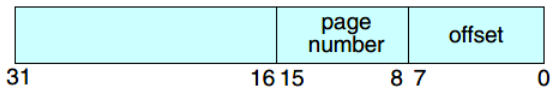
Project 8

Name: 刘　涵　之
ID : 519021910102

Project 8: Designing a Virtual Memory Manager

This project consists of writing a program that translates logical to physical addresses for a virtual address space of size $2^{16} = 65536$ bytes. The program will read from a file containing logical address 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. It requires us to use simulation to understand the steps involved in translating logical address to physical address, and this will include resolving page faults using demand paging, managing a TLB, and implementing a page-replacement algorithm.

The program will read a file containing several 32-bit integer numbers that represent logical addresses. However, the 16-bit addresses is the only thing that needs to be concerned, so we must mask the rightmost 16 bits of each logical addresses. These 16 bits are divided into an 8-bit page number and an 8-bit page offset. Hence, the addresses are structured as shown as:



Other specifics includes the following:

- 2^8 entries in the page table;
- Page size of 2^8 bytes;
- 16 entries in the TLB;
- Frame size of 2^8 bytes;
- 256 frames;
- Physical memory of 65536 bytes (256 frames $\times$ 256-byte frame size).

The program is to output the following values:

- The logical address being translated (the integer value being read from `addresses.txt`).
- The corresponding physical address (what your program translates the logical address to).
- The signed byte value stored in physical memory at the translated physical address.

We also provide the file `correct.txt`, which contains the correct output values for the file `addresses.txt`. You should use this file to determine if the program is correctly translating logical to physical addresses.

After completion, the program is to report the following statistics:

- Page-fault rate - The percentage of address references that resulted in page faults.
- TLB hit rate - The percentage of address references that were resolved in the TLB.

Since the logical addresses in `addresses.txt` were generated randomly and do not reflect any memory access locality, do not expect to have a high TLB hit rate.

Then use a smaller physical address space with 128 page frames rather than 256. This change will require modifying your program so that it keeps track of free page frames as well as implementing a page-replacement policy using either FIFO or LRU (Section 10.4) to resolve page faults when there is no free memory.

Design: My design for this task is:

- All the replacement (memory and TLB) is implemented with LRU algorithm.
- **swap_page_in()** function will first search for the empty frame. If there is no empty frame, LRU algorithm will be used to find a victim to swap. The victim will be deleted from TLB using **delete_TLB()** function.
- For translating a virtual address to a physical one, the program will first look up TLB using **get_frame_TLB()** function. If TLB miss, it will look up `page_table` with `page_table_inmem`. If page table miss, it will swap the page in from `backing_store` using **swap_page_in()** function.

The implementation of the virtual memory manager (`vmm.c`) is shown as follows.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define PAGE_NUM 256
#define PAGE_SIZE 256
#define FRAME_NUM 256
#define FRAME_SIZE 256
#define TLB_SIZE 16

int get_frame_TLB(int page_number);
void TLB_update(int page_number, int frame_number);
void delete_TLB(int page_number, int frame_number);

char memory[FRAME_NUM * FRAME_SIZE];

FILE *backing_store;
int total = 0, page_fault = 0, tlb_hit = 0;

int page_table[PAGE_NUM], page_table_inmem[PAGE_NUM];
int frame_count[FRAME_NUM];

void swap_page_out(int frame_number) {
    int page_number = -1;

        for (int i = 0; i < PAGE_NUM; i++) {
                if(page_table_inmem[i] > 0 && page_table[i] == frame_number) {
                        page_number = i;
                        break;
                }
    }
    if (page_number == -1) {
        for (int i = 0; i < PAGE_NUM; i++) {

```

```
        printf("%d, %d \n", page_table_inmem[i], page_table[i]);
    }
                fprintf(stderr, "[Err] Unexpectedddd Error!\n");

                exit(1);
        }
    page_table_inmem[page_number] = 0;

    //TLB delete
    delete_TLB(page_number, frame_number);

    return;
}

int swap_page_in(int page_number) {
    char buf[FRAME_SIZE];
    fseek(backing_store, page_number * FRAME_SIZE, 0);
    fread(buf, sizeof(char), FRAME_SIZE, backing_store);

    int target = -1;
    for (int i = 0; i < FRAME_NUM; i++) {
        if (frame_count[i] == 0) {
            target = i;
            break;
        }
    }
    if (target == -1) {
        for (int i = 0; i < FRAME_NUM; i ++) {
            if (frame_count[i] == FRAME_NUM) {
                target = i;
                break;
            }
        }
        swap_page_out(target);
    }
    for (int i = 0; i < FRAME_SIZE; ++ i) memory[target * FRAME_SIZE + i] = buf[i];
        for (int i = 0; i < FRAME_NUM; ++ i) if (frame_count[i] > 0) frame_count[i]++;
        frame_count[target] = 1;
        return target;
}


int get_val(int frame_number, int offset) {
    int val = (int)memory[frame_number * FRAME_SIZE + offset];

    for (int i = 0; i < FRAME_NUM; ++ i) {
        if (frame_count[i] != 0 && frame_count[i] < frame_count[frame_number])
                        frame_count[i]++;
    }
    frame_count[frame_number] = 1;

```

```
86      return val;
87  }
88
89  int get_frame(int page_number) {
90
91      // TLB
92      int tlb_frame = get_frame_TLB(page_number);
93      if (tlb_frame != -1) return tlb_frame;
94
95      if (page_table_inmem[page_number] > 0) {
96
97          // TLB
98          TLB_update(page_number, page_table[page_number]);
99
100                 return page_table[page_number];
101         } else {
102                 page_fault++;
103                 page_table[page_number] = swap_page_in(page_number);
104                 page_table_inmem[page_number] = 1;
105
106         // TLB
107         TLB_update(page_number, page_table[page_number]);
108                 return page_table[page_number];
109         }
110     return -1;
111 }
112
113
114 // TLB
115
116 int TLB_list[TLB_SIZE][2]; // 0 -> page; 1 -> frame
117 int TLB_count[TLB_SIZE];
118
119 int get_frame_TLB(int page_number) {
120     int frame = -1, loc = -1;
121     for (int i = 0; i < TLB_SIZE; ++ i)
122                 if (TLB_count[i] != 0 && TLB_list[i][0] == page_number) {
123                         frame = TLB_list[i][1];
124             loc = i;
125                         break;
126                 }
127
128     if (frame == -1) return -1; // TLB miss
129
130     tlb_hit++;
131     for (int i = 0; i < TLB_SIZE; i++)
132         if (TLB_count[i] != 0 && TLB_count[i] < TLB_count[loc]) {
133                         TLB_count[i]++;
134         }
135     TLB_count[loc] = 1;
136
```

```
137     return frame;
138 }
139 
140 void TLB_update(int page_number, int frame_number) {
141     int loc = -1;
142         for (int i = 0; i < TLB_SIZE; i++)
143                 if(TLB_count[i] == 0) {
144                         loc = i;
145                         break;
146                 }
147         if (loc == -1) {
148                 for (int i = 0; i < TLB_SIZE; i++)
149                         if(TLB_count[i] == TLB_SIZE) {
150                                 loc = i;
151                                 break;
152                         }
153         }
154 
155     for (int i = 0; i < TLB_SIZE; i++)
156                 if (TLB_count[i] > 0) TLB_count[i] += 1;
157 
158         TLB_count[loc] = 1;
159     TLB_list[loc][0] = page_number;
160     TLB_list[loc][1] = frame_number;
161     return;
162 }
163 
164 void delete_TLB(int page_number, int frame_number) {
165     int loc = -1;
166         for (int i = 0; i < TLB_SIZE; ++ i)
167                 if(TLB_count[i] == 0) {
168                         loc = i;
169                         break;
170                 }
171         if (loc == -1) return; // not find, no need to delete
172 
173     for (int i = 0; i < TLB_SIZE; ++ i)
174                 if (TLB_count[i] > TLB_count[loc]) TLB_count[i] -= 1;
175 
176         TLB_count[loc] = 0;
177 }
178 
179 
180 int main(int argc, char *argv[]) {
181     if (argc != 2) {
182         fprintf(stderr, "[Error] Unexpected input!\n");
183                 return 1;
184     }
185 
186     backing_store = fopen("BACKING_STORE.bin", "r");
187 
```

```
    for (int i = 0; i < PAGE_NUM; i++) {
        page_table[i] = 0;
        page_table_inmem[i] = 0;
    }
    for (int i = 0; i < FRAME_NUM; i++) {
        frame_count[i] = 0;
    }
    for (int i = 0; i < TLB_SIZE; i++) {
        TLB_list[i][0] = 0;
        TLB_list[i][1] = 0;
        TLB_count[i] = 0;
    }

        FILE *stream_in = fopen(argv[1], "r");
        FILE *stream_out = fopen("output.txt", "w");

    int virtual_addr;
    int page_number, offset, frame_number, val;

    while(~fscanf(stream_in, "%d", &virtual_addr)) {
        total += 1;

        virtual_addr = virtual_addr & 0x0000ffff;
        page_number = (virtual_addr >> 8) & 0x000000ff;
                offset = virtual_addr & 0x000000ff;

        frame_number = get_frame(page_number);
        val = get_val(frame_number, offset);

        fprintf(stream_out, "Virtual address: %d Physical address: %d Value: %d\n",
            virtual_addr, (frame_number << 8) + offset, val);
    }
    fprintf(stdout, "TLB hit rate: %.2f%%      Page fault rate: %.2f%%\n", 100.0 *
        tlb_hit / total, 100.0 * page_fault / total);

    fclose(stream_in);
    fclose(stream_out);

    return 0;
}
```

To check the result, i also implement a checker program (`judge.c`), the main function of it is to compare the value of the answer and the output. If the output is correct, this program will print **All correct!**, otherwise, it will print **Wrong answer!**. The code of `judge.c` is shown as follow.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
        FILE *stream_ans = fopen("output.txt", "r");
        FILE *stream_std = fopen("correct.txt", "r");
```

```

    int a, b, c;
    int ac = 0; // 0 => ac; 1 => wrong

    while(~fscanf(stream_ans, "Virtual address: %d Physical address: %d Value: %d\n", &a,
         &b, &c)) {
        int d, e, f;
        if (fscanf(stream_std, "Virtual address: %d Physical address: %d Value: %d\n", &d
            , &e, &f) == EOF) {
            printf("File length not match!\n");
                        ac = 1;
                        break;
        }
        if (c != f) {
            printf("Wrong answer!\n");
                        ac = 1;
                        break;
        }
    }

    if (ac == 0) printf("All correct!\n");

    fclose(stream_ans);
    fclose(stream_std);

    return 0;
}
```

Makefile for this task is shown as below:

```
CC=gcc
CFLAGS=-Wall

all: vmm.o judge.o
        $(CC) $(CFLAGS) -o vmm vmm.o
        $(CC) $(CFLAGS) -o judge judge.o

vmm.o: vmm.c
        $(CC) $(CFLAGS) -c vmm.c

judge.o: judge.c
        $(CC) $(CFLAGS) -c judge.c

clean:
        rm -rf *.o
        rm -rf vmm
        rm -rf judge
```

When the **FRAME_NUM** is 128, the execution result of the virtual memory manager is shown as follows:

```
misaka@MS-BVZPMBEQIPCD:/mnt/c/Projects/OS_Project/Project8/project8$ make all
gcc -Wall -c vmm.c
gcc -Wall -c judge.c
gcc -Wall -o vmm vmm.o
gcc -Wall -o judge judge.o
misaka@MS-BVZPMBEQIPCD:/mnt/c/Projects/OS_Project/Project8/project8$ ./vmm addresses.txt
TLB hit rate: 5.50%      Page fault rate: 53.90%
misaka@MS-BVZPMBEQIPCD:/mnt/c/Projects/OS_Project/Project8/project8$ ./judge
All correct!
```

图 1: Designing a Virtual Memory Manager (frame number is 128)

When the **FRAME_NUM** is 256, the execution result of the virtual memory manager is shown as follows. The page fault rate is dropped.

```
misaka@MS-BVZPMBEQIPCD:/mnt/c/Projects/OS_Project/Project8/project8$ make all
gcc -Wall -c vmm.c
gcc -Wall -c judge.c
gcc -Wall -o vmm vmm.o
gcc -Wall -o judge judge.o
misaka@MS-BVZPMBEQIPCD:/mnt/c/Projects/OS_Project/Project8/project8$ ./vmm addresses.txt
TLB hit rate: 5.50%      Page fault rate: 24.40%
misaka@MS-BVZPMBEQIPCD:/mnt/c/Projects/OS_Project/Project8/project8$ ./judge
All correct!
```

图 2: Designing a Virtual Memory Manager (frame number is 256)