RV College of Engineering®

Experiential Learning Phase -I

COURSE	Operating Systems(CS235AI)
TITLE	Design and Implementation of Copy on Write-Fork for Improved Performance

Go, change the world®

RV College of Engineering®

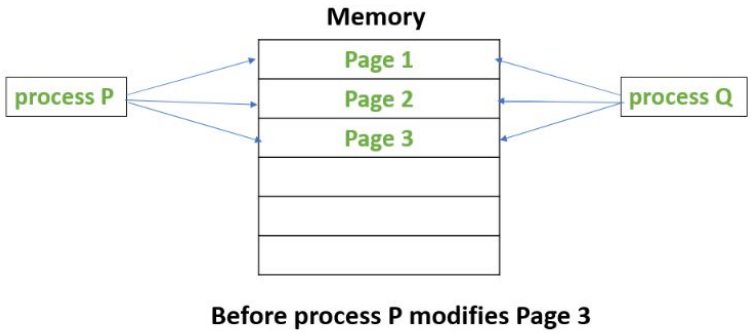
TEAM INTRODUCTION

Roll No.	USN	Name	Email Id
1.	1RV22CS210	Sundarakrishnan N	sundarakn.cs22@rvce.edu.in
2.	1RV22CS217	Tanmay Umesh	tanmayumesh.cs22@rvce.edu.in
3.	1RV22CS218	Tarun Bhupathi	tarunbhupathi.cs22@rvce.edu.in

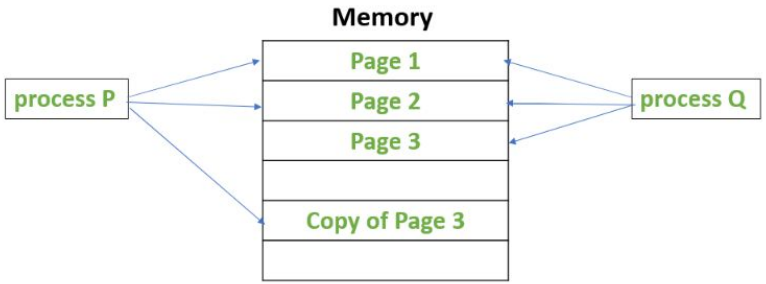
Copy on Write

- Copy on Write (COW) delays duplicating resources, such as memory or data structures, until changes are imminent.Multiple processes or users can efficiently share resources.

- COW only creates new copies when modifications are necessary, minimizing the overhead of memory and resource management.This approach allows for optimized resource utilization.



Before process P modifies Page 3



After process P modifies Page 3

Tools/API used

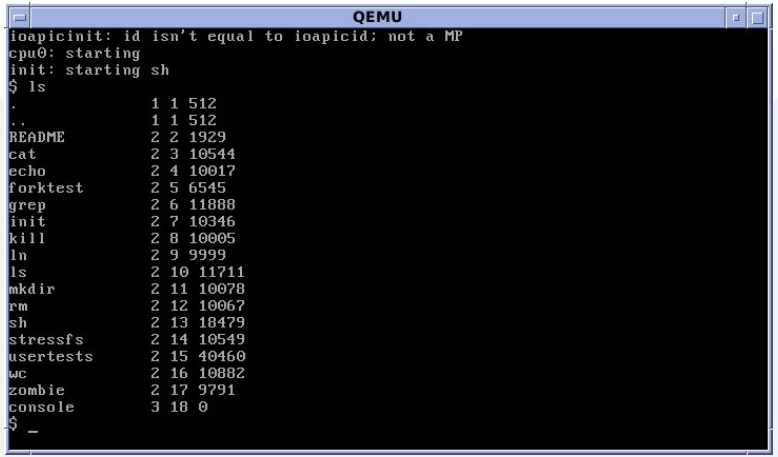
- xv6 Operating System
- System Calls (fork(), exec())
- Memory Management (Page Allocation,Deallocation and Page Tables)
- Concurrency Control (Locks, Semaphores)
- Testing Framework (Tools and Scripts)

xv6

- xv6, a Unix-like operating system kernel, serves as a foundational platform for understanding operating system principles.

- Its minimalist design provides a basic framework, enabling users to explore core concepts without the complexity of a full-fledged operating system

- Due to its educational design, xv6 lacks certain optimization features. While it includes basic file systems and memory management, functionalities such as Copy-on-Write and multi-user support are missing



RV College of Engineering®

Literature Survey

Go, change the world®

Title	Author	Inference
Understanding xv6	Marten van Steen	• It provides a practical and hands-on approach to learning about operating system concepts. • Relevance of COW Fork Discussion: Since the paper is expected to discuss COW fork, it would be important for the author to provide clear explanations of how this technique is implemented in xv6 and its significance in terms of performance and memory management. • The paper is structured to provide a comprehensive overview of xv6's kernel structures, system calls, internal mechanisms, and perhaps focuses on the Copy-On-Write (COW) fork technique, which is a significant feature of xv6.
Effect of copy-on-write memory management on the response time of UNIX fork operations	Jonathan M. Smith, Gerald Q. Maguire Jr.	• The study offers a focused analysis on 'copy-on-write' in UNIX fork(), emphasizing memory and execution time aspects. • Execution time dependence on memory copied during fork() is a key finding, highlighting a critical factor in 'copy-on-write' performance. • 'Copy-on-write' proves practically efficient, reducing real-time requirements and demonstrating its potential for enhancing overall system performance • 'Copy-on-write' proves effective, reducing real-time requirements for fork() operations, showcasing practical advantages for system efficiency.

Problem Statement

- Our system currently suffers from high memory usage and slow process creation times, especially when dealing with frequent forks and large memory footprints. This impacts overall performance and scalability.

- A Copy On Write Technique allows processes to initially share memory pages, only creating copies when modifications occur. This can significantly reduce memory usage and improve process creation speed.

RV College of Engineering®

Methodology

Go, change the world®

MAP PARENT'S PAGES

Update uvmcopy() to map the parent's physical pages into the child instead of allocating new ones. Clear the writable bit (PTE_W) in the PTEs for both child and parent on shared pages.

HANDLE COW PAGE FAULTS

Recognize page faults in usertrap(). On a write page fault for a COW page, allocate a new page, copy the content, and update the PTE with PTE_W set. Maintain read-only status for read only pages

IMPLEMENT REFERENCE COUNTING

Set reference count to one when allocating with kalloc(). Increment for shared pages during fork(), and decrement during page table drops. Modify kfree() to free only when the reference count is zero.

UPDATE COPYOUT()

Modify copyout() to handle COW pages similarly to page faults. Allocate a new page, copy the content, and update the PTE with PTE_W if the original page was writable.

RECORD COW MAPPING IN PTE

Use RISC-V PTE's reserved bits to record COW mappings in each PTE. Manage COW mappings during page faults and copying out data.

Implementation

1. **Shared Page Identification and Mapping:**

- During fork, instead of copying memory pages, the child inherits the parent's page table entries for user memory.
- Modify **uvmcopy()** in **vm.c** to iterate through the page tables.
- For each page, check writability using Page Table Entry permissions.If writable, create a copy during write attempts (handled later).
- If read-only (potential shared page), clear write permission and map it into the child's address space with user, execute, and read permissions.

```
10      if((pte = walk(old, 1, 0)) == 0)
12      if((*pte & PTE_V) == 0)
13        panic("uvmcopy: page not present");
14
15      *pte &= ~PTE_W;
16
17
18      *pte |= PTE_COW;
19
20
21
22      pa = PTE2PA(*pte);
23
24
25      acquire(&reflock);
26      reference_counter[pa/PGSIZE]+=1;
27      release(&reflock);
28
29      flags = PTE_FLAGS(*pte);
30
```

Implementation

2. Handle Copy-on-Write Page Faults

- Modification of page fault handler in **trap.c** to manage write attempts on shared pages.
- When a write fault occurs (due to a read-only page), the handler identifies the faulting page.
- The handler allocates a new physical page, copies the content from the shared parent page, and updates the child's page table entry to point to the new page.
- The reference count for the original shared page might need to be adjusted.

```
{
  else if(r_scause()== 15){
    if ((*my_pte&PTE_COW)){
      uint flags;
      char *mem;


      flags = PTE_FLAGS(*my_pte);
      flags |= PTE_W;
      flags &= ~PTE_COW;

      if((mem = kalloc()) == 0){

        p->killed=1;
        exit(-1);
      }


      memmove(mem, (char*)pa, PGSIZE);


      *my_pte = PA2PTE(mem) | flags;

      kfree((char*)pa);
      if((*my_pte&PTE_COW)&&reference_counter[pa/PGSIZE]==1)
      {

      *my_pte &= ~PTE_COW;
      *my_pte |= PTE_W;
      }
      p->trapframe->epc = r_sepc();
```

Implementation

3. Page Table Updates:

- Modify the code that modifies user memory pages to check for write permission before the write.
- If write permission is denied (read-only page), trigger a page fault.
- After a page fault is handled and a copy is created, update the child's page table entry to mark the page writable.

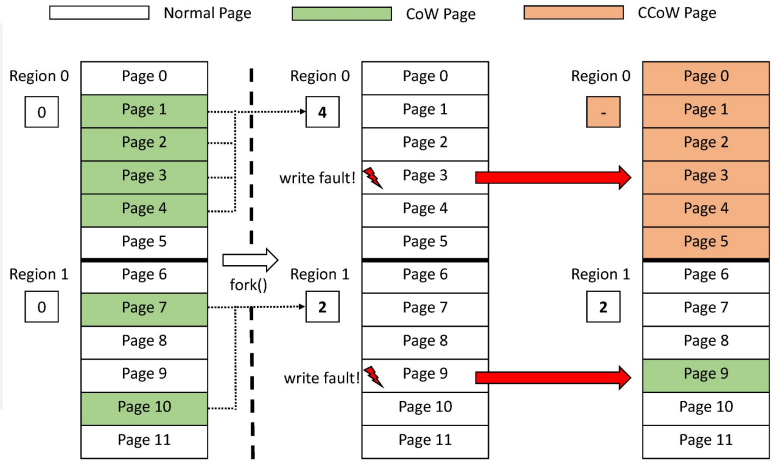
CoW Test

- A "cowtest" within operating systems, such as xv6, serves as a validation method to ensure the accurate implementation and functionality of Copy-On-Write (CoW) memory management.
- During the cowtest, memory pages are initially allocated to a parent process, and subsequently, a child process is created via forking.
- Both processes are expected to share the same memory pages. The test involves modifying the memory contents in one process while verifying that the changes do not affect the other.

Advantages

- COW simplifies page table management, particularly in scenarios involving forks and clones. The sharing of read-only pages avoids the need for immediate page table duplication, leading to more efficient memory utilization.

- Reduced memory usage where data modified by a process is copied, saving memory. Faster process creation where there is no initial copying, reducing creation time. COW simplifies memory management in these scenarios.



Applications of Copy on Write

- **Database cloning:**
  COW allows for creating database clones efficiently. Only data differences are copied, reducing cloning time and resource usage.
- **Process Forking:**
  When a process creates a child process (e.g., using fork in Unix-like systems), both initially share the same memory pages using COW. This saves memory and reduces process creation time, as copying occurs only when necessary (on write).
- **Live Migration:**
  By using COW snapshots, virtual machines can be migrated between hosts with minimal downtime, maintaining application state without significant memory copying.
- **Snapshots and Versioning:**
  File systems can leverage COW to create snapshots of directories or files. This allows for reverting to previous versions, recovering from accidental changes, or managing file versions efficiently.

References

1] Jonathan M. Smith and John Ioannidis, ‘‘Implementing remote fork() with checkpoint/restart,’’ IEEE Technical Committee on Operating Systems Newsletter,(February, 1989)

[2] Effects of copy-on-write memory management on the response time of UNIX fork operations Jonathan M. Smith,Gerald Q. Maguire, Jr.Computer Science Department, Columbia University, New York, NY 10027

[3]Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, "Operating Systems: Three Easy Pieces" by includes a section on xv6.

[4]“CCoW: Optimizing Copy-on-Write Considering the Spatial Locality in Workloads”, Minjong Ha,Sang-Hoon Kim,23 December 2021,

[5] "Copy-on-Write" by Ousterhout, et al. and "Copy-on-Write: Principles and Performance" by D. R. Engler and M. F. Kaashoek.