# EFFECTIVE PROGRAMMING TECHNIQUES – less watts

Ivana Hutařová Vařeková, Marcela Mašláňová,

Power Managment team, Brno

September 2009

# CONTENT

1. Design level
2. Implementation level
3. Compilation level
4. Profilers (bottleneck)
5. Materials, links

## Watt-Hunting competition

- **programmer competition**
- program the maximum of tasks and optimize them in 45 minutes
- use whatever language you want to get best results
- nice awards
- 13:00, 14:00, 15:00, 16:00 in lab B007

# 1. Design level

- used algorithm and its complexity
- crucial part of optimization
- different complexity classes
- sorting - **$O(n \cdot \ln(n))$**, **$O(n^2)$**

# 2. Implementation level

1. String functions
2. Threads in different languages
3. Memory allocation functions
4. Fsync
5. Wake-ups of disk

# 2. Implementation level – string functions I

```
char *strcpy(char *dest, const char *src);

char *strncpy(char *dest, const char *src, size_t n);

void *memcpy(void *dest, const void *src, size_t n);
```

# 2. Implementation level – string functions II

```
char *strcpy(char *dest, const char *src);
```

- per byte
- copy character from `src` to `dest`, finish when it finds the first occurrence of character `'\0'`

# 2. Implementation level – string functions III

```
char *strncpy(char *dest, const char *src,
                size_t n);
```

- per byte
- tests whether the character is `'\0'` and the length is `n`
- if `strncpy` reaches character `'\0'` before it write `n` chars to `dest`, it fills the rest of `n` characters with `'\0'`
- if `strncpy` writes `n` characters before it reaches character `'\0'` then it stops too – the result string is not end by `'\0'`

# 2. Implementation level – string functions IV

```
char *memcpy(void *dest, const void *src,
                size_t n);
```

- per word
- copy the memory until it reach the limit of size n

# 2. Implementation level – string functions V

- `strcpy/strncpy/memcpy` which copies 10/100/1000 character word to buffer of the same size, `n` is the length of copying word

500
50
5
10 100 1000 10000

strcpy
strncpy
memcpy

# 2. Implementation level – string functions VI

- `strcpy/strncpy/memcpy` which copies 2-character word to 10/100/1000 buffer, `n` is set to the size of buffer

50
5
10
100
1000
10000

strcpy
stncpy
memcpy

# 2. Implementation level – string functions VII

```
char *strcat(char *dest, const char *src);
```

- go through the whole `dest` character find the first occurrance of `'\0'`
- then copy the `src` there
- if we want to copy several strings and use a lot of `strcats` - the complexity is **$O(n^2)$**
- if we save the end position and `memcpy` the string there - the complexity is **$O(n)$**

# 2. Implementation level – string functions VIII

```
char *stpcpy(char *dest, const char *src);
```

- `stpcpy` – new in POSIX.2008
- copy a string returning a pointer to its end

```
char *to = buf;
to = stpcpy(to,"foo");
to = stpcpy(to,"bar");
```

# 2. Implementation level – string functions IX

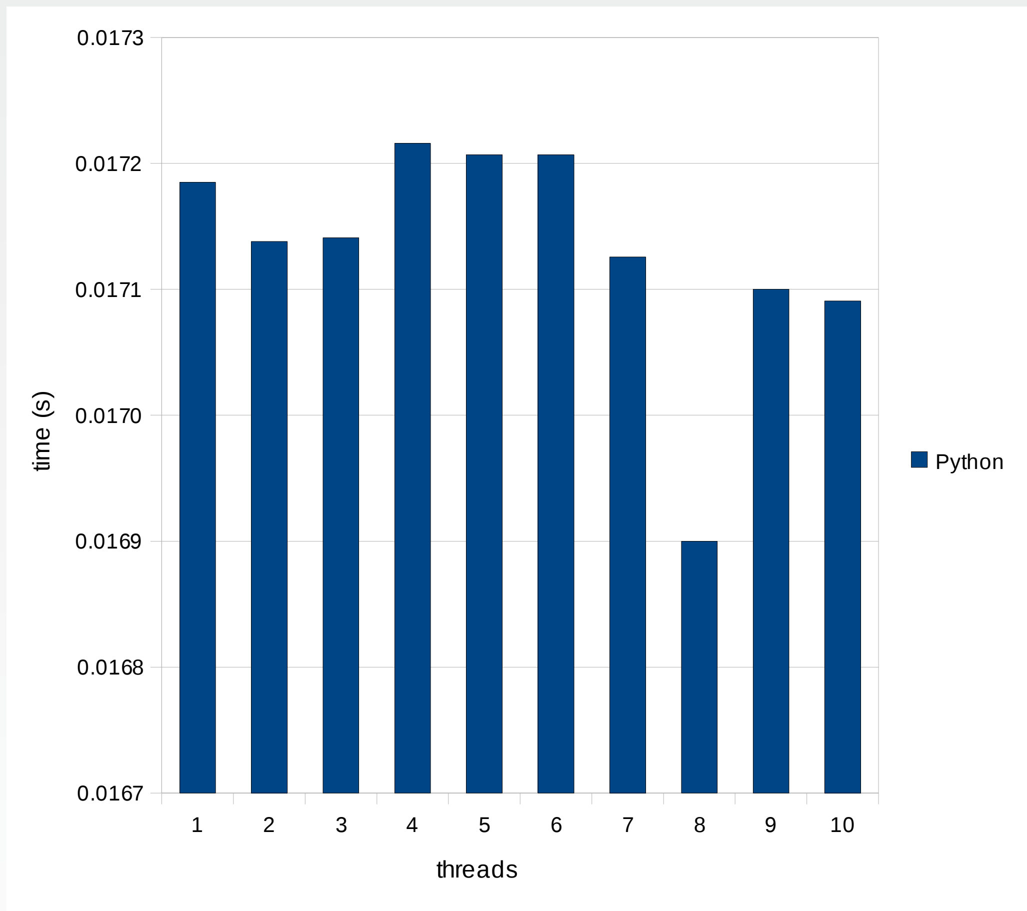
| cycles | 1000 | 2000 |
| --- | --- | --- |
| 2. method | 0.01 | 0.01 |
| strcat | 0.1 | 0.03 |

18
16
14
12
10
8
6
4
2
0

0 5000 10000 15000 20000 25000 30000 35000 40000 45000 50000

strcat  2. method

# 2. Implementation level - threads




Python

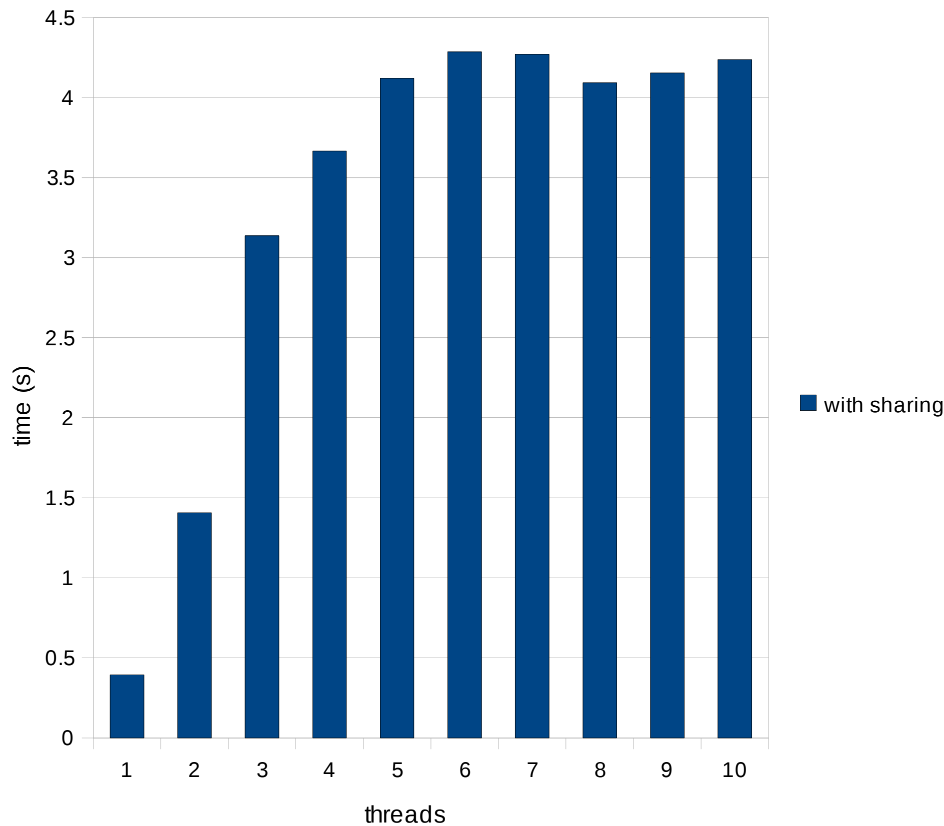
- Global Lock Interpreter (GIL)
- operations aren't safe without lock
- profitable only on I/O operations
- optimize with project unladen swallow

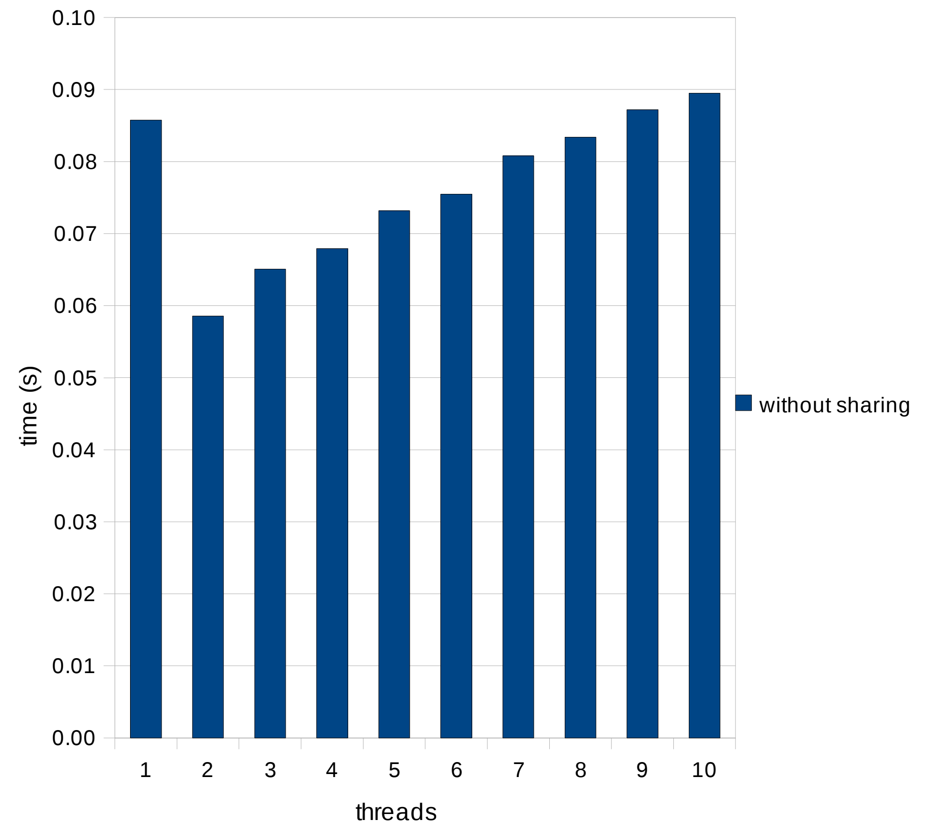
# 2. Implementation level - threads

- Perl
  - threads, ithreads
    - (don't) share data by default
    - user defines level of sharing
    - Problems:
      - without sharing – COW for every thread
      - with sharing – COW + create tied variables
      - fork could be better for unix only application

# 2. Implementation level - threads

## Multiply matrix 40x40



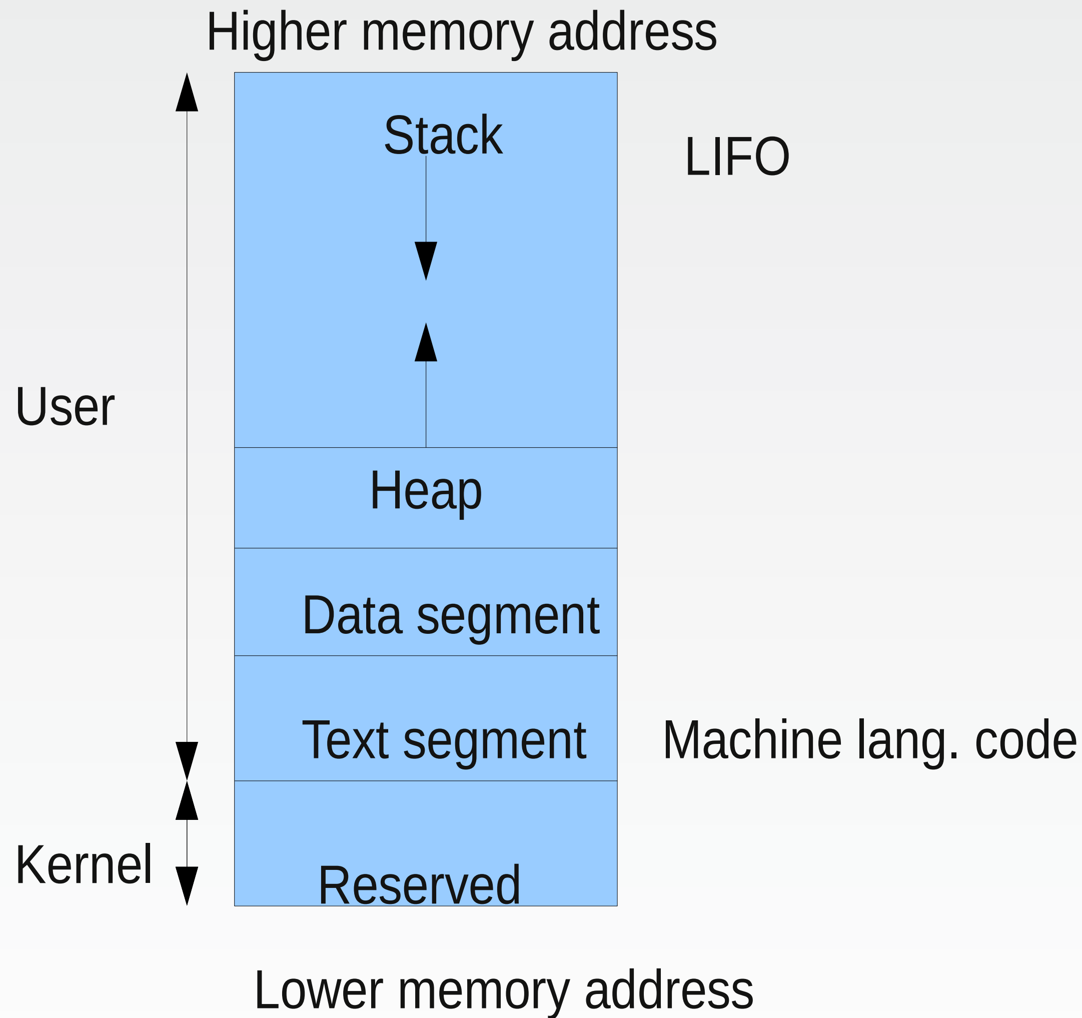
# 2. Implementation level - threads

- C
  - threads share the same global memory
  - each thread has his own stack
  - independent threads can access the same variables in memory
  - kernel doesn't create new copy of process memory space and file descriptors
    - standard fork is slower compared to threads
  - threads can take advantage of more CPU

# 2. Implementation level - memory

- Virtual memory divided into segments:
  - text – code, read-only data
  - data (=heap) – RW segment of binaries/libraries
  - stack
    - segment grows with variables but don't shrink with them
    - pointer at structure is better than structure (time)
    - allocation of larger locale variables dynamically (free'd after end of function/block)

# 2. Implementation level - memory



# 2. Implementation level - memory

```
 6482 pts/6    Ss+    0:00 bash -rcfile .bashrc

cat /proc/6482/maps
00400000-004d1000 r-xp 00000000 fd:00 39898                              /bin/bash
006d0000-006da000 rw-p 000d0000 fd:00 39898                              /bin/bash
...
008d9000-008e1000 rw-p 000d9000 fd:00 39898                              /bin/bash
00e9e000-00f01000 rw-p 00000000 00:00 0                                  [heap]
328ba00000-328ba1d000 r-xp 00000000 fd:00 13883                          /lib64/libtinfo.so.5.7
328ba1d000-328bc1d000 ---p 0001d000 fd:00 13883                          /lib64/libtinfo.so.5.7
328bc1d000-328bc21000 rw-p 0001d000 fd:00 13883                          /lib64/libtinfo.so.5.7
7fa10d08b000-7fa10d08c000 rw-p 00000000 00:00 0
7fa10d08c000-7fa10d08d000 r--p 0001f000 fd:00 32810                      /lib64/ld-2.10.90.so
7fa10d08d000-7fa10d08e000 rw-p 00020000 fd:00 32810                      /lib64/ld-2.10.90.so
7fa10d08e000-7fa10d08f000 rw-p 00000000 00:00 0
7fff4d262000-7fff4d277000 rw-p 00000000 00:00 0                          [stack]
7fff4d3ff000-7fff4d400000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

# 2. Implementation level - memory

- Data segment – allocation functions
  - calloc – allocates and sets the chunk to zero
  - malloc – allocates the chunk
  - realloc – changes size in both directions but increase could be time consuming (not enough memory after allocated chunks)
  - mmap – great for large files, immediately free'd by unmap
  - obstack – pool of memory containing stack objects

# 2. Implementation level - malloc_trim

- undocumented function, not in POSIX
- useful in specific cases (parsing with libxml2)
- memory isn't free'd after call free -> inefficient memory
- chunks are reserved for application

# 2. Implementation level – fsync

```
Bad example:
open and read file ~/.kde/myconfig
fd = open("~/.kde/myconfig",
O_WRONLY|O_TRUNC|O_CREAT)
write(fd, bufferOfNewData,
sizeof(bufferOfNewData))
close(fd)
```

- Call fsync too much vs. never
- Examples: frozen Firefox, empty files

```
Better example:
open("~/.kde/myconfig", O_WRONLY|O_TRUNC|
O_CREAT); read(myconfig);
fd = open("~/.kde/myconfig.suffix", O_WRONLY|
O_TRUNC|O_CREAT)
write(fd, bufferOfNewData,
sizeof(bufferOfNewData))
fsync; /* optional - to be 100% sure */
some_operation() {}
close(fd)
rename("~/.kde/myconfig", "~/.kde/myconfig~") /*
optional */
rename("~/.kde/myconfig.suffix",
"~/.kde/myconfig")
```

# 2. Implementation level - wake-ups

- wake-ups/spin disk
- bug hunt for power hungry application found some of them
- rewrite checks
  - better design
  - inotify

```
Package gdm
#define
SC_SECURITY_TOKEN_MONITOR_POLL_INTER
VAL 100      /* ms */
```

```
Package hplip rhbz#204725
/* in seconds */
        try:
-               loop(timeout=0.5)
+               loop(timeout=5.0)
```

# 2. Implementation level – wake-ups

- inotify watch predefined changes on file/directory since kernel 2.6.13
- since kernel 2.6.31 new API fanotify
- problem: number of watches
  /proc/sys/fs/inotify/max_user_watches

```
int fd;
fd = inotify_init();
int wd;
wd = inotify_add_watch(fd,
“./myConfigFile”, IN_MODIFY);
...
fd_set rdfs;
struct timeval tv;
int retval;
FD_ZERO(&rfds);
FD_SET(0, &rfds);

tv.tv_sec = 5; /* 5 sec */
value = select(1, &rdfs, NULL, NULL, &tv);
if (value == -1)
   perror(“select()”);
else {
   do_some_stuff();
}
...
```

# 3. Compilation level I

**gcc** optimization flags

- **–O0** - no optimization of result binary, optimized the time of compilation, implicit
- **–O1 (–O)** - optimization which does not want a lot of time and improve the speed of the result binary
- **–O2** - used in fedora flags, all optimization which improve the speed of result binary except of space-speed tradeoff

# 3. Compilation level II

- **-O3** - all optimizations which improve the speed and can't be problematic
- **-Os** - all optimizations which reduce the size of binary
- **-march=name** - name of the target architecture

# 4. Profilers (bottleneck) I

- "*We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.*"
  - Donald Knuth
- "*Bottlenecks occur in surprising places, so don't try to second guess and put in a speed hack until you have proven that's where the bottleneck is.*"
  - Rob Pike

# 4. Profilers (bottleneck) II

- `gprof`
  - needs to have profiling binary and the libraries it need recompiled with `–pg` flag
- `oprofile`
  - uses kernel module and daemon which have to be start
  - quite complicated managing
- `valgrind --tool=callgrind`

# 4. Profilers (bottleneck) III

```
valgrind --tool=callgrind [copt] program [popt]
```

- create the data file `callgrind.out.$pid`

```
callgrind_annotate  callgrind.out.$pid
```

- create the profiling for processes which finished

```
callgrind_control callgrind.out.$pid
```

- create the profiling for processes which are running

# 4. Profilers (bottleneck) IV

- useful options:

`--treshold=<0-100>`
set the percentage of counts we are interested in

`--tree=none|caller|calling|both`
print for each function their callers, the called function, both

`--inclusive=yes|no`
add subroutine cost to function calls

# 4. Profilers (bottleneck) V

```
1,274,566  PROGRAM TOTALS
--------------------------------------------------------------------------------
       Ir  file:function
--------------------------------------------------------------------------------
1,162,636  *  gzip-1.3.12/gzip.c:main [gzip-1.3.12/gzip]
       48  >   /usr/src/debug/glibc/signal/sigempty.c:sigemptyset (1x) [/lib/libc-2.9.so]
      180  >   /usr/src/debug/glibc/signal/sigaddset.c:sigaddset (6x) [/lib/libc-2.9.so]
      186  >   /usr/src/debug/glibc/signal/sigismem.c:sigismember (6x) [/lib/libc-2.9.so]
      902  >   gzip-1.3.12/util.c:gzip_base_name (1x) [gzip-1.3.12/gzip]
1,150,531  >   gzip-1.3.12/gzip.c:treat_file (1x) [gzip-1.3.12/gzip]

1,150,531  *  gzip-1.3.12/gzip.c:treat_file [gzip-1.3.12/gzip]
       77  >   gzip-1.3.12/lib/open-safer.c:open_safer (1x) [gzip-1.3.12/gzip]
      835  >   gzip-1.3.12/util.c:xunlink (1x) [gzip-1.3.12/gzip]
    2,024  >   gzip-1.3.12/gzip.c:open_and_stat (1x) [gzip-1.3.12/gzip]
1,136,108  >   gzip-1.3.12/zip.c:zip (1x) [gzip-1.3.12/gzip]
```

# 4. Profilers (bottleneck) VI

```
1,136,108  *  gzip-1.3.12/zip.c:zip [gzip-1.3.12/gzip]
       15  >   gzip-1.3.12/util.c:updcrc (1x) [gzip-1.3.12/gzip]
      864  >   gzip-1.3.12/util.c:flush_outbuf (1x) [gzip-1.3.12/gzip]
       63  >   gzip-1.3.12/util.c:gzip_base_name (1x) [gzip-1.3.12/gzip]
1,024,770  >   gzip-1.3.12/deflate.c:deflate (1x) [gzip-1.3.12/gzip]

1,024,770  *  gzip-1.3.12/deflate.c:deflate [gzip-1.3.12/gzip]
  269,438  >   gzip-1.3.12/trees.c:flush_block (1x) [gzip-1.3.12/gzip]
  378,069  >   gzip-1.3.12/deflate.c:longest_match (2082x) [gzip-1.3.12/gzip]
   98,223  >   gzip-1.3.12/trees.c:ct_tally (2443x) [gzip-1.3.12/gzip]
       77  >   gzip-1.3.12/deflate.c:fill_window (1x) [gzip-1.3.12/gzip]
```

# 4. Profilers (bottleneck) VII

```
1,274,566  PROGRAM TOTALS
--------------------------------------------------------------------------------
       Ir  file:function
--------------------------------------------------------------------------------
1,162,857  /usr/src/debug/glibc-/csu/libc-start.c:(below main) [/lib/libc-2.9.so]
1,162,636  gzip-1.3.12/gzip.c:main [gzip-1.3.12/gzip]
1,150,531  gzip-1.3.12/gzip.c:treat_file [gzip-1.3.12/gzip]
1,136,108  gzip-1.3.12/zip.c:zip [gzip-1.3.12/gzip]
1,024,770  gzip-1.3.12/deflate.c:deflate [gzip-1.3.12/gzip]
  378,069  gzip-1.3.12/deflate.c:longest_match [gzip-1.3.12/gzip]
  269,438  gzip-1.3.12/trees.c:flush_block [gzip-1.3.12/gzip]
  188,152  gzip-1.3.12/trees.c:compress_block [gzip-1.3.12/gzip]
  122,698  gzip-1.3.12/bits.c:send_bits [gzip-1.3.12/gzip]
  110,359  /usr/src/debug/glibc-20081113T2206/elf/rtld.c:_dl_start [/lib/ld-2.9.so]
  109,068  /usr/src/debug/glibc-20081113T2206/elf/rtld.c:dl_main [/lib/ld-2.9.so]
   98,223  gzip-1.3.12/trees.c:ct_tally [gzip-1.3.12/gzip]
```

# Materials, links

- Ulrich Dreppers slides

http://people.redhat.com/drepper

- gcc documentation

http://gcc.gnu.org/onlinedoc/gcc-4.4.1/gcc

- profilers documentation

http://sourceware.org/binutils/docs/gprof

http://oprofile.sourceforge.net/doc/index.html

- valgrind documentation

http://valgrind.org/docs/manual/cl-manual.html

# Materials, links

- Python threads

http://docs.python.org/c-api/init.html?highlight=gil

http://code.google.com/p/unladen-swallow/

- Perl threads

http://www.perlmonks.org/?node_id=288022

- fsync problems

http://thunk.org/tytso/blog/2009/03/15/dont-fear-the-fsync/

- inotify

http://www.linuxjournal.com/node/8478/print