



# L 06: CPU / Scheduling

# Fragen

- Memory Hierarchien?
- Elemente eines Adressraums?
- Unterschiede Swapping / Paging?
- Welche Segmente gibt es?
- Typen von Adressierungen?
- Wie funktioniert Laden eines Programms?
- Wie funktioniert Paging?
- Page Replacement Algorithmen?
- Was ist Cache Kohärenz?
- Transactional Memory?

# Prozess / Thread

- Prozess
  - hat genau einen Adressraum
  - hat einen oder mehrere Threads

# Prozess spezifische Daten

Priorität

Scheduling Parameter

User, Group, (Project)

Process ID, Parent PID, (Task)

Process Group

Signale

Startzeit, verbrauchte Zeit

Alarm Zeitpunkt

Segment-Informationen

(text, data, bss, stack, shared...)

# Thread spezifische Daten

- Program Counter
- Programm Status Wort (PSW)
- Register
- Thread-Stack (stack area, stack-pointer)
- Thread Status

# Threads

- schnellere Kommunikation via Adressraum
  - mehr Parallelität bei mehreren CPUs
  - schnellere Erzeugung als bei Prozess
  - besseres Programmiermodell (Java! C?)
  - POSIX Standard (*pthread*)
  - auch teilweise als *green threads*
- 
- LWP vs. Thread
    - 1:1 ist heute üblich
    - 1:n früher in Solaris (Vorteile, zu schwierig?)

# Prozess Lifecycle

- Erzeugen
  - Initial vom OS
  - Aus anderen Prozessen
- Beenden
  - Normale Beendigung (durch Programm)
  - Fehler Rückmeldung (durch Programm)
  - Fataler Fehler (durch OS)
  - killed (OS / Prozess)

# Prozess Lifecycle

- Status
  - *running*
  - *blocked*
  - *ready*
  - *suspended*

# Prozess Hierarchie

- Unix
  - init Prozess als Wurzel
    - started Dienste und Terminal Sessions
  - Starten per fork/vfork und exec
    - vfork beläßt Adressraum
  - Prozessgruppe bleibt verbunden
    - Signale, CPU-Verbrauch
    - Abtrennung per nohup

# Prozess Hierarchie

- Windows
  - Startup erzeugt Basisprozesse
  - Systemcall: CreateProcess
  - nicht weiter verbunden
- Mainframe
  - Startup startet wenige Basisprozesse
    - Job Scheduler ist einer davon
    - Basis – Jobs starten Rest

# Synchronisation

- Motivation:
  - *race condition, lost update*
  - *critical region*
- Implementierung:
  - Abschalten Interrupts
  - Busy Waiting mit *turn-Variable*
  - *lock-Variable* (HW Support notwendig)
    - busy waiting (*spin lock, spinning mutex*)
    - mit Suspendierung (*blocked state*)
  - HW pipe (Mainframe Channel to channel)

# Inter Process Communication (IPC)

- Signale
  - Unidirektionale Benachrichtigung
- Semaphore
  - Ressourcen mit Suspendierung
- Pipes, Named Pipes, Sockets
  - Unidirektionaler Datenkanal
- Message Queues (strukturierte Nachrichten)
- Rendezvous (send/receive)
- Shared Memory (+ Signalisierung)

# OS Synchronisation

- Neue Methoden:
  - Mach Messages
    - Unidirektionale Nachricht
    - Aufruf einer Funktion beim Empfänger
    - Parameter in übereignetem Memory Segment
  - Solaris Doors (auch Linux):
    - Aufruf von wartendem Thread beim Empfänger
    - Übergabe von Parametern in Segment
    - Antwort ebenfalls in Memory Segment

# Job Scheduling

- Mainframe Job Control Language
  - EXEC zum Ausführen eines Programms
  - DD zum Definieren von Link-Namen
  - Ablauf-Kontrolle  
(JOB, PROC, PEND, IF, THEN, ELSE, ENDIF, ...)

```
//JOB3      JOB (123456),MSGCLASS=X
//STEP1    EXEC PGM=IDCAMS
//DDIN      DD DISP=SHR,DSN=SYSPROG.SMF.AUSWERT
//DDOUT     DD DISP=(NEW,CATLG),DSN=SYSPROG.SMF.HISTORY(+1),
//          UNIT=SYSDA,SPACE=(CYL,(15,15),RLSE),DCB=*.DDIN
//SYSPRINT DD SYSOUT=*
//SYSIN     DD *
           REPRO INFILE(DDIN) OUTFILE(DDOUT)
/*
```

# Job Scheduling

- Unix
  - cron
  - at, batch
  - shell-Scripte
- Batch-Systeme
  - Sun Grid Engine (OpenSource)
  - Torque
  - ...

# Scheduler

- Varianten
  - Einzel-Prozess
  - Multitasking /-threading
    - cooperative Scheduler
    - preemptive

Heute: nur noch preemptive Scheduler

# Scheduler Varianten

- FIFO
- Round Robin
- Priority Scheduler
- Deadline Scheduler
- Shortest Process First
- Guaranteed Scheduler
- Lottery Scheduler
- Fair Share Scheduler
- Real Time Scheduler (hard, soft)

Realisierung heute: Round Robin + Priority

# Scheduler Affinity

## Multi Prozessor Systeme:

- Run Queue pro Thread in Core
- Run Queue pro Core
- Run Queue pro Socket
- Run Queue pro Systemboard
- Run Queue pro System

# Scheduler in Linux

- Realtime FIFO
- Realtime Round Robin
- Time Sharing

# Scheduler in Solaris

- SYS System Tasks (sched, pageout, fsflush)
- RT Real Time (RR + feste Prio)
- TS Time Sharing (round robin + gewichtete Prio)
- IA Interactive (wie TS + enqueue first)
- FX Fixed Priority (statt RT, einfacher)
  
- FSS Fair Share Scheduler
  - nutzt Prioritäten zur Bevorzugung
  - stellt konfigurierbares Verhältnis ein

# Tabellengesteuerter Scheduler

## Tabelle (Solaris TS Scheduler):

- 60 Userland Prioritäten
- pro Priorität
  - quantum      Zeitscheibe in Millisekunden
  - tqexp        Neue Priorität nach Ausnutzen Zeitscheibe
  - slpret        Priorität nach Sleeping (*blocked mode*)
  - maxwait      Max Wartezeit in Sekunden in *ready mode*
  - lwait         Neue Priorität nach maxwait Sekunden

# Ressource Management (neu)

- Weitere Unterteilung unter User/Group
  - Multiple Applikationen unter einer Userid
  - Unterschiedliche Behandlung
- Mainframe: Account-Number (+ SW)
- Unix (Solaris, teilweise Irix)
  - task:        arbiträre Gruppe von Prozessen
  - project:     Tag pro Task
    - Definition von Scheduling Parametern
    - Limits (Memory, CPU, File, Network)
    - Definition von Ressource Pool

# Pooling von Ressourcen

- Binden Prozess an CPU (pbind)
- Prozessor Set für ausschliessliche Nutzung
  - keine OS Prozesse
  - ggf. Interrupts (Solaris: ausschaltbar)
  - Solaris: psrset Kommando
  - Linux: cpuset Filesystem (Konfiguration wie procfs)
  - nur dynamisch (immer neu nach Reboot)

# Pooling von Ressourcen

- Ressource Pools
  - Definition einer Ablauf-Umgebung
  - Prozessor-Sets / Memory Sets
  - Scheduler
  - statisch (automatische Definition nach Konfig)
  - dynamisch anpassbar zur Laufzeit