diet libc

Felix von Leitner
`felix-dietlibc@fefe.de`

November 2001

Agenda

1. What is it?

2. Why?

3. Features

4. Goals

What is it?

- standard C runtime
- as small as possible, as fast as necessary
- implement stuff because it's *needed*, not because it's there
- aim for compliance to the SUSv2 and POSIX
- „be better than glibc“
- runs on x86, arm, sparc, alpha, ppc, mips, s390

Why?

- **because glibc sucks!** ;-)
- often, small software is superior (thttpd, djbware)
- Previous projects: libdjb, libowfat (replace large parts of libc)
- very basic security principle: separation
- many small programs are horribly inefficient with glibc
- create a libc that only has what is really necessary!

Are you stoned? Why not just enhance glibc?

„Wouldn’t that have been much easier?“

- glibc is beyond repair (`popen`, `inet_ntoa`, `inet_aton`, ...)
- Less hack value
- I have learned a lot about Unix and small software
- Ulrich Drepper is sometimes a little difficult to work with
- I probably wouldn’t be standing here right now ;-)

In short: enhancing glibc is all the pain with none of the gain.

Why not port the diet libc to *BSD?

- Personally, I don't like BSD.
- The BSD people are responsible for three thirds of the broken APIs that plague Unix today. The rest can be blamed on Sun.
- The BSD people would not accept it without $LICENSE==BSD (currently: GPL)
- Then, Microsoft could steal my code to make Windoze less obviously broken (can't let that happen).
- Technically, it would be easy as the code is portable (with minor exceptions like `openpty`).

Why is it GPL and not LGPL or BSD?

Because I don't want to be ripped off.

Everybody and his little brother is currently trying to do embedded stuff. Even Microsoft has jumped on the bandwagon and presented *Embedded NT* (giggle). Only two or three of those companies have contributed anything useful, the rest just sell other people's code.

So, if anyone wants to link a commercial application with the diet libc, he can talk to me about a commercial license.

Measuring efficiency: CPU cycles

Count CPU cycles for executing `main() { return 0; }`.

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc,char* argv[]) {
  long a,b;
  asm("rdtsc":"=a"(a)::"edx");
  if (!fork()) { execve(argv[1],argv+1,environ); exit(1); }
  wait(0);
  asm("rdtsc":"=a"(b)::"edx");
  printf("%lu cycles\n",b-a);
}
```

Measuring efficiency: memory usage

```
#include <stdio.h>

main() {
  FILE* f=fopen("/proc/self/statm","r");
  int size,resident,shared;
  if (fscanf(f,"%d %d %d",&size,&resident,&shared)==3)
    printf("%dK total size, %dK resident, %dK shared\n",
           size*4,resident*4,shared*4);
}
```

glibc vs. diet libc

	glibc	static glibc	diet libc
execve (mega-cycles)	1.340	0.330	0.215
binary size	4,751	432,883	1,528
binary size (stripped)	2,516	352,712	532
memory (shared)	296k	88k	12k
memory (non-shared)	52k	24k	20k

`ls -l` of 8 files linked against the diet libc takes 1.160 mega-cycles.

Of the 532 bytes, 110 bytes are code (rest: ELF headers and alignment).

Memory is allocated in 4k pages on ix86. 4k for the stack and 4k for `.bss` are unavoidable for normal C programs (`environ`, `errno`).

Does efficiency matter in the age of 2 GHz desktop CPUs?

Hell, yes! I wrote a small httpd called *fnord*. I run it from tcpserver (if you don't know: think inetd). I also wrote a http/ftp download util called *fget*.

```
for i in ‘seq 0 1000‘; do
  fget http://127.0.0.1:8000/index.html > /dev/null
done
```

configuration	output of time(1)
both diet	0.14s user 0.54s system 63% cpu 1.076 total
glibc fget	0.83s user 0.79s system 76% cpu 2.125 total
both glibc	0.73s user 0.95s system 53% cpu 3.114 total

That's roughly 2000 hits/second or 166.3 million hits/day (on my single-CPU 1.333 MHz Athlon desktop box)!

Why vary fget in the benchmark?

Think CGI! Many web servers are optimized for static content, but have to fork and exec for each CGI request!

This led to scripting languages being embedded or plugged into web servers and using multi-threading to tune performance. Both approaches sacrifice security for performance.

So, if you use *fnord* on an all-dynamic web server with CGI, you can still serve over 80 million hits/day, thanks to the diet libc!

Design Principles

- Good code is more important than programmer time.
- Don't copy. Rewrite! It's easier to kill bugs than to shave off bloat.
- Every byte counts.
- Don't do anything *„for completeness"* .
- Use every trick in the book to avoid dependencies
- Be multi-platform friendly (static headers)
- Dare to make platform dependent speed-ups if it's worth it

Supported Functionality

- all (?) system calls; opendir, readdir, closedir; POSIX signals
- gmtime, localtime (with /etc/localtime), mktime, ctime, asctime
- stdio including *printf and *scanf
- libpthread
- libdl and ld.so (x86 and ARM only so far, alpha quality)
- libm (x86 only, so far)

- malloc, realloc, free
- librpc (man, that was`^H^H^H`is ugly!)
- get(mnt,proto,serv,net,ut,gr,pw,sp)ent, syslog
- partial libresolv (dn_expand, res_mkquery, res_query...)
- gethostby(name,addr)/2, getaddrinfo, getnameinfo
- crypt including md5crypt
- regcomp, regexec!
- large file support, 32-bit UID support

- qsort and bsearch
- networking glue: ntoh[sl], hton[sl], if_indextoname, if_nametoindex
- ctype (isalpha, isalnum, tolower) for ASCII and latin1
- iconv (UTF8 <-> latin1 only so far)
- fnmatch, glob, ftw (not perfect yet)
- rand and *rand48
- mk[sd]temp, openpty, getopt, getopt_long, getopt_long_only

Enough already! ;-)

What’s Missing

- a wealthy and generous sponsor
- regerror
- ports to IA64, 68k PA-RISC and Super-H (hehe)
- locale (that won’t change, locale sucks)
- a little bit here and there

S/390 is in progress. Super-H will be done soon, if only because uC-libc supports it.

Minimal Internal Dependencies

- getservent and friends do not use fopen
- atexit handling is only linked in if atexit is used
- printf does not reference stdio (this was tricky)
- floating point support for printf can be configured away

Help programmers write small code

We use linker warnings to guilt programmers into writing better code:

1. warn if assert is used (may indicate debug code)
2. warn if stdio is used (bloat)
3. warn if sprintf is used (snprintf is better)
4. warn if *printf/*scanf is used (bloat, use -lowfat)
5. warn if system is used (security risk)
6. warn if mktemp/tempnam/tmpnam is used (race condition)

Using the diet libc

Installation:

```
$ make
[...]
$ sudo make install
```

Using:

```
$ diet gcc -o program main.c
$ CC="diet gcc -static -nostdinc" ./configure \
  --prefix=/opt/diet
```

Cross Compiler Friendliness

- no dynamically generated code or headers
- no dependencies on external headers
- the same header files for all platforms (no *bits/* directory like glibc)
- does not depend on kernel headers at all (no *linux/errno.h* like glibc)
- Easy integration: `diet arm-linux-gcc -o program main.c`

Which programs work?

Surprisingly many (besides my own software).

sysvinit, net-tools (ifconfig), util-linux (mount, fdisk), gzip, bzip2, wget, syslogd, busybox, boa, ash, thttpd, wireless-tools, hdparm, gpm, mtr, lame, mpg123, portmap, openssl, openssh, ncurses, zsh, gawk, less, webfsd, links (!), vim (!!), mutt (has regex issues), and a whole lot more.

A few utils were even written expressly for the diet libc: embutils, mininet, minit, fgetty, fget and fnord (from me) and diethotplug (will probably be part of kernel 2.5). But first: -lowfat.

libowfat

(It is spelled l-i-b-o-w-f-a-t, but it’s pronounced „-lowfat“ ;-))

libowfat reimplements the internal helper APIs from Dan Bernstein’s great software. Example:

```
#include <buffer.h>
int main() {
/* equivalent to: printf("%d\n",23); */
  buffer_putulong(buffer_1,23);
  buffer_putsflush(buffer_1,"\n");
}
```

printf: 5.8k, -lowfat: 1.2k

embutils

My answer to busybox.

Reimplements: arch, basename, cat, chmod, chown/chgrp, chroot, chvt, clear, cp, dd, df, dirname, dmesg, domainname, echo, env, false, head, hostname, id, install, kill, ln, ls, md5sum, mkdir, mknod, mv, pwd, rm, rmdir, sleep, sync, tar, tee, tr, true, tty, uname, uniq, wc, which, whoami, yes.

Also comes with SOS versions of cp, ln, ln -s, mv, rm.

mininet

Small implementations of ping and host. Also planned: ifconfig, route, arp, ...

minit

New init. Smaller and more powerful than sysvinit.

Features:

1. dependencies
2. dynamic service reconfiguration
3. does not use /proc, System V IPC, IP or Unix Domain Sockets
4. works completely on a read-only file system
5. still, /sbin/minit is only 7k

fgetty

Features:

1. Uses checkpassword (another Dan Bernstein invention)
2. supports pluggable authentication via checkpassword
3. comes with a checkpassword with /etc/shadow and MD5 passwords support
4. Still, /sbin/fgetty is only 7k

fget (http/ftp download util)

Features:

1. None, really. Can download a single URL and output it on stdout.
2. HTTP and FTP
3. IPv6 and IPv4
4. `http_proxy` and `ftp_proxy` or no proxy

fnord (web server)

Features:

1. run by tcpserver
2. static files only (CGI can be enabled with #define)
3. can rewrite URLs to enable gzip compression
4. virtual servers
5. surprisingly efficient
6. still only 15k

To Be Done

- make the dynamic linker and libm work for all platforms
- a test suite!
- look at more programs, e.g. X, kaffe ...
- write more diet versions of old programs
- dietlibc++?
- a diet Java runtime?

Tricks we used: unified syscalls

Share the actual syscall code. Each syscall just writes the syscall number to the register and then jumps to the shared code that copies the data from the stack/saves and shuffles registers, jumps into the kernel and sets `errno` if there was an error. That reduces each syscall to a few bytes.

Tricks we used: weak ELF symbols

ELF defines *„weak symbols“*. Weak symbols are only used to resolve references if there is no normal symbol of the same name. If a library contains both a weak and a normal symbol, the normal symbol is used.

If the code contains a weak symbol and a reference to it, the linker will not look for another symbol of that name.

What about uC-libc?

uC-libc is an older project than the diet libc.

It appears to be in the same efficiency class as the diet libc (i.e. much better than glibc), and supports Super-H. However, in every comparison I did, the diet libc produced smaller binaries and was normally faster, too.

What I dislike about uC-libc is that they copy so much code. That is the easier way, but it leads to the Dark Side Of The Force. If you copy code, you can't be much better then the place you copied the code from. Most of the code in uC-libc appears to be taken from other projects (stdio, libm and the dynamic loader, for example).

Your mileage may vary ;-)

Differences from glibc

1. openpty only works with devptsfs
2. getopt and friends don't sort argv
3. need `_BSD_SOURCE` to get `caddr_t`, `ulong`, `u_long` et al
4. need `_GNU_SOURCE` to get `u_int32_t` et al
5. the fake locale, of course
6. regcomp does not support non-extended regular expressions
7. (probably many more)

Where do we go from here?

The diet libc and libowfat are just the foundation. We need to provide more bloat-free building blocks. For example, the world needs a small standards compliant Bourne Shell!

The world has enough everything-linked-in ultra-glue packages like perl, python, tcl, mozilla, java. What is needed is small tools that do a small and well-defined job, but do it really good.

I'm planning to write an LDAP server (if you know LDAP on protocol and concept level, please come to me after the talk!)

If You Want To Help

The project home page is http://www.fefe.de/dietlibc/.

First, you should join the mailing list and read the TODO file from the latest CVS version.

If you are not interested in creating a test suite for the diet libc, the synergy would probably be greater if you helped with embutils or mininet, because other than that, what remains is mostly bug fixing.

Other than that: Sponsor the project! Buy a commercial license from me! Invite me and my family to your house in the caribbean! Have fun!

Or help me write that LDAP server...

Finally! The Last Slide!

Thanks for listening to my sermon!

For the slow writers: `http://www.fefe.de/dietlibc/`

Any questions?

People with too much time or money on their hands (preferably both) and LDAP experts: please join me after the questions for some small talk!