School of Computer and Information Science

《Operating-System-Concepts(10th)》

Project Report

Student information：

Name：张乐之 ID: 222022321102120
Major/Grade: 2022 计科中外 04 Time: 2024-11-30

Experiment information：

Topic:

Programming Projects in Chapter 10

Requirements:

1. Master the basic concepts and design ideas of memory management system through literature research and theoretical analysis. Experiment is carried out to verify the complex engineering problems of virtual memory management which combines the computer hardware and software.
2. Have the ability to select the technical route and development environment, and design the system implementation scheme. Master the system kernel-level virtual memory management design and implementation.

Procedure：

1. Break down the logical and physical address structure.
2. Specify the sizes and components: page table, TLB, main memory, and backing store.
3. Create structures for the page table, TLB, and physical memory.
4. Define the TLB search, TLB replacement policy (FIFO/LRU), and page fault handling process.
5. Extract the page number and offset from logical addresses.
6. Use the TLB for fast lookup, and consult the page table on misses.
7. Handle page faults by loading pages from the backing store.
8. Manage free frames in memory and load pages on demand from BACKING_STORE.bin.
9. Update the page table and TLB when new pages are loaded.
10. Measure TLB hit rate and page fault rate during address translation.
11. Validate outputs against the provided correct.txt file.
12. Implement page replacement policies (FIFO/LRU) for systems with limited physical memory.
13. Compile the program to work on a Linux system.
14. Test it using addresses.txt as input and analyze the performance.

Results：

（Code and Figures）

```
#include <stdio.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>

#define TLB_SIZE 16
#define PAGES 256
#define PAGE_MASK 255

#define PAGE_SIZE 256
#define OFFSET_BITS 8
#define OFFSET_MASK 255

#define MEMORY_SIZE PAGES *PAGE_SIZE

// Max number of characters per line of input file to read.
#define BUFFER_SIZE 10

struct tlbentry
{
      unsigned char logical;
      unsigned char physical;
};

// TLB is kept track of as a circular array, with the oldest element being overwritten
once the TLB is full.
struct tlbentry tlb[TLB_SIZE];
// number of inserts into TLB that have been completed. Use as tlbindex % TLB_SIZE for
the index of the next TLB line to use.
int tlbindex = 0;

// pagetable[logical_page] is the physical page number for logical page. Value is -1 if
that logical page isn't yet in the table.
int pagetable[PAGES];

signed char main_memory[MEMORY_SIZE];

// Pointer to memory mapped backing file
signed char *backing;

int max(int a, int b)
```

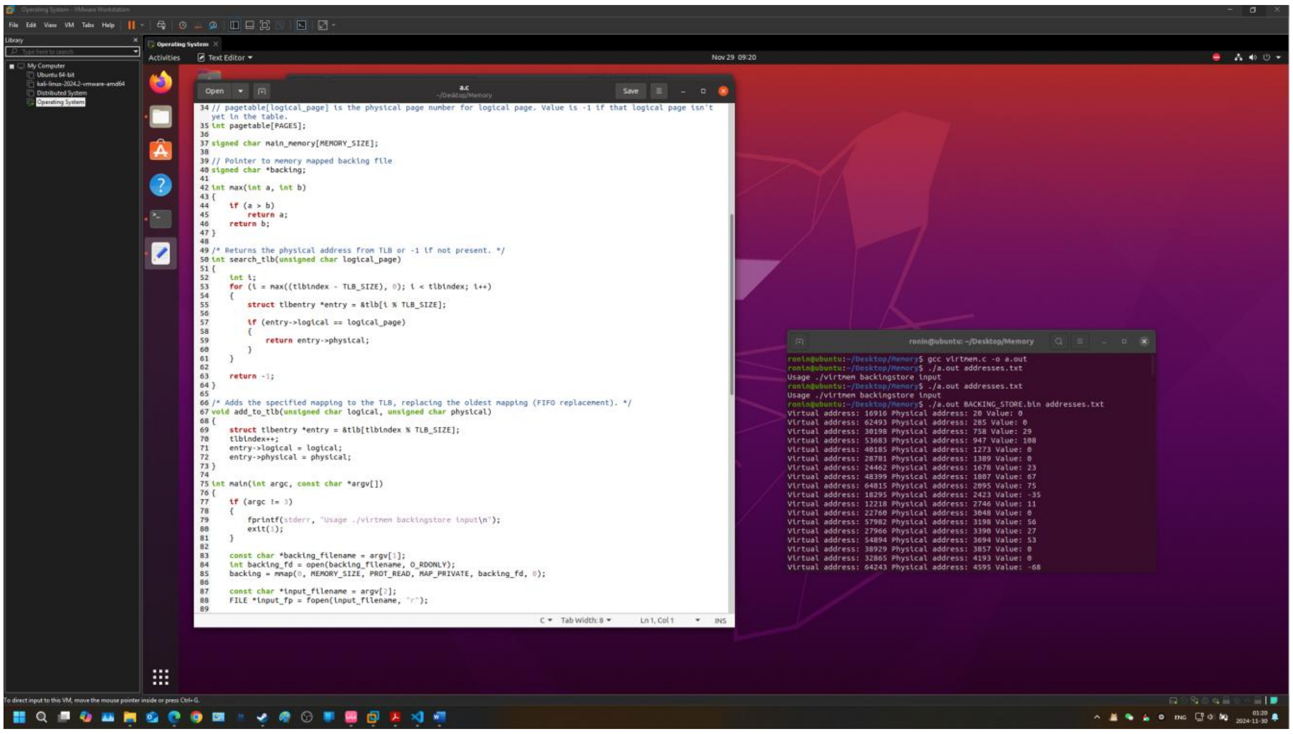
```
{
	if (a > b)
		return a;
	return b;
}

/* Returns the physical address from TLB or -1 if not present. */
int search_tlb(unsigned char logical_page)
{
	int i;
	for (i = max((tlbindex - TLB_SIZE), 0); i < tlbindex; i++)
	{
		struct tlbentry *entry = &tlb[i % TLB_SIZE];

		if (entry->logical == logical_page)
		{
			return entry->physical;
		}
	}

	return -1;
}

/* Adds the specified mapping to the TLB, replacing the oldest mapping (FIFO replacement).
*/
void add_to_tlb(unsigned char logical, unsigned char physical)
{
	struct tlbentry *entry = &tlb[tlbindex % TLB_SIZE];
	tlbindex++;
	entry->logical = logical;
	entry->physical = physical;
}

int main(int argc, const char *argv[])
{
	if (argc != 3)
	{
		fprintf(stderr, "Usage ./virtmem backingstore input\n");
		exit(1);
	}

	const char *backing_filename = argv[1];
	int backing_fd = open(backing_filename, O_RDONLY);
	backing = mmap(0, MEMORY_SIZE, PROT_READ, MAP_PRIVATE, backing_fd, 0);

	const char *input_filename = argv[2];
```

```c
FILE *input_fp = fopen(input_filename, "r");

// Fill page table entries with -1 for initially empty table.
int i;
for (i = 0; i < PAGES; i++)
{
      pagetable[i] = -1;
}

// Character buffer for reading lines of input file.
char buffer[BUFFER_SIZE];

// Data we need to keep track of to compute stats at end.
int total_addresses = 0;
int tlb_hits = 0;
int page_faults = 0;

// Number of the next unallocated physical page in main memory
unsigned char free_page = 0;

while (fgets(buffer, BUFFER_SIZE, input_fp) != NULL)
{
      total_addresses++;
      int logical_address = atoi(buffer);
      int offset = logical_address & OFFSET_MASK;
      int logical_page = (logical_address >> OFFSET_BITS) & PAGE_MASK;

      int physical_page = search_tlb(logical_page);
      // TLB hit
      if (physical_page != -1)
      {
            tlb_hits++;
            // TLB miss
      }
      else
      {
            physical_page = pagetable[logical_page];

            // Page fault
            if (physical_page == -1)
            {
                  page_faults++;

                  physical_page = free_page;
                  free_page++;
```

```
                        // Copy page from backing file into physical memory
                        memcpy(main_memory + physical_page * PAGE_SIZE, backing + 
logical_page * PAGE_SIZE, PAGE_SIZE);

                        pagetable[logical_page] = physical_page;
                  }

                  add_to_tlb(logical_page, physical_page);
            }

            int physical_address = (physical_page << OFFSET_BITS) | offset;
            signed char value = main_memory[physical_page * PAGE_SIZE + offset];

            printf("Virtual address: %d Physical address: %d Value: %d\n", 
logical_address, physical_address, value);
      }

      printf("Number of Translated Addresses = %d\n", total_addresses);
      printf("Page Faults = %d\n", page_faults);
      printf("Page Fault Rate = %.3f\n", page_faults / (1. * total_addresses));
      printf("TLB Hits = %d\n", tlb_hits);
      printf("TLB Hit Rate = %.3f\n", tlb_hits / (1. * total_addresses));

      return 0;
}
```

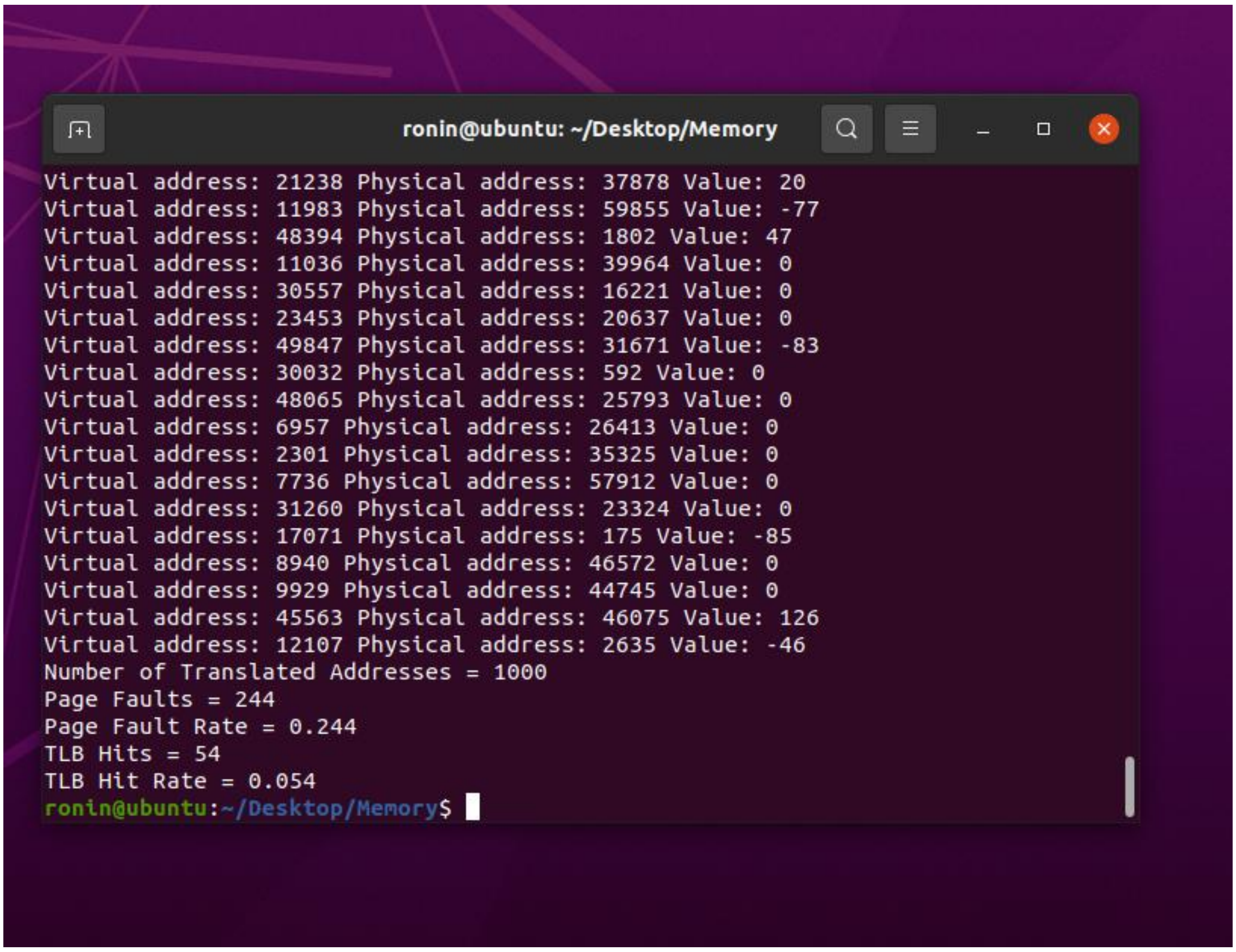
The virtual memory management system operates through a series of well-defined steps to ensure efficient and accurate address translation while maintaining system stability. The system utilizes a **search_tlb()** function to quickly locate the physical page corresponding to a given logical page. If the page is found in the TLB (a "hit"), translation proceeds immediately. Otherwise, a **page fault** handling mechanism is triggered, where the required page is loaded from the backing store into physical memory. To prevent resource exhaustion, a **page replacement policy** such as FIFO or LRU is used to evict old pages when memory is full.

Each new mapping is added to the TLB using the **add_to_tlb()** function, and the system continuously monitors its performance through metrics like TLB hit rate and page fault rate. After processing a request (e.g., translating an address), the system checks for safety and stability using consistent rules; unsafe conditions are detected, changes are rolled back, and requests are denied to prevent disruption. This ensures both accuracy in address resolution and robustness in system operations.



```c
// pagetable[logical_page] is the physical page number for logical page. Value is -1 if that logical page isn't
yet in the table.
int pagetable[PAGES];

signed char main_memory[MEMORY_SIZE];

// Pointer to memory mapped backing file
signed char *backing;

int max(int a, int b)
{
    if (a > b)
        return a;
    return b;
}

/* Returns the physical address from TLB or -1 if not present. */
int search_tlb(unsigned char logical_page)
{
    int i;
    for (i = max((tlbindex - TLB_SIZE), 0); i < tlbindex; i++)
    {
        struct tlbentry *entry = &tlb[i % TLB_SIZE];

        if (entry->logical == logical_page)
        {
            return entry->physical;
        }
    }

    return -1;
}

/* Adds the specified mapping to the TLB, replacing the oldest mapping (FIFO replacement). */
void add_to_tlb(unsigned char logical, unsigned char physical)
{
    struct tlbentry *entry = &tlb[tlbindex % TLB_SIZE];
    tlbindex++;
    entry->logical = logical;
    entry->physical = physical;
}

int main(int argc, const char *argv[])
{
    if (argc != 3)
    {
        fprintf(stderr, "Usage ./virtmem backingstore input\n");
        exit(1);
    }

    const char *backing_filename = argv[1];
    int backing_fd = open(backing_filename, O_RDONLY);
    backing = mmap(0, MEMORY_SIZE, PROT_READ, MAP_PRIVATE, backing_fd, 0);

    const char *input_filename = argv[2];
    FILE *input_fp = fopen(input_filename, "r");

```

```
ronin@ubuntu:~/Desktop/Memory$ gcc virtmem.c -o a.out
ronin@ubuntu:~/Desktop/Memory$ ./a.out addresses.txt
Usage ./virtmem backingstore input
ronin@ubuntu:~/Desktop/Memory$ ./a.out addresses.txt
Usage ./virtmem backingstore input
ronin@ubuntu:~/Desktop/Memory$ ./a.out BACKING_STORE.bin addresses.txt
Virtual address: 16916 Physical address: 20 Value: 0
Virtual address: 62493 Physical address: 285 Value: 0
Virtual address: 30198 Physical address: 758 Value: 29
Virtual address: 53683 Physical address: 947 Value: 108
Virtual address: 40185 Physical address: 1273 Value: 0
Virtual address: 28781 Physical address: 1389 Value: 0
Virtual address: 24462 Physical address: 1678 Value: 23
Virtual address: 48399 Physical address: 1807 Value: 67
Virtual address: 64815 Physical address: 2095 Value: 75
Virtual address: 18295 Physical address: 2423 Value: -35
Virtual address: 12218 Physical address: 2746 Value: 11
Virtual address: 22760 Physical address: 3048 Value: 0
Virtual address: 57982 Physical address: 3198 Value: 56
Virtual address: 27966 Physical address: 3390 Value: 27
Virtual address: 54894 Physical address: 3694 Value: 53
Virtual address: 38929 Physical address: 3857 Value: 0
Virtual address: 32865 Physical address: 4193 Value: 0
Virtual address: 64243 Physical address: 4595 Value: -68
```



```
Virtual address: 21238 Physical address: 37878 Value: 20
Virtual address: 11983 Physical address: 59855 Value: -77
Virtual address: 48394 Physical address: 1802 Value: 47
Virtual address: 11036 Physical address: 39964 Value: 0
Virtual address: 30557 Physical address: 16221 Value: 0
Virtual address: 23453 Physical address: 20637 Value: 0
Virtual address: 49847 Physical address: 31671 Value: -83
Virtual address: 30032 Physical address: 592 Value: 0
Virtual address: 48065 Physical address: 25793 Value: 0
Virtual address: 6957 Physical address: 26413 Value: 0
Virtual address: 2301 Physical address: 35325 Value: 0
Virtual address: 7736 Physical address: 57912 Value: 0
Virtual address: 31260 Physical address: 23324 Value: 0
Virtual address: 17071 Physical address: 175 Value: -85
Virtual address: 8940 Physical address: 46572 Value: 0
Virtual address: 9929 Physical address: 44745 Value: 0
Virtual address: 45563 Physical address: 46075 Value: 126
Virtual address: 12107 Physical address: 2635 Value: -46
Number of Translated Addresses = 1000
Page Faults = 244
Page Fault Rate = 0.244
TLB Hits = 54
TLB Hit Rate = 0.054
ronin@ubuntu:~/Desktop/Memory$
```