Betriebssysteme

SS 2015

Hans-Georg Eßer
Dipl.-Math., Dipl.-Inform.

Foliensatz U:
- Unix und Ulix
- Aufbau der Ulix-Quellcode-Dateien

v1.0-nl, 2015/05/12
(klassisch kommentierte Version)

Übersicht: BS Praxis und BS Theorie

A	Einführung Shell Dateiverwaltung	Software-Verwaltung	E
		Scheduler / Prioritäten	F
		Synchronisation	G
B	Filter C-Compiler	Speicherverwaltung	S
		Dateisysteme, Zugriffsrechte	T
C	Prozesse / Jobs Threads	Einführung Ulix	Folien U
		Ulix: Interrupts, Faults	V
D	Interrupts System Calls	Zusammenfassung	H

Unix und Ulix

Unix-Features: Überblick (1)

- Wie alle (modernen) BS bietet Unix eine Abstraktion der Hardware und ein Interface, über das Prozesse diese nutzen können
- Prozessor (Multi-Tasking, Scheduling)
- Prozessverwaltung (`fork`, `exec`, `wait`, `exit`)
- Festplatte/Diskette (Dateisystem)
- Prozess-Kommunikation und -Synchronisation
- Netzwerk: TCP/IP, Sockets
- Multi-User-Betrieb

Prozessor (Scheduling)

- Unix stellt die Ressource CPU mehreren Prozessen zur Verfügung
- je nach Unix-Version verschiedene Scheduling-Strategien
- Priorisierung von Prozessen (nice values), Steuerung über `nice()` und `setpriority()`
- kein Prozess kann die CPU monopolisieren

Prozessverwaltung (1)

- Prozesse bei Unix in Baumstruktur organisiert („Vater/Sohn“)
- Neue Prozesse werden mit `fork()` als ident. Kopie des aufrufenden Prozesses erzeugt
- Prozess kann mit `exec()` ein anderes Programm nachladen
- `exit()` beendet Prozess (mit Rückgabewert)
- `wait()`, `waitpid()` warten auf Ende eines Kindprozesses, Auswerten des Rückgabewerts

Prozessverwaltung (2)

- Intern verwaltet Unix Prozesse über Prozess-Kontroll-Blöcke (PCBs)
- jeder Prozess hat eigenen Adressraum (Speicherschutz), der zudem vom Adressraum des Unix-Kernels getrennt ist
- traditionell: keine Threads (nur über User-Level-Bibliothek)
- Context Switch wechselt von einem Prozess zum nächsten (Scheduler entscheidet, wann und zu welchem Prozess)

Dateisystem (1)

- Alle Unix-Dateisysteme haben gemeinsame Eigenschaften (→ Foliensatz T)
  - hierarchisches Dateisystem (Unterordner)
  - Medien über Mountpoints in Baum integriert
  - Inodes verwalten Eigenschaften und Blockliste einer Datei
  - Inodes enthalten *keinen* Dateinamen
  - Verzeichnisse sind spezielle Dateien mit Zuordnungen Dateiname → Inode-Nummer
  - freie Datenblöcke und freie Inodes in Bitmaps verwaltet
  - Superblock enthält Verwaltungsinformationen für gesamtes Dateisystem

Dateisystem (2)

- System Calls für Zugriff auf Dateien
  - `fd=open()` öffnet Datei, gibt file descriptor zurück
  - `read (fd, ...)` liest aus Datei
  - `write (fd, ...)` schreibt in Datei
  - `lseek (fd, ...)` springt zu Position in Datei
  - `close (fd)` schließt Datei
- Interne Umsetzung oft über Virtual Filesystem (VFS)
- unterstützt mehrere FS und diverse Hardware

IPC, Prozess-Synchronisation

- Signal-Mechanismus
  - Signal-Handler, `signal()`
  - Signal senden, `kill()`
- Auch BS selbst verwendet Signale
- Synchronisation über Mutexe und/oder Semaphore

Netzwerk

- TCP/IP (Verbindungen), UDP (verbindungslos)
- Sockets (Client/Server)
- Verwendung von Sockets ähnlich wie Zugriff auf Dateien,
  - `socket()`,
    TCP: `SOCK_STREAM`,
    UDP: `SOCK_DGRAM`
  - I/O multiplexing mit `select()`

Multi-User-Betrieb

- Unix kennt mehrere Benutzer, root (UID=0) ist Systemverwalter
- Anmeldung über Username, Passwort
- Benutzergruppen
- Dateizugriff: Zugriffsrechte (für Besitzer, Gruppe, sonstige)
- Signalversand: nur an eigene Prozesse

ULIX: Features, Einschränkungen (1)

ULIX 0.12 ist erste fertige Release.

- **Prozesse:**
  - ✓ `fork`, `exit`, `wait`, `exec` funktionieren; RR-Scheduler
  - ✘ aktuelle Version ohne Prioritäten (→ BA, fertig)
- **Dateisystem:**
  - ✓ VFS, Hardware: Disketten, IDE-Platten, Minix-v2-FS

ULIX: Features, Einschränkungen (2)

- **Signale:**
  - ✓ Datenstrukturen im PCB vorhanden, `signal()` trägt Handler ein, `kill()` verschickt Signal
  - ✓ Kernel passt Stack des Empfängerprozess an, damit er beim nächsten Aufruf den Signal Handler ausführt
- **Speicher:**
  - ✓ Paging: Seitentabellen für jeden Prozess, Kernel-Adressraum wird beim Wechsel in Kernel-Mode sichtbar; Auslagern / Wiedereinlagern von Seiten

ULIX: Features, Einschränkungen (3)

- **System Calls:**
  - ✓ System Call Interface, erlaubt Registrieren neuer Syscall-Handler. Standard-Syscalls für alle Kernel-Funktionen vorhanden User-Mode-Bibliothek enthält Funktionen, welche diese Syscalls aufrufen
- **Interrupts:**
  - ✓ Interrupt-Handler für Timer, FDC, IDE, Tastatur, serielle Schnittstelle

ULIX: Features, Einschränkungen (4)

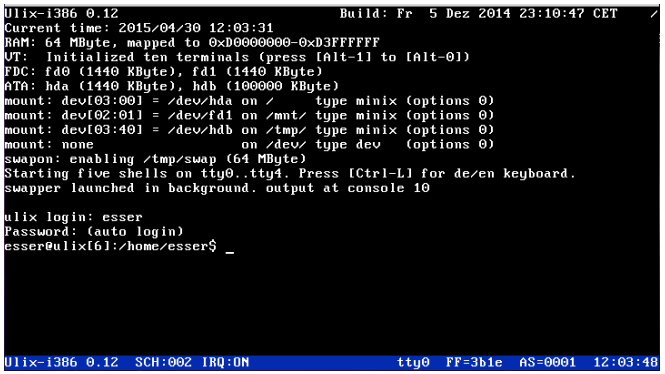
- **Zugriff auf Geräte:**
  - ✓ Floppy: DMA, Block-Transfer
  - ✓ Festplatte: PIO, Block-Transfer
  - ✓ „Serial Harddisk“: byte-weiser Transfer über serielle Schnittstelle
  - ✓ Gerätedateien (für Blockgeräte)
- **Netzwerk:**
  - ✘ nicht implementiert (in BA: IP via SLIP)

ULIX: Features, Einschränkungen (5)

- **Multi-User-Betrieb:**
  - ✓ alle nötigen Datenstrukturen vorhanden
  - ✓ Zugriffsrechte im Minix-Dateisystem
  - ✓ Anmeldung (Benutzer/Passwort), Benutzer-wechsel mit `su`
- **Terminals:**
  - ✓ ULIX startet Shells auf mehreren Terminals
  - ✘ kein Grafikmodus; Terminals laufen in 80x24-Textmodus (Zeile 25: BS-Statuszeile)

Ulix 0.12 Statistik

```
$ wc -l *.c *.asm *.h
     859 block.c
    1971 fs.c
     296 irq.c
     182 keyboard.c
     333 kshell.c
     761 memory.c
      23 module.c
     102 queues.c
     196 sched.c
      59 serial.c
     120 string.c
     266 sync.c
     622 syscall.c
     166 tables.c
     334 terminal.c
     562 threads.c
     207 timer.c
      56 ulix.c
     278 start.asm
     245 globals.h
      85 irq.h
       7 module.h
    1050 ulix.h
    8780 total
```



Aufbau des Ulix-Codes

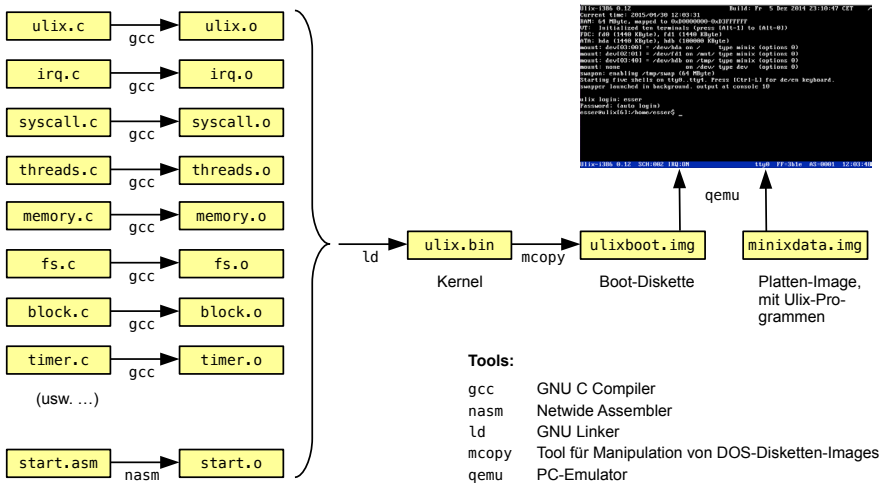
Ulix-Quellcode-Dateien (1)

- **ulix.c: Kernel-main-Funktion**
- globals.h: globale Variablen (Kernel-weit sichtbar)
- block.c: readblock, writeblock (und Hardware-nahe Implementation für Floppy- / IDE-Controller)
- fs.c: logische Dateisysteme (Minix, DevFS)
- **irq.c: Interrupt- und Fault-Behandlung**
- keyboard.c: Tastatur
- kshell.c: Kernel-Mode-Shell (Debugging)
- memory.c: Speicherverwaltung (Paging), Ein-/Auslagern von Seiten
- module.c: Erweiterungen (leer; für Bachelor-Arbeiten o. ä.)
- queues.c: Verwaltung der Bereit- / Blockiert-Warteschlangen

Ulix-Quellcode-Dateien (2)

- sched.c: Scheduler, Context Switch
- serial.c: serielle Schnittstellen (i. W.: Debugging)
- string.c: String-Behandlung (strcpy, strcmp etc.)
- sync.c: Mutexe und Semaphore
- **syscall.c: Behandlung von System Calls und alle spezifischen System Call Handler**
- tables.c: Daten (z. B. Standard-Mount-Tabelle, Tastaturbelegung)
- terminal.c: Verwaltung der 10 Text-Konsolen, printf
- threads.c: Prozesse und Threads, fork, exit, pthread_create etc.
- timer.c: Konfiguration des Timer-Chips, Timer-Interrupt-Handler
- start.asm: Assembler-Code

Ulix-Build-Prozess



Kernel-`main()`-Funktion in `ulix.c`

```
31 void main () {
32   initialize_all_locks ();          // locks and semaphores
33   initialize_all_queues ();         // ready and standard blocked queues
34   uartinit (1);                     // serial port (debugging)
35   initialize_memory ();             // paging, 1st part of setup
36   vt_clrscr ();                     // clear the screen
37   set_statusline (UNAME);
38   printf ("%s (non-LP version)          Build: %s\n", UNAME, BUILDDATE);
39   setup_irqs_and_faults ();         // interrupt and fault handlers
40   keyboard_install ();              // keyboard handler
41   initialize_timer ();              // timer handler
42   initialize_memory_late ();        // paging, 2nd part of setup
43   initialize_terminals ();          // ten virtual consoles
44   install_all_syscall_handlers ();  // load syscall table
45   install_filesystem();             // fs: enable serial disk, fd, hd, swap
46   initialize_module ();             // external module (if available)
47   asm ("sti");                      // enable interrupts
48   #ifdef START_KERNEL_SHELL
49     kernel_shell ();
50   #endif
51   printf ("Starting five shells on tty0..tty4. "
52           "Press [Ctrl-L] for de/en keyboard.\n");
53   start_program_from_disk ("/init"); // load flat binary of init
54   // never reach this line!
55 }
```

irq.c

syscall.c

Nächste Schritte

- Foliensatz D (neu): Grundlagen zu den Themen
  - Interrupt-Behandlung
  - Fault-Behandlung
  - System Calls
- Foliensatz V: Details zur Implementierung in Ulix