QNX

Echtzeitsysteme - Übungen zur Vorlesung

Fabian Scheler, Peter Ulbrich, Niko Böhm

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.informatik.uni-erlangen.de

8. Februar 2010

What is QNX?

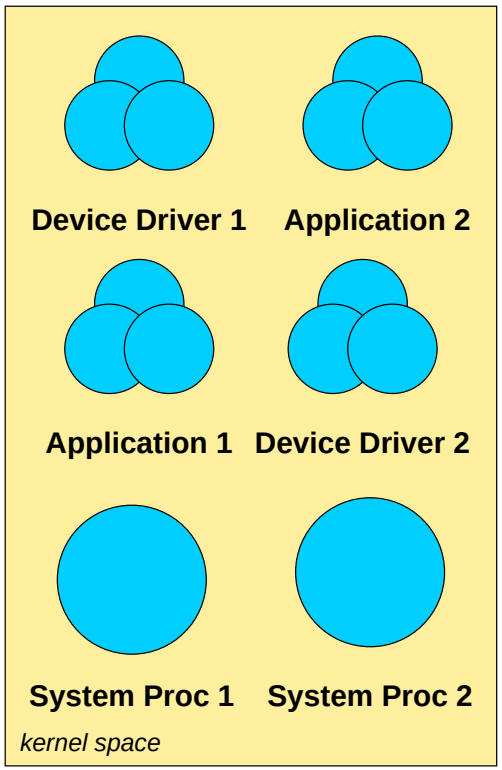
- Real-Time Operating System
- POSIX-compliant $\rightsquigarrow$ UNIX-like
- μ-Kernel Operating System
- available for a variety of 32-bit target architectures
  - x86, MIPS, PowerPC, SH-4 and ARM (ARM, StrongARM, xScale)

History of QNX

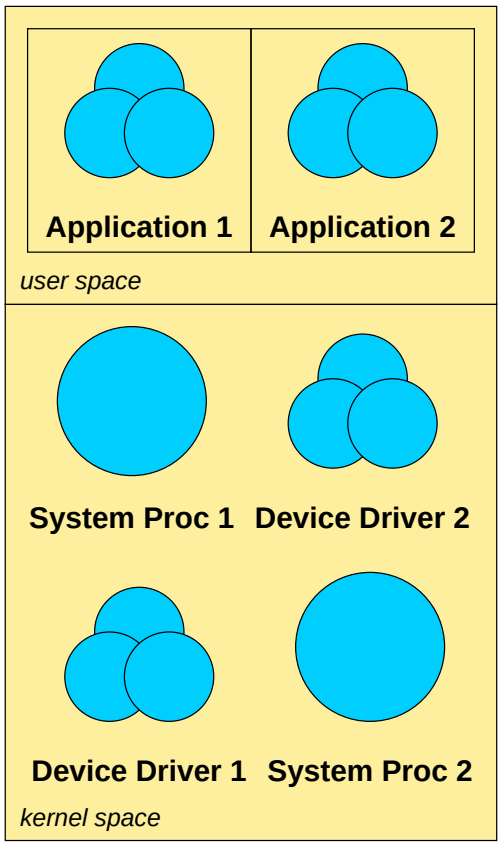
- **1980** Idea was born
  - after an OS lecture at Waterloo University
- **1980** Quantum Software Systems was founded
- **1982** first version of QNX
  - target architecture: Intel 8088
  - non-embedded OS
- **mid 1990s** QNX 4
  - POSIX-compliance
  - graphical user interface: PHOTON
- **end 1990s** QNX Neutrino
  - Linux-like
- **2004** QNX Software Systems taken over by Harman International

Library vs. Monolithic vs. μ-Kernel

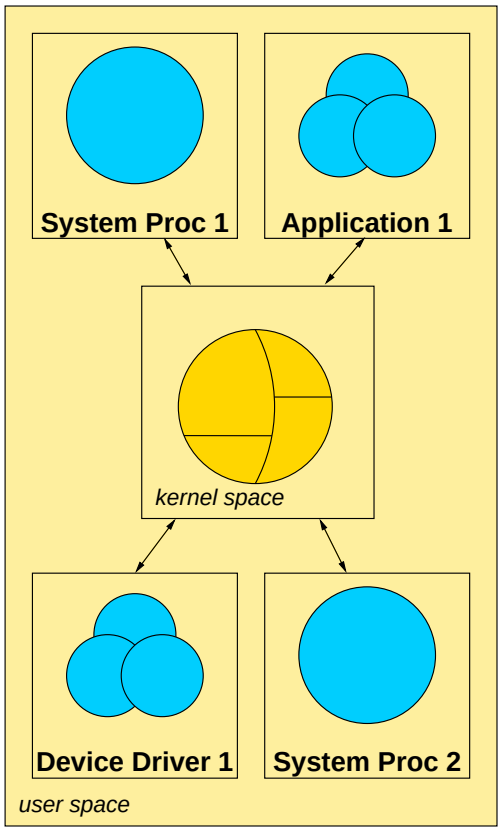
Library



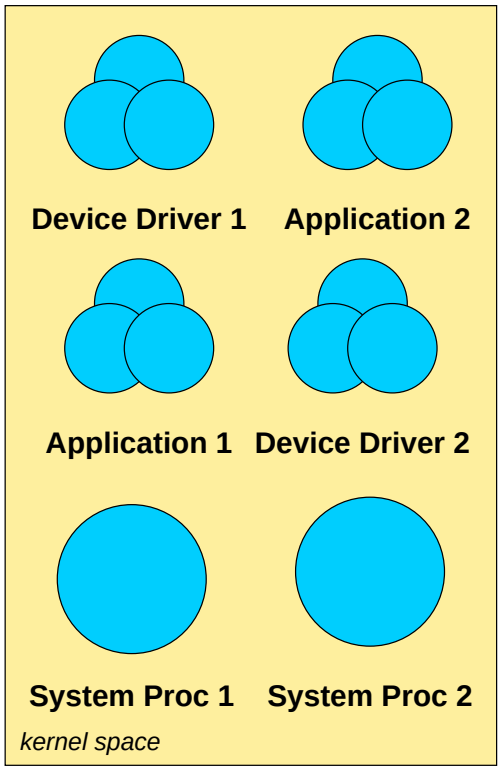
Monolithic



μ-Kernel

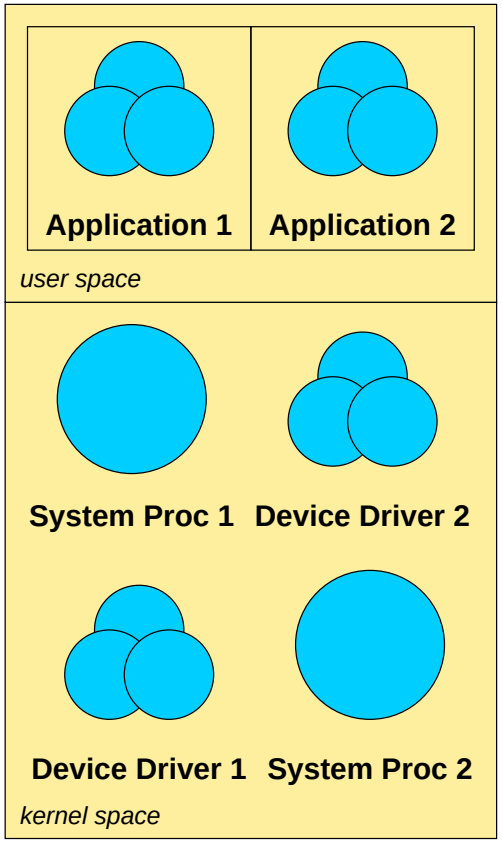


Library-based Executive



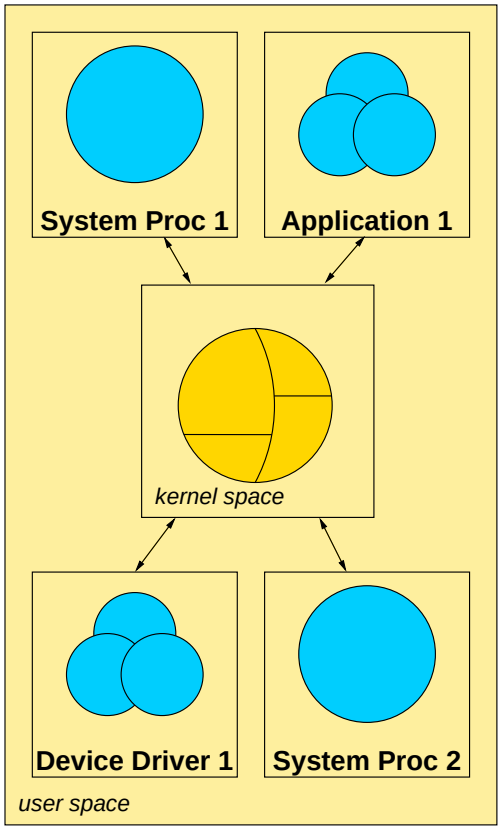
- single address space
- no isolation or protection
- kernel interface: procedure calls
- small, efficient kernels
- fault-prone

Monolithic Kernels



- user space and kernel space
- separated address spaces
- isolation and protection
  - between applications
  - between applications and the kernel
- no isolation or protectio
  - kernel and device drivers
- kernel interface: traps
- efficient
- inflexible

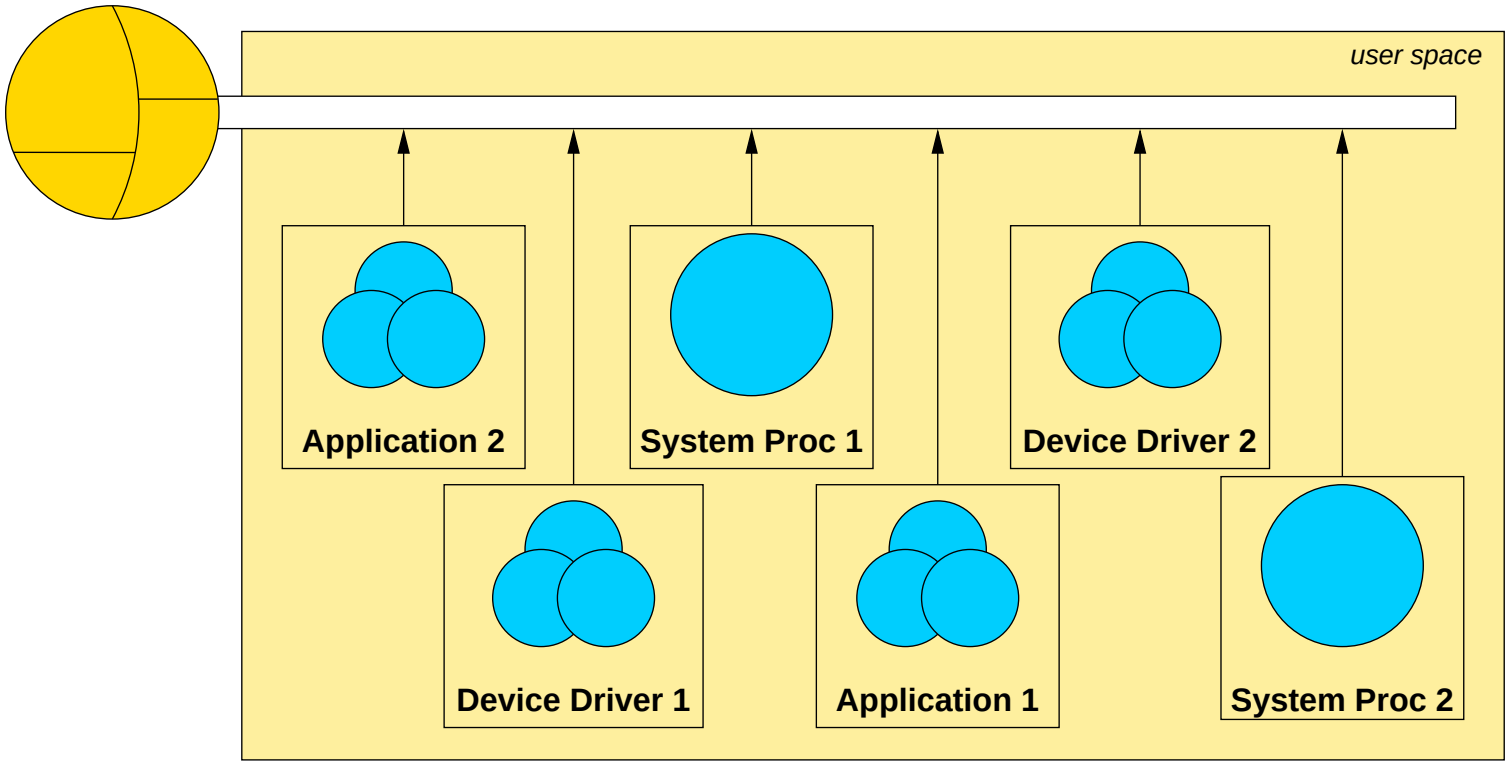
Microkernels



- user space and kernel space
- separated address spaces
- isolation and protection
  - between applications and
  - device drivers and
  - the kernel
- kernel interface: traps
- communication: message passing
- flexible
- supposed to be inefficient

The Role of the QNX Microkernel

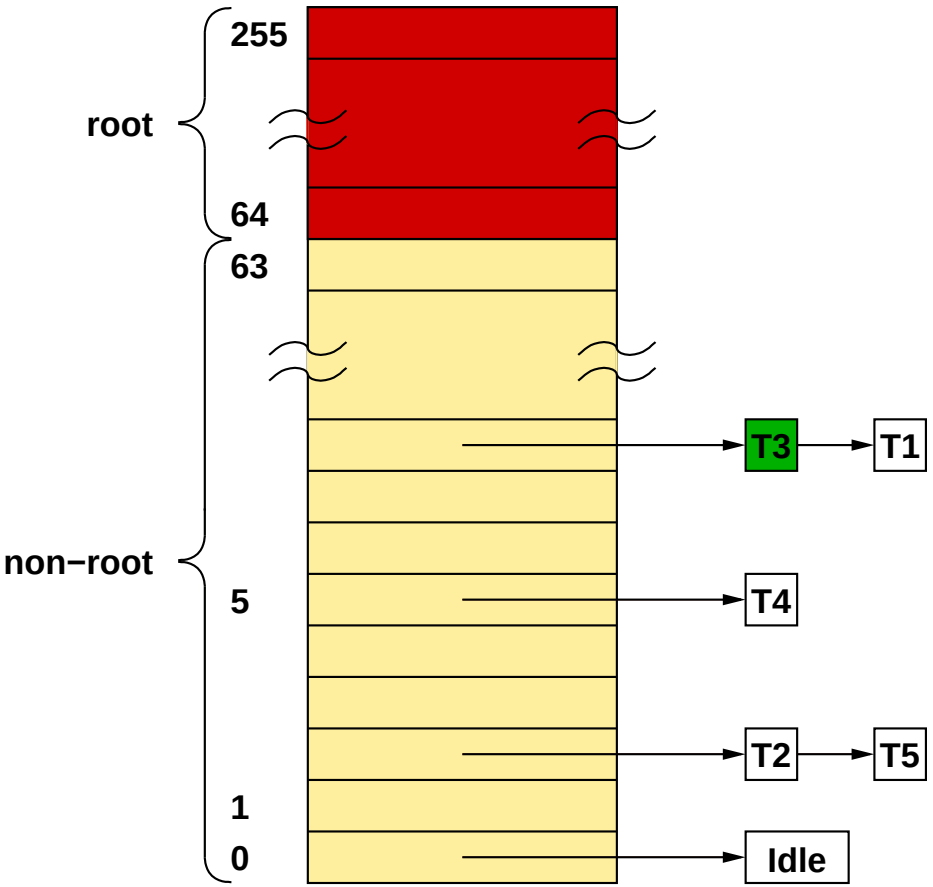
Provides a Software Bus



QNX Microkernel Services

- thread services (POSIX)
- signal services (POSIX)
- message-passing services
- synchronization services (POSIX)
- scheduling services (POSIX)
- timer services (POSIX)
- process management services

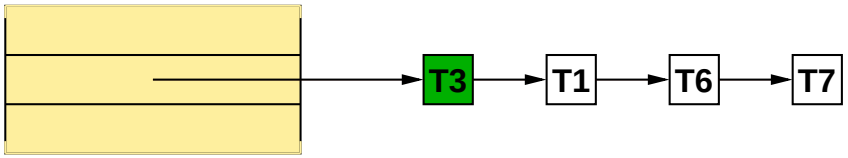
Thread Scheduling



- priority-driven (256 priorities)
- full preemptive
- threads are scheduled globally
- threads inherit parent priority
- context switch when a thread
  - blocks
  - is preempted
  - yields
- scheduling algorithms
  - FIFO
  - round-robin
  - sporadic

Thread Scheduling Algorithms

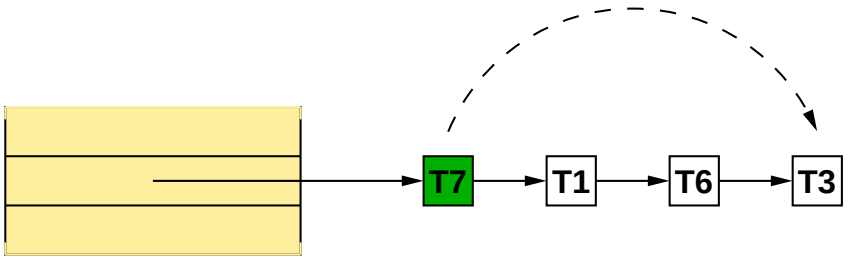
- FIFO



A thread runs until it

- relinquishes control
- is preempted

- Round-robin

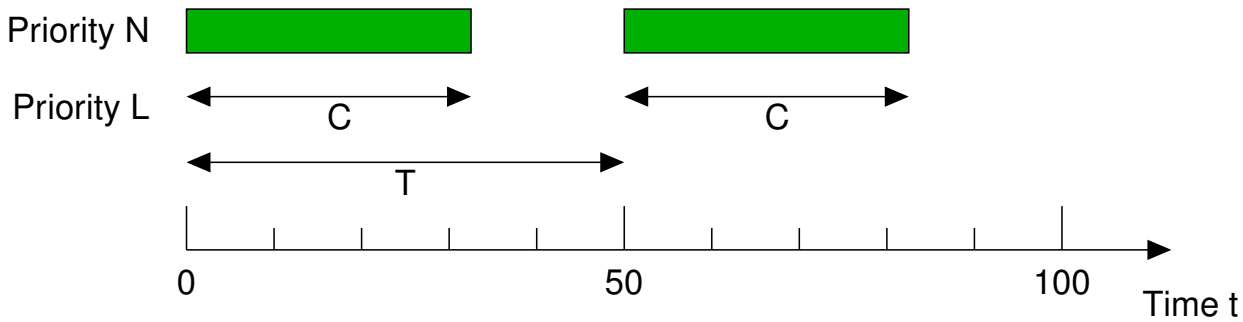


A thread runs until it

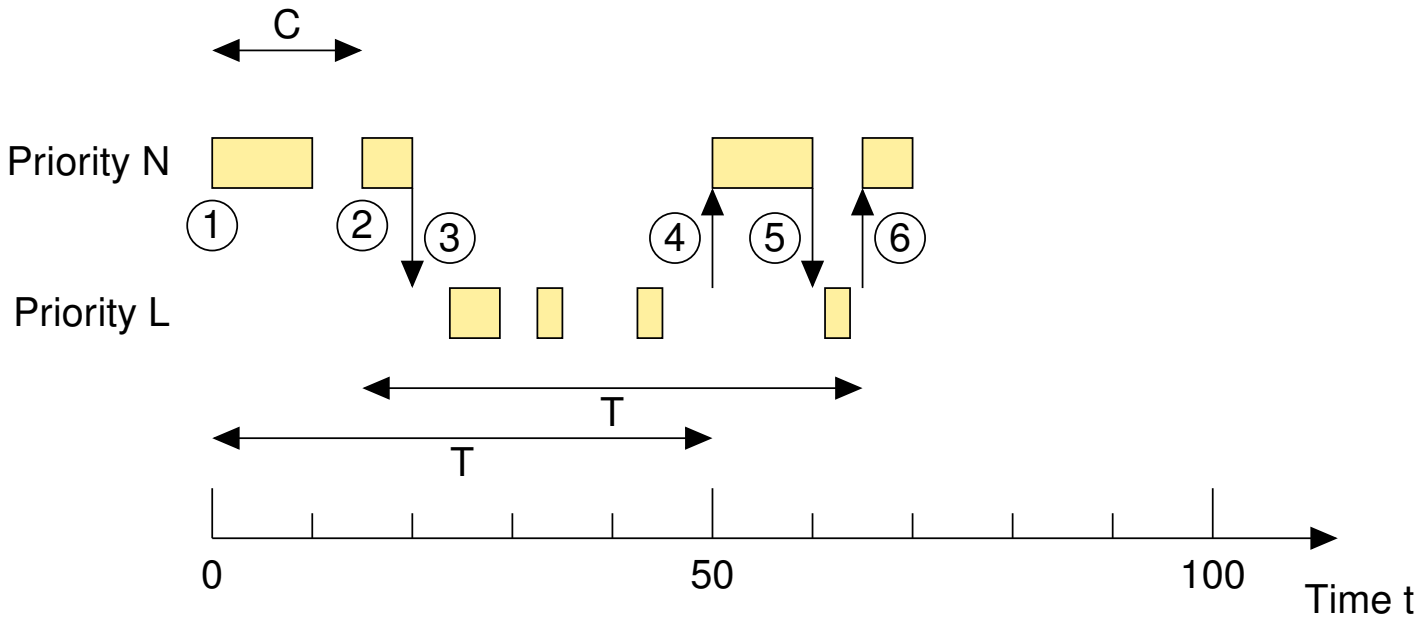
- relinquishes control
- is preempted
- comsumes its timeslice

Sporadic Scheduling

- Sporadic Server
  - initial budget (C)
  - foreground: normal priority (N) $\leadsto$ budget is available
  - background: low priority (L) $\leadsto$ budget is exhausted
  - replenishment period (T)
    - budget is replenished in chunks
    - number of pending replenishments can be bounded $\leadsto$ overhead
- equivalent to a periodic thread $T = (T, C)$



Sporadic Scheduling – $D_S = (50ms, 15ms)$

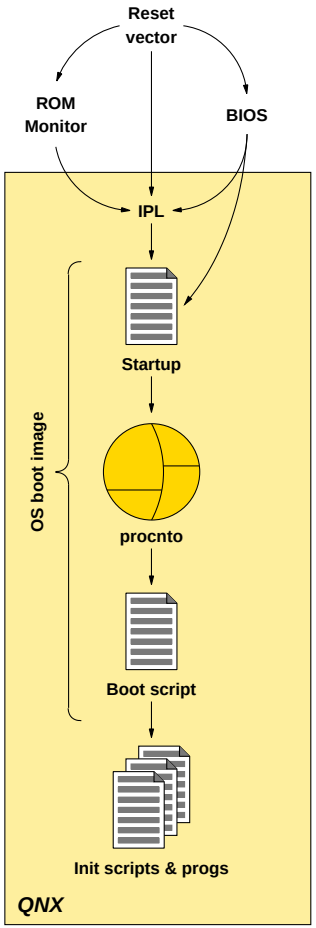


1. thread blocks after 10 ms $\rightsquigarrow$ replenishment of 10 ms at 50 ms
2. thread runs again at 15 ms $\rightsquigarrow$ replenishment of 5 ms at 65 ms
3. thread exhausts its budget at 20 ms $\rightsquigarrow$ drops to low priority L
4. 10 ms of threads budget are replenished $\rightsquigarrow$ back to priority N
5. thread exhausts its budget at 60 ms $\rightsquigarrow$ drops to low priority L
6. 5 ms of threads budget are replenished $\rightsquigarrow$ back to priority N

Thread Synchronization

- Mutexes (kernel, priority inheritance)
- Condition Variables (kernel)
- Barriers (on top of mutexes, Condition Variables)
- Sleepon Locks (on top of mutexes, Condition Variables)
- Reader/Writer Locks (on top of mutexes, Condition Variables)
- Semaphores (kernel)
- FIFO Scheduling (kernel)
- Send/Receive/Reply (kernel)
- Atomic Operations (processor or emulated by kernel)

Startup

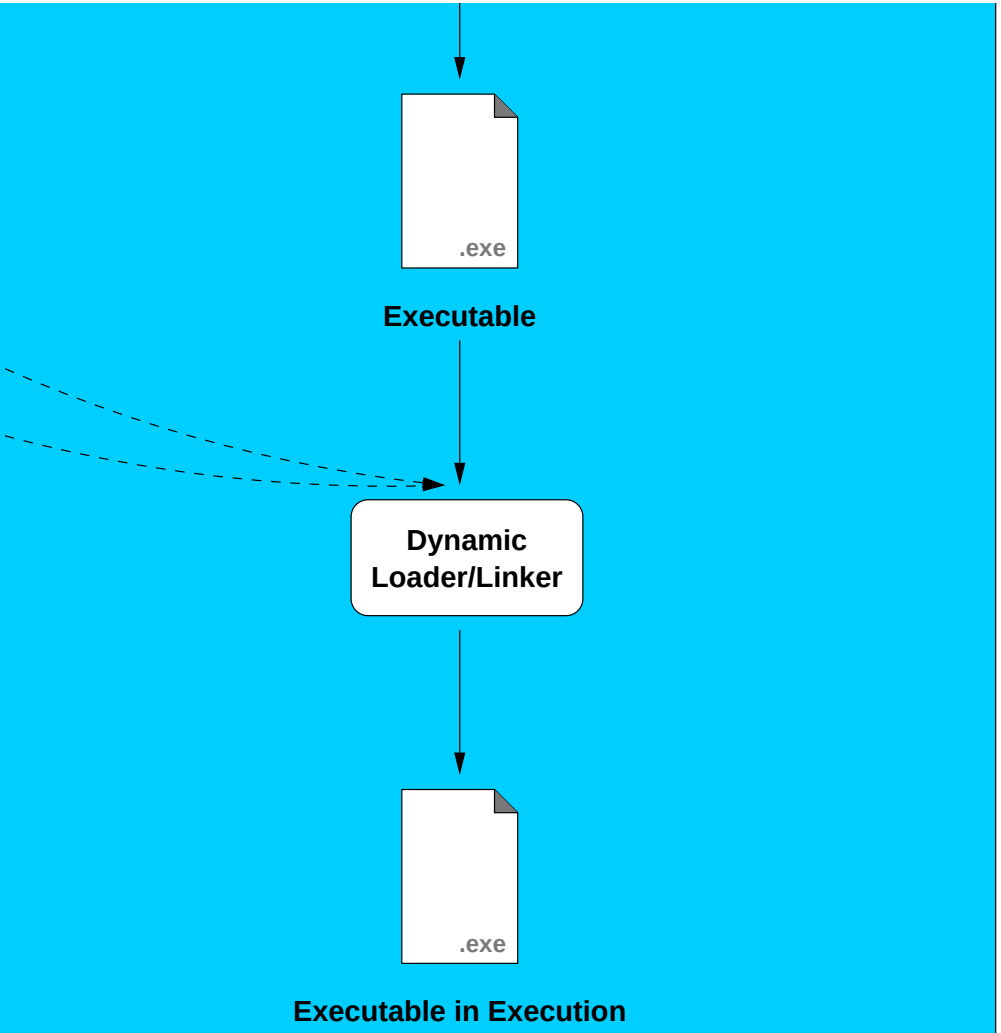


1. Initial programm loader (IPL)
   - configure memory controller, clock
   - setup processor, stack
   - locate OS image
   - load startup program
2. startup program
   - copy and decompress OS image
   - configure hardware
   - determine system configuration
   - establish callout-routines
   - start the OS kernel
3. Operating System Kernel
4. Boot Script
5. Applications

Available modules

- Filesystems (RAM, QNX4, DOS, CD-ROM, FFS3, NFS, CIFS, Ext2, Virtual, ETFS, ...)
- Character I/O (Console, Serial, Parallel, Pseudo terminals ...)
- Networking (QNet, TCP/IP)
- Power Management
- Photon microGUI
- XFree86
- tcsh
- ...

Development with QNX



Development with QNX