2. MMU and TLB

Young W. Lim

2021-07-06 Tue

Outline

1. Based on

2. Virtual memory
   - Memory Management Unit and Translation Lookaside Buffer

Based on

"Study of ELF loading and relocs", 1999
http://netwinder.osuosl.org/users/p/patb/public_html/elf_relocs.html

Compling 32-bit program on 64-bit gcc

- gcc -v
- gcc -m32 t.c
- sudo apt-get install gcc-multilib
- sudo apt-get install g++-multilib
- gcc-multilib
- g++-multilib
- gcc -m32
- objdump -m i386

TOC: Memory Management Unit

MMU and TLB

- Memory Management Unit (MMU)
  - hardware unit that translates a virtual address to a physical address
  - every memory reference is passed through the MMU
- Translation Lookaside Buffer (TLB)
  - a cache for the virtual-to-physical translations table of MMU
  - not needed for correctness
  - but source of significant performance gain

https://cseweb.ucsd.edu/classes/su09/cse120/lectures/Lecture7.pdf

MMU (Memory Management Unit) (1)

- MMU (memory-management unit) hardware
  - maps logical address to physical address
- OS together with MMU
  - the user program generates the logical address and
  - thinks that the program is running in this logical address
  - but to access physical memory for its execution,
    this logical address must be mapped
    to the physical address by MMU

https://www.geeksforgeeks.org/logical-and-physical-address-in-operating-system/

MMU (Memory Management Unit) (2)

- MMU is the hardware responsible for implementing virtual memory
- sits between the CPU core and memory
- usually the part of the physical CPU
- separate from the RAM controller
  DDR controller is a separate IP block

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf

MMU (Memory Management Unit) (3)

- transparently handles all memory accesses
  from load / store instructions
- maps memory acceses using virtual addresses
  to system RAM and peripheral hardware
- handles permissions
- generates an exception (page fault)
  on an invalid access

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

MMU (Memory Management Unit) (4)

- the MMU manages virtual address mappings
  - maps virtual addresses to physical addresses
- the MMU operates on basic units of memory : pages
  - page size varies by architecture
  - some architectures have configurable page sizes

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

MMU (Memory Management Unit) (5)

- common page sizes
  - ARM - 4k
  - ARM64 - 4k or 64k
  - MIPS - widely configurable
  - x86 - 4k

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

MMU (Memory Management Unit) (6)

- a page is
  - a unit of memory size
  - aligned at the page size
  - abstract
- a page frame refers to
  - a physical memory block
    which is page sized and page aligned
  - physical
- the pfn (page frame number) is often
  used to refer to physical page frames
  in the kernel

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf

MMU (Memory Management Unit) (7)

- the MMU operates on pages
- the MMU maps physical frames to virtual addresses
- a memory map for a process contains many mappings
- a mapping often covers multiple pages
- the TLB holds each mapping
  - virtual address
  - physical address
  - permissions

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf

TLB (Translation Lookaside Buffer)

- when CPU accesses a virtual address
  TLB is consulted by the MMU
  - if the virtual address is in the TLB,
    the MMU can look up the physical address
  - if the virtual address is not in the TLB,
    the MMU will generate a page fault exception
    and interrupt the CPU
  - if the virtual address is in the TLB,
    but the permissions are insufficient,
    the MMU will generate a page fault

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf

TLB (translation lookaside buffer) (1)

- Virtual Memory would not be very effective
  if every virtual memory address had to be translated
  by looking up the associated physical page in memory.
- the solution is to cache the recent translations
  in a Translation Lookaside Buffer (TLB)

`https://courses.cs.washington.edu/courses/cse378/00au/Lec28.pdf`

TLB (translation lookaside buffer) (2)

- the TLB is a small cache of
  the most recent virtual-physical mappings
- by checking here first, temporal locality is exploited
  to speed virtual address transaltion
  - while a virtual-to-physical translation is under way,
    the hardware checks to see if it has seen
    this translation recently

https://courses.cs.washington.edu/courses/cse378/00au/Lec28.pdf

TLB (translation lookaside buffer) (3)

- fast associative memory
  keeps most recent translations
  (logical page, page frame)
- determine whether non-offset part of LA
  (logical address) is in TLB (translation lookaside buffer)
  - if so, get corresponding frame num for physical address
  - if not, wait for normal memory translation (parallel)

`https://cseweb.ucsd.edu/classes/fa03/cse120/Lec08.pdf`

Translation cost with TLB

- cost is determined by
  - speed of memory : ~ 100 nsec
  - speed of TLB : ~ 20 nsec
  - hit ratio : fraction of refs satisfied by TLB, ~95%
- Speed with no address translation : 100 nsec
- Speed with address translation
  - TLB miss : 200 nsec (100% slowdown)
  - TLB hit : 120 nsec (20% slowdown)
  - avarage : 120 * .95 + 200 * .05 = 124 nsec

https://cseweb.ucsd.edu/classes/fa03/cse120/Lec08.pdf

TLB design issues

- the larger the TLB
  - the higher the hit ratio
  - the slower the response
  - the greater the expense
- TLB has a major effect on performance
  - must be flushed on context switches
  - alternative : tagging entries with PIDs
- MIPS: has only a TLB, no page tables
  - devote more chip space to TLB

`https://cseweb.ucsd.edu/classes/fa03/cse120/Lec08.pdf`

Basic TLB mappings (1)

- user virtual address space
  - mapped pages unmapped space
- physical address space
  - allocated frames
- TLB mapings
  - TLB entries (page, page frame)
  - virtually contiguous regions
    not physically contiguous

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

Basic TLB mappings (2)

- mappings to virtually contiguous regions
  do not have to be physically contiguous
- easy memory allocation
- almost all user space code does not need
  physically contiguous memory

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

Multiple processes

- each *process* has its own set of mappings
- the same virtual addresses in two different processes will likely be used to map different physical addresses
  - (page, page frame1) for process 1
  - (page, page frame2) for process 2

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

Shared memory (1)

- shared memory is easily implemented with an MMU
- simply map the same physical frame
  into two different processes
- the virtual addresses need not be the same
  - for pointers to values inside a shared memory region
    the virtual addresses must be the same

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf

Shared memory (2)

- the shared memory region can be mapped to different virtual addresses in each process
- the `mmap()` system call allows the user space process to request a specific virtual address to map the shared memory region
  - if the kernel cannot grant a mapping at this address, `mmap()` returns with failure

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

Page faults

- when a process acceses a region of memorythat is not mapped, the MMU will generate a page fault exception
- the kernel handles page fault exceptions regularly as part of its memory management design
- TLB can contain only the part of the required maps for a process
- page faults at context switch time
- lazy allocation

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

Lazy allocation (1)

- the kernel does not allocate pages immeidately
  that are requested by a process
- the kernel will wait until those pages are actually used
- lazy allocation to optimize a performance
  - if the requested pages may not be actually used,
    then the allocation will never happen

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf

Lazy allocation (2)

- when memory is requested for allocation,
  the kernel simply creates
  a record of the *request* in its page tables
  and then returns (quickly) to the process,
  without updating the TLB
- when that newly-allocated memory is actually accessed,
  the CPU will generate a page fault,
  because the CPU doesn't know about the mapping
  (no entry in the TLB)

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

Lazy allocation (3)

- in the page fault handler,
  the kernel uses its page tables
  to determine that the mapping is valid
  (from the kernel's point of view)
  yet unmapped in the TLB
- the kernel will allocate a physical page frame
  and update the TLB with the new mapping
- the kernel returns from the exception handler and
  user space program can resume

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

Lazy allocation (4)

- in a lazy allocation case, the user space program is never aware that the page fault happened
- the page fault can only be detected at the time that was lost to handle it
- for processses that are time-sensitive pages can be pre-faulted, or simply touched, at the start of execution
  - see also `mlock()` and `mlockall()`

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

Page tables (1)

- the entries in the TLB are a limited resource
- far more mappings can be made than can exist in the TLB at one time
- the kernel must keep track of all of the mappings at all times
- the krenel stores all these informations in the page tables
  `stuct_mm` and `vm_area_struct`

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

Page tables (2)

- since the TLB can only hold a limited subset of
  the total mappings for a process,
  some valid mappings will not have TLB entries
- when these addresses are touched
  the CPU will generate a page fault
  because the CPU has no knowledge of the mapping
  only the kernel does

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

Page tables (3)

- the page fault handler will
  - find the appropriate mapping for the offending addresses in the krenel's page tables
  - select and remove an existing TLB entry
  - create a TLB entry for the page containing the address
  - return to the user space process
    - observe the similarities to lazy allocation handling

`https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf`

Swapping (1)

- when memory utilization is high,
  the kernel may swap some frames to disk
  to free up RAM
- the MMU makes this possible
  - the kernel may copy a frame to disk and
    remove its TLB entry
  - the frame may be reused by another process

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf

Swapping (2)

- when the frame is needed again,
  the CPU will generate a page fault
  because the address is not in the TLB
- at a page fault time, the kernel can
  - put the process to sleep
  - copy the frame from the disk
    into an unused frame in RAM
  - fix the page table entry
  - wake the process

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf

Swapping (3)

- note that when the page is restored to RAM,
  it is not necessarily restored to the same physical frame
  where it originally was located (before being swapped out)
- the MMU will use the same virtual address though,
  so the user space program will not know the difference
  - this is why user space memory cannot
    typically be used for DMA

https://elinux.org/images/b/b0/Introduction_to_Memory_Management_in_Linux.pdf