

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

Oberon 07 compiler

Port to Linux/i386

Norayr Chilingaryan

chnorik@gmail.com

norayr.livejournal.com



List of topics

- Concepts
 - Why Linux/i386
 - Vision
- Implementation
 - Technical decisions
 - Code patterns
- Future plans
 - Future plans and improvements

Why Linux/Intel?

- Provide powerful and high performance alternative to traditional C to develop native Unix applications
 - Console applications
 - Server applications
 - Desktop/GUI applications
- x86 is a dominant desktop & server architecture

- **Standalone, commandline compiler with Unix integration**
 - Full workflow (development, debug, linking) possible in plain console
 - Remote development/debugging (server systems)
 - Automations of builds, patches, compilations...
 - Easy integration with graphical IDE's
- **No special environment/library requirements**
 - both for compiler and produced code should be native x86 object files
- **Library with support for native API (kernel syscalls)**
 - Unix system programming
 - `fork()`, `exec()`, `pipe()`, `mmap()`...
- **Support for native (read: C) calling conventions**

Is Oberon language alien to Unix environment?

finding Unix & Oberon things in common

- Minimalism
- Scalability
- Unpretentious hardware requirements
- High performance
- Modularity (to some extent)
- Dynamic loading of shared objects
- Enthusiastic usersbase

Combining spirit of Oberon with Unix traditions

- Lightning fast compilation of resulting binaries
- Short time to compile a compiler
- Reasonably small size & memory requirements of generated code
- Reasonably efficient, not overoptimized code

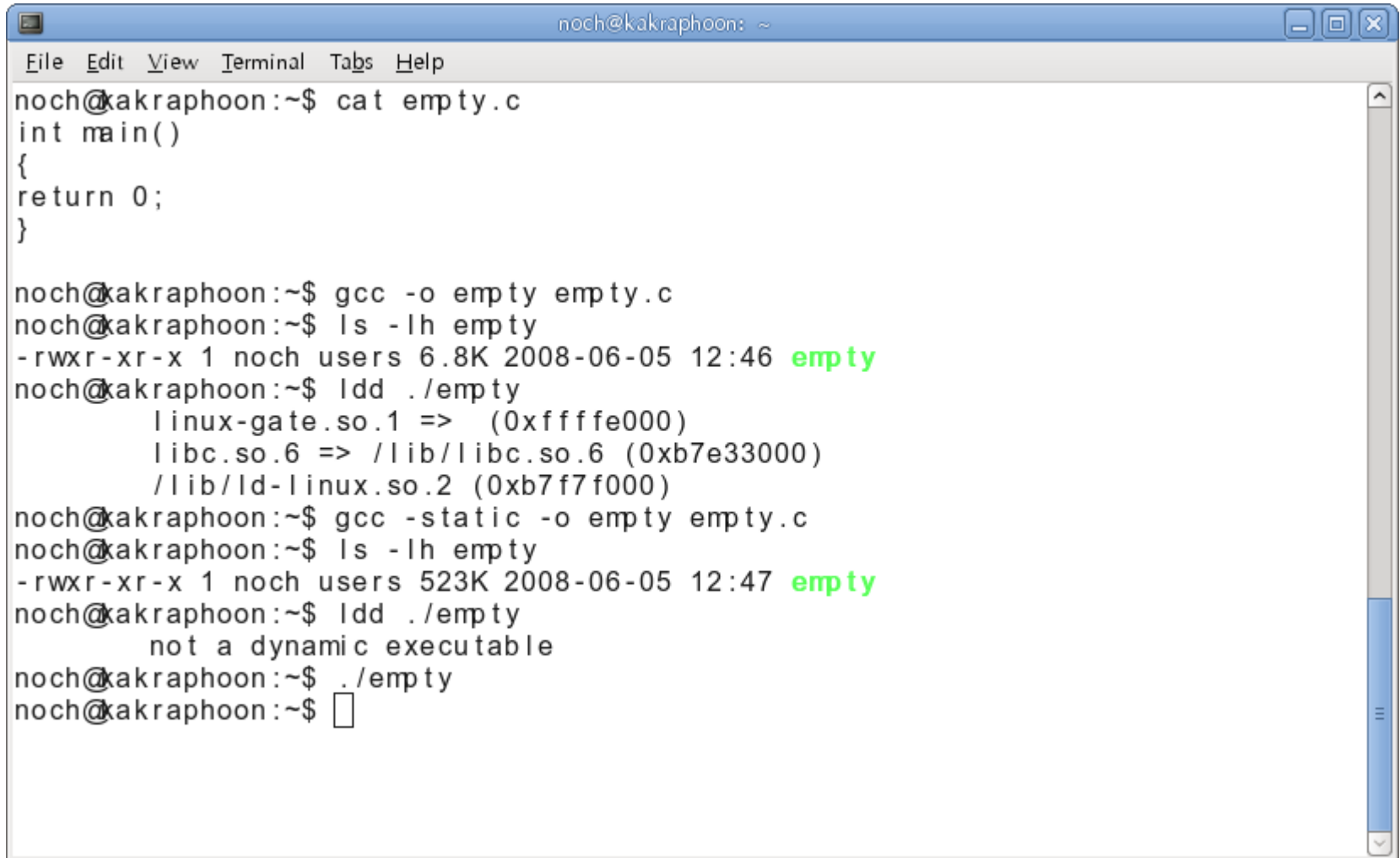
Finding compromise

- To not follow GNU/Linux traditions when it contradicts to Oberon spirit (see above)
 - Decided to not use gcc

To not use gcc

- Because it is not in the spirit of Oberon
 - Slow compilation time
 - produce overoptimized code
 - Recompilation of a compiler takes hours
 - Fat produced binaries
 - High memory requirements of produced binaries
 - Unstable interfaces

Fat code



```
noch@kakraphoon: ~  
File Edit View Terminal Tabs Help  
noch@kakraphoon:~$ cat empty.c  
int main()  
{  
return 0;  
}  
  
noch@kakraphoon:~$ gcc -o empty empty.c  
noch@kakraphoon:~$ ls -lh empty  
-rwxr-xr-x 1 noch users 6.8K 2008-06-05 12:46 empty  
noch@kakraphoon:~$ ldd ./empty  
linux-gate.so.1 => (0xffffe000)  
libc.so.6 => /lib/libc.so.6 (0xb7e33000)  
/lib/ld-linux.so.2 (0xb7f7f000)  
noch@kakraphoon:~$ gcc -static -o empty empty.c  
noch@kakraphoon:~$ ls -lh empty  
-rwxr-xr-x 1 noch users 523K 2008-06-05 12:47 empty  
noch@kakraphoon:~$ ldd ./empty  
not a dynamic executable  
noch@kakraphoon:~$ ./empty  
noch@kakraphoon:~$
```

optimizations: harm or benefit?

In the thread at LKML <http://lkml.org/lkml/2007/10/25/186> about troubles in Linux kernel development as a consequence of gcc strong optimizations Linus Torvalds stated:

I have to admit that for the last five years or so, I've really wanted some other compiler team to come up with a good open-source compiler. Exactly due to issues like this (Q: "Gcc creates bogus code that doesn't work!" A: "It's not bogus, it's technically allowed by the language specs that don't talk about xyz, the fact that it doesn't work isn't our problem").

Memory hunger

Pascal Free Pascal benchmarks | Gentoo : Intel® Pentium® 4 Computer Language Benchmarks Game

File Edit View Go Bookmarks Tabs Help

Back Forward Stop Reload Home History Bookmarks Smaller Larger

http://shootout.alioth.debian.org/gp4/benchmark.php?test=all&lang=fpascal&lang2=gpp

LKML: Nick Pig... Pascal Free Pasc...

How many times better?

How many times *faster* or *smaller* are the **Pascal Free Pascal** programs than the corresponding C++ GNU g++ programs?

| Program & Logs | Pascal Free Pascal <i>x times</i> better ~ C++ GNU g++ <i>x times</i> better | | |
|------------------------------------|---------------------------------------------------------------------------------|---------------------|---------------------|
| | Faster | Smaller: Memory Use | Smaller: GZip Bytes |
| binary-trees | 1.1 | 1.7 | ~1.4 |
| chameneos-redux | No program | | |
| fannkuch | 1.2 | 8.8 | ~1.3 |
| fasta | ~1.4 | 2.1 | 1.0 |
| k-nucleotide | ~1.3 | ~1.1 | ~1.7 |
| mandelbrot | ~2.0 | 9.0 | 2.1 |
| meteor-contest | ~1.8 | ~1.7 | ~1.0 |
| n-body | ~1.2 | 3.4 | 1.3 |
| nsieve | ~1.0 | 1.1 | 1.0 |
| nsieve-bits | ~1.1 | 1.2 | 1.0 |
| partial-sums | ~1.0 | 3.1 | 1.1 |
| pidigits | 1.0 | 1.7 | ~1.2 |
| recursive | ~1.7 | 2.2 | 1.2 |
| regex-dna | No program | | |
| reverse-complement | ~1.3 | 1.0 | 1.1 |
| spectral-norm | ~1.2 | 3.5 | 1.0 |
| startup | 3.0 | | 1.6 |
| sum-file | 1.3 | 9.3 | 1.4 |
| thread-ring | | | No GNU g++ |

about Pascal Free Pascal

Enter a web address to open, or a phrase to search for

Unstable interfaces

From gcc mailing list:

«interfacing to gcc internals is strongly discouraged unless this is going to be part of gcc itself. We can't allow outside projects to use gcc internals. We can't guarantee stability of interfaces, and we also need to prevent people from trying to violate the GPL. This is an FSF policy. In order to protect the value of GCC, and in order to prevent people from using devious methods to circumvent the GPL, we are not allowed to let outside projects use gcc internals»

<http://gcc.gnu.org/ml/gcc/2003-07/msg00247.html>

«Of course, the code is GPL, so you can write your own interfaces if you want, but we will not be able to accept the patches. GCC changes at such a rapid pace that it is very expensive to maintain your own patches, and hence this discourages most people from trying. If someone is able to do this using existing gcc features, then we may obfuscate the feature to prevent this use.»

<http://gcc.gnu.org/ml/gcc/2003-07/msg00437.html>

Technical decisions

■ Generate assembly

- It makes life easier (c) :)
- No external dependency (like libelf)
- Not necessary to implement elf object file format generation
- Good abstraction for all object file formats (elf, a.out, coff, misc, pe)
- Simplifies linking

■ Use GNU assembler as a backend

- «as» available on most platforms, present in major Linux distributions by default
- low level constructs (movb, movsbl)
- Compatible with traditional for Unix AT&T syntax

What is Linux API ?

- GNU C library
 - + exists on all desktop/server Linux distributions
 - - wrapper interface to kernel
 - - may be replaced by other implementations in some cases
(for example uclibc for embedded systems)
- Linux (POSIX compliant) kernel calls?

Why avoid wrappers?

And why avoid GNU libc?

- If libc change, we must change with it
- libc bugs will affect our work
- It is faster to call kernel calls directly
 - Instead of libc like wrapper functions, or mono/.net like environments
- More portable
 - In case of Linux this means supporting whole range of possible builds, not only selected distributions

Choice of tools

- **ooc (optimizing oberon-2 compiler) ooc.sf.net**
 - Uses C as a backend assembler
 - - Generates very optimized and unreadable code, which is very hard to debug
 - + Outstanding usability, automatic compilation of all necessary dependency modules,
 - + Comprehensive error messages at runtime, mentioning error position in code
 - + Good Library
- **Ofront (OP2 port by Josef Templ)**
 - Uses C as a backend assembler
 - + Generates very readable code which is easy to debug
 - - Requires writing makefile
 - - Limited library

Debugging GSA optimized code

The screenshot shows the DDD (Data Display Debugger) interface. The title bar indicates the current file is `DDD: /home/noch/freeoberon/obj/OSAP.c`. The menu bar includes `File`, `Edit`, `View`, `Program`, `Commands`, `Status`, `Source`, and `Data`. The toolbar contains icons for `Lookup`, `Find`, `Clear`, `Watch`, `Print`, `Display`, `Plot`, `Show`, `Rotate`, `Set`, and `Undo`. The main window displays the following C code:

```
126: OSAP__Get((void*)(OOC_INT32)&OSAP__syn);
127:
127:   i0 = OSAP__syn;
127:   i0 = i0==31;
127:   if (i0) goto 112_loop;
132:
132: OSAP__Check(52, (OOC_CHAR8*)"no ;", 5);
133:
133: OSAP__Declarations(10, (void*)(OOC_INT32)&OSAP__dc, (void*)(OOC_INT32)&dunny, 1u);
133:   i0 = OSAP__syn;
133:   i1 = i0==66;
133:   if (!i1) goto 142;
137_loop:
137_loop: OSAP__ProcedureDecl();
137_loop: OSAP__Check(52, (OOC_CHAR8*)"no ;", 5);
137_loop:   i0 = OSAP__syn;
137_loop:   i1 = i0==66;
137_loop:   if (i1) goto 137_loop;
142:
142: OSAP__Header();
142:   i0 = i0==67;
```

Below the code, the debugger's output shows assembly instructions and GDB commands:

```
x.label=-5(%ebp)
x.ilabel=
x.and = FALSE
(gdb) next
(gdb) next
(gdb) next
(gdb) [
```

On the right side of the window, there is a control panel titled "DDD" with the following buttons:

- Run
- Interrupt
- Step
- Stepi
- Next
- Nexti
- Until
- Finish
- Cont
- Kill
- Up
- Down
- Undo
- Redo
- Edit
- Make

Debugging Ofront produced code

The screenshot shows the DDD (Data Display Debugger) interface. The title bar indicates the current file is `DDD: /home/noch/treeoberon/src/OSAP.c`. The menu bar includes `File`, `Edit`, `View`, `Program`, `Commands`, `Status`, `Source`, `Data`, and `Help`. The toolbar contains icons for `Lookup`, `Find`, `Clear`, `Watch`, `Print`, `Display`, `Plot`, `Show`, `Rotate`, `Set`, and `Undo`.

The main window displays the source code for `foc.c:116`. The code is as follows:

```
                OSAS_Mark("id expected", 12L);
            }
        } else {
            __MOVE(inp1d, inp1d1, 32);
        }
        OSAB_Import(inp1d, inp1d1);
        OSAG_Import(inp1d);
        if (OSAP_sym == 40) {
            OSAS_Get(&OSAP_sym);
        } else if (OSAP_sym == 31) {
            OSAS_Mark("comma missing", 14L);
        }
    }
    OSAP_Check(52, "no ;", 5L);
}
OSAP_Declarations(10, &OSAP_dc, &dummy, 1);
while (OSAP_sym == 66) {
    OSAP_ProcedureDecl();
    OSAP_Check(52, "no ;", 5L);
}
OSAG_Header();
if (OSAP_sym == 67) {
```

The assembly window at the bottom shows the following instructions:

```
x.label=-5(%ebp)
x.ilabel=
x.and = FALSE
movl %ebp,%esp
popl %ebp
(gdb) [
.type test_test,
```

On the right side, a floating window titled "DDD" contains a control panel with the following buttons:

- Run
- Interrupt
- Step
- Stepi
- Next
- Nexti
- Until
- Finish
- Cont
- Kill
- Up
- Down
- Undo
- Redo
- Edit
- Make

Specific code templates avoiding register usage when possible

$a := 5$

```
movl $5, _a
```

$a := b + 5$

```
movl _b, %eax  
addl $5, %eax  
movl %eax, _a
```

$a := b \text{ DIV } c$

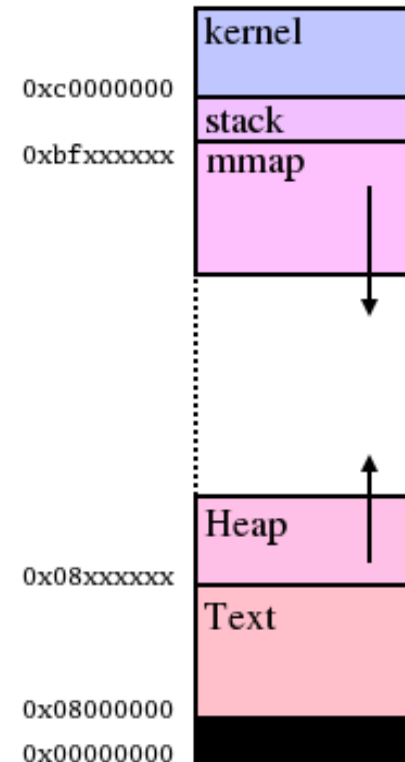
```
movl _b, %eax  
cld  
idivl _c  
movl %eax, _a
```

Memory layout & management

```
.section .data
.comm mmap_, 24
.section .text
movl $0, mmap_
# length of requested memory
#movl $65535, mmap_+4
# read, write, PROT_WRITE | PROT_READ, 0x02, 0x01
movl $3, mmap_+8
# map anonymously (0x20), map_private (0x2)
movl $34, mmap_+12
# fd, -1 for portability
movl $-1, mmap_+16
# offset is ignored
movl $0, mmap_+20
```

```
NEW(p)      movl $memsize, mmap_+4
            movl $90, %eax
            leal mmap_, %ebx
            int $0x80
            movl %eax, _ptr
```

```
DISPOSE(p)  movl $91, %eax
            movl _ptr, %ebx
            movl (%ebx), %ecx
            movl (%ecx), %ecx
```



Porting to other x86 platforms

It is necessary to change only 2 procedures in generator module:

```
PROCEDURE PutExit(i : INTEGER);  
PROCEDURE New*(VAR x, y: Item);
```

Low level library example, kernel interface

```
MODULE Unix;  
PROCEDURE write*(CONST s : ARRAY OF CHAR; I : INTEGER);  
BEGIN  
    ASM  
        movl 8(%ebp), %ecx  
        movl 16(%ebp), %edx  
        movl $1, %ebx  
        movl $4, %eax  
        int $0x80  
    END;  
END write;  
END Unix.
```

Low level library example, libc interface

```
MODULE Unix;  
PROCEDURE write*(CONST s : ARRAY OF CHAR; I : INTEGER);  
BEGIN  
    ASM  
        movl 8(%ebp), %ecx  
        pushl %ecx  
        call printf  
    END;  
END write;  
  
END Unix.
```

Standard Library example

```
MODULE Out;  
IMPORT Unix, IntStr;  
PROCEDURE String* (CONST s : ARRAY OF CHAR);  
VAR i : INTEGER;  
BEGIN  
    i := 0;  
    WHILE s[i] # 0X DO  
        INC(i);  
    END;  
    Unix.write(s, i)  
END String;  
...
```

Compile & run example programs benchmarks

Future plans & improvements

- Automatically resolve module hierarchy
(make file is not necessary)
- Improve register allocation
- Dynamic module loading(Pos-independent code)
- Improve usability, add debugging possibility,
separate directories for libs, config file etc...
- Compile other existing Oberon libraries
- Port to other x86 Unix systems
- Port to other architectures

Thank you!