An Overview of Scheduling in the FreeBSD Kernel

Brought to you by

Dr. Marshall Kirk McKusick

EuroBSD Conference 2021
19 September 2021

Gushaus Campus of Vienna's Technical University
Vienna, Austria

Scheduling Classes

Threads are divided into five scheduling classes

Priority	**Class**	**Thread type**
0 – 47	ITHD	Bottom-half kernel (interrupt)
48 – 79	REALTIME	Real-time user
80 – 119	KERN	Top-half kernel
120 – 223	TIMESHARE	Time-sharing user
224 – 255	IDLE	Idle user

- Higher values of priority imply lower levels of service

- ITHD and KERN classes are managed by the kernel

- REALTIME and IDLE classes are managed by user processes

- TIMESHARE class management shared by kernel and user processes

Scheduling Choices

Real time

- processes set specific priorities
- kernel does not change priorities

Interactive scheduler (ULE)

- processor affinity
- kernel sets priority based on interactivity score

Share scheduler (4BSD)

- multi-level feedback queues
- kernel changes priority based on run behavior

Idle scheduler

- administrator set specific priorities
- kernel does not change priorities

Run Queues

The 4BSD scheduler uses a single global set of run queues organized from highest to lowest priority (priorities 0 - 255)

The ULE scheduler uses three sets of run queues for each CPU:

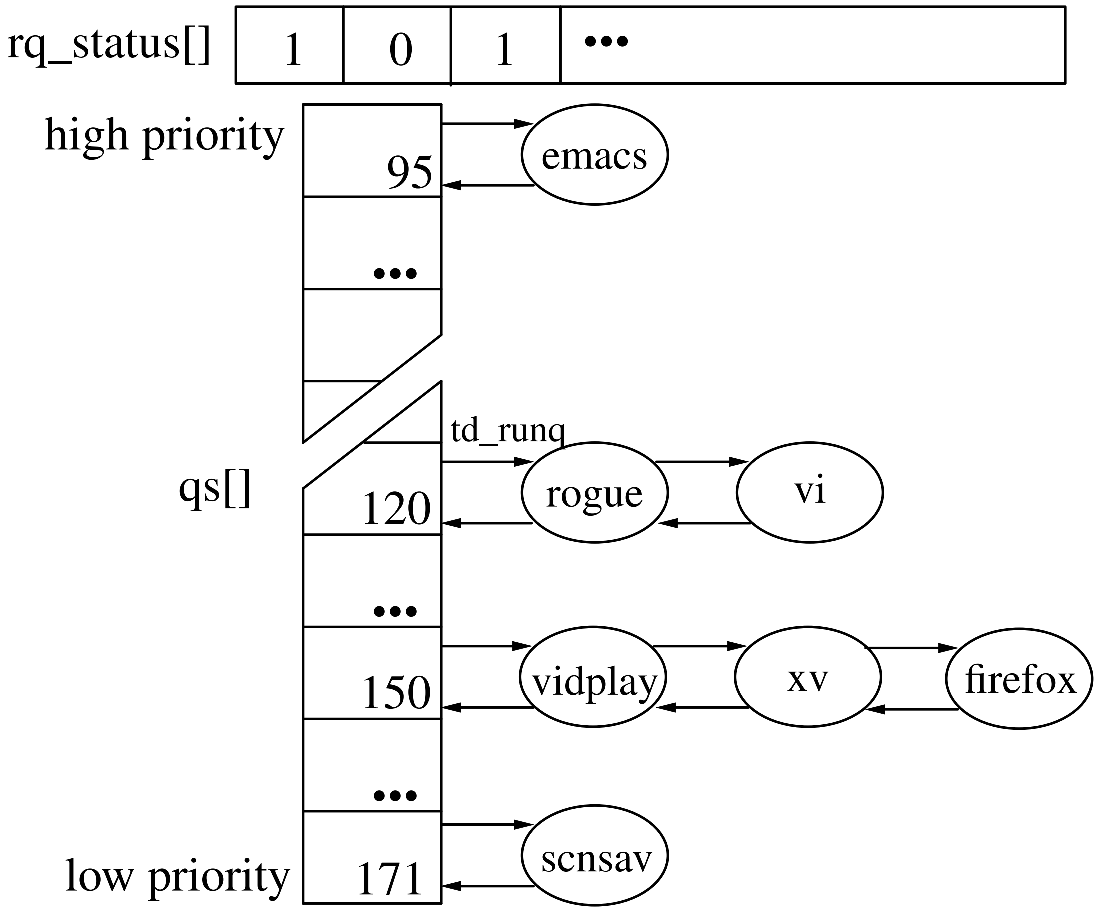
1) real-time queue has kernel, real-time, and timeshare threads classified as interactive and is organized from highest to lowest priority (priorities 0 - 171)

2) batch queue has timeshare threads classified as batch and is organized as a calendar queue (priorities 172 - 223)

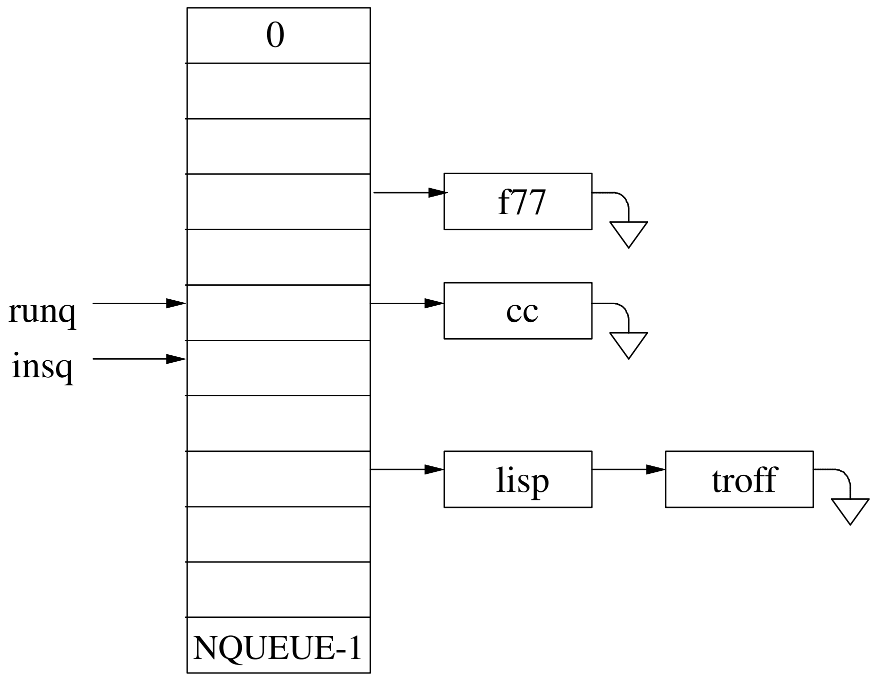
3) idle queue has idle threads and is organized from highest to lowest priority (priorities 224 - 255)

Priority-based Queues

- Run queues contain all runnable threads in memory except the running threads.
  - Threads are placed at the end of the appropriate queue by *setrunnable*( ).
  - Threads are removed from their queue by *remrunqueue*( ) and (when run) by *sched_switch*( ).



Calendar-based Queues



- run as a circular queue
- NQUEUE = number of batch priorities + 1
- run at runq until queue empty, then increment
- insert at:
  (insq + priority - MINBATPRI) % NQUEUE
- increment insq
  - every 10 milliseconds
  - with runq when incremented runq == insq

Run Priority

Run selection order:

1) If any real-time threads, select from first thread in highest priority non-empty queue

2) If any batch-queue threads, run calendar queue starting from first thread at current (runq) entry

3) If any idle-queue threads, select from first thread in highest priority non-empty queue

Low-level Scheduling

- Choosing to context switch
  - For priority-based queues, run all threads in top queue round robin, switching every 10 milliseconds but such that every thread will run within 50 milliseconds (a queue with ten threads will use 5 millisecond time slices)
  - For calendar-based queues, run all threads in current slot until empty giving every one a time slice of the same duration as used for priority-based queues
  - When a thread other than the currently running thread attains better priority, switch immediately

Scheduling context switch

Context switches are requested:

- when blocking voluntarily (sleep)
- when higher-priority process becomes runnable (wakeup, setrunnable), mostly from interrupt level
- once per time slice (roundrobin)

Request is posted by:

- setting NEEDRESCHED flag in td_flags

Request is processed:

- after return from interrupt, if idling, or
- at end of current system call or trap (if any)

If rescheduling is involuntary, process places itself at end of run queue before switch

Context Switching (*sched_switch*)

Under kernel control, switch from one thread to another:

- save context of current thread
  - registers
  - address space
- choose next thread to run
  - find highest-priority queue with threads
  - remove first thread from queue
  - if no queue contains threads, unblock interrupts and repeat selection
- clear NEEDRESCHED flag
- restore context of new thread

ULE Scheduler Goals

Identify and give low latency to interactive threads

- allow brief bursts of activity
- differentiate time waiting for CPU from time waiting for user input

Only migrate threads when necessary (processor affinity)

Understand CPU hierarchy; in decreasing order, preference is to run on:

- same CPU
- a CPU on same chip
- a CPU on same board
- a CPU in same chassis

Constant scheduling time independent of number of threads to be scheduled

Differentiating Interactive versus Batch

Scheduling variables

p_nice	-20 to +20 (0 is normal)
td_runtime	recent CPU utilization
td_slptick	recent voluntary sleep time
td_priority	current priority
td_user_pri	priority when running user level

- td_runtime accumulates running ticks
- td_slptick accumulates sleeping ticks
- decay td_runtime and td_slptick when their sum exceeds five seconds

  td_runtime = (td_runtime / 5) * 4;
  td_slptick = (td_slptick / 5) * 4;
- recompute priority when thread is awakened or accumulates a tick
- Interactive if td_slptick exceeds td_runtime

Selecting CPU on Which to Run

When a thread becomes runnable search for best CPU as follows:

- Threads with hard affinity to a single CPU or short-term binding pick only allowed CPU
- Interrupt threads that are being scheduled by their hardware interrupt handlers are scheduled on the current CPU if their priority is high enough to run immediately
- Starting from the last CPU on which the thread ran, walk down hierarchy until a CPU is found with valid affinity that can run the thread immediately
- Search whole system for least-loaded CPU running a lower-priority thread
- Search whole system for least-loaded CPU
- If search offers better CPU choice than last CPU on which thread ran, switch to it (the longer the sleep the more willing to move)

Rebalancing CPU Loads

Periodically need to rebalance threads between CPUs

When CPU idles, it looks for other CPUs from which it can steal work

When job added to a CPU with excessive load, it looks for other CPUs to which it can push work

Approximately once per second load balancer moves a thread from the busiest CPU to the least busy CPU

Questions

References:

- Marshall Kirk McKusick, George Neville-Neil, and Robert Watson, “The Design and Implementation of the FreeBSD Operating System, 2nd Edition”, Section 4.4 (pages 114-126)

Marshall Kirk McKusick

<mckusick@mckusick.com>

http://www.mckusick.com

Notes

FreeBSD Kernel Internals on Video

This 40-hour course is the detailed version of this introductory video and provides a complete background of the FreeBSD kernel. It covers all the topics in the book. In addition, it covers other related topics including performance measurement and system tuning. The first video provides an introduction to the FreeBSD community. The remaining videos consist of fifteen lectures on the FreeBSD kernel that align with the book chapters. There are assigned readings to be completed before viewing each lecture. The first thirteen lectures have a set of exercises to be done after each video is viewed. Follow-up comments on the exercises are provided at the beginning of the lecture following the one in which they are assigned.

The syllabus for the the course is as follows:

0) Preface: an introduction to the FreeBSD community
1) Introduction: kernel terminology and basic kernel services
2) Kernel-resource management: locking
3) Processes: process structure and process management
4) Security: security framework and policies, Capsicum, and jails
5) Virtual memory: virtual-memory management, paging, and swapping
6) Kernel I/O system: multiplexing I/O, support for multiple filesystems, the block I/O system (buffer cache), and stackable filesystems
7) Devices: special files, pseudo-terminal handling, autoconfiguration strategy, structure of a disk device driver, and machine virtualization
8) Local filesystem implementation: fast filesystem (FFS)
9) Local filesystem implementation: zettabyte filesystem (ZFS)
10) Remote filesystem implementation: network filesystem (NFS)
11) Interprocess communication: concepts and terminology, basic IPC services, system layers and interfaces, and code review of a simple application that demonstrates use of the IPC and network facilities
12) Network layer: IPv4 and IPv6 protocols, firewalls, and routing
13) Transport layer: TCP and SCTP
14) System startup: boot loaders, kernel startup, and system launch; system measurement tools
15) System tuning: performance measurement and system tuning

In addition to the preface and fifteen lecture videos, you also receive a copy of the course notes containing copies of all the overhead slides used in the course, an extensive set of written notes about each lecture, a set of weekly readings from this textbook, thirteen sets of exercises (along with answers), and a set of papers that provide supplemental reading to the text.

Tiered pricing is available for companies, individuals, and students. On-site courses can be arranged. For up-to-date information on course availability and pricing or to place an order, see the Web page at

http://www.mckusick.com/courses/

Advanced FreeBSD Course on Video

The 46-hour course provides an in-depth study of the source code of the FreeBSD kernel. It is aimed at users who already have a good understanding of the algorithms used in the FreeBSD kernel and want to learn the details of each algorithm's implementation. Students are expected to have either taken this class or a similar class taught by the instructor or to have throughly read and understood "The Design and Implementation of the FreeBSD Operating System, Second Edition" (published by Pearson Education's Addison-Wesley Professional division). They are also expected to have a complete background in reading and programming in the C programming language. Students will not need to prove relationship with a source license holder, as the course is based on the non-proprietary kernel sources released by the FreeBSD project.

The class consists of fifteen lectures on the FreeBSD kernel source code. The lecture topics are:

1) Organization, overview of source layout
2) Kernel header files
3) System calls and file opening
4) Pathname translation and file creation
5) Vnode interface mechanics, writing to an FFS file
6) Write to a ZFS file
7) Opening, using, and closing locally connected sockets
8) User datagram protocol and routing
9) TCP algorithms
10) Fork, exit, and exec
11) Signal generation and delivery, scheduling
12) Virtual memory header files, file mapping
13) Page fault service, pageout processing
14) NFS client and server operation
15) Multiplexing with select, system startup

In addition to the fifteen lecture videos, you also receive a CD-ROM with a copy of the FreeBSD kernel source covered in the lectures and a copy of the lecture notes.

Tiered pricing is available for companies, individuals, and students. For up-to-date information on course availability and pricing or to place an order, see the Web page at

http://www.mckusick.com/courses/

FreeBSD Networking from the Bottom Up on Video

This course describes the FreeBSD networking stack. It is made up of a series of lectures derived from tutorials given by George Neville-Neil.

The class currently consists of five lectures, though additional lectures are being developed. The current lecture topics are:

1) Device Drivers: how to write and maintain network drivers in FreeBSD. By way of example it uses the Intel Gigabit Ethernet driver (*igb*). The lecture covers the basic data structures and APIs necessary to implement a network driver in FreeBSD. It is specific enough that given a device and a manual, you should be able to develop a working driver on your own.
2) The IPv6 Stack: an in-depth discussion and code walk-through of version 6 of the IP protocols, describing and dissecting the paths that packets take from the driver layer up to the socket layer of the network stack. The lecture covers the four paths packets travel through the network stack: reception, transmission, forwarding, and error handling.
3) Routing: packet forwarding and routing subsystems in FreeBSD. The routing and forwarding code are the glue that keeps the networking stack together, connecting the network protocols, such as IPv4 and IPv6, to their underlying data link layers and making sure that packets are sent to the correct next hop in the network. Topics in the lecture include the Routing Information Base (RIB), Forwarding Information Base (FIB), and the systems that interact with them. Also covered are routing sockets and the RIB/FIB APIs, the address-resolution protocol (ARP), Neighbor Discovery (ND6), the Common Address Redundancy Protocol (CARP), the IP firewall and traffic shaper control program (*ipfw*), and the packet filter interface (*pfil*).
4) Packet Processing Frameworks: The FreeBSD Kernel has several different packet processing frameworks—software that is meant to transform packets but which are not traditionally considered to be network protocols. It is these packet processing frameworks that are often the basis for new products built with FreeBSD. This lecture covers all of the packet processing frameworks, including the Berkeley Packet Filter (BPF), IP Firewall (IPFW), Dummynet, Packet Filter (PF), Netgraph, and netmap. It discusses the appropriate use of each framework and takes a walk through the relevant sections of each framework. Working examples of extensions to each framework are given so that students can see how to build new systems with and around the frameworks that are present in the kernel.
5) A Look Inside FreeBSD Using DTrace. DTrace is a modern system that gives software developers the ability to add low overhead tracing that is always available to programs that they are creating, modifying, and debugging. The desired tracing is described and controlled with an advanced scripting language. This tutorial covers the basics of DTrace, including basic and advanced uses. Using a set of worked examples, students learn to add tracing to user space and kernel space systems. The tutorial includes a set of short labs carried out on virtual machines that give the students hands-on experience working with DTrace.

Each lecture may be purchased separately and comes with a copy of its course notes. Tiered pricing is available for companies, individuals, and students. For up-to-date information on course availability and pricing or to place an order, see the Web page at

http://www.mckusick.com/courses/

CSRG Archive CD-ROMs

Thanks to the efforts of the volunteers of the "UNIX Heritage Society" (see http://www.tuhs.org) and the willingness of Caldera to release 32/V under an open source license (see http://www.mckusick.com/csrg/calder-lic.pdf), it is now possible to make the full source archives of the University of California at Berkeley's Computer Systems Research Group (CSRG) available.

The archive contains four CD-ROMs with the following content:

CD-ROM #1—Berkeley Systems 1978–1986

1bsd	2.9pucc	4.1.snap	4.2buglist
2.10	2bsd	4.1a	4.3
2.79	3bsd	4.1c.1	VM.snapshot.1
2.8	4.0	4.1c.2	pascal.2.0
2.9	4.1	4.2	pascal.2.10

CD-ROM #2—Berkeley Systems 1987–1993

4.3reno	4.4BSD-Lite1	net.1
4.3tahoe	VM.snapshot.2	net.2

CD-ROM #3—Final Berkeley Releases

4.4	4.4BSD-Lite2

CD-ROM #4—Final /usr/src including SCCS files

Contrib	admin	games	local	sys
Makefile	bin	include	old	usr.bin
README	contrib	lib	sbin	usr.sbin
SCCS	etc	libexec	share	

The University of California at Berkeley wants you to know that these CD-ROMs contain software developed by the University of California at Berkeley and its many contributors.

The CD-ROMs are produced using standard pressing technology, *not* with write-once CD-R technology. Thus, they are expected to have a 100-year lifetime rather than the 10–20 years expected of CD-R disks. The CDs are sold only in complete sets; they are not available individually. The price for the 4-CD set is $99. The contents of the original four CD-ROMs plus some additional early UNIX distributions is available on a single DVD using 100-year lifetime M-DISC technology for $149.00. The archive can be ordered from

http://www.mckusick.com/csrg/

The compilation of this archive is copyright © 1998 by Marshall Kirk McKusick. You may freely redistribute it to anyone else. However, I appreciate you buying your own copy to help cover the costs that I incurred in producing the archive.

History of UNIX at Berkeley

Learn the history of the BSD (Berkeley Software Distributions) from one of the key developers who brings the history to life, complete with anecdotes and interesting footnotes to the historical narrative.

Part I is titled "Twenty Years of Berkeley UNIX: From AT&T-Owned to Freely Redistributable." The history of UNIX development at Berkeley has been recounted in detail by Marshall Kirk McKusick in his chapter in the O'Reilly book Open Sources: *Voices from the Open Source Revolution* and is now recounted in part one of this video. It begins with the start of the BSD community at the University of California at Berkeley in the late 1970s. It relates the triumphs and defeats of the project and its releases during its heydays in the 1980s. It concludes with the tumultuous lawsuit ultimately settled in Berkeley's favor, which allowed the final release in 1992 of 4.4BSD-Lite, an open-source version of BSD.

Part II is titled "Building and Running An Open-Source Community: The FreeBSD Project." It tells the story of the independent development by the FreeBSD project starting from the open-source release from Berkeley. The FreeBSD project patterned its initial community structure on the development structure built up at Berkeley. It evolved and expanded that structure to create a self-organizing project that supports an ever growing and changing group of developers around the world. This part concludes with a description of the roles played by the thousands of volunteer developers that make up the FreeBSD Project of today.

Dr. Marshall Kirk McKusick's work with UNIX and BSD development spans over thirty years. It begins with his first paper on the implementation of Berkeley Pascal in 1979, goes on to his pioneering work in the eighties on the BSD Fast File System, the BSD virtual memory system, and the final release of 4.4BSD-Lite from the University of California Berkeley Computer Systems Research Group. Since 1993, he has been working on FreeBSD, adding soft updates, snapshots, and the second-generation Fast Filesystem to the system. A key figure in UNIX and BSD development, his experiences chronicle not only the innovative technical achievements, but also the interesting personalities and philosophical debates in UNIX since its inception in 1970.

The price for the video is $19.95. The video can be ordered from

http://www.mckusick.com/history/