Lab Report of Project 8
Designing a Virtual Memory Manager

Name: Li Dongyue   Student Number: 516030910502
Computer System Engineering (Undergraduate), Fall 2018

December 29, 2018

1 Introduction

This project consists of writing a program that translates logical to physical addresses for a virtual address space of size $2^{16} = 65536$ bytes. This program reads from a file containing logical addresses and, using a TLB and a page table, will translate each logical address to its corresponding physical address and output the value of the byte stored at the translated physical address. We are to simulate the process of translating logical to physical addresses. This includes:

- Resolve page faults using demand paging
- Manage a TLB
- Implement a page-replacement algorithm

We are provided with the file `addresses.txt`, which contains integer values representing logical addresses ranging from 0 to 65535 (the size of the virtual address space). Our program will open this file, read each logical address and translate it to its corresponding physical address, and output the value of the signed byte at the physical address. Furthermore, out program is required to to report the following statistics: Page-fault rate and TLB hit rate.

2 Implementation

In the implementation of my program, I create a list `page_table` to represent page table and a two-dimensional list `tlb` to represent Translation Look-aside Buffer (TLB). Furthermore, I create an array `memory` to store physical memory data.

2.1 Initialization

In my program, I first initialize the page table and TLB. In initialization, I set all the values in page table and TLB to empty represented as -1. Then I map the given `.bin` file into memory:

```c
store_fd = open(store_file, O_RDONLY);
store_data = mmap(0, MEM_SIZE, PROT_READ, MAP_SHARED, store_fd, 0);
/* Check that the mmap call succeeded. */
if (store_data == MAP_FAILED) {
    close(store_fd);
    printf("Error mmapping the backing store file!");
    exit(EXIT_FAILURE);
}
```

virtual_mem.c

2.2 Input Processing

Then I move on to process input file. The virtual address is 16 bit, composed of 8 bit page number and 8 bit page offset. The program loops through the input file one line at a time and get the page number and offset number from the virtual address.

```c
/* Calculate and return the page number. */
int get_page_number(int virtual) {
    /* Shift virtual to the right by n bits. */
    return (virtual >> PAGE_NUM_BITS);
}

/* Calculate and return the offset value. */
int get_offset(int virtual) {
    /* Mask is decimal representation of 8 binary 1 digits.
     * It is actually 2^8-1. */
    int mask = 255;

    return virtual & mask;
}
```

virtual_mem.c

2.2.1 Managing TLB

After getting page number and page offset, first, I find the frame number in TLB to check whether it is in TLB. If it is in TLB, the program does not update the TLB and fetch the value directly from memory. Otherwise, the program look for page number in page table and update the TLB.

```c
int consult_tlb(int page_number) {
    /* If page_number is found, return the corresponding frame number. */
    for (int i = 0; i < TLB_ENTRIES; i++) {
        if (tlb[i][0] == page_number) {
            /* TLB hit! */
            tlb_counter++;

            return tlb[i][1];
        }
    }

    /* If page_number doesn't exist in TLB, return -1. */
    /* TLB miss! */
    return -1;
}
```

virtual_mem.c

2.2.2 Updating the TLB

My program use a **FIFO** policy to update TLB. I use circular array to implement the queue and always insert a new TLB entry in the back. In each update operation, the program update the queue tail accordingly and insert the new entry in the queue tail.

```c
void update_tlb(int page_number, int frame_number) {
    /* Use FIFO policy. */
    if (tlb_front == -1) {
        tlb_front = 0;
        tlb_back = 0;

        /* Update TLB. */
        tlb[tlb_back][0] = page_number;
        tlb[tlb_back][1] = frame_number;
    }
    else {
        tlb_back = (tlb_back + 1) % TLB_ENTRIES;

        /* Insert new TLB entry in the back. */
        tlb[tlb_back][0] = page_number;
        tlb[tlb_back][1] = frame_number;
    }

    return;
}
```

virtual_mem.c

2.2.3 Resolving Page Fault and Implementing Page Replacement

If the given virtual memory value is not in TLB, the program checks the value in page table.

```c
int consult_page_table(int page_number) {
    if (page_table[page_number] == -1) {
        fault_counter++;
    }

    return page_table[page_number];
}
```

virtual_mem.c

If no page fault occurs, the program calculate the physical address as frame number plus the offset. Then the program update the TLB accordingly and fetch the value directly from memory.

```c
/* No page fault. */
physical = frame_number + offset;

/* Update TLB. */
update_tlb(page_number, frame_number);

/* Fetch the value directly from memory. */
value = memory[physical];
```

virtual_mem.c

If page fault occurs, the program will read in a 256-byte page from the file `BACKING_STORE.bin` and store it in an available page frame in the physical memory. The program check if a free frame exits. If yes, the program will:

- store the page from store file into memory frame
- update page table with correct frame number
- update TLB

```c
/* Success, a free frame exists. */
/* Store the page from store file into memory frame. */
memcpy(memory + mem_index,
        store_data + page_address, PAGE_SIZE);

frame_number = mem_index;
physical = frame_number + offset;

value = memory[physical];

/* Update page_table with correct frame number. */
page_table[page_number] = mem_index;

/* Update TLB. */
update_tlb(page_number, frame_number);


if (mem_index < MEM_SIZE - FRAME_SIZE) {
    mem_index += FRAME_SIZE;
}
else {
    mem_index = -1;
}
```

virtual_mem.c

if no free frame in memory exists, the program will initiate the page replacement. In this program, physical memory is the same size as the virtual address space. Therefore, no page replacement is needed.

The whole code is in attached file.

3 How to run the program

In this section, I describe how to run this program.

Compile with: `gcc virtual_mem.c -o virtual_mem.o -std=c99 -lm`

Run with: `./virtual_mem.o addresses.txt output.txt BACKING_STORE.bin`