POTATOES

Practical **O**riented **T**e**A**ching **T**ool,

Operating (and) **E**ducating **S**ystem

Worum geht es?

Ein Betriebssystem für x86-Rechner in C und Assembler

Ziele

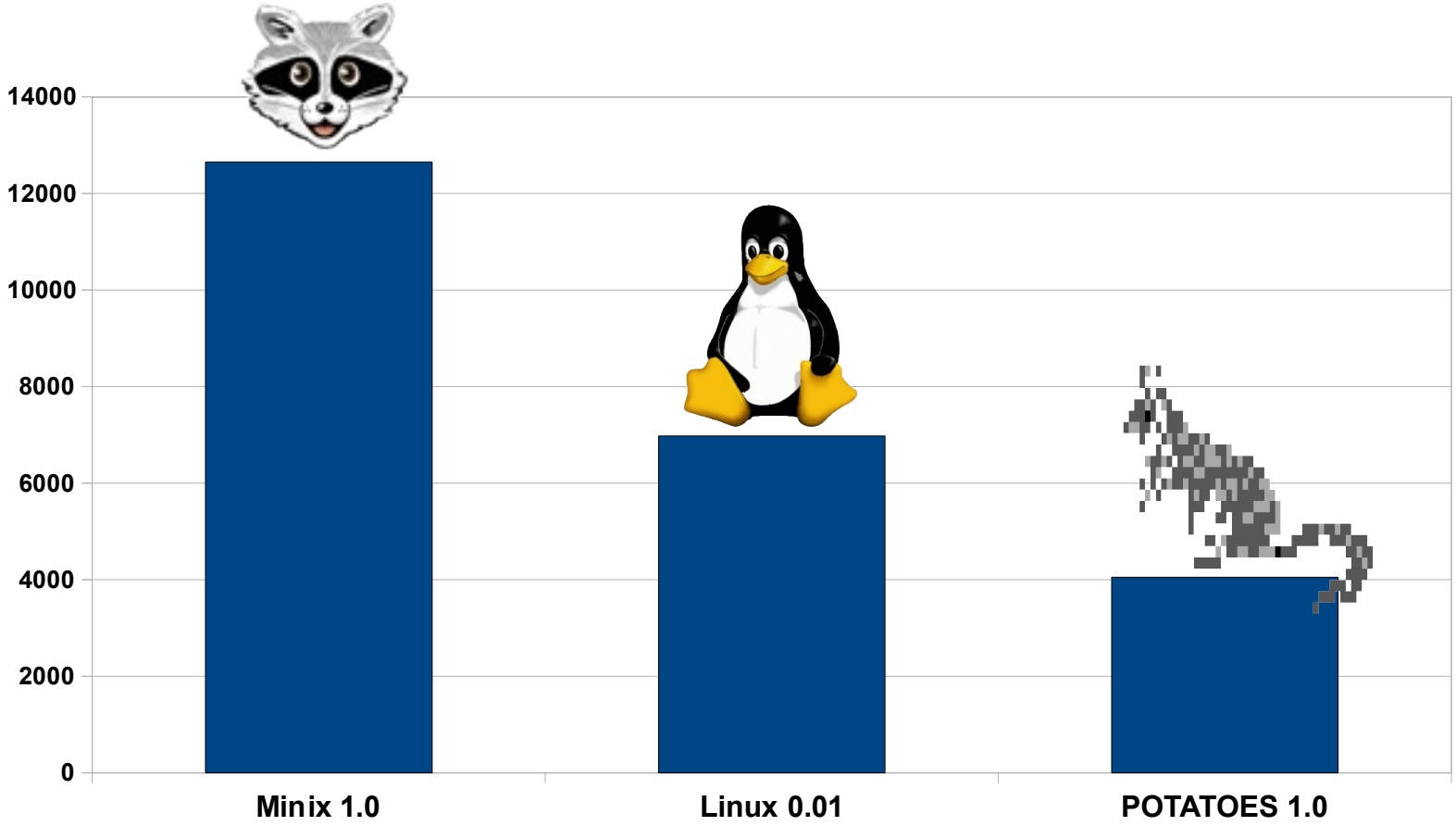
- **KISS:**

  Keep It Short & Simple

- Ein einfach verständliches open-source Betriebssystem für Lehr- bzw. Lernzwecke und eigene Experimente

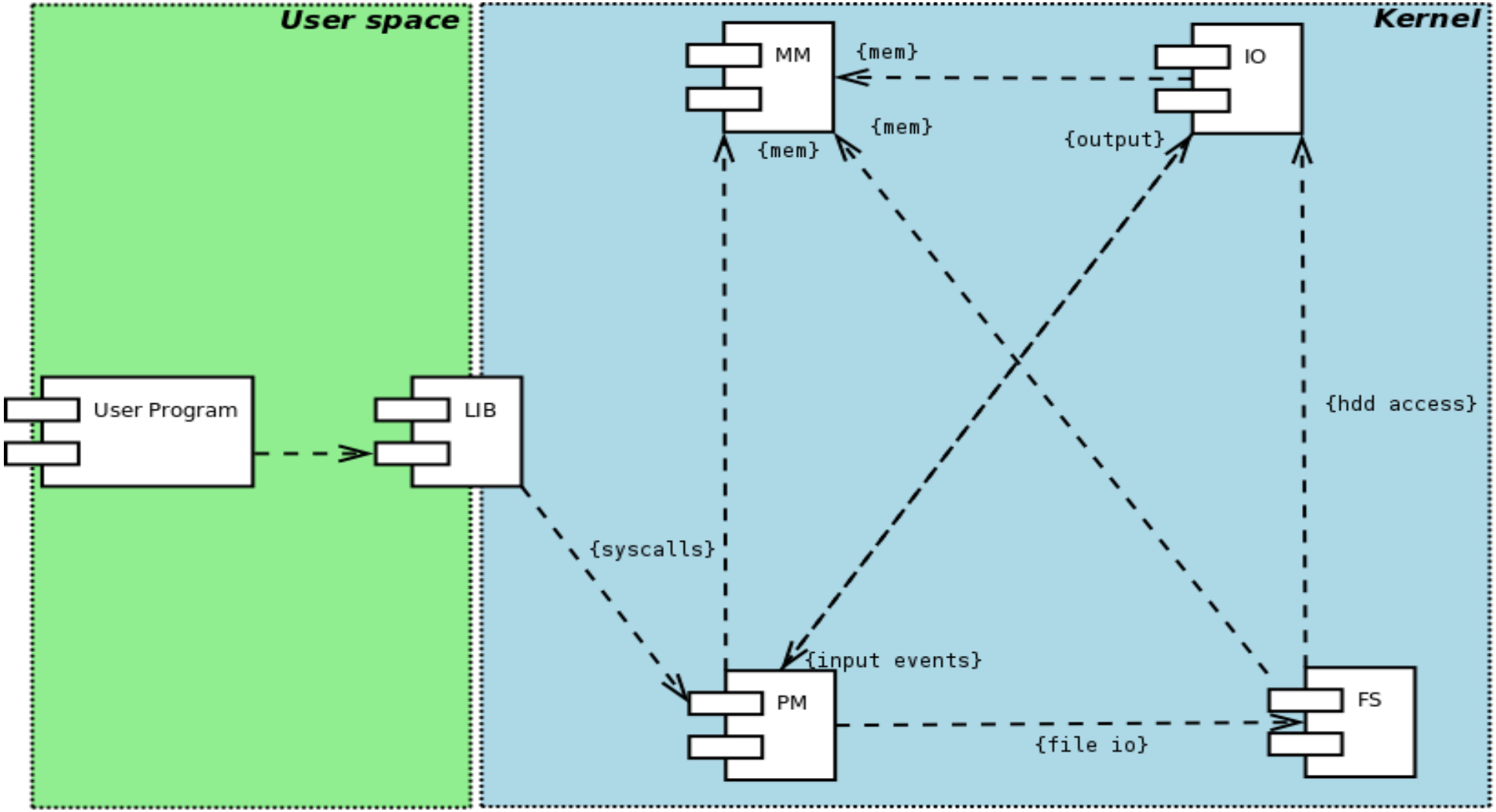
POTATOES im Vergleich

- Lines of Code (Kernel)



```
void main() {
    char* bar = malloc(foo);
    int foo = 42;

    while (TRUE)
        chaos(bar);

    return foo | 0x42;
}
```

POTATOES: Practical Oriented TeAching Tool, Operating (and) Educating System



Prozessverwaltung

- **Multitasking**
  - Round-robin Scheduler mit festem Zeitfenster
  - Gepufferte, separate Ein-/Ausgabe für jeden Prozess
- **System Calls**
  - Am POSIX-Standard orientiert
- **C-Bibliothek**
  - Teilmenge der libc
- **Gerätedateien** (device files)
  - /dev/clock, stdin, stdout, …

Speicherverwaltung

- **Virtuelle Adressierung**
  - Paging (ohne Swapping)
- **Heapverwaltung**
  - *first-fit* Allokation

Ein-/Ausgabe

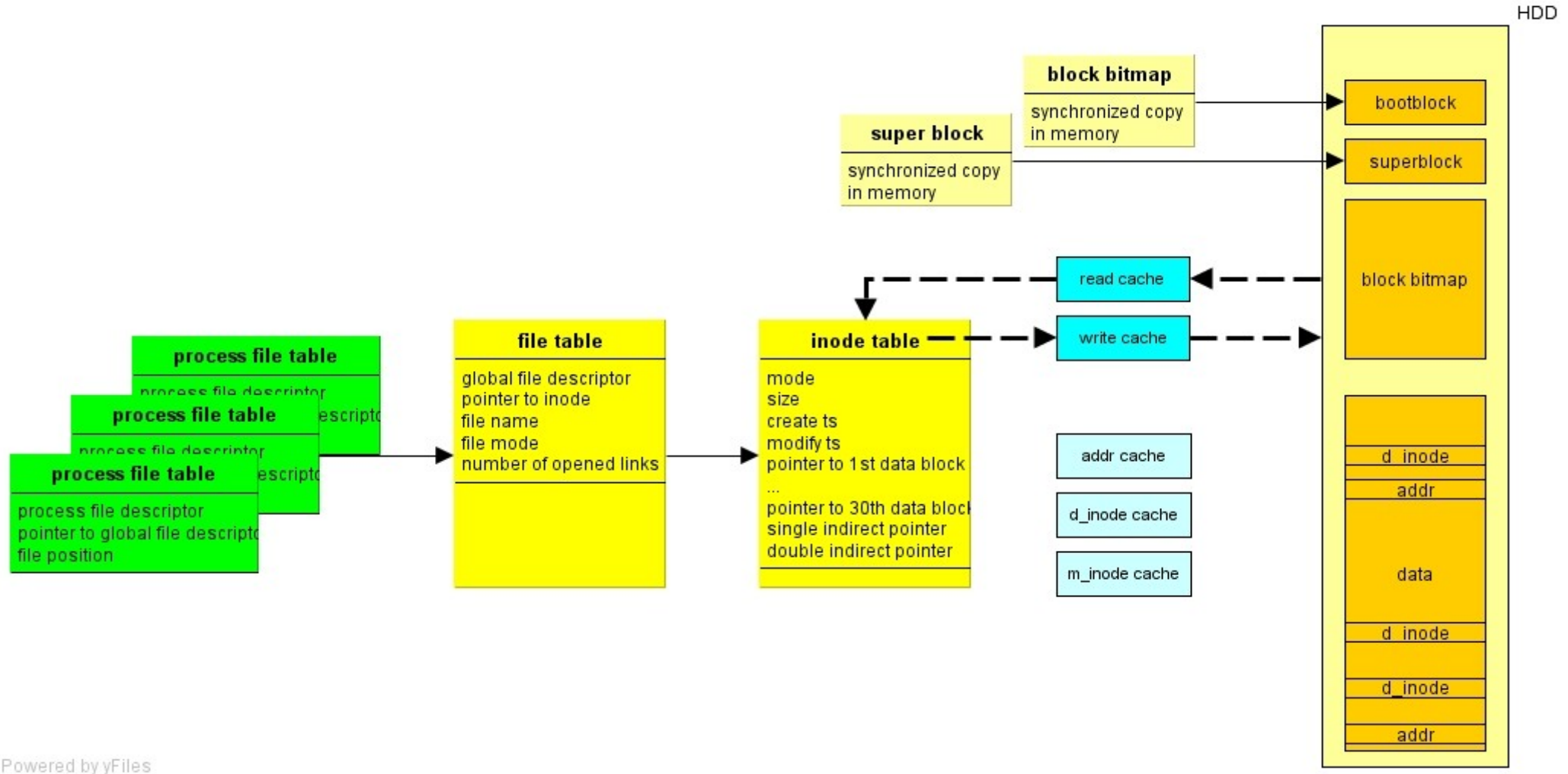
- **Hardwaretreiber**
  - Zeitgeber, Echtzeituhr
  - ATA-Festplattenzugriff
  - Bildschirm (VGA)
  - Tastatur
- **Interruptverwaltung**
- **Virtuelle Konsolen**

Dateisystem

- hierarchischer Aufbau
- basierend auf **inode-Konzept** (zweifach-indirekt)
- keine statische inode-Tabelle
- standardisiertes API:
  - create(), open(), read(), ...

```
void main() {
    char* bar = malloc(foo);
    int foo = 42;

    while (TRUE)
        chaos(bar);

    return foo | 0x42;
}
```

POTATOES: Practical Oriented TeAching Tool, Operating (and) Educating System



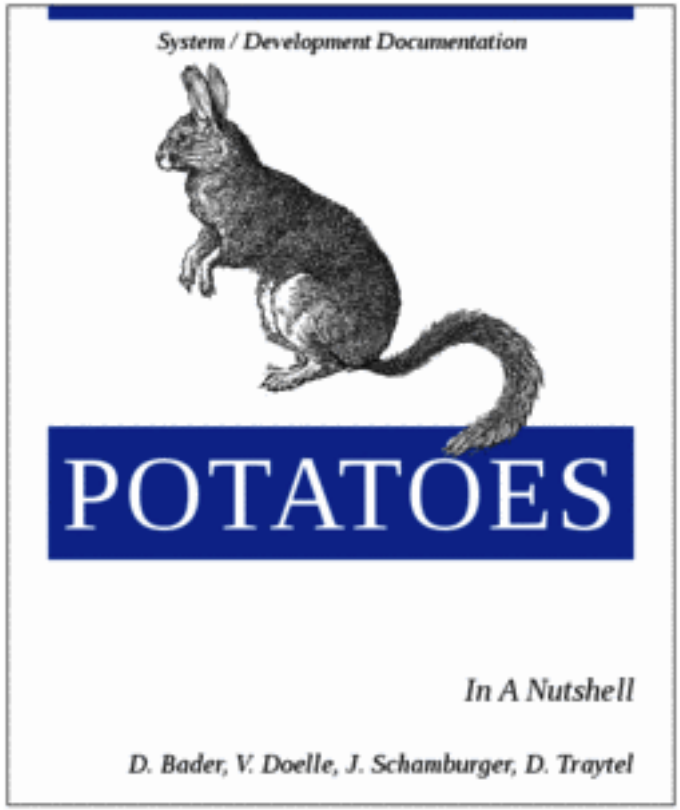
Powered by yFiles

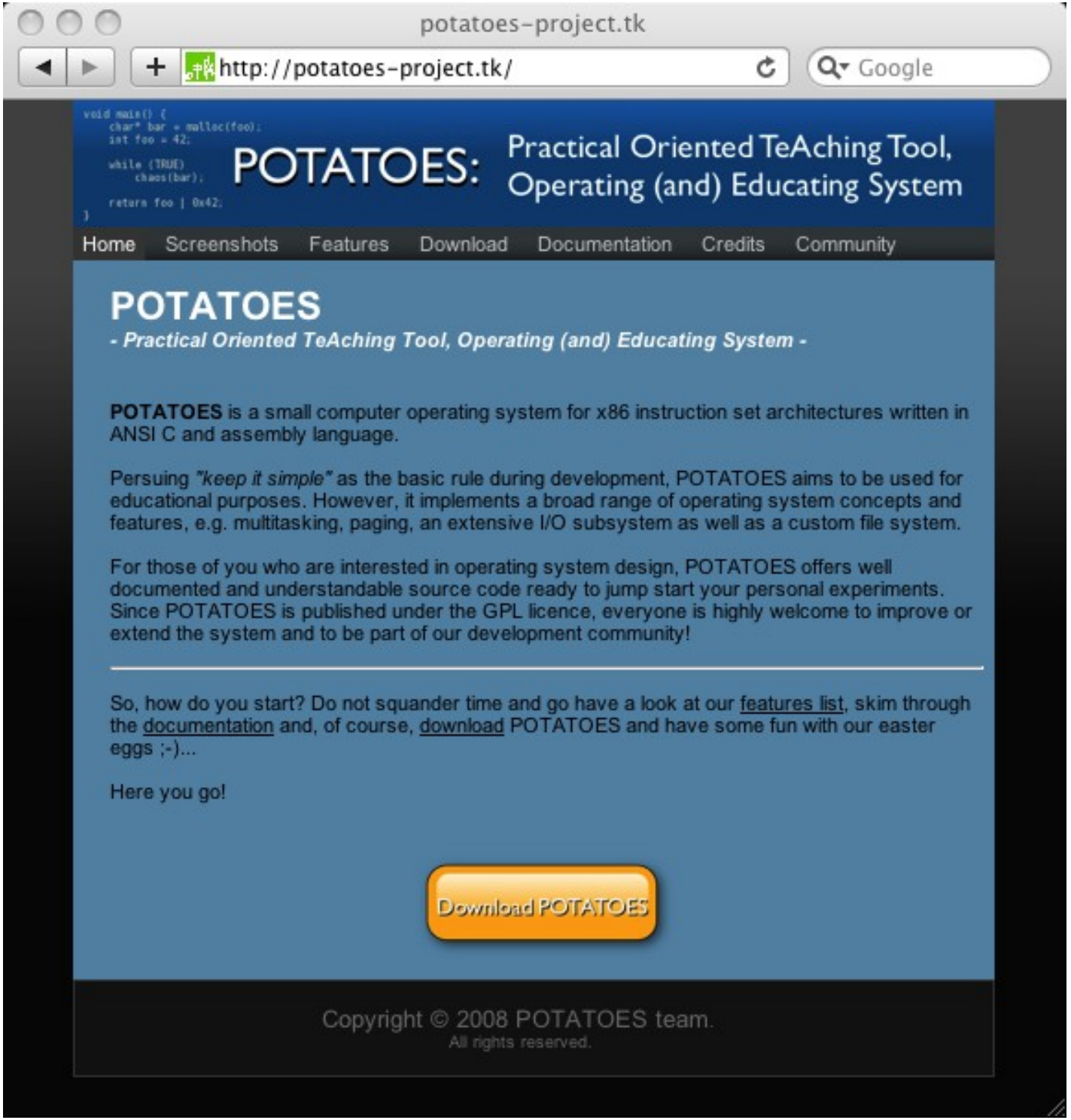
Entwicklungsumgebung

- umfangreiches Makefile zur Erzeugung von
  - Installationsimages
  - Dokumentation
  - Testläufen
- System ist lauffähig in zahlreichen Emulatoren:
  - Bochs, VirtualBox, qemu, …
- … und auch auf echter Hardware!

Dokumentation

- vollständig kommentierter Quellcode
- automatisch generiertes Referenzhandbuch
- Community-Homepage
  - www.potatoes-project.tk




potatoes-project.tk
http://potatoes-project.tk/
Google
POTATOES: Practical Oriented TeAching Tool, Operating (and) Educating System
Home
Screenshots
Features
Download
Documentation
Credits
Community
POTATOES
- Practical Oriented TeAching Tool, Operating (and) Educating System -
POTATOES is a small computer operating system for x86 instruction set architectures written in ANSI C and assembly language.
Persuing "keep it simple" as the basic rule during development, POTATOES aims to be used for educational purposes. However, it implements a broad range of operating system concepts and features, e.g. multitasking, paging, an extensive I/O subsystem as well as a custom file system.
For those of you who are interested in operating system design, POTATOES offers well documented and understandable source code ready to jump start your personal experiments. Since POTATOES is published under the GPL licence, everyone is highly welcome to improve or extend the system and to be part of our development community!
So, how do you start? Do not squander time and go have a look at our features list, skim through the documentation and, of course, download POTATOES and have some fun with our easter eggs ;-)...
Here you go!
Download POTATOES
Copyright © 2008 POTATOES team.
All rights reserved.

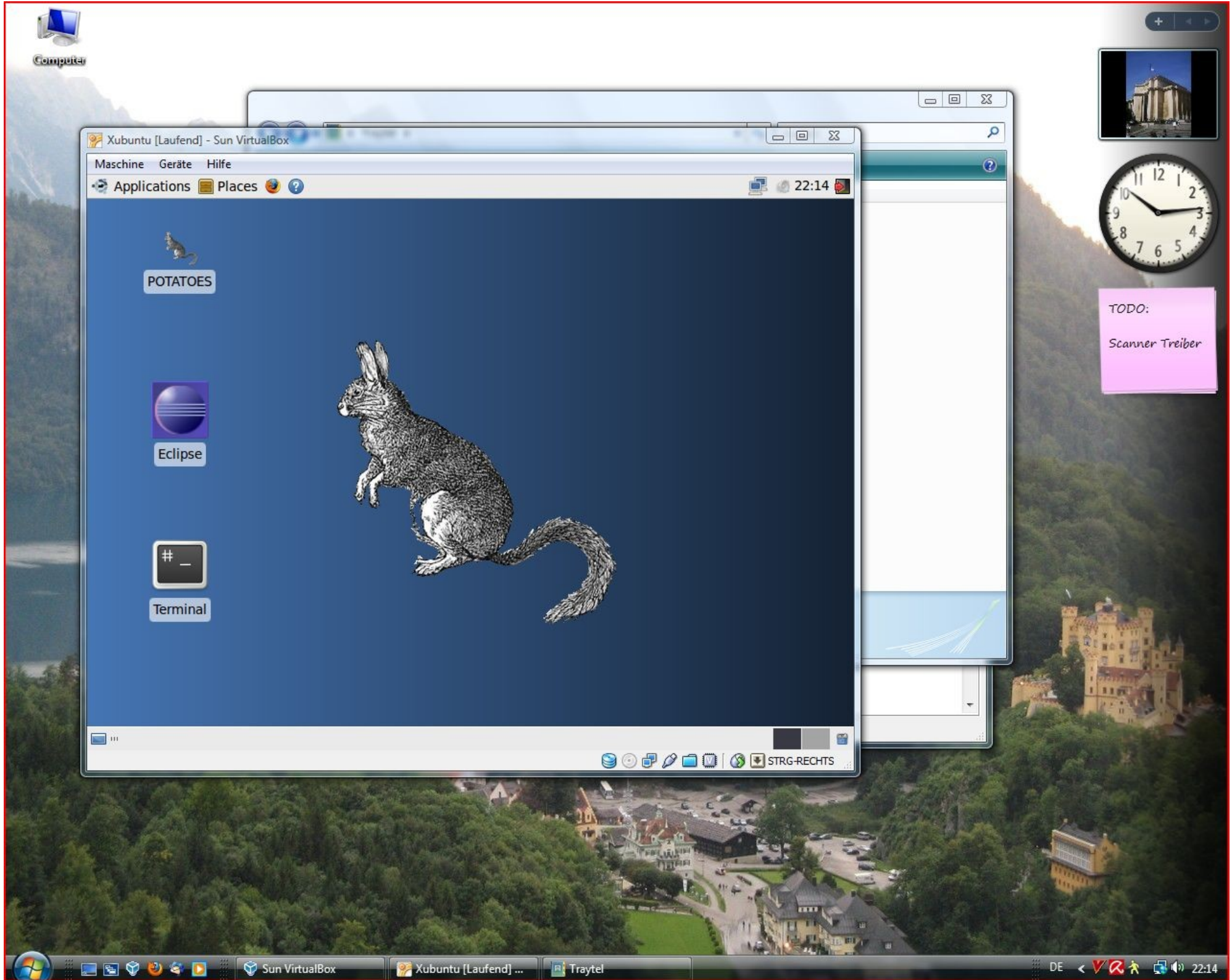
Demonstration

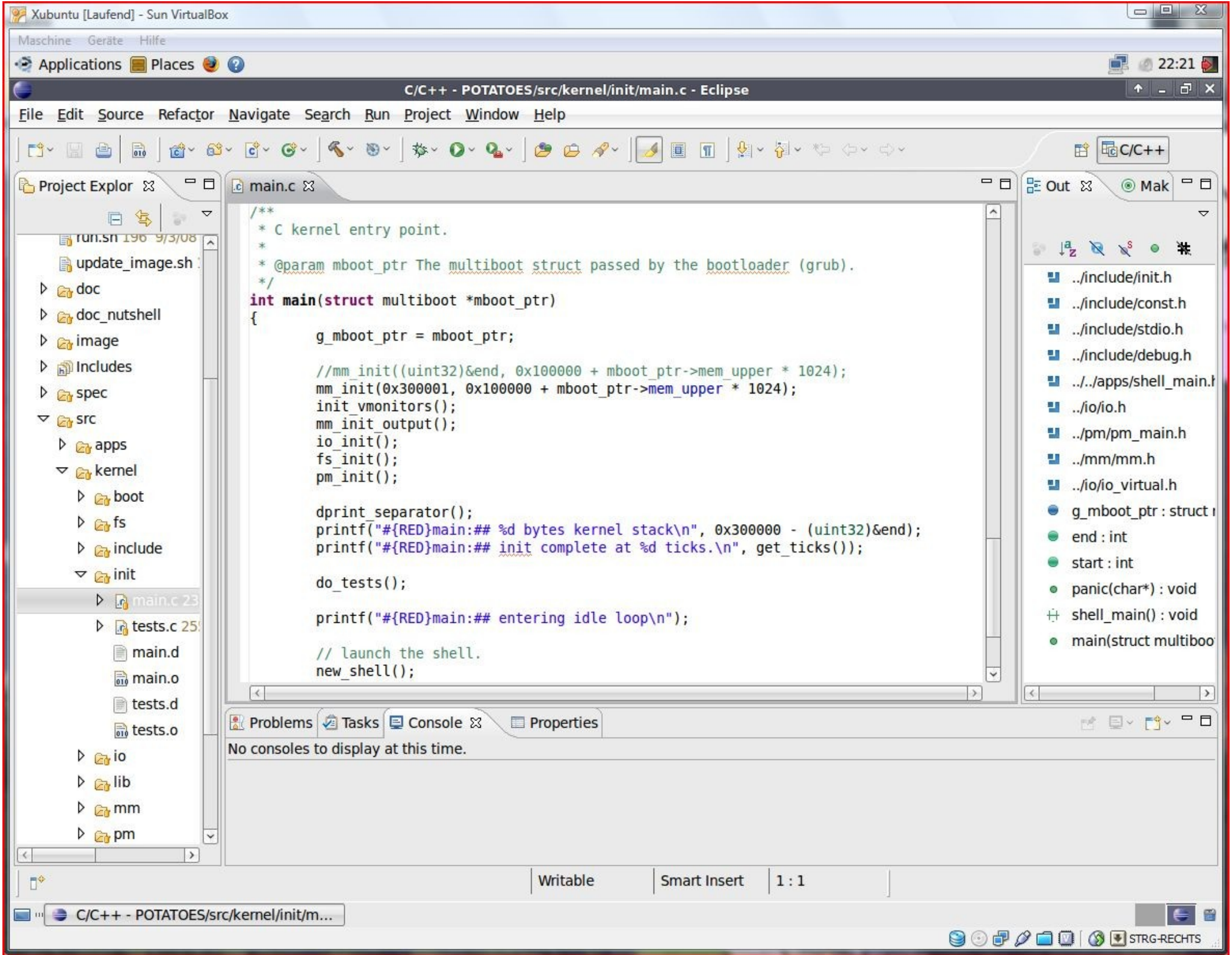
```
= DEBUG MONITOR ========================================= SUN 06.10.2008 22:23:16 =
pm: init
pm: setting up kernel task...
pm: 9 syscalls registered
pm: creating /dev
pm: registered device /dev/null
pm: registered device /dev/stdout
pm: registered device /dev/stdin
pm: registered device /dev/framebuffer
pm: registered device /dev/keyboard
pm: registered device /dev/brainfuck
pm: registered device /dev/clock
pm: scheduler initialized
------------------------------------------------------------------------------
main: 1985280 bytes kernel stack
main: init complete at 15 ticks.

Global keyboard shortcuts:
        CTRL + r ==> ralph_wiggum()
        CTRL + a ==> threadA_test()
        CTRL + b ==> threadB_test()
        CTRL + + ==> switch_monitor_up()
        CTRL + - ==> switch_monitor_down()
        CTRL + s ==> make_snapshot()
------------------------------------------------------------------------------
```

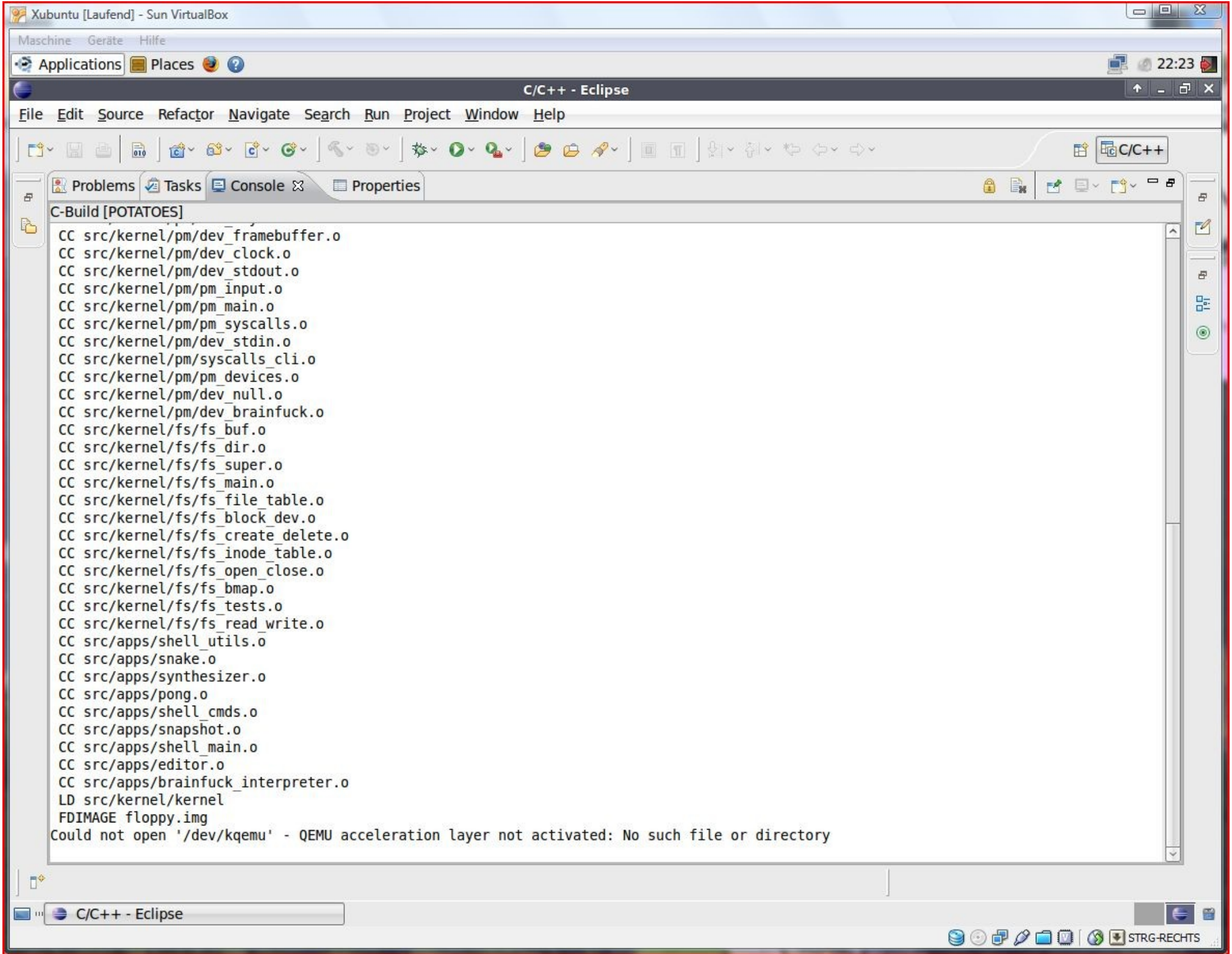
Arbeitsumgebung

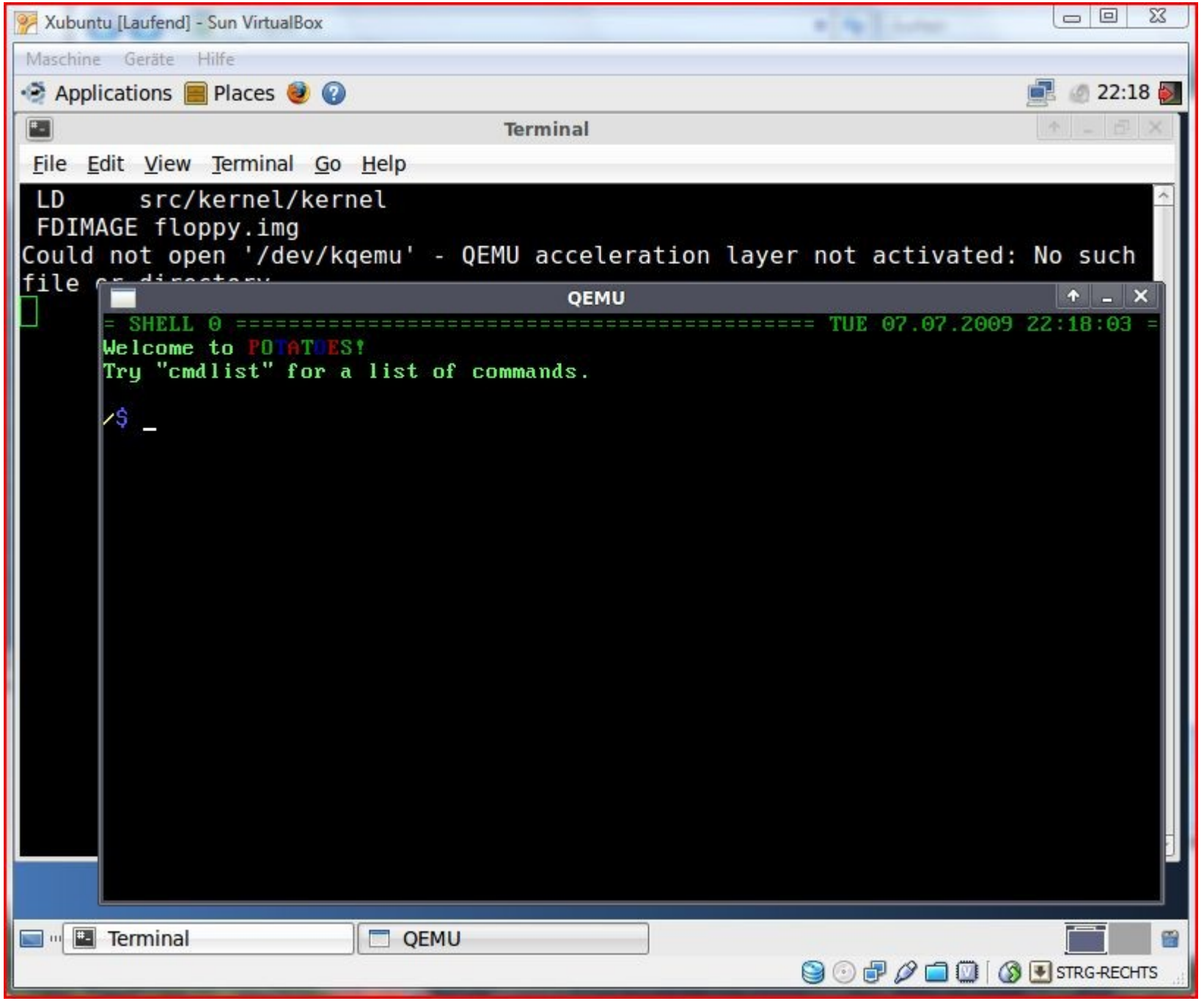
- Vorkonfigurierte Entwicklungsumgebung
  - Virtuelles Festplattenimage mit *Ubuntu*-Linux
  - *Eclipse* IDE mit C-Unterstützung
  - Komplette Toolchain: *GCC*, *NASM*, *latex*, …
- Plattformübergreifend ausführbar mittels (kostenloser) Virtualisierungsumgebung *VirtualBox*

➔ ***„Installation per Doppelklick“***


Computer
Xubuntu [Laufend] - Sun VirtualBox
Maschine Geräte Hilfe
Applications Places
22:14
POTATOES
Eclipse
Terminal
STRG-RECHTS
TODO:
Scanner Treiber
Sun VirtualBox
Xubuntu [Laufend] ...
Traytel
DE
22:14


Xubuntu [Laufend] - Sun VirtualBox
Maschine Geräte Hilfe
Applications Places
22:21
C/C++ - POTATOES/src/kernel/init/main.c - Eclipse
File Edit Source Refactor Navigate Search Run Project Window Help
C/C++
Project Explor
update_image.sh
doc
doc_nutshell
image
Includes
spec
src
apps
kernel
boot
fs
include
init
main.c
tests.c
main.d
main.o
tests.d
tests.o
io
lib
mm
pm
main.c
/**
* C kernel entry point.
*
* @param mboot_ptr The multiboot struct passed by the bootloader (grub).
*/
int main(struct multiboot *mboot_ptr)
{
g_mboot_ptr = mboot_ptr;
//mm_init((uint32)&end, 0x100000 + mboot_ptr->mem_upper * 1024);
mm_init(0x300001, 0x100000 + mboot_ptr->mem_upper * 1024);
init_vmonitors();
mm_init_output();
io_init();
fs_init();
pm_init();
dprint_separator();
printf("#{RED}main:## %d bytes kernel stack\n", 0x300000 - (uint32)&end);
printf("#{RED}main:## init complete at %d ticks.\n", get_ticks());
do_tests();
printf("#{RED}main:## entering idle loop\n");
// launch the shell.
new_shell();
Out
Mak
../include/init.h
../include/const.h
../include/stdio.h
../include/debug.h
../../apps/shell_main.h
../io/io.h
../pm/pm_main.h
../mm/mm.h
../io/io_virtual.h
g_mboot_ptr : struct
end : int
start : int
panic(char*) : void
shell_main() : void
main(struct multiboo
Problems
Tasks
Console
Properties
No consoles to display at this time.
Writable
Smart Insert
1 : 1
C/C++ - POTATOES/src/kernel/init/m...
STRG-RECHTS


Xubuntu [Laufend] - Sun VirtualBox
Maschine Geräte Hilfe
Applications Places
22:23
C/C++ - Eclipse
File Edit Source Refactor Navigate Search Run Project Window Help
C/C++
Problems Tasks Console Properties
C-Build [POTATOES]
CC src/kernel/pm/dev_framebuffer.o
CC src/kernel/pm/dev_clock.o
CC src/kernel/pm/dev_stdout.o
CC src/kernel/pm/pm_input.o
CC src/kernel/pm/pm_main.o
CC src/kernel/pm/pm_syscalls.o
CC src/kernel/pm/dev_stdin.o
CC src/kernel/pm/syscalls_cli.o
CC src/kernel/pm/pm_devices.o
CC src/kernel/pm/dev_null.o
CC src/kernel/pm/dev_brainfuck.o
CC src/kernel/fs/fs_buf.o
CC src/kernel/fs/fs_dir.o
CC src/kernel/fs/fs_super.o
CC src/kernel/fs/fs_main.o
CC src/kernel/fs/fs_file_table.o
CC src/kernel/fs/fs_block_dev.o
CC src/kernel/fs/fs_create_delete.o
CC src/kernel/fs/fs_inode_table.o
CC src/kernel/fs/fs_open_close.o
CC src/kernel/fs/fs_bmap.o
CC src/kernel/fs/fs_tests.o
CC src/kernel/fs/fs_read_write.o
CC src/apps/shell_utils.o
CC src/apps/snake.o
CC src/apps/synthesizer.o
CC src/apps/pong.o
CC src/apps/shell_cmds.o
CC src/apps/snapshot.o
CC src/apps/shell_main.o
CC src/apps/editor.o
CC src/apps/brainfuck_interpreter.o
LD src/kernel/kernel
FDIMAGE floppy.img
Could not open '/dev/kqemu' - QEMU acceleration layer not activated: No such file or directory
C/C++ - Eclipse
STRG-RECHTS

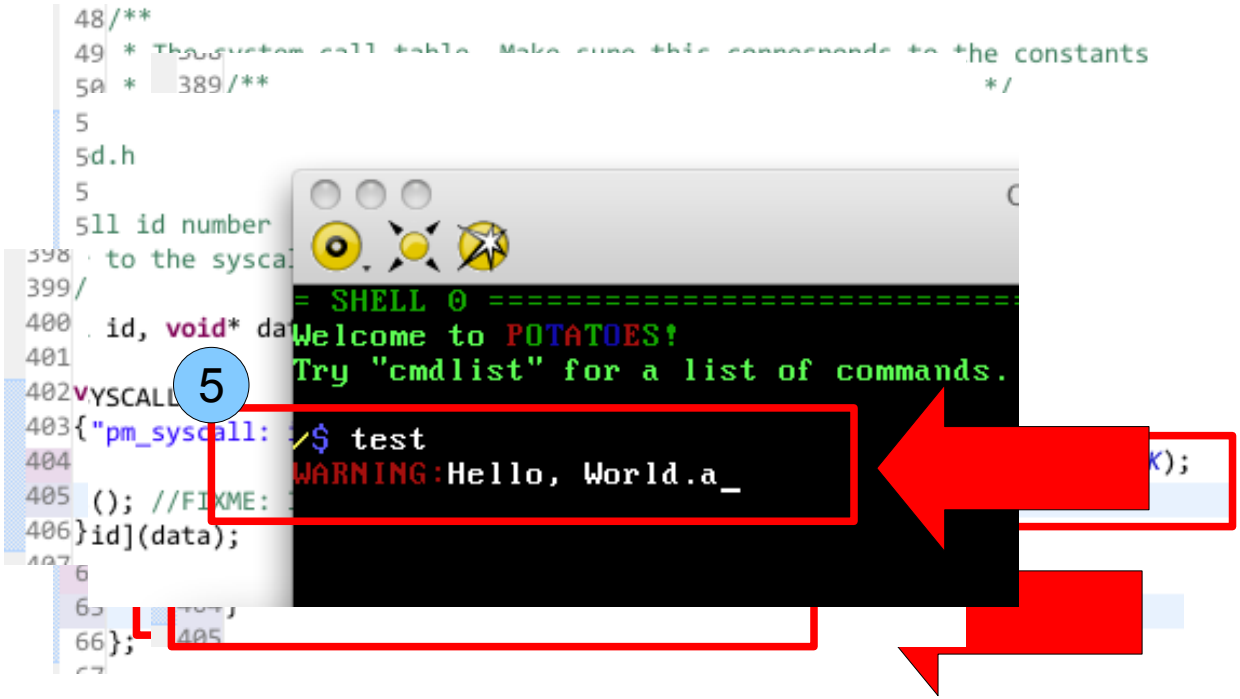

Xubuntu [Laufend] - Sun VirtualBox
Maschine Geräte Hilfe
Applications Places
22:18
Terminal
File Edit View Terminal Go Help
LD src/kernel/kernel
FDIMAGE floppy.img
Could not open '/dev/kqemu' - QEMU acceleration layer not activated: No such
file
QEMU
= SHELL 0 ========== TUE 07.07.2009 22:18:03 =
Welcome to POTATOES!
Try "cmdlist" for a list of commands.
/$ _
Terminal
QEMU
STRG-RECHTS

Beispielaufgabe (Systemaufruf)

1. In dieser Aufgabe soll ein neuer Systemaufruf `void _log_warning(char *msg)` programmiert werden. Dieser soll bei seiner Ausführung den Text „`Warning:`“ und den im Parameter `char *msg` übergebenen String in roter Schrift auf der Konsole ausgeben.

    a) Legen Sie die Prozedur `void sys_logwarn(void *data)` an, welche die kernelseitige Implementation unseres Systemaufrufs darstellt. Die Prozedur soll in dieser Teilaufgabe noch nicht ausprogrammiert werden, es reicht also, wenn Sie zunächst eine Prozedur mit leerem Rumpf implementieren. Registrieren Sie den neuen Systemaufruf im Array `syscall_table[]`. Modifizieren Sie dazu die Dateien `src/kernel/pm/pm_syscalls.h` und `src/kernel/pm/pm_syscalls.c`.

Beispielaufgabe - Umsetzung



Potentielle Aufgabenbereiche

Speicherverwaltung
- Seitenverwaltung
- Virtuelle Adressierung
- Allozierungsalgorithmus
- ...

Dateisystem
- Schichtmodell
- Inode-Konzept
- Gerätedateien
- ...

Ein- / Ausgabe
- Gerätetreiber
  - Monitor, Tastatur
  - Timer, RTC
- ...

Nebenläufigkeit
- Threads
- Synchronisation
  - Locking
  - Semaphoren
- ...

Prozessverwaltung
- Prozesskonzept
- Scheduling
- Interruptbehandlung
- Systemaufrufe
- ...

Systemnahe Programmierung
- C, Assembler
  - Speicherverwaltung
  - Pointer
- Compiler, Binder, Lader
- ...