Operating System Lab Project 5 Report

Reza Abdoli
Mohammad Reza Alavi
Farbod Azimmohseni
Commit Id: b110194752ed99fe0e14408cec5e8cd803c259fd

January 14, 2024

Questions

1. **VMA** keeps track of memory mapping in linux. Linux has this as a structure that describes a single memory area over a contiguous interval in a given address space. Each area have set of operations and permissions. Noncontiguous memories are linked lists of this struct. Vma needs to keep the start and end address.

2. In hierarchical page tables, you do not need to have all tables in the table at once. Instead If necessary, you load tables you need. Although it increases memory access overhead it uses less memory because you can separate page tables to each exactly fit in one page. Because it is type of hashing it also reduces search time.

3. Address bits in all tables except for the last layer keep the address of next layer page table. The last layer keeps the physical frame number. Flag bits keep some control information such as who can access this page or whether if you can read or write on memory. Dirty flag indicates if page has changed since modified. Also we have a access bit to check if we have used this entry before or not.

4. **P**hysical.Kalloc allocates a new page and returns a pointer to the block of memory allocated.

5. it maps a virtual memory to a physical memory. **mempages** is used in functions like **setupkvm** which allocates and maps kernel part of process. All functions that set memory of process use **mempages** to create the mapping.

6. **walkpgdir** gets a page directory and an address and returns its pagetable. It somewhat does the work of **MMU**.

7. **allocuvm** takes a page directory and start address of virtual memory and number of pages you want to allocate. It checks whether this memory range is valid. Then using kalloc it makes physical pages and add these entries to page table.

8. It starts by creating a new address space. It firsts load program to memory usuing **allocvm** and **loadvm** and then allocates two pages one of them inaccessible and the other one for user stack.

Adding shared memory

```
Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta
t 58
init: starting sh
$ sharedmem_test
refs : 2Values in child: 1
refs : 2Values in child: 2
refs : 2Values in child: 3
refs : 2Values in child: 4
refs : 2Values in child: 5
refs : 2Values in child: 6
refs : 2Values in child: 7
refs : 2Values in child: 8
refs : 2Values in child: 9
refs : 2Values in child: 10
Final Value: 10
$ _
```

Figure 1: user test program

1. In **open** system call we check to see if some process has been shared on this memory, if yes we will expand the virtual memory of process then map it to the physical address using **mappages**. If no one has already shared this memory we allocate a frame using **kalloc** and keep the pointer to it in our table. Other parts of process is like the first case.

2. For **close** system call we need to free the virtual address of the process and unmap it. To do so we check all virtual addresses we have using **walkpgdir** and find its page table then we set it to zero. This operation will free the virtual memory. If no one have reference to this memory we free it using **kfree**.