    msg = "READY=1"
    if not (addr and sock):
        return False
    try:
        retval = sock.sendto(msg, addr)
    except socket.error:
        return False
    return (retval > 0)
```

MODIO
Secure & efficient energy information

Infrastructure: systemd

Systemd fixes:

- Running
- Watchdog
- Restarting

systemd: watchdog

```
def watchdog_ping(addr, sock):
    msg = "WATCHDOG=1"
    if not (addr and sock):
        return False
    try:
        retval = sock.sendto(msg, addr)
    except socket.error:
        return False
    return (retval > 0)
```

MODIO
Secure & efficient energy information

Infrastructure: systemd

- We like systemd
- Replaced init+runit
- Gained functionality

Recap

Python is a solid choice

Recap

Python lacks some tooling

1. “crash later”
2. “Atomic read & unlink”
3. Type hints

Recap

Python <u>**works**</u> in embedded

Questions?

P.S. Code samples under MIT licence.

MODIO
Secure & efficient energy information