Notes for linux release 0.01

0. Contents of this directory

linux-0.01.tar.Z - sources to the kernel
bash.Z - compressed bash binary if you want to test it
update.Z - compressed update binary
RELNOTES-0.01 - this file

1. Short intro

This is a free minix-like kernel for i386(+) based AT-machines. Full source is included, and this source has been used to produce a running kernel on two different machines. Currently there are no kernel binaries for public viewing, as they have to be recompiled for different machines. You need to compile it with gcc (I use 1.40, don't know if 1.37.1 will handle all __asm__-directives), after having changed the relevant configuration file(s).

As the version number (0.01) suggests this is not a mature product. Currently only a subset of AT-hardware is supported (hard-disk, screen, keyboard and serial lines), and some of the system calls are not yet fully implemented (notably mount/umount aren't even implemented). See comments or readme's in the code.

This version is also meant mostly for reading - ie if you are interested in how the system looks like currently. It will compile and produce a working kernel, and though I will help in any way I can to get it working on your machine (mail me), it isn't really supported. Changes are frequent, and the first "production" version will probably differ wildly from this pre-alpha-release.

Hardware needed for running linux:

- 386 AT
- VGA/EGA screen
- AT-type harddisk controller (IDE is fine)
- Finnish keyboard (oh, you can use a US keyboard, but not without some practise :-)

The Finnish keyboard is hard-wired, and as I don't have a US one I

cannot change it without major problems. See kernel/keyboard.s for details. If anybody is willing to make an even partial port, I'd be grateful. Shouldn't be too hard, as it's tabledriven (it's assembler though, so ...)

Although linux is a complete kernel, and uses no code from minix or other sources, almost none of the support routines have yet been coded. Thus you currently need minix to bootstrap the system. It might be possible to use the free minix demo-disk to make a filesystem and run linux without having minix, but I don't know...

2. Copyrights etc

This kernel is (C) 1991 Linus Torvalds, but all or part of it may be redistributed provided you do the following:

- Full source must be available (and free), if not with the distribution then at least on asking for it.

- Copyright notices must be intact. (In fact, if you distribute only parts of it you may have to add copyrights, as there aren't (C)'s in all files.) Small partial excerpts may be copied without bothering with copyrights.

- You may not distibute this for a fee, not even "handling" costs.

Mail me at "torvalds@kruuna.helsinki.fi" if you have any questions.

Sadly, a kernel by itself gets you nowhere. To get a working system you need a shell, compilers, a library etc. These are separate parts and may be under a stricter (or even looser) copyright. Most of the tools used with linux are GNU software and are under the GNU copyleft. These tools aren't in the distribution - ask me (or GNU) for more info.

3. Short technical overview of the kernel.

The linux kernel has been made under minix, and it was my original idea to make it binary compatible with minix. That was dropped, as the differences got bigger, but the system still resembles minix a great deal. Some of the key points are:

- Efficient use of the possibilities offered by the 386 chip. Minix was written on a 8088, and later ported to other machines - linux takes full advantage of the 386 (which is nice if you /have/ a 386, but makes porting very difficult)

- No message passing, this is a more traditional approach to unix. System calls are just that - calls. This might or might not be faster, but it does mean we can dispense with some of the problems with messages (message queues etc). Of course, we also miss the nice features :-p.

- Multithreaded FS - a direct consequence of not using messages. This makes the filesystem a bit (a lot) more complicated, but much nicer. Coupled with a better scheduler, this means that you can actually run several processes concurrently without the performance hit induced by minix.

- Minimal task switching. This too is a consequence of not using messages. We task switch only when we really want to switch tasks - unlike minix which task-switches whatever you do. This means we can more easily implement 387 support (indeed this is already mostly implemented)

- Interrupts aren't hidden. Some people (among them Tanenbaum) think interrupts are ugly and should be hidden. Not so IMHO. Due to practical reasons interrupts must be mainly handled by machine code, which is a pity, but they are a part of the code like everything else. Especially device drivers are mostly interrupt routines - see kernel/hd.c etc.

- There is no distinction between kernel/fs/mm, and they are all linked into the same heap of code. This has it's good sides as well as bad. The code isn't as modular as the minix code, but on the other hand some things are simpler. The different parts of the kernel are under different sub-directories in the source tree, but when running everything happens in the same data/code space.

The guiding line when implementing linux was: get it working fast. I wanted the kernel simple, yet powerful enough to run most unix software. The file system I couldn't do much about - it needed to be minix compatible for practical reasons, and the minix filesystem was simple enough as it was. The kernel and mm could be simplified, though:

- Just one data structure for tasks. "Real" unices have task information in several places, I wanted everything in one

place.

- A very simple memory management algorithm, using both the paging and segmentation capabilities of the i386. Currently MM is just two files - memory.c and page.s, just a couple of hundreds of lines of code.

These decisions seem to have worked out well - bugs were easy to spot, and things work.

4. The "kernel proper"

All the routines handling tasks are in the subdirectory "kernel". These include things like 'fork' and 'exit' as well as scheduling and minor system calls like 'getpid' etc. Here are also the handlers for most exceptions and traps (not page faults, they are in mm), and all low-level device drivers (get_hd_block, tty_write etc). Currently all faults lead to a exit with error code 11 (Segmentation fault), and the system seems to be relatively stable ("crashme" hasn't - yet).

5. Memory management

This is the simplest of all parts, and should need only little changes. It contains entry-points for some things that the rest of the kernel needs, but mostly copes on it's own, handling page faults as they happen. Indeed, the rest of the kernel usually doesn't actively allocate pages, and just writes into user space, letting mm handle any possible 'page-not-present' errors.

Memory is dealt with in two completely different ways - by paging and segmentation. First the 386 VM-space (4GB) is divided into a number of segments (currently 64 segments of 64Mb each), the first of which is the kernel memory segment, with the complete physical memory identity-mapped into it. All kernel functions live within this area.

Tasks are then given one segment each, to use as they wish. The paging mechanism sees to filling the segment with the appropriate pages, keeping track of any duplicate copies (created at a 'fork'), and making copies on any write. The rest of the system doesn't need to know about all this.

6. The file system

As already mentioned, the linux FS is the same as in minix. This makes crosscompiling from minix easy, and means you can mount a linux partition from minix (or the other way around as soon as I implement mount :-). This is only on the logical level though - the actual routines are very different.

> NOTE! Minix-1.6.16 seems to have a new FS, with minor modifications to the 1.5.10 I've been using. Linux won't understand the new system.

The main difference is in the fact that minix has a single-threaded file-system and linux hasn't. Implementing a single-threaded FS is much easier as you don't need to worry about other processes allocating buffer blocks etc while you do something else. It also means that you lose some of the multiprocessing so important to unix.

There are a number of problems (deadlocks/raceconditions) that the linux kernel needed to address due to multi-threading. One way to inhibit race-conditions is to lock everything you need, but as this can lead to unnecessary blocking I decided never to lock any data structures (unless actually reading or writing to a physical device). This has the nice property that dead-locks cannot happen.

Sadly it has the not so nice property that race-conditions can happen almost everywhere. These are handled by double-checking allocations etc (see fs/buffer.c and fs/inode.c). Not letting the kernel schedule a task while it is in supervisor mode (standard unix practise), means that all kernel/fs/mm actions are atomic (not counting interrupts, and we are careful when writing those) if you don't call 'sleep', so that is one of the things we can count on.

7. Apologies :-)

This isn't yet the "mother of all operating systems", and anyone who hoped for that will have to wait for the first real release (1.0), and even then you might not want to change from minix. This is a source release for those that are interested in seeing what linux looks like, and it's not really supported yet. Anyone with questions or suggestions (even bug-reports if you decide to get it working on your system) is encouraged to mail me.

8. Getting it working

Most hardware dependancies will have to be compiled into the system, and there a number of defines in the file "include/linux/config.h" that you have to change to get a personalized kernel. Also you must uncomment the right "equ" in the file boot/boot.s, telling the bootup-routine what kind of device your A-floppy is. After that a simple "make" should make the file "Image", which you can copy to a floppy (cp Image /dev/PS0 is what I use with a 1.44Mb floppy). That's it.

Without any programs to run, though, the kernel cannot do anything. You should find binaries for 'update' and 'bash' at the same place you found this, which will have to be put into the '/bin' directory on the specified root-device (specified in config.h). Bash must be found under the name '/bin/sh', as that's what the kernel currently executes. Happy hacking.

Linus Torvalds "torvalds@kruuna.helsinki.fi"
Petersgatan 2 A 2
00140 Helsingfors 14
FINLAND