OPERATING SYSTEMS

PROJECT REPORT

Data Compartmentalization

CS18B042 RAMIREDDY PRAHLADHA REDDY
CS18B063 YAKKATI MAHESH
CONTRIBUTIONS 50-50

Objective:

The goal of this project is to add Compartment aware page tables in riscv-pk to achieve Data Compartmentalization.

Methodology:

- Extend the page table entry to contain compartment id in it
- Adding a system call that takes a compartment id as an argument and creates and maps a new page with compartment id and returns its virtual address.
- We tried two ways to implement it
    a) **M1:** Without taking the help of existing do_mmap and do_brk function
    b) **M2:** Using existing functions and extending them
- modifying the spike to identify the compartment mismatch

Coding Details:

- **Adding a new system call**
    - Adding a new system call (char*) my_alloc (int id)
    - Giving it a number 288 in syscall.h
    - Mapping number with function in syscall.c
    - Adding a scall.s file in application folder which helps in invokes system call when my_alloc is called in userspace
    - My_alloc will call do_my_alloc() (new function) which is added in mmap.c whole implementation is done in that function

---

- **Implementation of system call**
    - I tried implementing in two ways
    - **M1**
    - Creating a walk for the current.brk
    - Creating a new physical page
    - Mapping current. brk to the newly created page with all permissions(read write execute .....)
    - Using reserved 10 bits of the page table entry to store the compartment id
    - Incrementing the current. brk by page size
    - Changed pte_ppn so that compartment id is neglected when converting page table entry to physical address
    - **M2**
    - Using do_brk to extend virtual address and creating a new page
    - Modifying do_brk so that compartment id can be inserted into page table entry
    - Modifying do_mmap so that when it is called from do_brk it can assign compartment id to the page
    - Adding a new field compartment id in the metadata of the page (vrm_t) so that when the page fault happens it can store it in the page table entry(as do_mmap was implemented based on demand paging)
    - modifying the handle_page_fault function so that it puts compartment id into reserved 10 bits of page table entry when page fault happens ()
    - Changing every instance of do_brk and do_mmap so that they have an extra argument compartment id (setting it to default compartment id)
    - Changed pte_ppn so that compartment id is neglected when converting page table entry to physical address
- **Modifications in spike**
    - Modified the walk function in mmu.cc
    - Printing an error message and aborting when compartment id in page table entry and mcurrcap don't match (except when compartment id = 0|1 because it is used by shared libraries)
    - Modified the conversion between page table entry to physical address so that it neglects the compartment id bits

Evaluation and Results:

Test case	Output
One function in compartment 0	Creating accessing data in compartment 0: NO COMPARTMENT MISMATCH Creating accessing data in compartment 1: NO COMPARTMENT MISMATCH Creating accessing data in compartment 5: COMPARTMENT MISMATCH
One function in compartment 1	Creating accessing data in compartment 0: NO COMPARTMENT MISMATCH Creating accessing data in compartment 1: NO COMPARTMENT MISMATCH Creating accessing data in compartment 5: COMPARTMENT MISMATCH
One function in compartment 5	Creating accessing data in compartment 0: NO COMPARTMENT MISMATCH Creating accessing data in compartment 1: NO COMPARTMENT MISMATCH Creating accessing data in compartment 5: NO COMPARTMENT MISMATCH Creating accessing data in compartment 6: COMPARTMENT MISMATCH
Two functions main(compartment 5) Foo (compartment 6) Calling foo from main	(everything in foo) Creating accessing data in compartment 0: NO COMPARTMENT MISMATCH Creating accessing data in compartment 1: NO COMPARTMENT MISMATCH

	Creating accessing data in compartment 6: NO COMPARTMENT MISMATCH Creating accessing data in compartment 5: COMPARTMENT MISMATCH
Two functions main(compartment 6) Foo (compartment 1) Calling foo from main	(everything in foo) Creating accessing data in compartment 0: NO COMPARTMENT MISMATCH Creating accessing data in compartment 1: NO COMPARTMENT MISMATCH Creating accessing data in compartment 6: NO COMPARTMENT MISMATCH Creating accessing data in compartment 5: COMPARTMENT MISMATCH
Applying sorting algorithm on page	sorted

Advantages and disadvantages of two methods:

- In M1 there is no intermediate page to store the metadata this is suitable for the current project
- But if we want to change system call such that it may allocate multiple pages method 2 will be efficient
- Both the methods will work with default compartment id as 0 but if we want to change that to different value M2 will be more useful

Future work:

- **Multi compartmental objects**
    - In our implementation, we added a system call that takes a compartment id and maps to it but if we want to create a page that maps to multiple compartments that will not work

    - We can archive this by mapping same data page to different virtual addresses and putting different compartment ids in page table entries
    - This can be archived by maintaining a list of all addresses that are mapped to the same data in the object and trying each address until we get an address that doesn't throw any error
    - By this, we can use the same data in different compartments that it is mapped to
- **Modifiable default_compartmentid**
    - In our implementation default compartment id was set to zero(as while booting every register is set to zero and the first 10 bits are not touched)
    - If we are able to change default compartment if by just changing a macro it will be useful to people who work in a different direction to synchronise with this work
- **Allocating in lesser granularity**
    - In our implementation, we are creating a page whenever we are invoking my_alloc but we can extend it in such a way that its functionality is similar to malloc
    - We can archive this by allocating a page only if the past allocated page is full or there is no page corresponding to the compartment id
    - We need to keep track of the amount of remaining space in each page and try to store data in the old page if it can fit into it
    - This way we can use memory properly
- **Semi sharable data**
    - In the present implementation, if we want to share data in a compartment with another compartment we need to copy data into compartment 0 and share it
    - Since we need only 8 bits to store compartment id we can use a bit in remaining two reserved bits to mark data as semi sharable
    - If that bit is set to 1 then it is readable in all compartments and editable only in the compartment that it belongs to
    - This way we can share data as a read-only one with other compartments