Week Two

Communication Software

Host based communications

- Workstations serve as the source or destination for communications
- User and kernel mode
  - User mode refers to user space which typically serve as the application layer
  - Kernel mode is closest to the hardware and typically implements physical layer
  - Trade-off between memory protection and overall system performance
- Network Interfaces

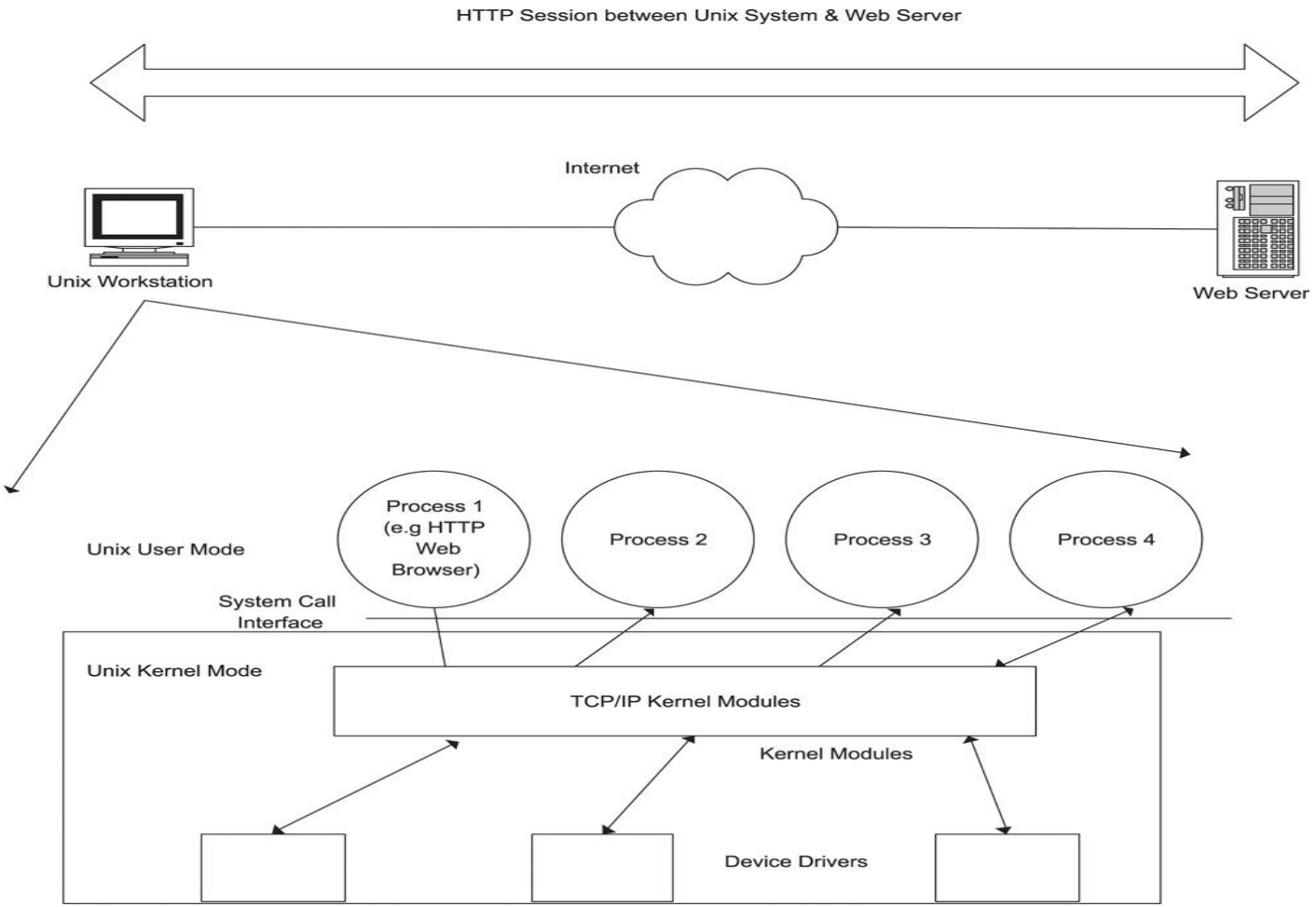
Host based communications

- Memory Protected Operation Systems
  - User Application Processes
    - runs on user space and make no assumption on HW
    - Interacts with kernel using well defined system calls
    - Context of a process is its state, defined by its text, data, registers and entries in process table
      - The **process table** is a data structure maintained by the operating system to facilitate context switching and scheduling, etc..
      - Each entry in the table called a **context block**, contains information about a process such as: process name and state, priority, registers, and a semaphore it may be waiting on and an entry to the page table.
  - Kernel
    - Isolates user from HW idiosyncrasies
    - Kernel switches the context when it executes another process

Host based communications

- Kernel and User Process Context
  - Moving between kernel mode and user mode is not context switch -- It is done usually via system calls or interrupts
  - When kernel services an interrupt, it does it in the context of executing process (it does not schedule a new process to handle interrupts)

Host based communications



Streams Architecture

- Used in several Unix hosts to implement network protocol stack
- Provides flexibility by providing dynamic “add and drop” of functional modules
- Set of kernel-resident system calls and resources to facilitate creation of data paths between kernel and user mode

Streams Architecture

- Streams Messages
  - Each message consists of multiple message blocks… each block points to a data block
  - Multicasting of message blocks -> more than one message block can point to the same data block
  - Memory management of data blocks are controlled by a reference count for memory efficiency- when count is zero data block is released
- Streams were the first host-based support for protocol stacks which could be dynamically loaded and unloaded (Kernel Loadable Modules)
  - Critical for systems that require kernel rebuild

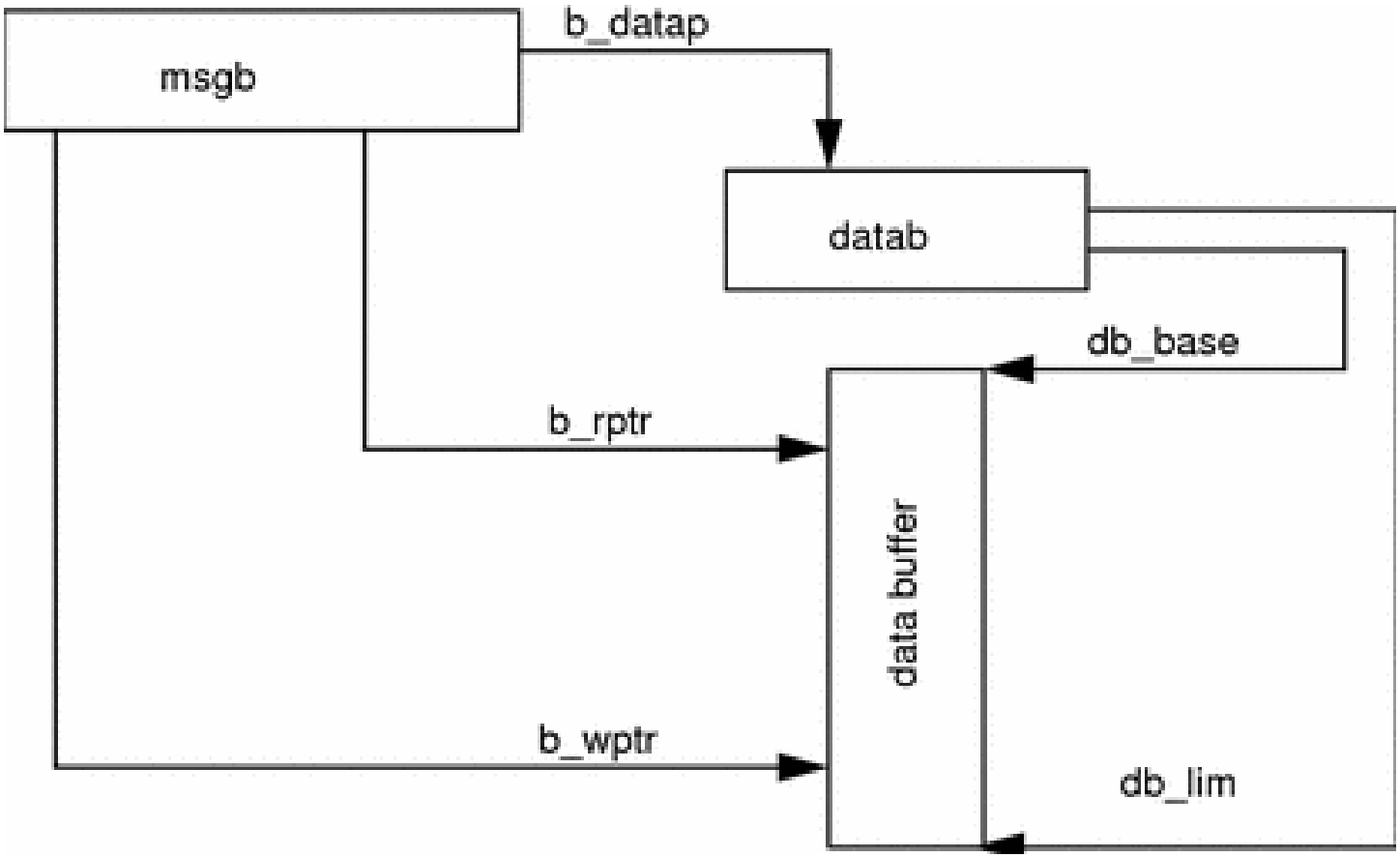
Streams Architecture

- **Streams message and data block structures.**
- struct msgb *b_next; /* Ptr to next msg on queue */
- struct msgb *b_prev; /* Ptr to prev msg on queue */
- struct msgb *b_cont; /* Ptr to next message blk */
- unsigned char *b_rptr; /* Ptr to first unread byte*/
- unsigned char *b_wptr; /* Ptr to first byte to write*/
- struct datab *b_datap; /* Ptr to data block */
- unsigned char b_band; /* Message Priority */
- unsigned short b_flag; /* Flag used by stream head */

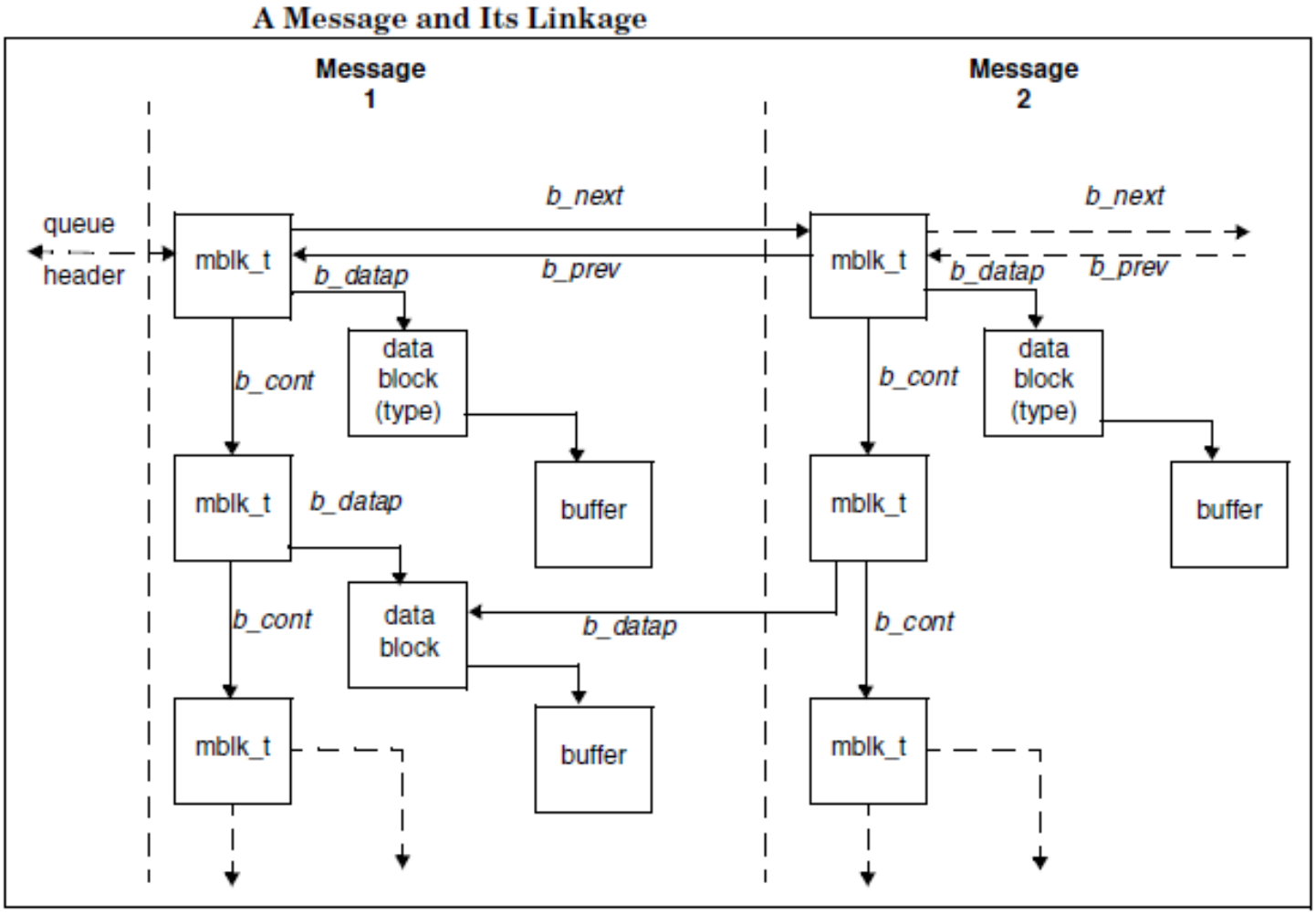
- **The data block organization is as follows:**

- unsigned char *db_base; /* Ptr to first byte of buffer */
- unsigned char *db_lim; /* Ptr to last byte (+1) of buffer*/
- dbref_t db_ref; /*Reference count- i.e.# of ptrs*/
- unsigned char db_type; /* message type */

Streams Architecture

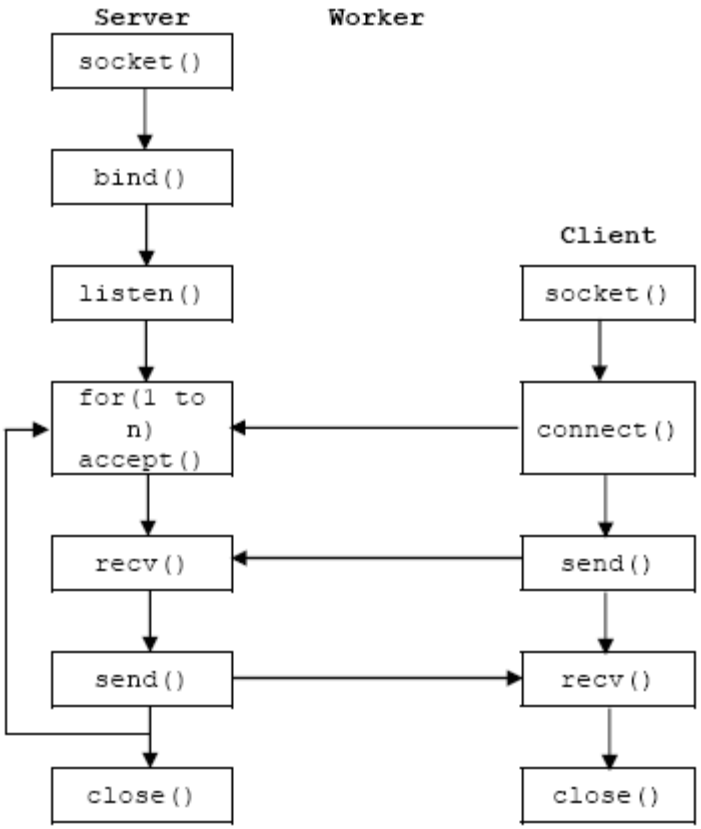


Streams Architecture

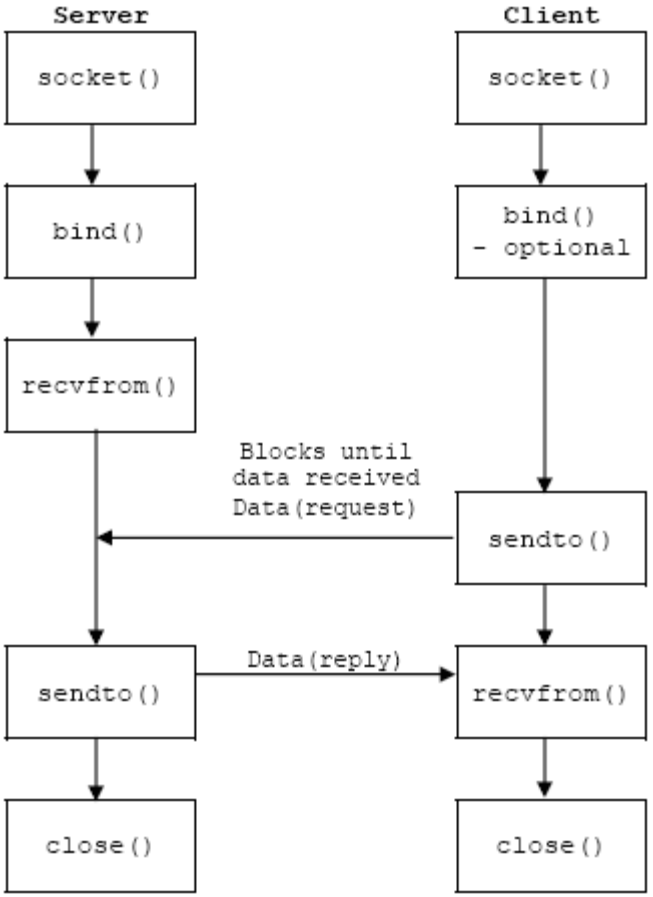


Streams Socket Types

- TCP



UDP

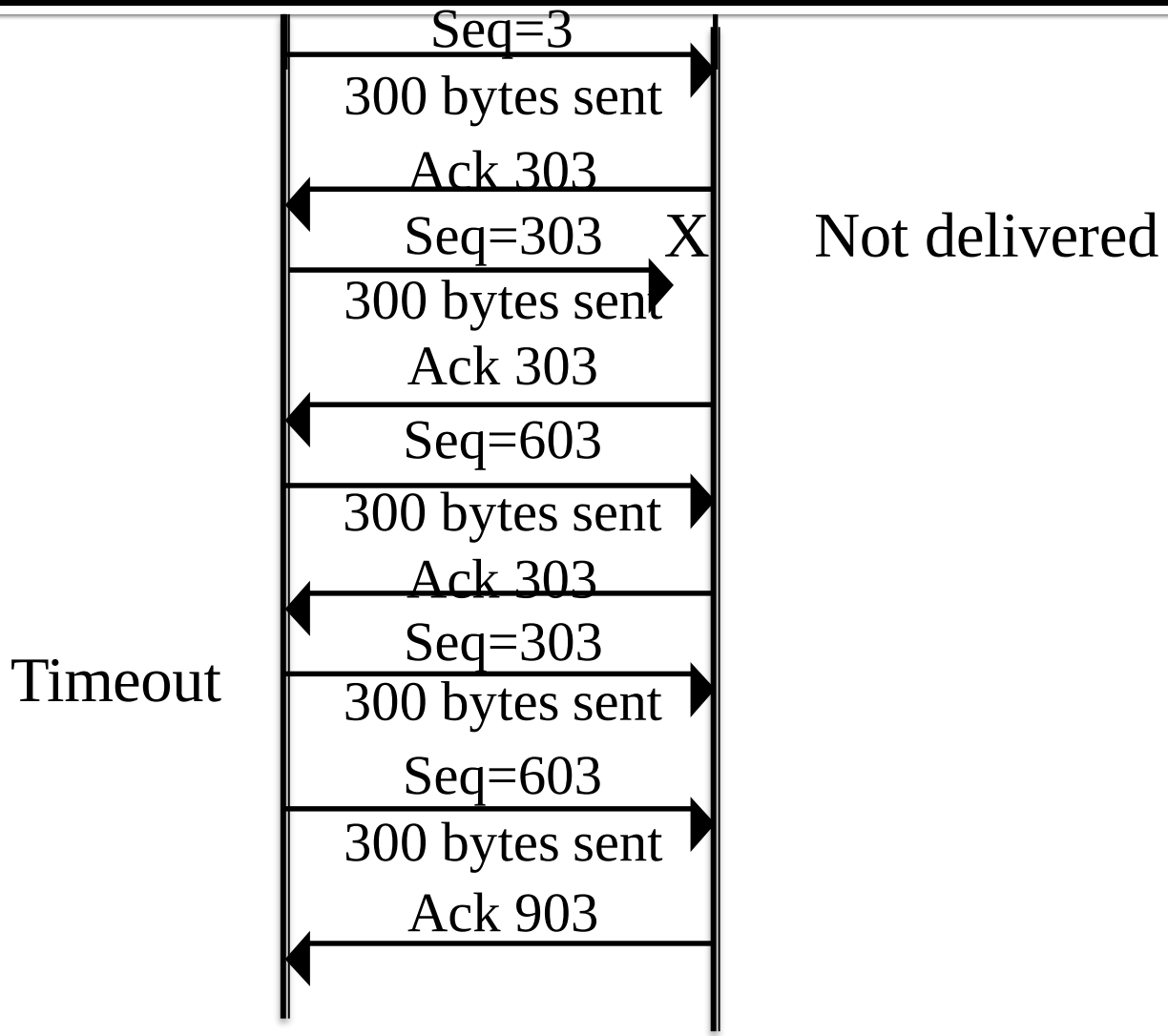


- https://www.tutorialspoint.com/unix_sockets/socket_server_example.htm
- https://www.tutorialspoint.com/unix_sockets/socket_client_example.htm

TCP

- **Communicating with TCP**
  - The client and server can not know how many bytes are sent in each write.
  - Delivered chunks are not always the same size as in the original write.
  - Reads must be handled in a loop to cope with stream sockets.

TCP Retransmission



UDP

- **Communicating with UDP**
  - UDP data is always a complete message (datagram).
  - Receiver receives the complete datagram unless fewer bytes are read.
  - Reading in a loop for a single datagram is pointless with UDP

A Simple Client Call

- To make a connection, a client must:
  - select UDP or TCP…
  - determine a server's IP address…
  - determine the proper port…
  - make the socket call…
  - make the connect call (TCP)…

Homework

- Write a client and server program using TCP or UDP to perform the following:
  - Client program sends 2 numbers to the server
  - Server waits 15 seconds and replies with the sum of the two numbers
  - Server should handle servicing up to 4 clients

Issues with Host Based Systems

- Not high performance
  - Very little scope for hardware acceleration
  - Inherent limitations of a host based OS
  - Software built for functionality and not performance
- Can be mitigated by:
  - Moving performance critical functions to kernel
  - Re-tooling scheduler of host OS to meet real time performance criteria

Embedded Communications Software

- An embedded communication device is a dedicated piece of hardware
- Common characteristics:
  - Usually runs on a real time OS
  - Limited memory or flash based memory
  - Diskless or limited hard disk space
  - Provides terminal (RS-232) or Ethernet interface for control and configuration
  - Hardware acceleration facility

RTOS

- Why RTOS?
  - RTOS vendors support complete lifecycle of the development environment
  - Vendors tune to RTOS for performance efficiency since they support the embedded processor as well
  - Has efficient and performance tuned infrastructure components for system software

RTOS

- Board Support Package (BSP)
  - Package that contains all board H/W-specific code that conforms to a specific operating system
    - Boot loader
    - OEM adaptation layer
    - Device drivers for all devices on the board
    - Optional root file system

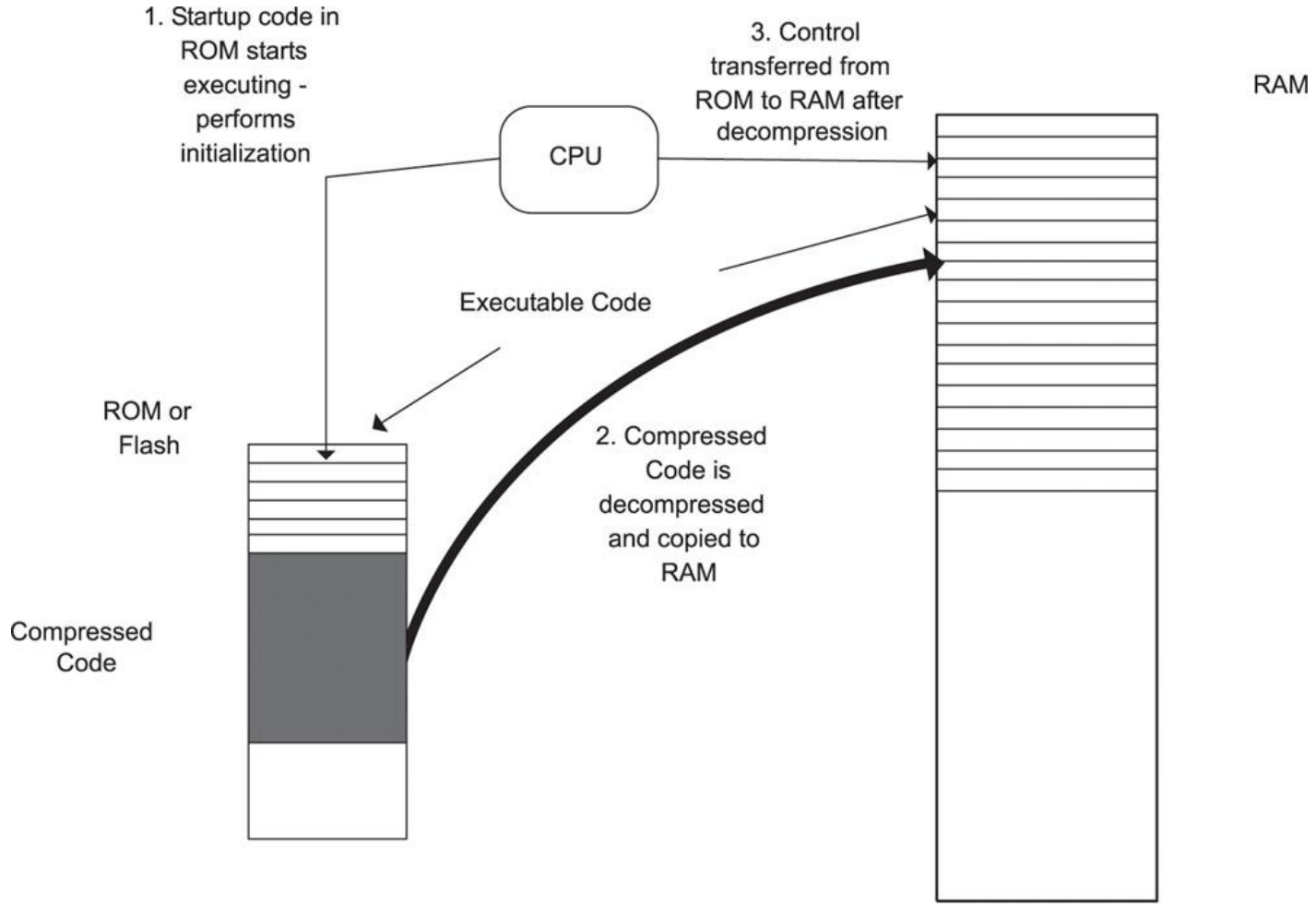
RTOS

Issue	Standard RTOS	Proprietary RTOS
Performance for a specific application	Less Optimized	More optimized
Maintenance	Responsibility of RTOS vendor	Responsibility of developer
Portability to multiple processors	Provided by RTOS vendor via separate packages	Has to be provided by developer for each processor
Support for standard Ethernet/serial devices	Provided as part of RTOS package for board	Has to be developed
Modifiability	Can be done only if RTOS vendor provides source code	Easily modifiable since source code is available internally
Tool Chain Support	Supported by RTOS vendor	Need to build using third-party development tools
Standard Interfaces, IPC mechanisms and APIs	Provided by RTOS vendor	Has to be designed in by the developer
Cost	High in dollar terms	Low in upfront dollar terms but could be high because of development effort/time and debugging

Memory

- Embedded devices typically don't have a disk drive and boot off a Programmable ROM or flash
- RAM space is used by the software application and RTOS for data structures, dynamic buffering and memory requirements

Memory


1. Startup code in ROM starts executing - performs initialization
3. Control transferred from ROM to RAM after decompression
RAM
CPU
Executable Code
ROM or Flash
2. Compressed Code is decompressed and copied to RAM
Compressed Code

Memory

- Unlike a host-system OS like Unix, RTOS does not have a defined kernel mode
  - Memory protection between tasks is therefore implemented only when appropriate
- Memory corruption bugs are manifested in indirect ways. Hence:
  - Difficult to identify a reason for a system crash and to identify which function caused a memory crash
  - Therefore recommended to use memory protection if available in a Memory Management Unit (MMU) or the RTOS

.

EPROM Configuration and Image Download

- Configuration is often stored in local non-volatile memory like an EEPROM or flash
  - Example: Router's IP addresses of its interfaces; routing protocols timer values, peer information.
- Permit the download of a new image to upgrade the existing image on the flash.
  - Field upgrades to avoid shipping the system back to the manufacturer for bug fixes and upgrades.
  - The new image is downloaded to a separate area of the flash so that the system can recover in case the download is unsuccessful or if the new image has problems

Device Issues

- Embedded communications devices do not come with a monitor and a keyboard.
- The only way to communicate with the embedded device is through a serial port or Ethernet.
- The communications device typically has a Command Line Interface (CLI), which allows the user to type commands for configuring the device.
- Ethernet port is typically used for communicating with the device for management purposes.
- The TCP/IP stack runs over this Ethernet port, so that the network administrator can telnet to the device over this port to access the CLI. This facility is used by network administrators to manage the device from a remote location.

Device Drivers

- Some RTOSes have their own communications stacks integrated or available as an additional package. These stacks interface to the hardware drivers through standard interfaces.

Function	Description
`open ()`	causes the device to be made active
`close ()`	results in the device being made inactive
`read ()`	is used for reading data received by the device
`write ()`	is used for writing data to the device
`ioctl ()`	is used for configuring and controlling the device

HW/SW Partitioning Guidelines

- Partitioning is driven by varying and often conflicting optimization constraints:
  - Size – Performance tradeoff
    - Minimize power, Minimize cost, Minimize computation time, Minimize communication time, etc…
    - Usage of memory and caching which is performance parameter
  - Fast and Slow Path
    - Path followed by most of the packets through the system is Fast while path taken by least number of packets is Slow path
    - Separation between fast and slow path serves as basis for hardware acceleration

HW/SW Partitioning Guidelines

- Boot loader is usually in EEPROM or boot ROM or flash
- DRAM is used to hold executable code and packets/buffer during transmission of packets
- SRAM is usually used to store tables for caching, since it requires faster lookup but SRAM occupies more space and is more expensive than DRAM

HW/SW Partitioning Guidelines

- Network software running on single processors are insufficient when:
  - Data rates to be supported are high and/or
  - A large number of interfaces are to be supported
- Solution?

Hardware Acceleration

- H/W acceleration used for FAST path processing
- ASIC (Application Specific Integrated Circuit)
  - Allows designers to add specific functionality
  - Expensive to develop
  - Proprietary and allows for intellectual property protection
- Network Processors (Vector Processor DSPs)
  - Is a programmable ASIC optimized for networking
  - Common functions required for packet processing are optimized and implemented using a reduced instruction set
  - Protocols evolve making programmable H/W an important tool in packet processing.

Control and Data Planes

- Classical planar networking architecture partitioning - three planes:
  - **Data**: work required for basic operations
  - **Control**: communication with peers and tables for correct operation of data plane
  - **Management**: control and configuration of system
- Hardware acceleration implemented in data plane's fast path

HW and SW acceleration check list

- Design the code to be modular
- Separate the fast-path and slow-path implementation of the data plane
- Maximize performance in the data plane
- Handle all exception processing in software
- Ensure that interrupt processing code is efficient
- Do not restrict the design such that only certain parts of the data plane may be built in hardware. Network processor devices can move several data plane functions into software.

HW and SW acceleration check list

- Ensure that performance calculations are made for the system both with and without hardware acceleration
  - This is also one way to determine the effectiveness of the partitioning.
- When interfacing to hardware, use generic APIs instead of direct calls to manipulate registers on the controller.
  - This will ensure that only the API implementations need to be changed when a new hardware controller is used. Applications will see no change since the same API is used.

References

- Designing Embedded Communications Software, by T. Sridhar,
- IT Architectures and Middleware – 2nd edition. Chris Britton, Peter Bye. Addison-Wesley
- Tanenbaum & van Steen Distributed Systems: Principles and Paradigms, 2nd ed.
- Advanced Programming in Unix Environment , W. Richard Stevens & Stephen A. Rago
- Middleware for Communications, Qusay H. Mahmoud
- Middleware Architecture with Patterns and Frameworks, Sacha Krakowiak