Designing a Virtual Memory Manager

Overview

This project consists of writing a program that translates logical to physical addresses for a virtual address space of size 216 = 65,536 bytes. Your program will read 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. Your learning goal is to use simulation to understand the steps involved in translating logical to physical addresses. This will include resolving page faults using demand paging,managing a TLB,and implementing a page-replacement algorithm.

- 28 entries in the page table
- Page size of 28 bytes
- 16 entries in the TLB
- Frame size of 2 8 bytes
- 256 frames
- Physical memory of 65,536 bytes (256 frames×256-byte frame size)

Program Structure

Data structure

I simply use arrays to represent page table and TLB.

```
int page_table[PAGE_TABLE_ENTRIES];
int TLB[TLB_ENTRIES][2];
char memory[MEM_SIZE];

int mem_idx;
int tlb_pt;
bool tlb_full;


void initMemory()
{
    mem_idx = 0;
    printf("Memory inited.\n");
}

void initPageTable()
{
    for(int i = 0; i < PAGE_TABLE_ENTRIES; i++)
    {
        page_table[i] = -1;
    }
    printf("Page table inited.\n");
}


void initTLB()
{
    tlb_pt = 0;
    tlb_full = 0;
    for(int i = 0; i < TLB_ENTRIES; i++)
    {
        TLB[i][0] = -1;
        TLB[i][1] = -1;
    }
    printf("TLB inited.\n");
}
```

Then, I implement `requestTLB()` and `requestPageTable()` to represent the process of the manager consulting TLB and page table for the result and `updateTLB()` to update TLB with new pages and frames (here I use FIFO to update TLB).

```
int requestPageTable(int page_num)
{
    return page_table[page_num];
}


int requestTLB(int page_num)
{
    if(tlb_full)
    {
        /* if TLB is full */
        for(int i = 0; i < TLB_ENTRIES; i++)
        {
            if(TLB[i][0] == page_num)
                return TLB[i][1];
        }
        return -1;
    }
    else
    {
        /* if TLB isn't full */
        for(int i = 0; i < tlb_pt; i++)
        {
            if(TLB[i][0] == page_num)
                return TLB[i][1];
        }
        return -1;
    }
}
```

TLB Miss and Page fault

If the page number hasn't been recorded in TLB, a TLB miss occurs. The manager turns to the page table.

If the page number hasn't been recorded in the page table, a page fault occurs.

If a page fault occurs

1. store the page into memory frame.
2. update page table
3. update TLB

The size of physical memory is the same as the size of the virtual address space—65,536bytes—so you do not need to be concerned about page replacements during a page fault. Later, we describe a modification to this project using a smaller a mount of physical memory; at that point,a page-replacement strategy will be required.

```
page_num = getPage(address);
offset = getOffset(address);
frame_num = requestTLB(page_num);
if(frame_num != -1)
{
    tlb_hit++;
    physical = frame_num * FRAME_SIZE + offset;

    value = memory[physical];
}
else
{
    frame_num = requestPageTable(page_num);
    if(frame_num != -1)
    {
        physical = frame_num * FRAME_SIZE + offset;
        value = memory[physical];
        updateTLB(page_num, frame_num);

    }
    else
    {
        page_address = page_num * PAGE_SIZE;
        page_fault++;
        if(mem_idx != -1)
        {
            /* if there is still a free frame available */
            memcpy(memory + mem_idx,store_data + page_address, PAGE_SIZE);
            physical = mem_idx + offset;
            value = memory[mem_idx + offset];
            //printf("physical %d %d\n",physical,value);
            page_table[page_num] = mem_idx >> OFFSET_BITS;
            updateTLB(page_num,mem_idx >> OFFSET_BITS);
            if(mem_idx < MEM_SIZE - FRAME_SIZE)
            {
                mem_idx += FRAME_SIZE;
            }
            else
            {
                /* the memory is used up */
```

```
                    mem_idx = -1;
                }
            }
            {
                /* replacement */
            }

        }
    }
```

How to run the program

To compile the files, enter `gcc -o test virtual_mem.c`

To run it, enter `./test addresses.txt`