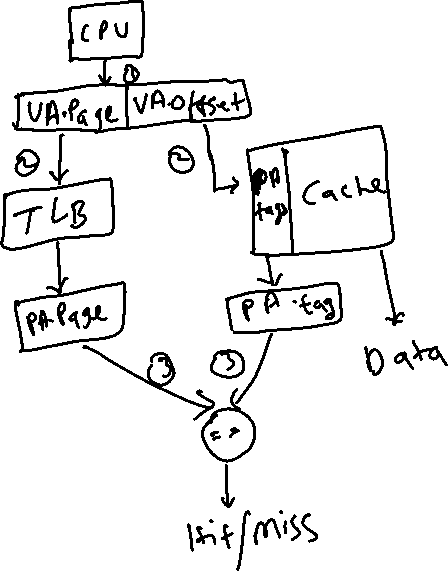
Q1. Consider a processor that supports virtual memory. It has a virtually indexed physically tagged cache, TLB, and page table in memory. Explain what happens in such a processor from the time the CPU generates a virtual address to the point where the referenced memory contents are available to the processor.

A1.

**Happy case:**

1. In Virtually Indexed Physically Tagged (VIPT) cache data is indexed by virtual address (VA) and tagged by physical address (PA).
2. Processor generates a VA.
3. The key idea in VIPT is to check TLB and Cache at the same time.
4. Processor separates VA into VA.Page and VA.Offset.
5. VA.Page is looked up in TLB and converted to PA.Page (or miss).
6. Processor looks up VA.Offset in Cache, uses VA.Offset as index, finds Cache entry (or miss). Cache entry is tagged with physical address (PA), PA.Tag.
7. PA.Page and PA.Tag are compared. If they are equal, we get a hit (or miss).



**TLB Miss – Page Table hit:**

1. Page table is used to find PTE (or miss).
2. Page table is loaded with PTE.
3. Rest is as happy case.

**TLB Miss – Page Table miss:**

1. If PTE is not present in Page Table, we get a page fault exception.
2. A3 for more details.

**VIPT cache miss:**

1. VA.Offset is appended to PA.Page to find location in physical memory.

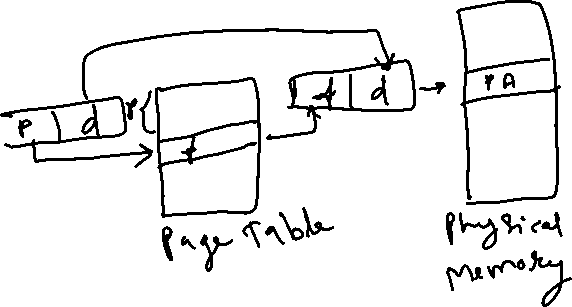
**Comparator miss:**

1. VA.Page is translated to PA.Page using TLB or Page Table or disk loaded to physical memory.
2. VA.Offset is appened to PA.Page to find PA.

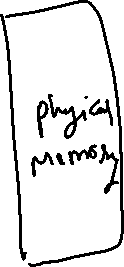
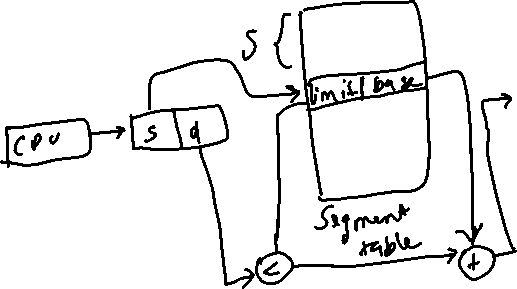
Q2. Distinguish between segmentation and paging.

A2.

**Paging:**



**Segmentation:**

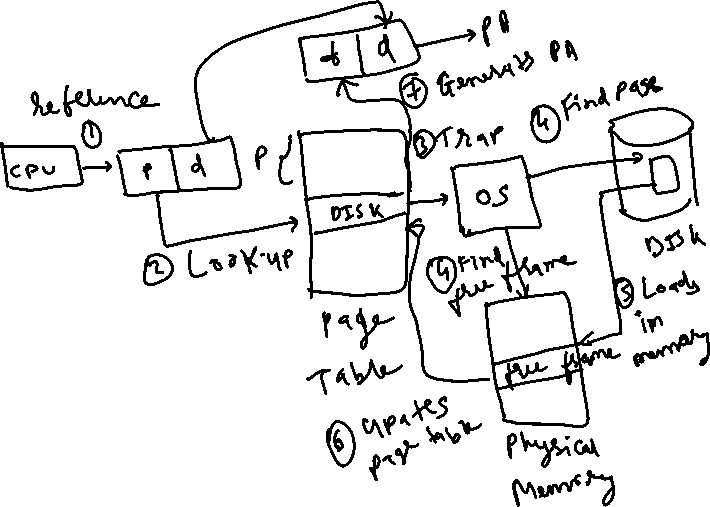


**Differences:**

|  |  |  |
| --- | --- | --- |
| **BASIS** | **PAGING** | **SEGMENTATION** |
| **Basic** | A page is of fixed block size. | A segment is of variable size. |
| **Fragmentation** | Paging may lead to internal fragmentation. | Segmentation may lead to external fragmentation. |
| **Address** | The user specified address is divided by CPU into a page number and offset. | The user specifies each address by two quantities a segment number and the offset (Segment limit). |
| **Size** | The hardware decides the page size. | The segment size is specified by the user. |
| **Table** | Paging involves a page table that contains base address of each page. | Segmentation involves the segment table that contains segment number and offset (segment length). |

Q3. 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.

A3.



1. CPU generated a virtual address (VA).
2. Page is looked up in Page Table.
3. If this page is marked as invalid (DISK) a trap (interrupt) is generated.
4. OS Operations:
   1. OS directs IO(Disk) Manager to find the page.
   2. At the same time OS directs Memmory Management Unit (MMU) to find a free frame. This step may involve a page replacement if required.
5. OS Loads the found page in the found free frame.
6. Page table is updated for this VA.
7. Physical address is generated and returned to the CPU for consumption.

Q4. Explain the following terms: working set of a process, thrashing, paging daemon, swapper, loader, and linker.

A4.

**Working set of process:**

1. Working set relies on Locality of Reference model, which claims that as process executes, process moves from locality to locality (locality is set of pages actively used together).
   1. Spatial locality - higher probability that nearby memory accessed by subsequent instructions. Examples are arrays, local variables, and code instructions also exhibit spacial locality.
   2. Temporal locality - higher probability that same memory accessed again within a short time period.
2. Pages referenced by a process recently (last (delta) seconds) are the working set.
3. Pages outside working set may be discarded. Working sets rely on concept of balance sets to ensure memory not overcommitted.
4. Divide runnable processes into active and inactive groups. Active processes called the balance set.
5. Working set strategy prevents thrashing while keeping the degree of multiprogramming high. The difficulty is keeping the working set updated.

**Thrashing:**

1. Problem which can occur when the system has committed (most/all) of available physical memory pages,

system must select page(s) to reuse (by moving dirty/changed pages to storage).

When paging overhead becomes excessive, system spends more time moving pages between memory and secondary storage than actual process execution.

**Paging Daemon:**

1. Process that runs and provides services to the OS, identifying pages that are candidates for discard and dirty write to secondary storage), and allocate for reuse.
2. Handles the VM cache replacement policy. Concerned with keeping physical memory available for allocation/use.

**Swapper:**

1. Swap entire processes out of memory and to secondary storage; supplanted by paging daemon which reuses parts of memory rather than entire processes.

**Loader:**

1. The part of an OS responsible for loading programs and libraries.
2. A linker on VM systems may not load program into memory, but simply declare a mapping to the VM between memory allocate and the executable file.
3. The VM may then fill memory on demand when program execution hits memory locations, and unused code may never be loaded to memory.
4. Tasks include validation, memory requirements, copy program image to memory, copy command line arguments to the stack, initialize registers, and jump to the program entry point.
5. Dynamic linking loaders load programs and link shared libraries to running programs.

**Linker:**

1. A linker or link editor takes object files produced by a compiler, and resolves references.
2. Object and library files are combined using symbols:
   1. local - internal to object file that can be used for relocation
   2. external defined - which allow other modules to call/reference
   3. external undefined - which reference other modules and include those libraries in output (may only link the referenced modules)
3. Arranges the object files in the program address space, and relocate code by replacing fixed base addresses with retargeted absolute loads, jumps, stores.
4. A final linker pass may be needed when hardware does not have virtual memory to perform relocation, or the executable is position independent.
5. Dynamic linking delays some relocation and symbol resolution until program is run.

Q5. Explain page coloring and how it may be used in memory management by an operating system.

A5.

1. Page Coloring is also known as Cache Coloring.
2. Assign colors to Physical memory page.
3. Same color pages map to the same lines or sets in the cache.
4. Different colors are assigned different cache lines.
5. Number of colors = (cache size) / (page size \* associativity)
6. Page color = (page number) modulo (number of colors)
7. Attempts to allocate contiguous free pages from the CPU cache viewpoint, reduce cache conflicts.
8. Goal is to increase cache usage by selecting pages that do not contend with neighbor pages.
9. Avoids sequential VM pages contending for the same cache line (exploits spatial locality).
10. Virtual memory made as deterministic as physical memory with regards to cache performance.
11. On page fault, allocate a physical page with the same color as the virtual page.
12. Adds significant complexity to the VM memory allocation subsystem.
13. OS keeps separate per-color free lists.

Q6. Explain clearly the costs associated with a process context switch.

A6.

1. Process context switch is computationally expensive as it requires border crossings between user process space and kernel space.
2. Context switch occurs when hardware or software interrupt occurs due to system call or timer.

Consider switching from process P1 to process P2:

* 1. A hardware protection level change so kernel code/data can be access.
  2. CPU state for process P1 (registers, flags) saved to process memory.
  3. Saving process P1 memory maps, to process memory.
  4. FPU (and other dedicated) hardware context are saved too.
  5. CPU cache line(s) are marked invalid.
  6. OS scheduler algorithm runs to determine next process (P2).
  7. MMU loaded with address space for process P2.
  8. Switch to OS dedicated stack, mark stack position to allow for nested interrupts.
  9. FPU (and other dedicated) hardware context restored.
  10. Loading process P2 memory maps.
  11. CPU state for process P2 is restored.
  12. Protection level change to user level.
  13. Interrupt return performed to transfer control to P2.

Other costs not strictly part of the context switch include:

1. Extra cache flushes and page faults which occur since memory may have been paged out.
2. Extra physical page flushes to save changed (dirty) pages.

Q7. Explain the functionality of the different layers found in the network protocol stack of an operating system such as Linux.

A7.

Linux(Unix) defines a 4-layer network stack (compared to the OSI Reference model 7-layer protocl stack).

**(4) Application Layer (OSI Session, Presentation, Applicatio Layers):**

1. Application layer provides end applications and API for building programs.
2. Provides applications services such as FTP, HTTP, NFS, SMTP, SNMP, DNS, your program, etc.
3. Provides a system call interface - library functions (read/write, open/close, socket, connect, etc) for opening and closing connections, sending and receiving data messages/streams, which are used to build application layer programs.

**(3) Transport Layer (OSI Transport Layer):**

1. Transport layer provides quality of service and reliability, a data stream or packets.
2. TCP (transmission control protocol) connection oriented, provides error detection and retransmission, packet reordering, acknowledgement, flow control, congestion control. between endpoints (hosts).
3. UDP (user datagram protocol) connectionless, unreliable, (more deterministic) provides packet transmission, unreliable, application must perform desired connection management (message ordering reassembly, error detection, acknowledgement, flow control, etc).
4. Converts Application data between TCP/UDP packets (add/remove TCP/UDP header)

**(2) Network (Internet) Layer (OSI Network Layer):**

1. Network (or Internet) layer handles packet routing via logical addressing.
2. Includes IP, ICMP, IGMP protocols, provides IP routing, computes checksum, encapsulates TCP/UDP packets into IP packets prepended with IP Header.
3. Network layer is responsible for how data is sent across network, provides routing information,
4. divides data into packets (and reassembles data).
5. Converts TCP/UDP packets between Network packets (add/remove IP header)

**(1) Link Layer (Ethernet) (OSI Data Link and Physical Layers):**

1. Provides device driver and network interface card (NIC) interface (kernel interface).
2. Device interface transports data between kernel/device drivers, network data blocks are divided into packets suitable for the physical layer to transport.
3. Passes IP packets to physical device (ethernet) where physical headers added/removed.
4. Packets are moved between host and NIC memory, system is interrupted when data send/receive done.
5. Transmits bits between endpoints of a network connection.
6. Converts Link packets between Network packets (add/remove Link/Ethernet header).