CPython Internals Documentation

Release 0.0

Victor Stinner

Jun 14, 2017

Contents

Contents:

CHAPTER 1

Price of the portability

Misc

- Timers: PEP 418
- Inheritance of file descriptors and handles: PEP 446 (and PEP 433)
- Timezone
- SSL/TLS x509 root certificates
- use recent OS features: detected in configure, detected at runtime
- doc: “Availability: xxx”
- os.confstr(), os.sysconf()
- os.sysconf(“SC_PAGESIZE”)
- os.cpucount()
- OSError, VMSError, WindowsError, IOError => OSError
- os.closerange()
- os.sendfile(): different API depending on the OS!
- time.strftime()
- tzdata

POSIX vs real world

- Linux adds many new features not part of the POSIX standards
- (Systemd uses new Linux-only features like cgroups, not portable on FreeBSD, OpenBSD, etc.)
- _GNU_SOURCE: asprintf(), canonicalize_file_name()

- realpath(NULL)
- "Undefined" parts of POSIX standards: kept for efficiency?
- AIX

Windows

- POSIX API
- POSIX API has fewer features
- Windows console
- Windows ANSI, OEM and console code pages
- Atomic rename file, os.replace()
- One binary for all Windows versions: use LoadLibrary()
- os.fsencode() error handler and Windows versions
- Socket: no file descriptor, SOCKET_T

UNIX

- accept4() returns ENOSYS
- open(O_CLOEXEC) and socket(SOCK_CLOEXEC) on old Linux kernels

select, pipes

- test_signal: "OS doesn't report write() error on the read end of a pipe"
- select() limited to sockets on Windows
- select(), poll(), epoll(), devpoll(), kqueue(), etc. => selectors => asyncio
- async I/O operations: no Python API yet?
- select.poll()
  - ifdef HAVE_BROKEN_POLL
  - #ifdef __APPLE__ select_have_broken_poll()

Threads and signals

- Hard
  - low-level OS functions, syscalls
  - threads
  - signals
  - fork, exec

- many tests skipped on old versions of operating systems
- OpenBSD older than 5.2 implemented threads in user-space
- FreeBSD older than 7.0
  - "Issue #12392 and #12469: send a signal to the main thread doesn't work before the creation of the first thread on FreeBSD 6"
- "Issue #18238: sigwaitinfo() can be interrupted on Linux (raises InterruptedError), but not on AIX"
- Signal orders
- HP-UX11
- depending on the OS, a signal sent to the pid is received by the mainthread or a random thread
- timeout on locks: drop support for thread APIs different than pthread and nt
- pthread_sigmask()
- pthread_kill()

CHAPTER 2

Bugs found by the Python project

Kernel bugs

- Crash with mmap and sparse files on Mac OS X
- test_os on Linux 2.6.?35? return code
- Many issues with threads+signals on OpenBSD and old versions of FreeBSD: test disabled on these platforms
- FreeBSD: when close(fd) on a fifo fails with EINTR, the file descriptor is not really closed

Compiler bugs

- Visual Studio: PGO on 64 bits
- Visual Studio PGupdate duplicated functions:
  - "Disable COMDAT folding in Windows PGO builds."
  - http://bugs.python.org/issue8847#msg166935
  - http://hg.python.org/cpython/rev/029cde4e58c5
- Error with gcc-4.6 -O1 -ftree-vectorize
  - "Python 3.2 doesn't compile correctly with -O3"
  - http://gcc.gnu.org/ml/gcc-help/2011-01/msg00136.html
  - http://gcc.gnu.org/bugzilla/show_bug.cgi?id=47271
  - http://gcc.gnu.org/viewcvs?view=revision&revision=169233
- Apple Clang bug
  - llvm-gcc-4.2 miscompiles Python (XCode 4.1 on Mac OS 10.7)

CHAPTER 3

C hacks in CPython source code

Detect integer overflows

Before memory allocation.

In overallocate:

```
if (newlen <= (PY_SSIZE_T_MAX - newlen / 4))
    newlen += newlen / 4;
```

Avoid undefined behaviour

Issue #7406:

```
-    x = a + b;
+    /* casts in the line below avoid undefined behaviour on overflow */
+    x = (long)((unsigned long)a + b);
```

Usage of goto

Something like try/finally.

Optimisations

Unicode.

stringlib

- fastsearch.h: Bloom filter

Computed goto

Python/ceval.c

Python types

- (size_t,) PY_SIZE_MAX
- Py_ssize_t, PY_SSIZE_T_MAX
- PY_LONG_LONG (#ifdef HAVE_LONG_LONG)

Type aliasing

PyCompactUnicodeObject:

```
typedef struct {
    PyASCIIObject _base;
    Py_ssize_t utf8_length;     /* Number of bytes in utf8, excluding the
                                 * terminating \0. */
    char *utf8;                 /* UTF-8 representation (null-terminated) */
    Py_ssize_t wstr_length;     /* Number of code points in wstr, possible
                                 * surrogates count as two code points. */
} PyCompactUnicodeObject;
```

Macros

- Add assertions using “a,b” syntax. Ugly example:

```
#define PyUnicode_READY(op)                       \
    (assert(PyUnicode_Check(op)),                 \
     (PyUnicode_IS_READY(op) ?                    \
      0 : _PyUnicode_Ready((PyObject *)(op))))
```

- Py_SAFE_DOWNCAST
- Py_ARRAY_LENGTH(array): check at compile time if the argument is an array
- do { ... } while (0), ex: Py_DECREF()

Examples:

```
#define Py_RETURN_TRUE return Py_INCREF(Py_True), Py_True
#define PyBool_Check(x) (Py_TYPE(x) == &PyBool_Type)
```

Memory allocation

- pymalloc
- free list: unbound method, MemoryError instance, frame, list, set, tuple
- overallocate
  - list: 12.5% new_allocated = (newsize >> 3) + (newsize < 9 ? 3 : 6);
  - string formatting: 25% newlen += newlen / 4;
  - depend on the performance of realloc(): fast on Linux, slow on Windows (older than Windows 7)
- Unicode: PyUnicodeWriter, PyAccu

dict

- hash()
- hash vulnerability

timsort

xxx

CHAPTER 4

CPython is fast

Optimizations implemented in CPython.

Singletons

- Integers in range [-5; 256]
- Empty string
- Single Latin-1 letter

IO

- Efficient buffering: FileIO.read() and FileIO.readall()
- HTTP and socket buffering

Others

- peephole
  - x in {1, 2, 3} => frozenset (constant)
- Overalllocation
  - list
  - bytearray?
  - PyUnicodeWriter
- Free list

- method cache
- stringlib
  - process long per long, instead of byte per byte
  - use goto

ceval.c

- computed goto: label + goto
- fast locals: f_localsplus of a frame

Unicode

- PEP 393: efficient storage for ASCII
- ASCII strings only require to copy N bytes to copy N characters: 2 or 4 times faster than applications using UTF-16 or UCS-4
- findchar: use memchr(), even for UCS-2 and UCS-4

dict

- specialized for int key
- specialized for str key
- hash(str)

list

- timsort

implemented in C

- decimal of Python 3.3 is 120x faster than the Python implementation of Python 3.2

CHAPTER 5

PEP 393 performances

Writing efficient code manipulating Unicode is harder since the PEP 393. The problem is to respect the canonical form.

If you preallocate an ASCII buffer but you need to write a Latin1 character, you have to convert the ASCII string to Latin1 which means copying all already written characters. It is inefficient especially if the Latin1 characters occurs at the end. If you preallocate an UCS4 buffer, but the result is UCS2, you have to "shrink" the buffer from UCS4 to UCS2, which means copying all characters.

To not having to widen or shrink your buffer, you can scan your input to compute the maximum character before allocating the buffer. In practice, processing the input twice may be slower.

Another problem is the length of the result. Getting the length of str%args and str.format(args) require to do the work twice: once to get the length, once to write characters. Both approaches were tested (*), and processing the output twice is too slow.

For efficient code, you should be optimistic and enlarge or widen the buffer on demand. When the output length is unknown, it is better to overallocate the buffer.

The _PyUnicodeWriter API helps to implement such function:

- Widen the buffer on demand
- Enlarge the buffer on demand
- Minimum length and overallocation of the buffer can be configured
- Avoid completly the need of a buffer when the output is only composed of one string
- Delay allocation of the buffer until the first write. It helps to compute the length and kind of the buffer, because the length and kind cannot always be computed before the first write. It avoids also allocating a buffer is no write is done at all (ex: error before writing the first characters).
- Give a direct access to the buffer for best performances

CHAPTER 6

Indices and tables

- genindex
- modindex
- search