FreeBSD Internals
Day 1
An Overview

Joel Isaacson
Ascender Technologies Ltd

Today's Lectures

- Today we will have two sessions
  - Morning 9:00-12:00
  - Afternoon 14:00-17:00
- I have tried to retain some flexibility and interactivity.
- If you have any requests about the next lecture, please speak to me.

Historical Notes

- It all started with Unix over 30 years ago.
- Ken Thompson wrote it as a private research project on an idle PDP-7.
- Dennis Ritchie join the effort and designed the C language for the project.
- The system has rewritten in C with some small amount of assembly code.

Historical Notes

- V6 Unix was released in 1976 and was the first version widely available outside Bell Labs.
- It also was distributed in source code.
- It was very powerful for it's time.
- It had an elegant design.

Historical Notes

- The UNIX system was distinguished from other operating systems in three important ways:
  - It was written in a high-level language.
  - It was distributed in source form.
  - I provided powerful primitives normally provided on more expensive hardware.

Historical Notes

- The most influential of the groups working on UNIX was the CSRG (Computer Science Research Group) at Berkeley at the University of California.
- They received a large grant from DARPA for creating a platform for their projects.

Historical Notes

- A graduate student, Bill Joy, took the technical lead and was the chief architect.
- They released the 3BSD (Berkeley Software Distribution).
- The next major release was the 4BSD release.

Historical Notes

- Eventually the Berkeley group noticed that they had written the majority of code in the distribution.
- ATT had a minority stake in the code.
- The CSRG then decided to free BSD from the ATT license by rewriting the parts that were under the ATT license.

Historical Notes

- The BSD networking release 2 (1991) was a partial release without any ATT code which contained the TCP/IP stack code.
- Within 6 month of this release Bill Jolitz rewrote the missing parts of the kernel and released the 386/BSD distribution.

Historical Notes

- The 386/BSD distribution was beyond the capabilities of Bill Jolitz to manage.
- The BSD effort splintered and splintered again into three distributions:
  - NetBSD
  - FreeBSD
  - OpenBSD

Historical Notes
The Lawsuite

- In 1992 ATT sued BSDI and subsequently the University of California for copyright violation.
- During the trial the rights to the UNIX system was sold to Novell which wanted to terminate the lawsuite.

Historical Notes
The Lawsuit

- The lawsuit was settled with BSD retaining the rights to distribute their system.
- The 4.4BSD-Lite release (June 1994) was released unencumbered by ATT (Novell) licensing.
- The CSRG group in Berkeley disbanded.

Historical Notes

- It is interesting to note that the lawsuit caused people not to use the various BSD distributions.
- People turned to a rather obscure Unix-like system called Linux.
- BSD never again regained the lead.

It's The License Dummy

- The BSD's have a license that allows pretty much anything to be done with the sources.
- There is no need to return changes to the source to the copyright holder.
- On the other hand Linux has a more restrictive license.

FreeBSD

- The most popular BSD derivative is FreeBSD.
- It has moderate popularity.
- Traditionally the networking component (the TCP/IP stack) is the strongest part of the BSD systems.

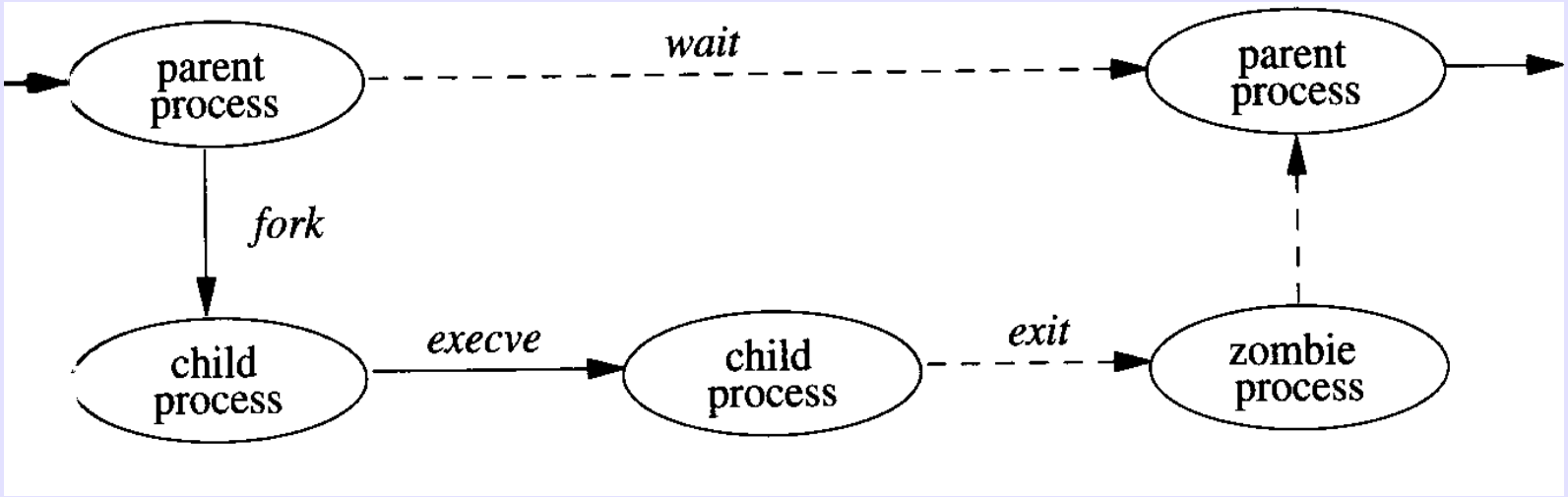
User level Interface

- We first will deal with the user interface.
- FreeBSD inherits the user API from Unix (POSIX).
- It has remained closer to the original Unix design that Linux.
- Linux has gone further than FreeBSD in terms of originality.

Process Creation

- User processes are managed and created with the standard UNIX fork/exec API.
- FreeBSD differs from Linux in the thread model:
  - FreeBSD has lightweight processes
  - Linux has only heavyweight processes

Process Creation



Signals

- The process can define signal handlers that will be called when signals are received.
- BSD defined a reliable signal model that allows signals to be masked and handled much as interrupts are handled in hardware.

Process Groups.

- Processes are organized into process groups.
- Processes inherit their process group from their parents.
- Signals can be sent to process groups.
- A group of processes can all be controlled by “job-control”

Memory Management

- The process space is initially divided into four logical segments:
  - Text- read/only
  - Data- read/write
  - Bss- read/write
  - Stack- read/write

Memory Management

- The stack segment is extended automatically.
- The program is broken into a number of fixed sized (4k) pages.
- Programs do not have to have pages resident in order to run.
- A demand paging system is used.

Memory Management

- The user process maintains its memory space with the **mmap** system call.
- Mmap deals with paged aligned memory segments.
- Regions can be either private (copy-on-write) or shared.

I/O Systems

- The basic I/O model consists of a sequence of bytes.
- There are no access methods.
- No particular structure is imposed by the kernel.
- I/O from one program can be redirected to any other program.

File Descriptors

- Descriptors represent underlying objects supported by the kernel:
  - Files – Persistent data usually on disks
  - Pipes – Linear array of bytes
  - Fifos – Named Pipes
  - Sockets – Interprocess and intercomputer communication

File Descriptors

- Most process have three standard descriptors open on creation:
  - 0 – standard input
  - 1 – standard output
  - 2 – standard error
- When a process terminates all process descriptors are closed.

Socket IPC

- The 4.2 BSD kernel introduced the IPC mechanism known as sockets.
- Today all operating systems have adapted this API for networking.
- Without the 4.2 reference TCP/IP implementation the internet might not have become what it is today.

Kernel Services

- The FreeBSD kernel can be viewed as a service provider to user processes.
- Processes usually access these services through system calls.
- Some kernel services are implemented as processes that execute in kernel mode.

System Processes

- All FreeBSD processes originate from a single process created at kernel startup.
- Kernel process function wholly within the kernel.
- The kernel starts a kernel process for each device that handle interrupts. This is different than the Linux interrupt model.

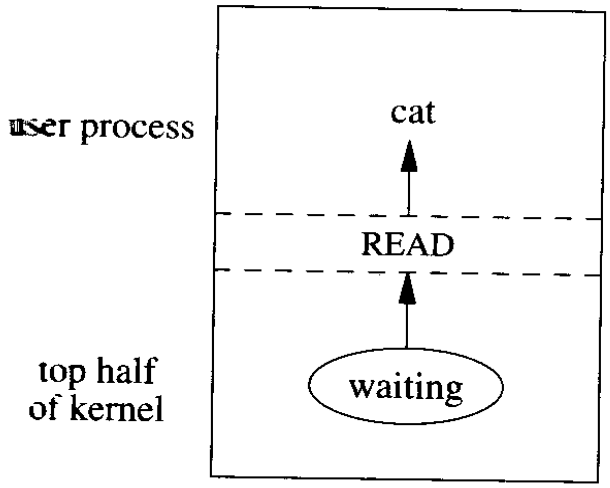
System Entry

- Entrance into kernel mode can be categorized according to the event that initiates it:
  - Hardware interrupt
  - Hardware trap
  - Software trap

Two Halves Make a Whole

- The kernel can be divided into a *top half* and a *bottom half*.
- It is important to note that the direction of top-bottom is inverted as opposed to the Linux nomenclature.

Kernel Structure



Preemptive scheduling;
cannot block;
runs on user stack in user address space.

Runs until blocked or done;
can block to await a resource;
runs on per-process kernel stack.

Scheduled by interrupts;
can block to await a resource;
runs on per-interrupt kernel stack.

Differences From Linux

- As opposed to Linux, entry into the kernel always involves a stack switch.
- Even interrupt handlers have their own context and stack.
- The process (thread) scheduling priority has a different class for top-kernel processes.

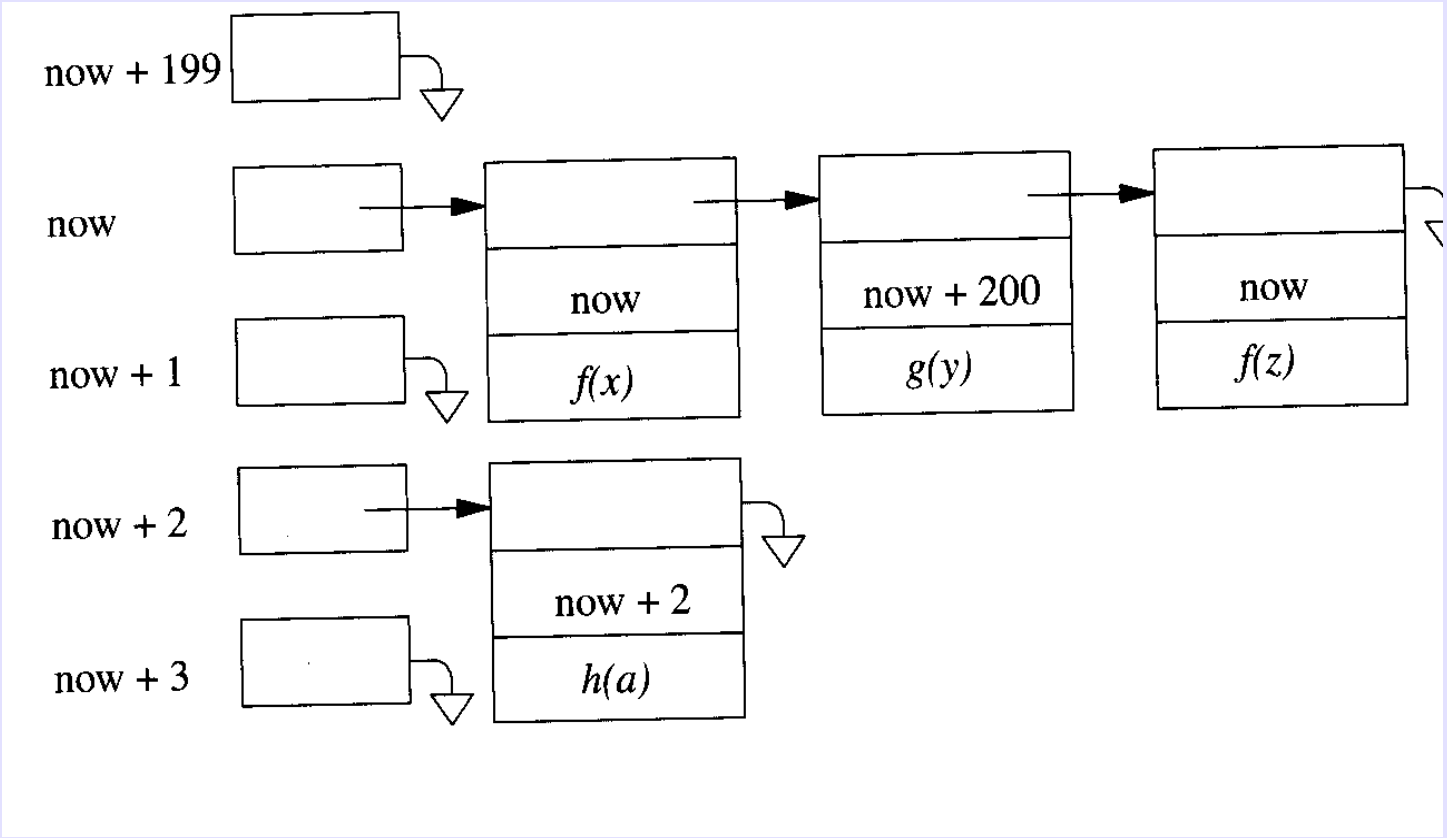
Entries into Kernel

- There are three major types of handlers for kernel entries:
  - syscall() - for a system call.
  - trap() - for hardware traps and software-initiated traps.
  - interrupt handlers.

Timer Handling

- Timer event scheduling is done using a discrete multiple event queue.
- Each timer event is put into a queue at an offset from the current time click modular the size of the event queues.
- The softclock() routine is called to execute the timer callback.

Timer Handling



Process Layout

- Each process starts with four memory areas: text, bss, data and stack.
- Other memory areas are added using the mmap system call.
- The shared library facility is implemented via mmap by a method that is very similar to the method used by linux.

Process Layout

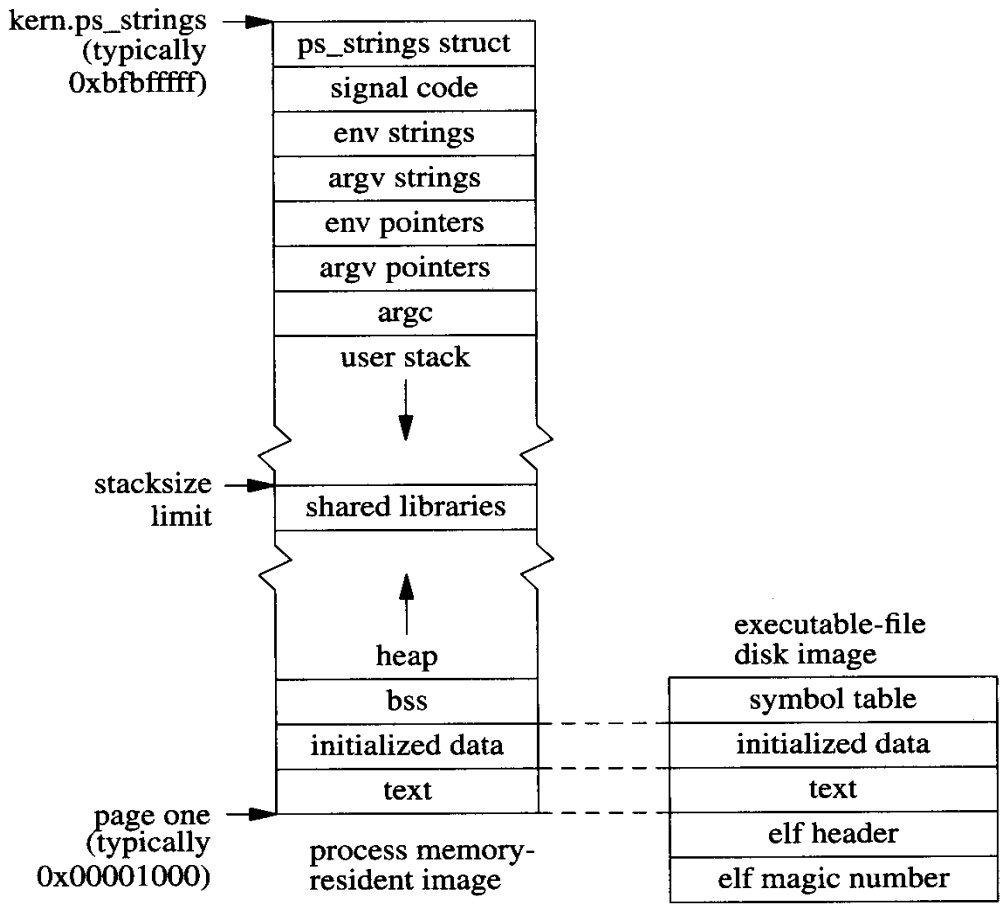
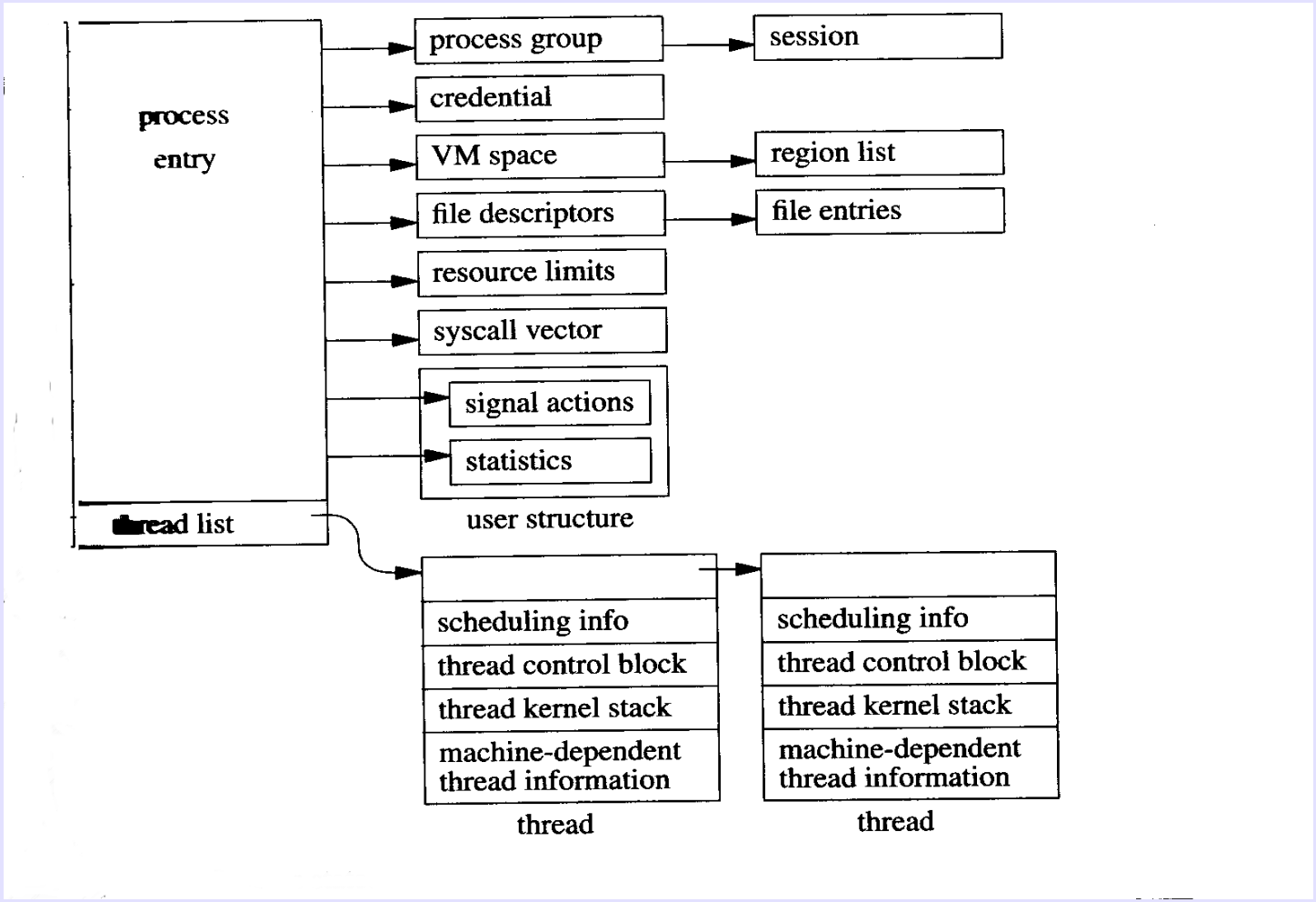


Figure 3.3 Layout of a FreeBSD process in memory and on disk.

Execution Framework

- Executable files have their execute bits set.
- Each executable file has an exec header with magic numbers for a particular type of executable.
- There are two types of execution models: direct or interpreter.

Process State



Thread Scheduling Classes

Thread-scheduling classes.

Range	Class	Thread type
0 – 63	ITHD	Bottom-half kernel (interrupt)
64 – 127	KERN	Top-half kernel
128 – 159	REALTIME	Real-time user
160 – 223	TIMESHARE	Time-sharing user
224 – 255	IDLE	Idle user

Locking Hierarchy

- There is a hierarchy of locks
  - hardware – test and set
  - spin lock
  - sleep mutex – reschedule
  - lock manager – interprocess synchronization
  - witness – partially ordered locks