Perl Buildoptionen und Performance

Hintergrund

- Upgrade GeNUGate von OpenBSD 4.4 auf 4.6
- damit verbunden Upgrade von perl5.8.8 auf perl5.10.x
- plötzlich war ein spürbarer Performanceunterschied
- ähnliches Problem schon mal mit Upgrade von OpenBSD 3.7
- Ursachenfindung brachte Erkenntnisse über Buildoptionen, cc, ld, objdump...

benutzter Benchmark

```
1  for( my $i=0;$i<100000;$i++ ) {
2      my @x = map { $i } (1..100);
3      my $x = join('',@x);
4      @x = split('',$x);
5  }
```

- macht nichts anderes als Speicher zu verbrauchen und freizugeben
- nicht wirklich untypischer Code
- Tests mit ungepatchtem perl5.10.1 auf Ubuntu 10.04 und OpenBSD 4.6

usemymalloc

- weist Perl an, sein eigenes malloc statt das des Systems zu benutzen
- verwirrend benannte Option (my perl vs. my system)
- Das malloc von Perl ist besser auf die Zugriffspattern von Perl optimiert, das systemweite malloc ist jedoch besser auf das System angepasst.

usemymalloc - Ubuntu 10.04

- aus hints/linux.sh: `The system malloc() is about as fast and as frugal as perl's.`
- system perl wird mit usemymalloc=n gebaut
- Benchmark 15% schneller mit Perls eigenem malloc

usemymalloc - OpenBSD 4.6

- aus hints/openbsd.sh: `OpenBSD has a better malloc than perl...`
- system Perl wird mit default usemymalloc=n gebaut
- Benchmark braucht etwa 2,5 mal so lang wie mit usemymalloc=y !!
- verbraucht wesentlich mehr Speicher
- auf aktuellem OpenBSD (>3.7) kann man von dem systemeigenen malloc im Zusammenhang mit Perl nur abraten!

usethreads

- baut ein Perl, in welchem man Threads nutzen kann
- perl Threads nicht das was viele annehmen:
  - nur explizites Sharing
  - Thread erstellen sehr teuer
  - das zusätzliche Locking kostet Performance, auch wenn keine Threads benutzt werden
- perl Default ist ohne Threads

usethreads Ubuntu 10.04

- System perl ist mit Threads (entgegen Default)
- Benchmark 15% schneller ohne Threads

usethreads OpenBSD 4.6

- System perl ist ohne Threads
- ist auch besser so: mit Threads braucht der Benchmark 2,5 mal so lang!

useshrplib

- baut shared Library, die von anderen Applikationen und auch von Perl selber benutzt wird
- hat Overhead beim Laden von Perl (Relocation) und evtl. auch beim Ausführen da verlinkter Code indirekt angesprungen wird

useshrplib Ubuntu 10.04

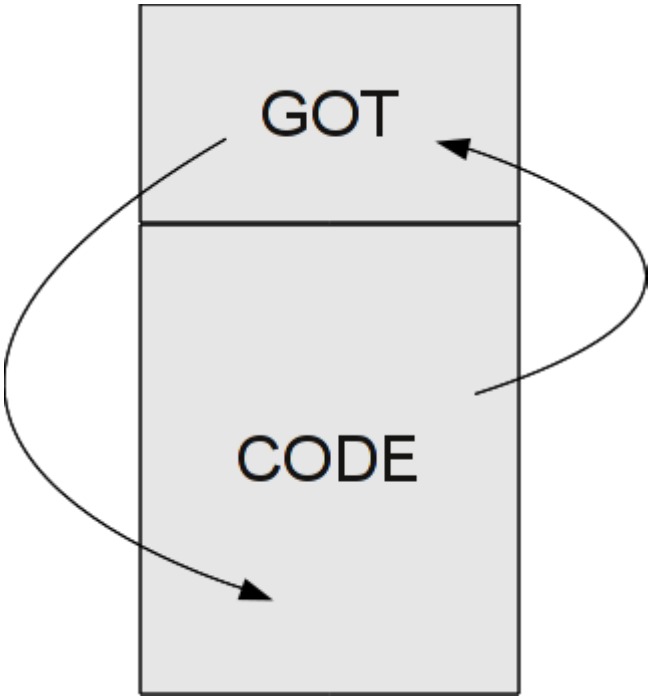
- System perl ist laut perl -V mit useshrplib gebaut
- es gibt auch eine libperl.so.
- Aber: ldd sagt, das libperl.so nicht verlinkt ist :)
- Benchmark mit useshrplib braucht ca. 20% länger als ohne

useshrplib OpenBSD 4.6

- System perl ist mit useshrplib gebaut
- Benchmark zeigt, das perl mit useshrplib ca. 25% länger als ohne braucht

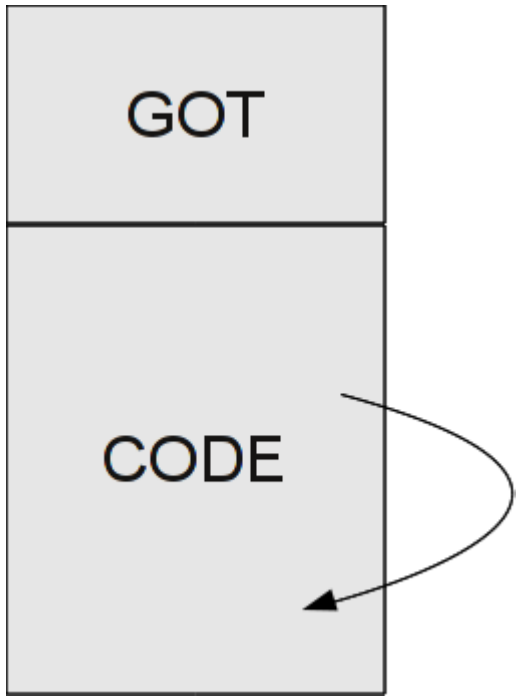
useshrplib Erklärung

- objdump -R libperl.so zeigt R_386_JUMP_SLOT für z.B. Symbol Perl_malloc
- d.h. es wird jedesmal über Sprungtabelle (GOT = Global Offset Table) aufgerufen



useshrplib Idee

- statt R_386_JUMP_SLOT wollen wir R_386_PC32, d.h. den direkten Aufruf



- mehr Erklärungen: http://www.codeproject.com/KB/library/elf-redirect.aspx

useshrplib Lösung

- -fPIC/-fpic: `Such code accesses all constant addresses through a global offset table (GOT)`
- cccdlflags='', d.h. *.c nicht mit -fPIC compilieren
- lddlflags+='-fPIC', d.h. *.o mit -fPIC zu *.so linken
- objdump -R libperl.so | grep Perl_malloc -> R_386_PC32, d.h. Funktion wird direkt aufgerufen
- Benchmark zeigt dann gleiche Performance wie ohne useshrplib

Zusammenfassung

- das mit dem System ausgelieferte Perl ist suboptimal bzw. stark beeinträchtigt.
- daher eigenes Perl bauen:
  - bei Ubuntu 10.04 bis zu 30% Gewinn
  - bei OpenBSD bis zu 170% Gewinn

strerror und locale

```
$fd = IO::Socket::INET->new(...);
$fd->blocking(0);
if ( ! sysread($fd,... )) {
  if ($!{EAGAIN}) { .. try later.. }
```

- strace zeigt Zugriff auf LC_MESSAGES Katalog wenn LANG!=C
- das kann bei I/O belastetem System die Sache spürbar verlangsamen
- Auswege:
  - LANG=C
  - i18n Unterstützung aus libc entfernen

Fragen?