OS-II: CS3523

Programming Assignment 5: xv6 Paging

Name: Aparna. S. Kaushik

Roll No.: CO22BTECH11003

Understanding System Calls and Learnings from Assignment

Introduction:

System calls are an essential mechanism in operating systems that allow user-level processes to request services from the kernel. These services include tasks such as reading from files, writing to the console, managing memory, and more. In this report, we will discuss our understanding of how system calls work and the key learnings gained from completing the assignment involving the implementation of new system calls in the xv6 kernel.

Understanding System Calls:

A system call is a controlled entry point into the kernel, typically exposed to user-space programs via libraries like libc in Unix-like operating systems. When a user-space program makes a system call, it triggers a context switch from user mode to kernel mode, allowing the kernel to perform privileged operations on behalf of the user process. This context switch involves several steps:

1. **Trap Entry:** The user-space program invokes a system call by executing a special trap instruction (e.g., syscall in x86), causing a trap into the kernel.
2. **Kernel Mode Transition:** Upon receiving the trap, the processor switches to kernel mode, granting access to privileged instructions and data.
3. **System Call Handler:** The kernel identifies the requested service based on the system call number provided by the user-space program. It then dispatches the appropriate system call handler function.
4. **Execution of System Call:** The system call handler executes the requested operation, interacting with kernel data structures and hardware as necessary.
5. **Return to User Space:** After completing the system call, the kernel switches the processor back to user mode, returning control to the user-space program.

Learnings from Assignment:

Completing the assignment involving the implementation of new system calls in the xv6 kernel provided valuable insights into several key concepts:

1. **Kernel Programming:** The assignment required modifying the kernel code to implement new system calls, enhancing our understanding of kernel development and system-level programming techniques.

2. **Memory Management:** Implementing system calls such as pgtPrint() and pgaccess() involved interacting with the page tables and understanding the virtual memory subsystem of the operating system. This deepened our knowledge of memory management concepts such as page tables, page faults, and demand paging.

3. **ELF File Format:** Understanding the ELF file format was essential for modifying the exec() function to support demand paging. We learned about program headers, memory layout, and dynamic memory allocation in executable files.
4. **Trap Handling:** Modifying the trap handling mechanism in the kernel to support demand paging enabled us to grasp the intricacies of handling page faults and implementing demand-driven memory allocation strategies.
5. **Debugging Techniques:** Debugging kernel-level code introduced us to various debugging techniques specific to operating system development, such as using print statements, kernel panics, and inspecting page tables.

Conclusion:

In conclusion, the assignment provided a hands-on learning experience in kernel programming, system call implementation, and memory management concepts. By understanding how system calls work and delving into the internals of the xv6 kernel, we gained valuable insights into operating system design and implementation, paving the way for deeper exploration and understanding of complex systems-level concepts.

Task-1: Printing the page table entries

Modify the xv6 kernel to implement a new system call named pgtPrint() which will print the page table entries for the current process. Since the total number of page table entries can be very large, the system call should print the entries only if it is valid and the page is allowed access in user mode.

Observations:

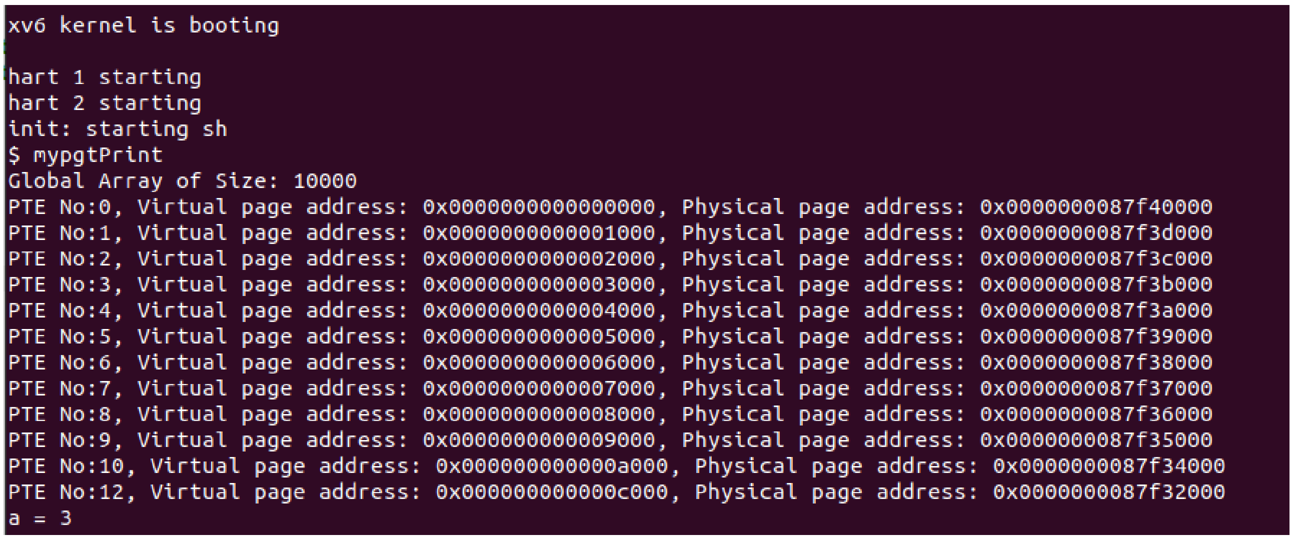
- When a large size local array (arrLocal[N]) is declared, only a few entries are printed during the page table traversal.
- Conversely, when a large size global array (arrGlobal[N]) is declared, more entries are printed during the traversal.
- Each execution of the program results in different physical page addresses assigned to the virtual pages of the array.
- Despite the changes in physical page addresses, the number of valid entries remains consistent for subsequent executions of the program.

Reasoning:

- The differences in the number of printed entries between the local and global arrays can be attributed to differences in scope, access patterns, compiler optimizations, and memory allocation requirements. While local arrays are subject to more aggressive optimizations due to their limited scope, global arrays may require more memory allocation and have broader access patterns, leading to different behaviors during page table traversal.
- The change in the number of valid PTEs is expected because declaring a large size global array increases the memory footprint of the process, potentially requiring additional pages to be mapped in the virtual address space.

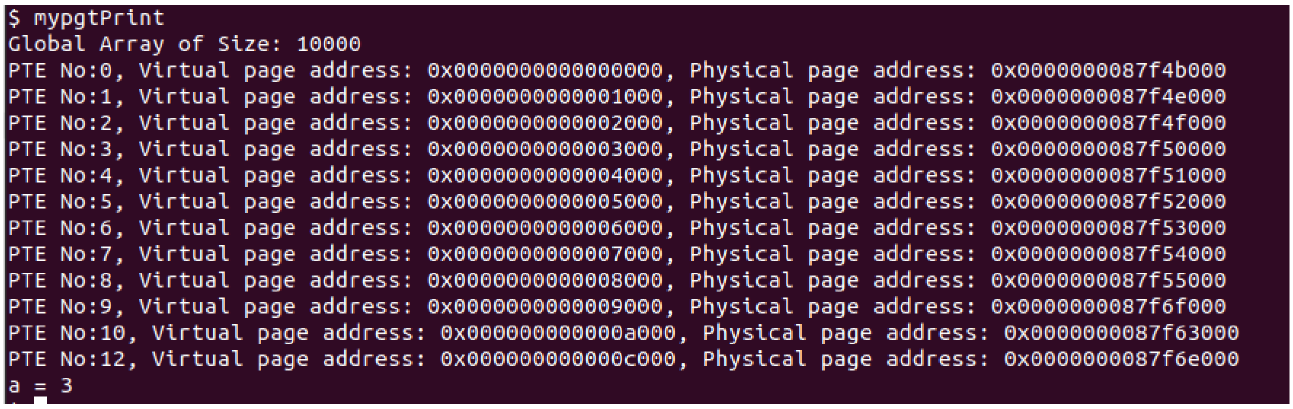
- The variation in physical page addresses across executions is due to the demand-driven nature of memory allocation in the xv6 kernel. As demand paging is employed, physical pages are allocated dynamically based on the access patterns and memory availability during each execution.
- The consistency in the number of valid entries indicates that the kernel efficiently manages memory allocation and mapping for the global array, ensuring that the required pages are available and mapped appropriately for subsequent executions.

Outputs:

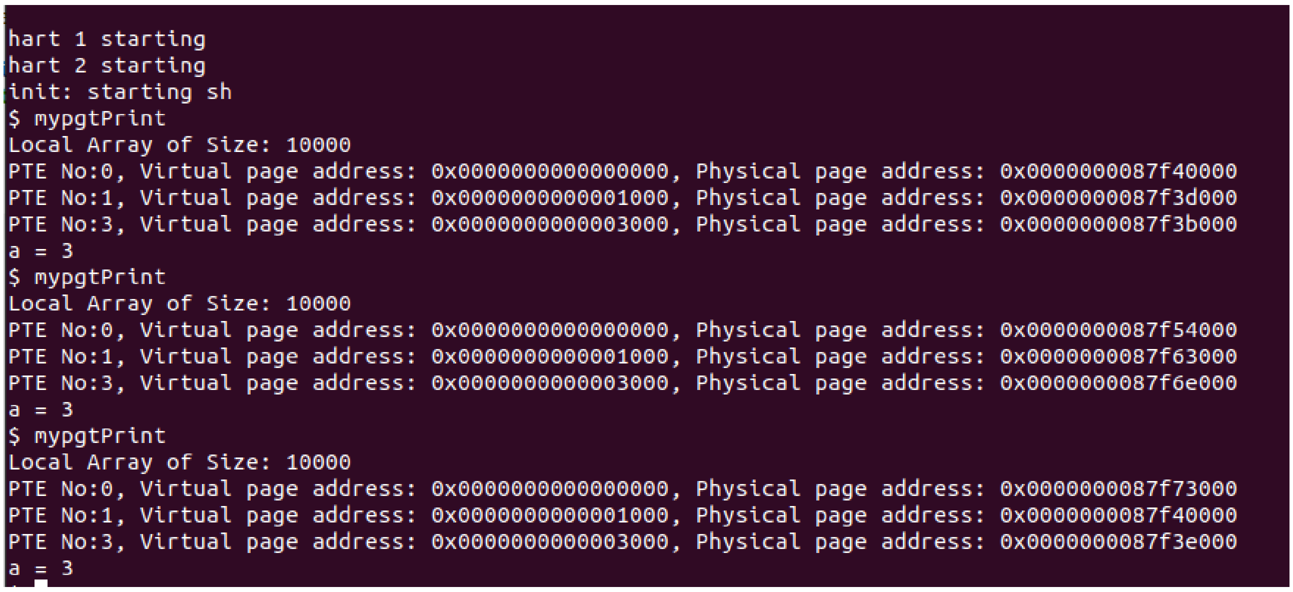


```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ mypgtPrint
Global Array of Size: 10000
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f40000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f3d000
PTE No:2, Virtual page address: 0x0000000000002000, Physical page address: 0x0000000087f3c000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f3b000
PTE No:4, Virtual page address: 0x0000000000004000, Physical page address: 0x0000000087f3a000
PTE No:5, Virtual page address: 0x0000000000005000, Physical page address: 0x0000000087f39000
PTE No:6, Virtual page address: 0x0000000000006000, Physical page address: 0x0000000087f38000
PTE No:7, Virtual page address: 0x0000000000007000, Physical page address: 0x0000000087f37000
PTE No:8, Virtual page address: 0x0000000000008000, Physical page address: 0x0000000087f36000
PTE No:9, Virtual page address: 0x0000000000009000, Physical page address: 0x0000000087f35000
PTE No:10, Virtual page address: 0x000000000000a000, Physical page address: 0x0000000087f34000
PTE No:12, Virtual page address: 0x000000000000c000, Physical page address: 0x0000000087f32000
a = 3
```



```
$ mypgtPrint
Global Array of Size: 10000
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f4b000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f4e000
PTE No:2, Virtual page address: 0x0000000000002000, Physical page address: 0x0000000087f4f000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f50000
PTE No:4, Virtual page address: 0x0000000000004000, Physical page address: 0x0000000087f51000
PTE No:5, Virtual page address: 0x0000000000005000, Physical page address: 0x0000000087f52000
PTE No:6, Virtual page address: 0x0000000000006000, Physical page address: 0x0000000087f53000
PTE No:7, Virtual page address: 0x0000000000007000, Physical page address: 0x0000000087f54000
PTE No:8, Virtual page address: 0x0000000000008000, Physical page address: 0x0000000087f55000
PTE No:9, Virtual page address: 0x0000000000009000, Physical page address: 0x0000000087f6f000
PTE No:10, Virtual page address: 0x000000000000a000, Physical page address: 0x0000000087f63000
PTE No:12, Virtual page address: 0x000000000000c000, Physical page address: 0x0000000087f6e000
a = 3
```



```
hart 1 starting
hart 2 starting
init: starting sh
$ mypgtPrint
Local Array of Size: 10000
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f40000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f3d000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f3b000
a = 3
$ mypgtPrint
Local Array of Size: 10000
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f54000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f63000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f6e000
a = 3
$ mypgtPrint
Local Array of Size: 10000
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f73000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f40000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f3e000
a = 3
```

Task-2: Implement demand paging

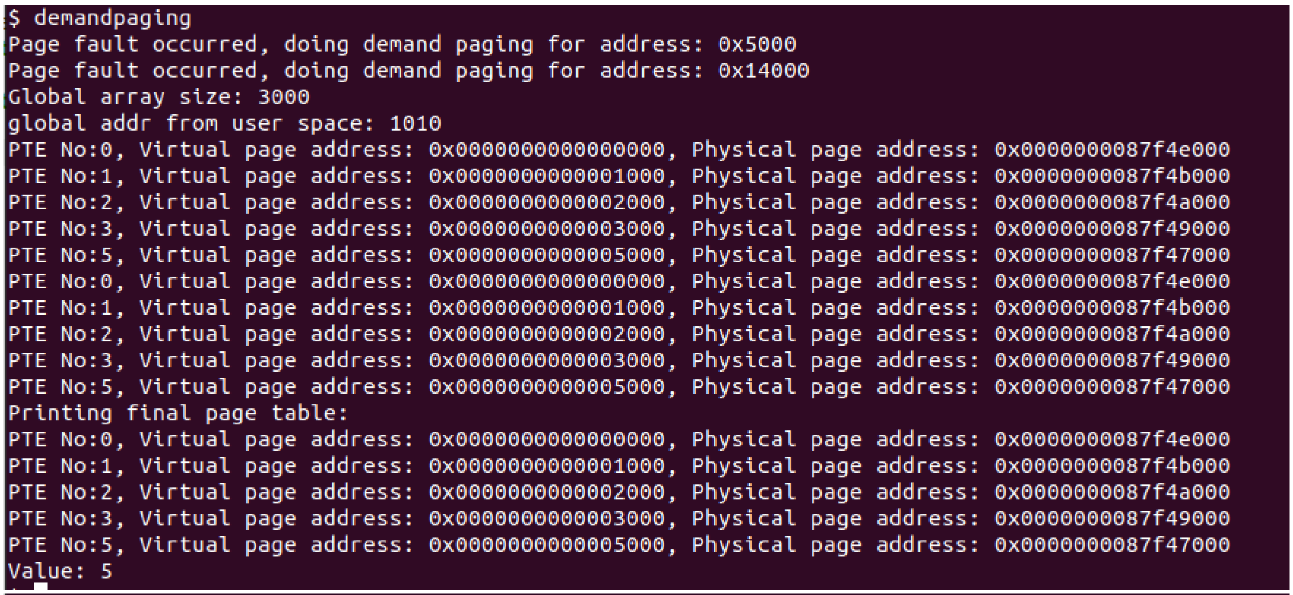
Observations:

- Both executions encounter page faults at addresses `0x5000` and `0x14000`, triggering demand paging.
- Despite similar execution conditions, the number of page table entries differs between the two executions.
- In the first execution (`Global array size: 5000`), there are 7 page table entries printed.
- In the second execution (`Global array size: 3000`), there are 5 page table entries printed.

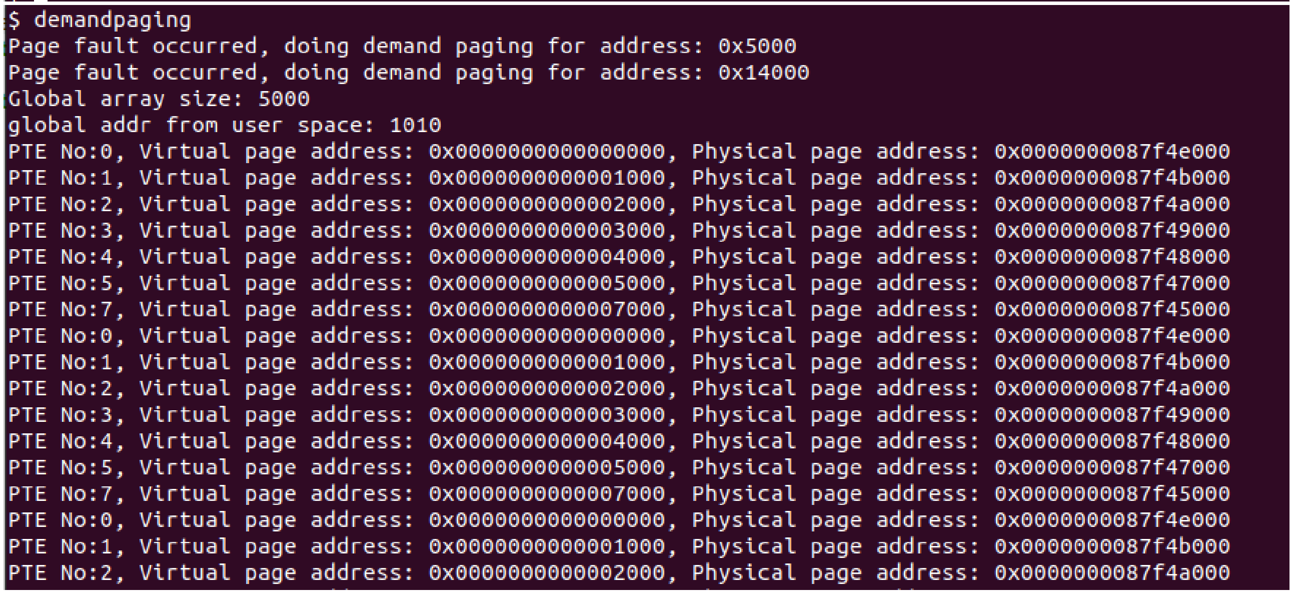
Reasoning:

- The page faults occurring at addresses `0x5000` and `0x14000` indicate attempts to access memory that is not currently mapped to physical pages. Demand paging allocates physical memory for these addresses on-the-fly to fulfill the access requests.
- The discrepancy in the number of page table entries likely stems from differences in memory allocation patterns between the two executions.
- The larger global array size in the first execution may have led to the allocation of more physical pages, resulting in additional page table entries.
- Conversely, the smaller global array size in the second execution may have resulted in fewer physical pages being allocated, hence fewer page table entries.

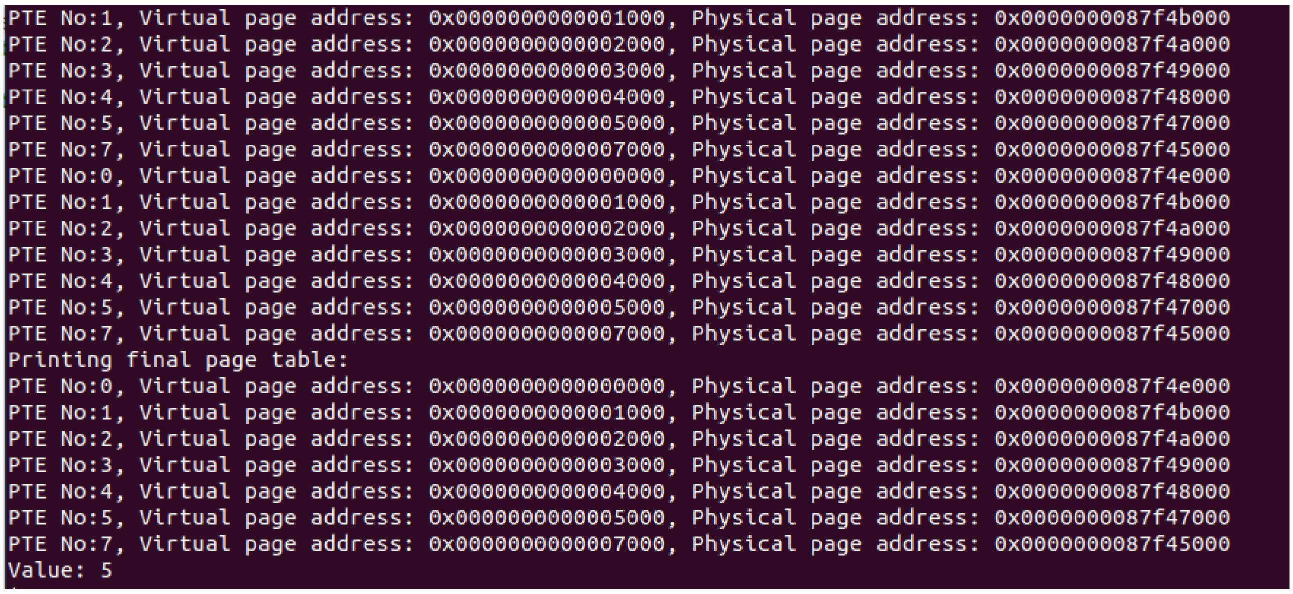
Outputs:



```
$ demandpaging
Page fault occurred, doing demand paging for address: 0x5000
Page fault occurred, doing demand paging for address: 0x14000
Global array size: 3000
global addr from user space: 1010
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f4e000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f4b000
PTE No:2, Virtual page address: 0x0000000000002000, Physical page address: 0x0000000087f4a000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f49000
PTE No:5, Virtual page address: 0x0000000000005000, Physical page address: 0x0000000087f47000
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f4e000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f4b000
PTE No:2, Virtual page address: 0x0000000000002000, Physical page address: 0x0000000087f4a000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f49000
PTE No:5, Virtual page address: 0x0000000000005000, Physical page address: 0x0000000087f47000
Printing final page table:
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f4e000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f4b000
PTE No:2, Virtual page address: 0x0000000000002000, Physical page address: 0x0000000087f4a000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f49000
PTE No:5, Virtual page address: 0x0000000000005000, Physical page address: 0x0000000087f47000
Value: 5
```



```
$ demandpaging
Page fault occurred, doing demand paging for address: 0x5000
Page fault occurred, doing demand paging for address: 0x14000
Global array size: 5000
global addr from user space: 1010
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f4e000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f4b000
PTE No:2, Virtual page address: 0x0000000000002000, Physical page address: 0x0000000087f4a000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f49000
PTE No:4, Virtual page address: 0x0000000000004000, Physical page address: 0x0000000087f48000
PTE No:5, Virtual page address: 0x0000000000005000, Physical page address: 0x0000000087f47000
PTE No:7, Virtual page address: 0x0000000000007000, Physical page address: 0x0000000087f45000
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f4e000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f4b000
PTE No:2, Virtual page address: 0x0000000000002000, Physical page address: 0x0000000087f4a000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f49000
PTE No:4, Virtual page address: 0x0000000000004000, Physical page address: 0x0000000087f48000
PTE No:5, Virtual page address: 0x0000000000005000, Physical page address: 0x0000000087f47000
PTE No:7, Virtual page address: 0x0000000000007000, Physical page address: 0x0000000087f45000
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f4e000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f4b000
PTE No:2, Virtual page address: 0x0000000000002000, Physical page address: 0x0000000087f4a000
```



```
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f4b000
PTE No:2, Virtual page address: 0x0000000000002000, Physical page address: 0x0000000087f4a000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f49000
PTE No:4, Virtual page address: 0x0000000000004000, Physical page address: 0x0000000087f48000
PTE No:5, Virtual page address: 0x0000000000005000, Physical page address: 0x0000000087f47000
PTE No:7, Virtual page address: 0x0000000000007000, Physical page address: 0x0000000087f45000
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f4e000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f4b000
PTE No:2, Virtual page address: 0x0000000000002000, Physical page address: 0x0000000087f4a000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f49000
PTE No:4, Virtual page address: 0x0000000000004000, Physical page address: 0x0000000087f48000
PTE No:5, Virtual page address: 0x0000000000005000, Physical page address: 0x0000000087f47000
PTE No:7, Virtual page address: 0x0000000000007000, Physical page address: 0x0000000087f45000
Printing final page table:
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f4e000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f4b000
PTE No:2, Virtual page address: 0x0000000000002000, Physical page address: 0x0000000087f4a000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f49000
PTE No:4, Virtual page address: 0x0000000000004000, Physical page address: 0x0000000087f48000
PTE No:5, Virtual page address: 0x0000000000005000, Physical page address: 0x0000000087f47000
PTE No:7, Virtual page address: 0x0000000000007000, Physical page address: 0x0000000087f45000
Value: 5
```

Task-3: Implement logic to detect which pages have been accessed and/or dirty

Observations:

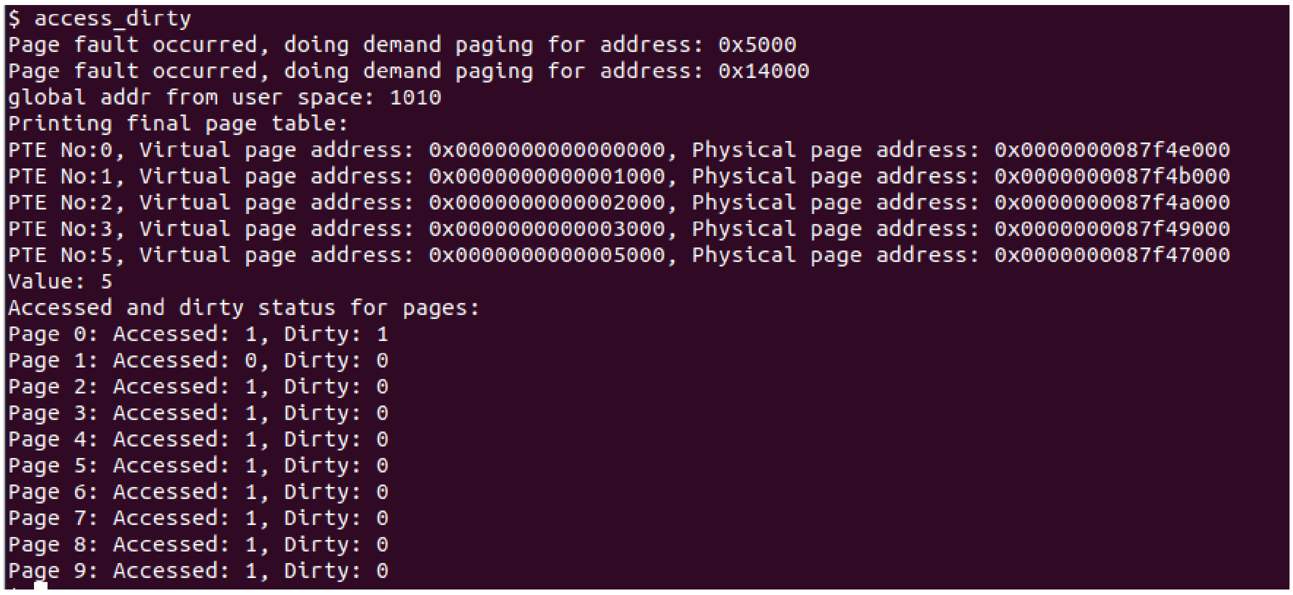
- The program encounters page faults at addresses 0x5000 and 0x14000, triggering demand paging.
- After demand paging, the final page table is printed, displaying the virtual and physical page addresses along with their respective page table entry numbers.
- The accessed and dirty status for each page is reported.

Reasoning:

- The accessed and dirty status for each page reflects their usage and modification during program execution.

- The accessed bit (A) in the page table entry (PTE) is set when the corresponding page is accessed, indicating recent usage.
- The dirty bit (D) in the PTE is set when the corresponding page is modified, indicating that the page content has been altered since it was loaded into memory.
- In the provided output, Page 0 is accessed and dirty, while other pages are accessed but not dirty.
- Page 1 is neither accessed nor modified.
- This indicates that Page 0 has been both accessed and modified during program execution, whereas the other pages have only been accessed but not modified.

Output:



```
$ access_dirty
Page fault occurred, doing demand paging for address: 0x5000
Page fault occurred, doing demand paging for address: 0x14000
global addr from user space: 1010
Printing final page table:
PTE No:0, Virtual page address: 0x0000000000000000, Physical page address: 0x0000000087f4e000
PTE No:1, Virtual page address: 0x0000000000001000, Physical page address: 0x0000000087f4b000
PTE No:2, Virtual page address: 0x0000000000002000, Physical page address: 0x0000000087f4a000
PTE No:3, Virtual page address: 0x0000000000003000, Physical page address: 0x0000000087f49000
PTE No:5, Virtual page address: 0x0000000000005000, Physical page address: 0x0000000087f47000
Value: 5
Accessed and dirty status for pages:
Page 0: Accessed: 1, Dirty: 1
Page 1: Accessed: 0, Dirty: 0
Page 2: Accessed: 1, Dirty: 0
Page 3: Accessed: 1, Dirty: 0
Page 4: Accessed: 1, Dirty: 0
Page 5: Accessed: 1, Dirty: 0
Page 6: Accessed: 1, Dirty: 0
Page 7: Accessed: 1, Dirty: 0
Page 8: Accessed: 1, Dirty: 0
Page 9: Accessed: 1, Dirty: 0
```