Topics in Operating Systems

Kernel Virtual Address

Debadatta Mishra, CSE, IITK

Kernel Virtual Memory

- Why not treat kernel as an isolated MM context?

Kernel Virtual Memory

- Why not treat kernel as an isolated MM context?
  - Require MM context loading/unloading on user-kernel context switch
  - In kernel context, user data is accessed (a lot!) why?
  - Even worse, user data of many processes accessed
  - In X86, a small part of the kernel can not be isolated as HW does not perform MM context switch
- Requirement: efficient memory isolation between user and kernel

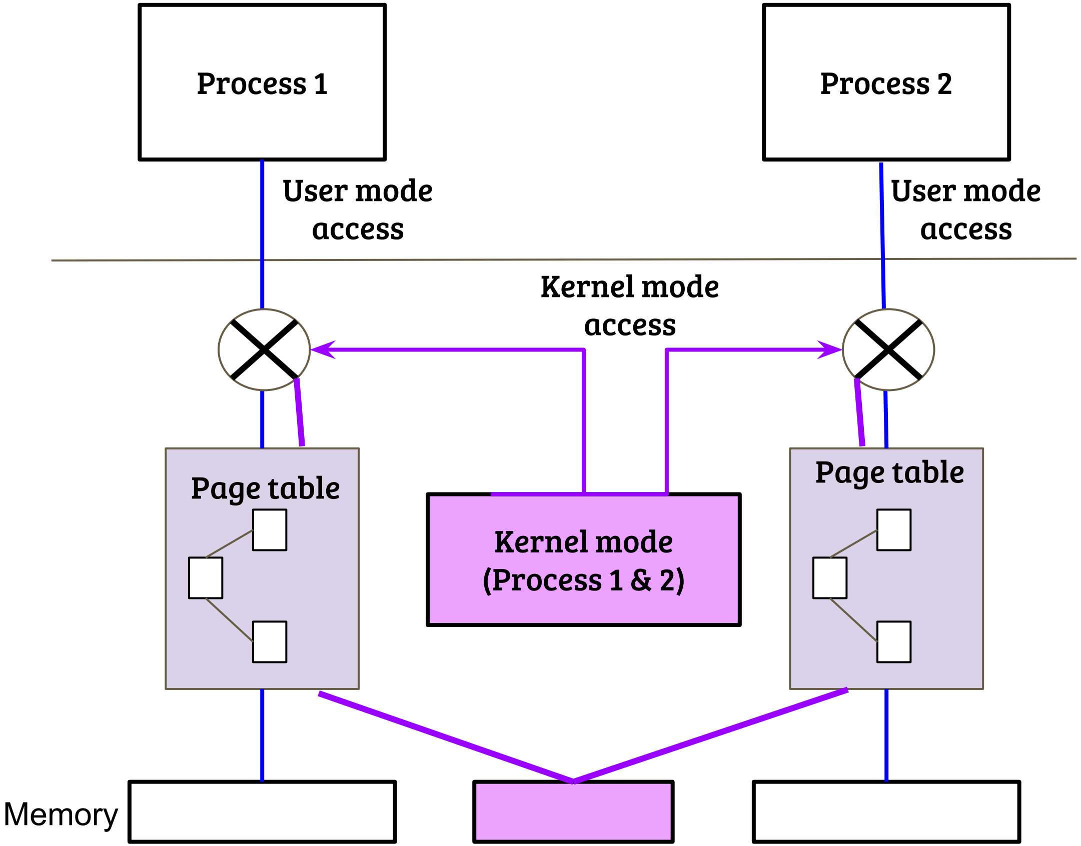
Kernel Virtual Memory

- Why not treat kernel as an isolated MM context?
    - Require MM context loading/unloading on user-kernel context switch
    - In kernel context, user data is accessed (a lot!) why?
    - Even worse, user data of many processes can be accessed
    - In X86, a small part of the kernel can not be isolated as HW does not perform MM context switch
- Requirement: efficient memory isolation between user and kernel
    - Let kernel use the same MM context of the user process
    - No context switch, no problems of accessing user data
- However,

Kernel Virtual Memory

- Why not treat kernel as an isolated MM context?
  - Require MM context loading/unloading on user-kernel context switch
  - In kernel context, user data is accessed (a lot!) why?
  - Even worse, user data of many processes can be accessed
  - In X86, a small part of the kernel can not be isolated as HW does not perform MM context switch
- Requirement: efficient memory isolation between user and kernel
  - Let kernel use the same MM context of the user process
  - No context switch, no problems of accessing user data
- However,
  - Kernel VM change propagation? Compromised Isolation!

Issue of Kernel VM propagation



- Kernel virtual address mapping should be present in both process page tables.
- Ex: If kernel allocates memory while serving syscall from process-1, process-2 in kernel mode should see it !
- How to design?

Constraints: Processes and memory are dynamically created and destroyed

Linux strives on family values!

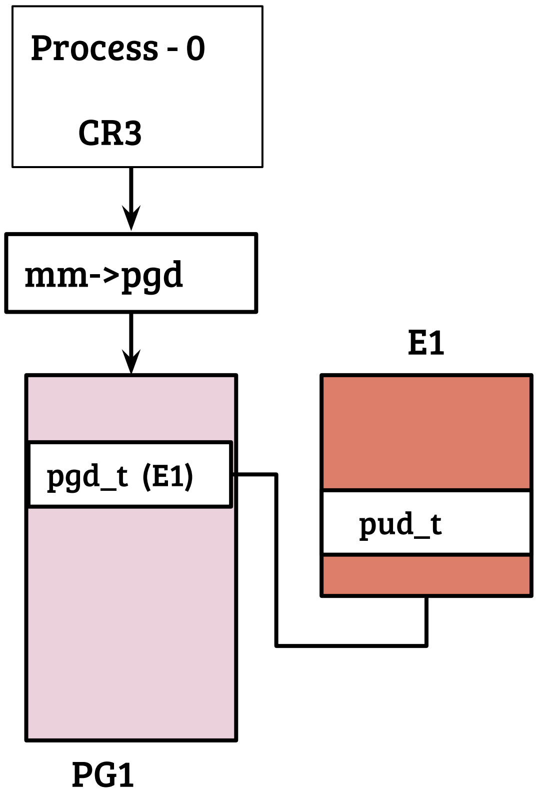
- A child process page table inherits the kernel mappings of the parent
- By implication, the inheritance tree is rooted at the first process
- Mapping changes ⟶ update mapping in every process?
  - Does not look good!

Linux strives on family values!

- A child process page table inherits the kernel mappings of the parent
- By implication, the inheritance tree is rooted at the first process
- Mapping changes → update mapping in every process?
  - Does not look good!

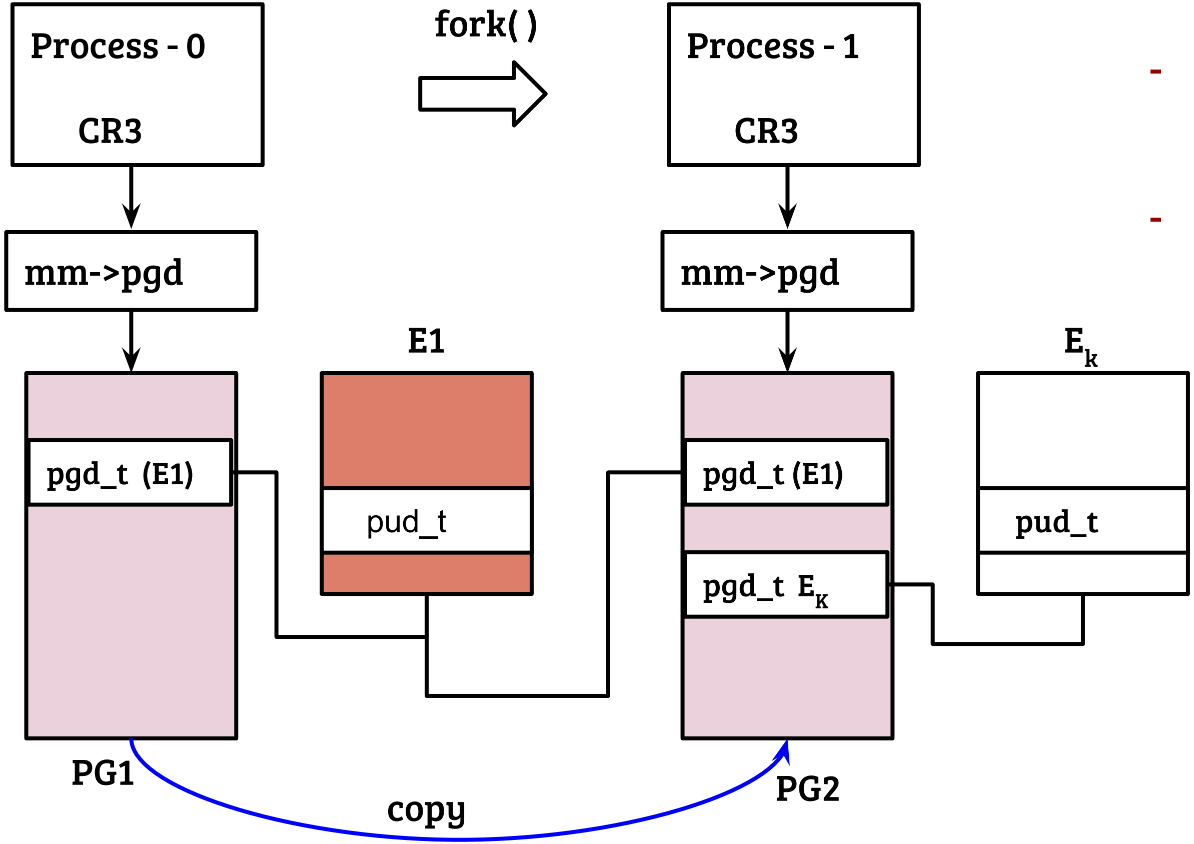
Solution: Every process owns its own **pgd** entries but inherits the kernel **pgd** entries from the parent :-)

Solution overview



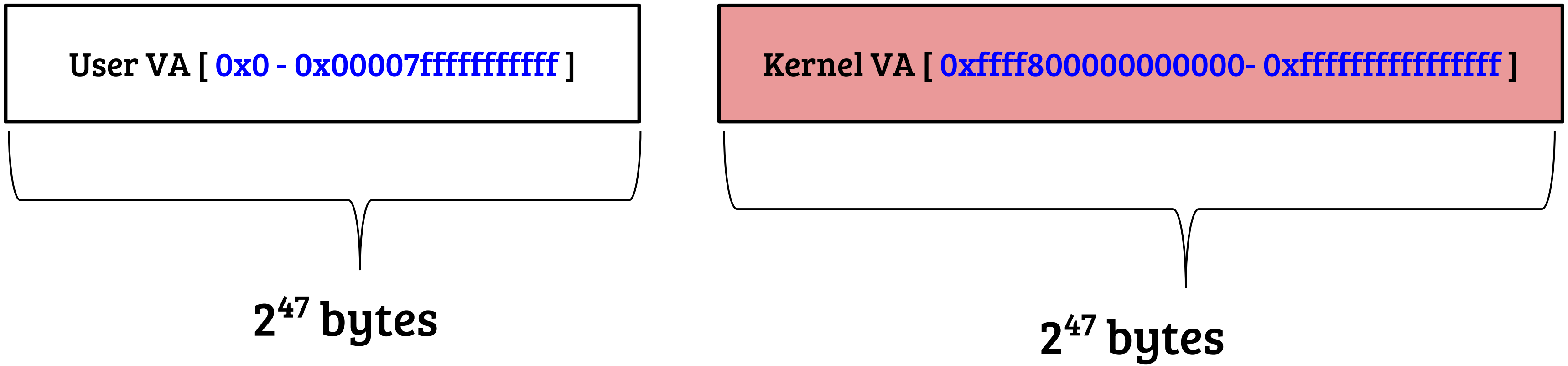
- One (or more) entries in PGD-level (level-4) reserved for kernel mapping
- How many?
- Depends on VA-range covered by one entry and the kernel VA size

Solution overview



- All updates to E1 are visible across all the processes
- So we are at peace! Not really.

Virtual memory layout (x86_64)



User VA [0x0 - 0x00007fffffffffff]

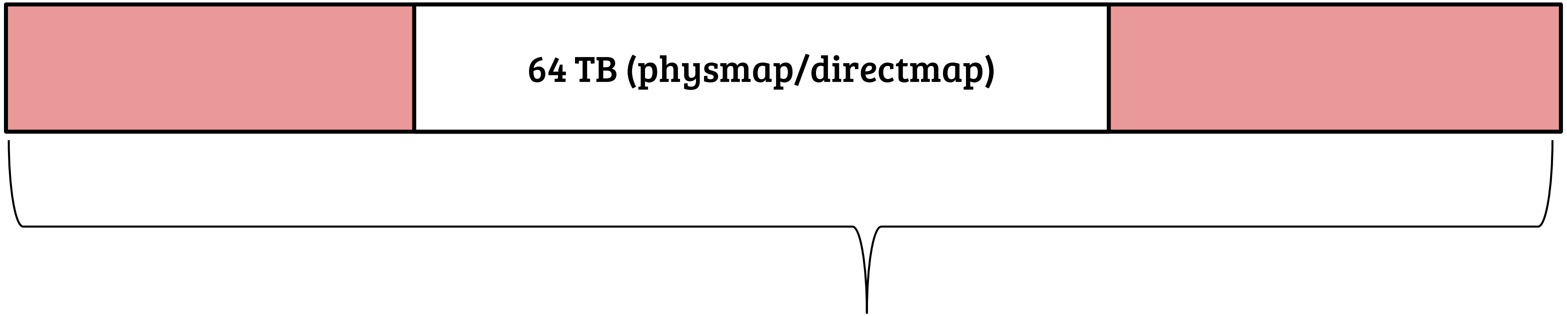
Kernel VA [0xffff800000000000- 0xffffffffffffffff]

2^{47} bytes

2^{47} bytes

- User virtual addresses use the LSB 47 bits
- Kernel virtual address does not start from 0x8000000000, but from 0xffff800000000000
- Why? What about page table translation (it is only 48 bits VA) ?

Virtual memory layout (x86_64)



Kernel VA [0xffff800000000000- 0xffffffffffffffff]

2^{47} bytes

- Direct map virtual address maps the whole physical memory
- Every PFN has a direct mapped kernel virtual address
- How kernel updates the page tables in PF handler? (answered!)