ADVANCED OPERATING SYSTEMS

Assignment -1

SUBMITTED BY:
Manish Choudhary
902982487

Question 1: Consider a processor that supports virtual memory. It has a virtually indexed physically tagged cache, TLB, and page table in memory. Explain what happens in such a processor from the time the CPU generates a virtual address to the point where the referenced memory contents are available to the processor.

Answer 1:

The following activities take place from the time the CPU generates virtual address to the point where the referenced memory contents are available to the processor:

1) The CPU generates a virtual address of the form:

Virtual Page Number(VPN)	Page Offset

2) As the cache into consideration is virtually indexed and physically tagged, TLB lookup and cache look up can be performed in parallel.

a) TLB Lookup:

- Generally implemented as fully associative cache and thus, each entry is of the form:

Tag	Data

(There may be some other metadata bits like valid bit.)

- Converts virtual page number (Tag) to physical frame number (data) by performing parallel tag matching against the virtual page number (some most significant bits in the virtual address generated by the CPU).

b) Cache Lookup:

- Each cache entry is of the form:

Valid bit	Tag	Data

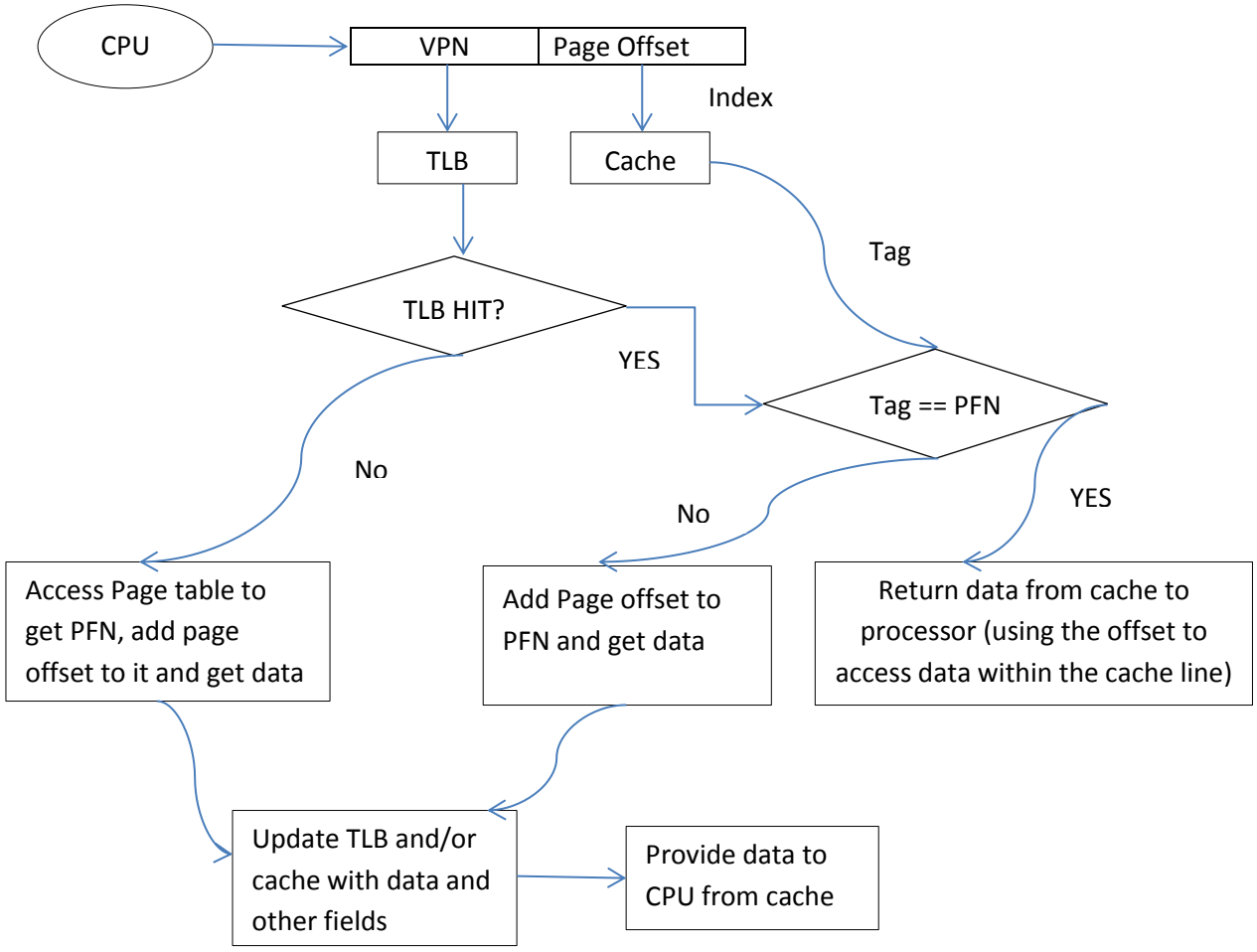
Each cache line may be defined as set (in case of set associative cache) providing multiple candidate locations in cache for each memory location. Here, the tag is derived from the physical address.

- The cache interprets the virtual address as:

VPN/TAG	Index	Block Offset

- Cache index and block offset are derived from the unchanging part of the virtual address (page offset part).
- Index is used to search for the corresponding cache line (all possible spots, referred to as a set).

3) If the tag is not found in the TLB, it waits for the miss service.
4) If the tag is found in TLB, the corresponding PFN is matched against the tag value of the cache line found in the cache.
5) If the tag matches and the valid bit is set (cache hit), the corresponding data is provided to the processor.
6) If the tag matching fails (cache miss), the physical frame number corresponding to the virtual page number stored in TLB is added to the page offset to access the memory location in the main memory.



7) In case of TLB miss, the page table corresponding to the running process is accessed with the virtual page number as the index. The stored value is physical frame number which is added to the offset to get the memory address in the main memory from where the data is retrieved (In case of demand paging: only if the valid bit is set i.e. page is present in the main memory. Otherwise, page fault may occur). After this, TLB and cache get updated as per the implementation with the data and other fields. Data is provided to CPU from cache.

Question 2: Distinguish between segmentation and paging.

Answer 2:

Paging:

- A memory-management scheme
- Physical memory is divided into fixed-sized blocks called **frames**
- logical memory is divided into blocks(called **pages**) of the size same as that of frames
- Pages of a process are loaded into available memory frames on execution
- Every address generated by the CPU is divided into two parts: a **page number** and a **page offset**.
- Page number is used as an index into a **page table** which contains the base address of each page in physical memory.
- Corresponding base address is combined with the page offset get the desired physical memory address
- Different types of paging – Hierarchical page tables, Hashed page table, Inverted page table

Advantages:

- Permits the physical address space of a process to be non-contiguous
- Avoids external fragmentation.
- Solves the problem of fitting memory chunks of varying size onto the backing store
- Facilitates sharing of common code
- Provides protection against memory access by other processes as separate page table is maintained per process and processes can't access memory outside their page table

Disadvantages:

- Internal fragmentation: on an average half page per process gets wasted
- Paging increases the context switch time.

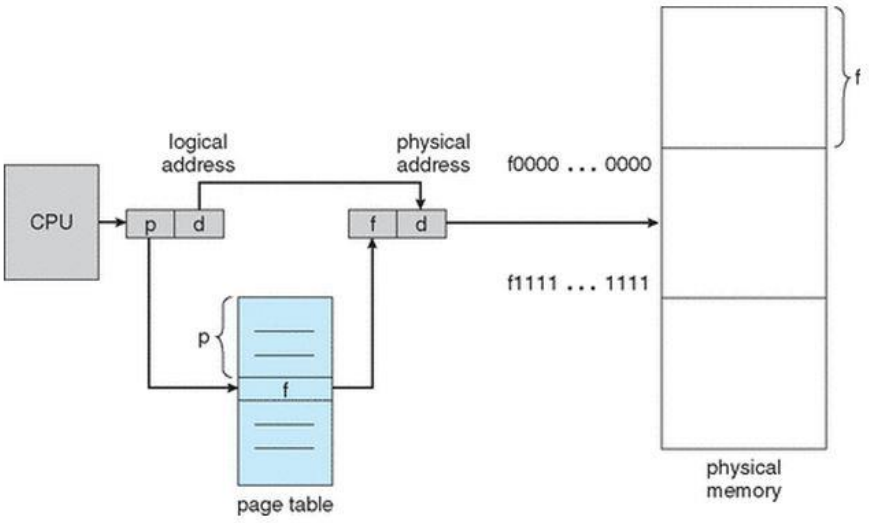


Figure: Paging [1]

Segmentation

- a memory-management scheme
- Supports **user view of memory** where memory is represented as a collection of variable sized segments and the length of each segment is governed by its purpose.
- Logical address space is collection of these segments.
- Each logical address is consists of **segment number** and the **offset** within the segment
- The segments are constructed by the compiler and numbered by the loader.
- **Segmentation table** is used for mapping the user defined two-dimensional address to one dimensional physical address.
- Segmentation table has entries in the form of **segment base** (the starting physical address of the segment) and **segment limit** (the length of the segment).
- Segment number is used as an index to segmentation table and the offset is compared against the segment limit. If it is exceeds the limit, trap to the operating system.
- Legal offset is added to the segment base to get the desired physical address in the main memory.

Advantages:

- Program components into individual segments at user discretion
- Modular approach
- No internal fragmentation

Disadvantage:

- External fragmentation possible because of variable sized segments.
- Difficult system optimization

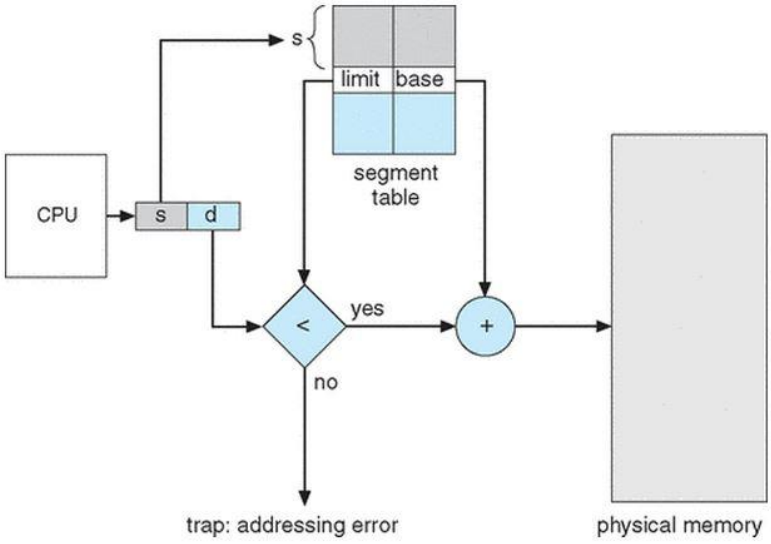


Figure: Segmentation [1]

Comparison:

PAGING	SEGMENTATION
Virtual memory can be greater than the physical memory.	Virtual memory can be greater than the physical memory.
User not restricted by the size of physical memory.	User not restricted by the size of physical memory.
User specifies a single address which is partitioned by hardware into page number and offset, all hidden from the user.	User is aware of the segmentation.
Fixed sized pages and frames with no relation among pages	Variable sized segments decided by the programmer, with each segment has some specific purpose
No support for modular design	Supports modular design which is useful for object oriented programming
Internal fragmentation possible	No internal fragmentation
No external fragmentation	External fragmentation possible because of variable sized segments
Easier to optimize the system as a whole as the page size is a system attribute	More difficult to optimize the system as a whole as the user has the control over the size of the segments

---

Question 3: Explain all the actions from the time a process incurs a page fault to the time it resumes execution. Assume that this is the only runnable process in the entire system.

Answer 3:

Page Fault:

Page fault is the term related to demand paging which is based on the idea of loading the parts of the program in the main memory only on demand which results in a better memory utilization.

To implement demand paging, the page table entry consists of an extra one bit field “Valid”. Bit value 1 means the page is present in the main memory and bit value 0 means that the page is not in the main memory.

If the valid bit is 0, it is called page fault and in this case, the missing page has to be brought into the main memory from the disk by the page fault handler.

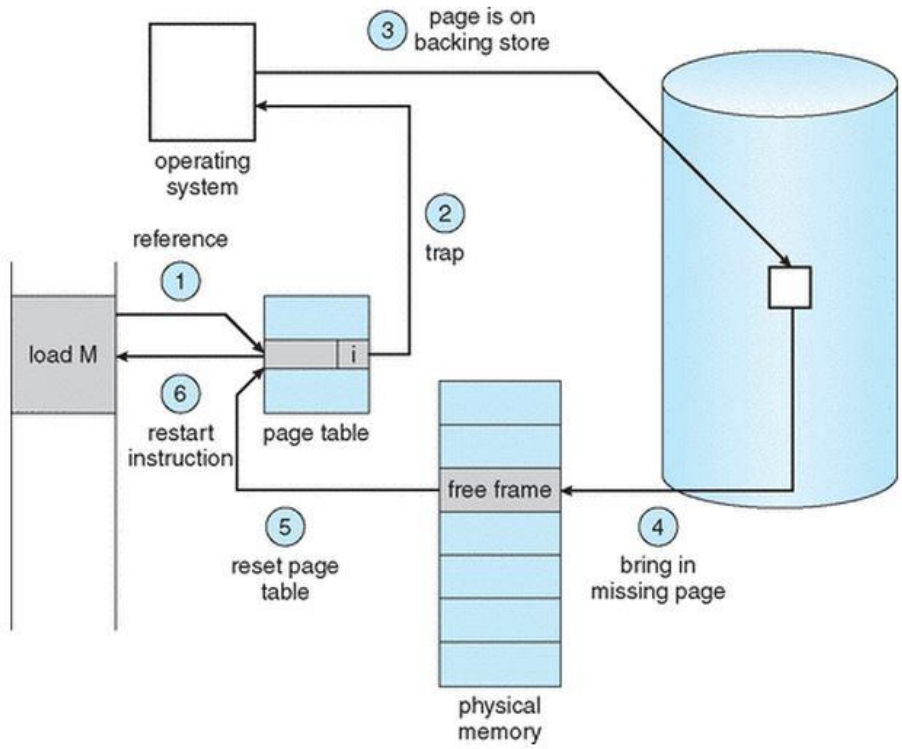


Figure: Steps in Page Fault Handling [1]

Following actions take place in case of a page fault (more detailed, not in sync with the diagram):

1) The page fault handler is called with some background processing like saving the state of the process.
2) **Find a free page frame:**
   - The page fault handler looks up the free list of page frames to get a free frame in which faulting page can be copied.
   - If the free list of page frames is empty, space has to be created to bring back the faulting page.
3) **Pick the victim page:**
   - To create the space and victim page is selected
   - Determine the victim process using the frame table
   - If it is a clean page (copies in main memory and disk are same), set the page table entry as invalid (no need to save the contents of this page).
   - If it is a dirty page, the page is written back to the disk at a location determined with disk map
4) **Load the faulting page:**
   - Using the disk map, the faulting page is read from the disk into the selected page frame.
5) **Update the page table for faulting process and the frame table:**
   - The value of frame in the page table entry is set to the selected page frame
   - The valid bit is set for this page table entry
   - The frame table entry for the selected page frame is updated
6) **Restart the faulting process:**
   - The faulting process is placed in the ready queue
   - Being the only runnable process, it is selected by the scheduler for execution

Question 4: Explain the following terms: working set of a process, thrashing, paging daemon, swapper, loader, and linker.

Answer 4:

A) Working Set of a process

- Defined to determine the locus of program activity (based on the principle of locality) i.e. the pages accessed by a process in a reasonably sized window of time
- As the locus of program activity changes over time, the working of a process also changes
- The size of the working set is the number of most recent distinct pages touched by a process in a window of time
- Working set model is used to control the impact of thrashing.
- If a page is in active use, it would be in the working set.
- If it is not used, it will be dropped from the set after window size time after the last reference
- Thus, it is good estimate of locality
- OS monitors the working set of each process and allocates it enough frames
- If there are extra frames, another process can be initiated
- If the sum of the working-set sizes exceeds the total number of available frames, the operating system selects a process to suspend
- This controls thrashing and provides high CPU utilization

B) Thrashing

- A situation when the system is not getting the useful work done
- Processes spend more time paging than computing
- A process page faults if it doesn't have enough number of frames to support the pages currently in active use
- To serve this, a frame is provided to it depending upon global or local allocation
- If global allocation is used, frame may be taken from some other process which may result in another page fault by the victim process.
- If local allocation is used, the same process would page fault as the page moved out was in active use.
- As more processes need the service of paging device, they get queued up and the ready queue becomes empty resulting in lower CPU utilization
- Seeing lower CPU utilization, the scheduler increases the degree of multi-programming resulting in further page faults and decease in CPU utilization.
- This high paging activity is called thrashing and result in overall lower CPU utilization
- After a point, the CPU utilization drops sharply resulting in a state where no useful work gets done.

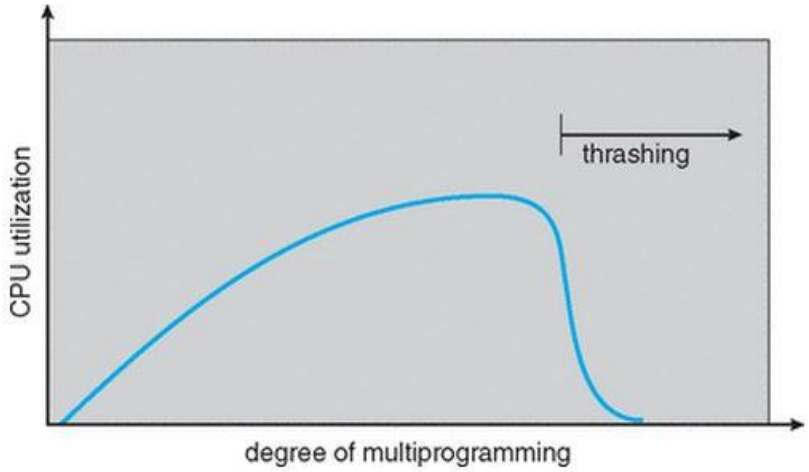


Figure: Thrashing [1]

C) Paging Daemon

- Runs as a background process responsible for some housekeeping task
- Tries to maintain a pool of free clean page frames
- Wakes up periodically to check whether the pool of free frames on the free-list is below some minimum threshold(minimum number of frames kept ready for allocation to satisfy page faults)
- If yes, page replacement algorithm is run to free up more frames to meet this minimum threshold

D) Swapper

- Moves an inactive process(waiting for I/O) from the memory to the disk
- Moves the ready-to-run but swapped out process (I/O request completed) from the disk to the memory
- If the paging daemon is not able to keep up the minimum number of frames in the free-list, swapper performs a more drastic action by swapping out the entire processes

E) Loader

- A part of the operating system
- Takes the disk representation of the program and creates the memory footprint(loads into the memory)
- Reads the contents of the executable file containing the program instructions into memory, and then carries out other required preparatory tasks to prepare the executable for running

F) Linker

- Takes the output of compilation which is an executable(object files)
- Links together the object files with the libraries
- Output is a binary representation of the program which is ready for execution on the processor
- Static and dynamic(late) linking
- Dynamic linking postpones the linking of shared/library files till run time.

Question 5: Explain page coloring and how it may be used in memory management by an operating system.

Answer 5:

Page Coloring

- Allocation of free pages that are contiguous from the CPU cache's point of view
- Careful allocation of pages such that each virtual page maps to a physical page which will be at a different location in the cache
- Physical memory pages are "colored" so that pages with different "colors" have different positions in CPU cache memory
- When allocating sequential pages in virtual memory for processes, the kernel collects pages with different “colors” and maps them to the virtual memory
- In this way, the adjacent pages of a process don’t content for the same location in the cache
- Another way of saying this, color both virtual and physical memory and allocate pages to the frame with color same as that of the page

Use in Memory Management

- Performance optimization designed to ensure that accesses to contiguous pages in virtual memory make the best use of the processor cache
- In case of physically tagged cache, two adjacent pages in virtual address space may not exist in the adjacent locations in cache or even can contend for the same cache line, resulting in poor cache performance and large number of cache overwrites
- Page coloring is used as an attempt to maximize the total number of pages cached by the processor
- Physical memory is colored in accordance to the cache size such that the sequential pages in virtual memory do not contend for the same cache line.
- Makes virtual memory as deterministic as physical memory in regard to cache performance
- Helps to get the benefit of spatial locality
- Allows having larger virtually indexed and physically tagged cache, independent of the page size.

Example of page coloring

Suppose the size of the cache (physically tagged) is equivalent to 10 pages. We have a process with 10 pages for allocation of frames. If the frame number 12 gets allocated to the page 0 of the process and if the next free frames are 22, 23…, 22 won’t be allocation to the page 1. Instead frame 23 would be assigned to the page 1. This is done because frame 22 would map to the same location in the cache as that of frame 12 resulting in non-optimal caching.

Question 6: Explain clearly the costs associated with a process context switch.

Answer 6:

Context Switch

- Switching the CPU to another process
- State save and state restore operations
- The context of the old process is saved in its PCB and the saved state of a new process scheduled to run is loaded

- Essential feature of multitasking operating system
- It is a pure overhead as the system does no useful work while switching

Cost

- Requires considerable processor time, which can be on the order of nanoseconds for each of the tens or hundreds of switches per second
- Process context includes all the information/states required to run a program:
    - User Context: code, data, stack, heap
    - Register Context: Values stored in all the registers, instruction pointer, stack pointer etc.
    - OS resources such as open files, signal related data structure
    - Other process management information
    - User context is saved in memory and other information is stored in process control block:
        - The process state.
        - The program counter, PC.
        - The values of the different registers.
        - The CPU scheduling information for the process.
        - Memory management information regarding the process.
        - Possible accounting information for this process.
        - I/O status information of the process.

    Saving/restoring all this information takes time.

- Interrupt, interrupt sensing, interrupt handling in kernel mode, saving the process state, restoring the state of other process, changing some pointer values and switching back to user mode incur cost.
- Depending on the implementation, the cost may also include flushing the cache, TLB etc.

Question 7: Explain the functionality of the different layers found in the network protocol stack of an operating system such as Linux.

Answer 7:

The functionality of different layers of Internet Protocol Stack:

1) **Application Layer:**
    - Provides support for network-based applications such as e-mail, file transfer, web browser
    - Protocols like HTTP(web), SMTP(e-mail), FTP(file transfer) can be employed on this
2) **Transport Layer:**
    - An abstraction to application layer
    - Provides end-to-end communication
    - Reliable and in-of-order delivery of packets with acknowledgements
    - Fragmentation/Defragmentation
    - TCP and UDP are two most common protocols on this layer
    - Addressing/ Multiplexing

- Flow control
- Congestion Avoidance

3) Network Layer

- Routing of packets
- Internet protocol employed on this layer
- Routing tabling formation, route identification
- IP address is used to route the packets

4) Link Layer

- hop to hop transfer of frames
- Ethernet, Token Ring, 802.11(Wi-Fi) can be employed on this layer
- Hop to hop frame error detection
- Logical link establishment between two nodes
- Frame traffic control

5) Physical Layer

- transmission and reception of the unstructured raw bit stream over a physical medium
- Responsible for physically moving the bits of the packets from one node to another
- Different media such as copper wires, a coaxial cable, an optical fibre etc.
- Data encoding

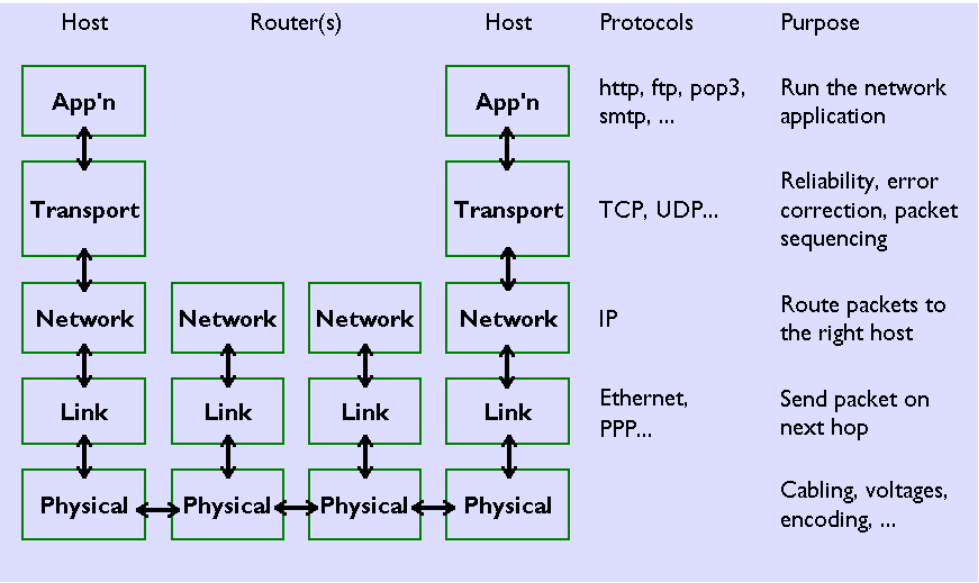


Figure: Internet Protocol Stack

REFERENCES:

[1] Abraham Silberschatz , Peter B. Galvin, Greg Gagne. Operating System Concepts, 8th Edition.
[2] Umakishore Ramachanran, William D. Leahy Jr. Computer Systems: An Integrated Approach to Architecture and Operating Systems.
[3] Wikipedia.