Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра системного програмування
і спеціалізованих комп'ютерних системи

Лабораторна робота №3

з дисципліни
«Архітектура комп'ютерів 2. Програмне забезпечення»

Виконв:
студент групи КВ-84
Воєводін Ілля петрович

Перевірив:

Молчанов О. А.

Київ – 2021

Загальне завдання

Завдання лабораторної роботи наступне: реалізувати програму мовою С, що виконує зчитування послідовності команд (програми) з файлу і заміняє віртуальні адреси на фізичні в командах, що визначаються варіантом. Тип організації пам'яті також визначається варіантом. Заміна адреси відбувається у випадку, якщо сторінка та/або сегмент знаходиться в оперативній пам'яті (ОП). Якщо потрібна віртуальна сторінки та/або сегмент відсутній в ОП, тоді має бути виведене повідомлення про помилку відсутності сторінки/сегменту, й аналіз команд має бути продовжено. Таблиця сторінок/сегментів задається у файлі формату CSV.

Програма має містити наступні компоненти:

1. Модуль зчитування таблиці сторінок/сегментів з файлу CSV і створення внутрішнього представлення відповідної таблиці (або таблиць);

2. Модуль з реалізацією функцій зчитування, аналізу і зміни команд з бінарного файлу, що виконує заміну віртуальних адрес в зчитаних командах на фізичні;

3. Модуль тестування, що містить тести реалізованої програми.

В результаті виконання лабораторної роботи має бути підготовлено звіт, який містить:

1) титульний аркуш;
2) загальне завдання лабораторної роботи;
3) варіант і завдання за варіантом (інформація про варіанти наведена в п. 4);
4) лістинг реалізованої програми (перших двох модулів);
5) результати тестування програми для декількох наборів тестових даних.

Тести мають виконувати перевірку всіх пунктів основного завдання, завдання за варіантом іпродемонструвати коректність роботи реалізованої програми.

Завдання за варіантом 12

Тип організації пам'яті	Параметри віртуальної адреси	Список команд
сегментно-сторінкова	РС: 2 Кбайт, РТД: 2048	1, 2, 3, 4, 5, 24, 25, 26, 28

Система команд:

№	Команда	Код команди (0x)	Опис
1	MOV <reg1>, <reg2>	1A /reg1 /reg2	перемістити значення з регістру <reg1> у регістр <reg2>
2	MOV <reg>,<addr>	1B 0 /reg /addr	перемістити значення з ОП за адресою <addr> у регістр <reg>
3	MOV <addr>,<reg>	1B 1 /reg /addr	перемістити значення з регістру <reg> в ОП за адресою <addr>
4	ADD <reg1>,<reg2>	01 /reg1 /reg2	додавання значення з регістру <reg1> до значення з регістру <reg2> і збереження результату в регістрі <reg1>
5	ADD <reg1>,<reg2>,<addr>	02 0 /reg /addr	одавання значення з регістру <reg> до 4-байтового значення з ОП за адресою <addr> і збереження результату в регістрі <reg>

24	JG <shift>	94 /shift	перехід за 1-байтовим відносним зміщенням <shift> у випадку, якщо ZF = 0 і SF = OF
25	JG <addr>	95 /addr	перехід за 4-байтовою адресою <addr> у випадку, якщо ZF = 0 і SF = OF
26	CMP <reg1>, <reg2>	80 /reg1 /reg2	порівняння двох значень і встановлення відповідних прапорців
28	MOV <reg>,<lit16>	1C 1 /reg /lit16	переміщення 2-байтового числа у регістра <reg>

Модуль зчитування таблиці сторінок/сегментів з файлу CSV і створення внутрішнього представлення відповідної таблиці (або таблиць)

Input.h

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define _CRT_SECURE_NO_WARNINGS
#define ArrSize 50

#pragma warning(disable : 4996)

typedef struct segments_table {
	char* pages_table_path;
	int records;
	int segment_number;

	struct segment_table* next_seg;
} segment_table;


typedef struct pages_table {
	int bit;
	int page_number;
	int frame;

	struct pages_table* next_page;
} page_table;


segment_table* parse_segment_table(char* filename);
segment_table* add_segment(char* path, int record, int seg_num, segment_table** seg_node);

page_table* parse_pages_table(char* filename);
page_table* add_page(int bit, int page_num, int frame, page_table** pages_node);
```

Input.c

```
#include "input.h"
```

```
segment_table* parse_segment_table(char* filename)
{
	segment_table* Head = NULL;
	segment_table* seg_node = NULL;

	FILE* file = fopen(filename, "r");
	int seg_num = 0;
	char buf[ArrSize];
	char* Path = (char*)malloc(sizeof(char) * ArrSize);
	while (fgets(buf, ArrSize, file))
	{
		int ind = 0;
		for (int i = 0; i < ArrSize; i++)
		{
			Path[i] = NULL;
		}
		char record[ArrSize] = { NULL };

		for (int i = 0; buf[i]; i++)
		{
			if (buf[i] != ';') {
				Path[i] = buf[i];
				Path[i + 1] = '\0';
			}
			else
				break;
			ind++;
		}

		int j = 0;
		for (int i = ind + 1; buf[i]; i++)
		{

			if (buf[i] != ';') {
				record[j] = buf[i];
				j++;
			}
			else
				break;
			ind++;
		}

		add_segment(Path, atoi(record), seg_num, &seg_node);
		seg_num++;

		if (Head == NULL)
			Head = seg_node;
	}
	fclose(file);
	return Head;
}

page_table* parse_pages_table(char* filename)
{
	page_table* Head = NULL;
	page_table* page_node = NULL;

	FILE* file = fopen(filename, "r");
	char buffer[ArrSize];
```

```
	while (fgets(buffer, ArrSize, file))
	{
		int ind = 0;
		char page_num[ArrSize] = { NULL };
		char frame[ArrSize] = { NULL };
		char bit[1] = { NULL };
		for (int i = 0; buffer[i]; i++)
		{
			if (buffer[i] != ';')
				page_num[i] = buffer[i];
			else {
				ind = i + 1;
				break;
			}
		}

		bit[0] = buffer[ind];
		ind += 2;

		int j = 0;
		for (int i = ind; buffer[i]; i++)
		{
			if (buffer[i] != ';')
			{
				frame[j] = buffer[i];
				j++;
			}
			else
				break;
		}

		add_page(atoi(bit), atoi(page_num), atoi(frame), &page_node);

		if (Head == NULL)
			Head = page_node;
	}
	fclose(file);
	return Head;
}

segment_table* add_segment(char* path, int record, int seg_num, segment_table** seg_node)
{
	segment_table* tmp = *seg_node;
	if (*seg_node == NULL)
		(*seg_node) = (segment_table*)malloc(sizeof(segment_table));
	else
	{
		while (tmp->next_seg)
			tmp = tmp->next_seg;
	}
	tmp = (segment_table*)malloc(sizeof(segment_table));

	tmp->pages_table_path = path;
	tmp->records = record;
	tmp->segment_number = seg_num;
	tmp->next_seg = NULL;
	(*seg_node)->next_seg = tmp;
	(*seg_node) = tmp;

	return (*seg_node);
}
```

```
page_table* add_page(int bit, int page_num, int frame, page_table** page_node) {
	page_table* tmp = *page_node;
	if (*page_node == NULL)
		(*page_node) = (page_table*)malloc(sizeof(page_table));
	else
	{
		while (tmp->next_page)
			tmp = tmp->next_page;
	}
	tmp = (page_table*)malloc(sizeof(page_table));

	tmp->bit = bit;
	tmp->page_number = page_num;
	tmp->frame = frame;
	tmp->next_page = NULL;
	(*page_node)->next_page = tmp;
	(*page_node) = tmp;

	return (*page_node);
}
```

Модуль з реалізацією функцій зчитування, аналізу і зміни команд з бінарного файлу, що виконує заміну віртуальних адрес в зчитаних командах на фізичні

data.h

```
#include "addr_check.h"

void data_analyze(char* data_filename, page_table* page_node, segment_table* seg_node);
```

data.c

```
#include "data.h"

void data_analyze(char* dat_filename, page_table* page_node, segment_table* seg_node)
{
	FILE* file = fopen(dat_filename, "rb");

	int i = 0;
	unsigned __int8 byte;
	long lst_result[ArrSize] = { NULL };

	while (fread(&byte, sizeof(byte), 1, file))
	{
		switch (byte)
		{
		case 0x1A:
		{
			printf("%X ", byte);
			strcat(lst_result, "MOV ");

			fread(&byte, sizeof(byte), 1, file);
			printf("%X ", byte);
			char lst_buf[ArrSize] = "";
			sprintf(lst_buf, "R%X ", byte);
			strcat(lst_result, lst_buf);

			fread(&byte, sizeof(byte), 1, file);
			printf("%X ", byte);
			sprintf(lst_buf, "R%X", byte);
```

```
        strcat(lst_result, lst_buf);
        break;
}
case 0x1B:
{
        printf("%X ", byte);
        strcat(lst_result, "MOV ");

        fread(&byte, sizeof(byte), 1, file);
        printf("%X ", byte);

        char lst_buf[ArrSize] = "";
        if (byte == 1)
        {
                unsigned __int32 addr;
                fread(&addr, sizeof(addr), 1, file);
                printf("%.8X ", addr);
                sprintf(lst_buf, "[0x%.8X] ", addr);
                strcat(lst_result, lst_buf);

                fread(&byte, sizeof(byte), 1, file);
                printf("%X ", byte);
                sprintf(lst_buf, "R%X ", byte);
                strcat(lst_result, lst_buf);

                addr = address_checkout(addr, page_node, seg_node);
                break;
        }

        fread(&byte, sizeof(byte), 1, file);
        printf("%X ", byte);
        sprintf(lst_buf, "R%X ", byte);
        strcat(lst_result, lst_buf);

        unsigned __int32 addr;
        fread(&addr, sizeof(addr), 1, file);
        printf("%.8X ", addr);
        addr = address_checkout(addr, page_node, seg_node);
        sprintf(lst_buf, "[0x%.8X] ", addr);
        strcat(lst_result, lst_buf);
        break;
}
case 0x01:
{
        printf("%.2X ", byte);
        strcat(lst_result, "ADD ");

        char lst_buf[ArrSize] = "";
        fread(&byte, sizeof(byte), 1, file);
        printf("%X ", byte);
        sprintf(lst_buf, "R%X ", byte);
        strcat(lst_result, lst_buf);

        fread(&byte, sizeof(byte), 1, file);
        printf("%X ", byte);
        sprintf(lst_buf, "R%X ", byte);
        strcat(lst_result, lst_buf);
        break;
}
case 0x02:
{
```

```
        printf("%.2X ", byte);
        strcat(lst_result, "ADD ");

        fread(&byte, sizeof(byte), 1, file);
        printf("%X ", byte);

        char lst_buf[ArrSize] = "";
        fread(&byte, sizeof(byte), 1, file);
        printf("%X ", byte);
        sprintf(lst_buf, "R%X ", byte);
        strcat(lst_result, lst_buf);

        unsigned __int32 addr;
        fread(&addr, sizeof(addr), 1, file);
        printf("%.8X ", addr);
        addr = address_checkout(addr, page_node, seg_node);
        sprintf(lst_buf, "[0x%.8X] ", addr);
        strcat(lst_result, lst_buf);
        break;
}
case 0x94:
{
        printf("%X ", byte);
        strcat(lst_result, "JG ");

        char lst_buf[ArrSize] = "";
        fread(&byte, sizeof(byte), 1, file);
        printf("%X ", byte);
        sprintf(lst_buf, "%X ", byte);
        strcat(lst_result, lst_buf);
        break;
}
case 0x95:
{
        printf("%X ", byte);
        strcat(lst_result, "JG ");

        char lst_buf[ArrSize] = "";
        unsigned __int32 addr;
        fread(&addr, sizeof(addr), 1, file);
        printf("%.8X ", addr);
        addr = address_checkout(addr, page_node, seg_node);
        sprintf(lst_buf, "[0x%.8X] ", addr);
        strcat(lst_result, lst_buf);
        break;
}
case 0x80:
{
        printf("%X ", byte);
        strcat(lst_result, "CMP ");

        fread(&byte, sizeof(byte), 1, file);
        printf("%X ", byte);
        char lst_buf[ArrSize] = "";
        sprintf(lst_buf, "R%X ", byte);
        strcat(lst_result, lst_buf);

        fread(&byte, sizeof(byte), 1, file);
        printf("%X ", byte);
        sprintf(lst_buf, "R%X ", byte);
        strcat(lst_result, lst_buf);
```

```
                        break;
                }
                case 0x1C:
                {
                        printf("%X ", byte);
                        strcat(lst_result, "MOV ");

                        char lst_buf[ArrSize] = "";
                        fread(&byte, sizeof(byte), 1, file);
                        printf("%X ", byte);

                        fread(&byte, sizeof(byte), 1, file);
                        printf("%X ", byte);
                        sprintf(lst_buf, "R%X ", byte);
                        strcat(lst_result, lst_buf);

                        unsigned __int16 num;
                        fread(&num, sizeof(num), 1, file);
                        printf("%.4X ", num);
                        sprintf(lst_buf, "%.4X ", num);
                        strcat(lst_result, lst_buf);
                        break;
                }
                default:
                        break;
                }

                printf(":\n\t// %s //\n", lst_result);

                for (int j = 0; lst_result[j]; j++)
                        lst_result[j] = NULL;
        }

        fclose(file);
}
```

Addr_check.h

```
#include "input.h"

int address_checkout(unsigned __int32 virt_addr, page_table* pagesLst, segment_table*
segmentsLst);
```

addr_check.c

```
#include "addr_check.h"
#define BIN8(x) BIN___(0##x)
#define BIN___(x)                                       \
        (                                                \
        ((x / 01ul) % 010)*(2>>1) +                      \
        ((x / 010ul) % 010)*(2<<0) +                     \
        ((x / 0100ul) % 010)*(2<<1) +                    \
        ((x / 01000ul) % 010)*(2<<2) +                   \
        ((x / 010000ul) % 010)*(2<<3) +                  \
        ((x / 0100000ul) % 010)*(2<<4) +                 \
        ((x / 01000000ul) % 010)*(2<<5) +                \
        ((x / 010000000ul) % 010)*(2<<6) +               \
        ((x / 0100000000ul) % 010)*(2<<7)                \
        )
```

```
#define BIN16(x1,x2) \
    ((BIN___(x1)<<8)+BIN___(x2))

#define BIN32(x1,x2,x3,x4) \
    ((BIN___(x1)<<24)+(BIN___(x2)<<16)+(BIN___(x3)<<8)+BIN___(x4))

int address_checkout(unsigned __int32 virtual_address, page_table* page_node, segment_table*
seg_node)
{
	segment_table* seg_tmp = seg_node;
	page_table* page_tmp = page_node;

	long seg_check = BIN32(011111111, 011100000, 000000000, 000000000);
	unsigned __int32 real_seg = (virtual_address & seg_check);
	real_seg = (real_seg >> 21);

	if ((0 > real_seg)||(real_seg >= 7))
	{
		printf("\n[Error]: Can't make real adress of [0x%.8X] because segment '%d'
dosen't exist!", virtual_address, real_seg);
		return virtual_address;
	}
	else
	{
		while (seg_tmp->next_seg)
		{
			if (seg_tmp->segment_number == real_seg)
				break;
			seg_tmp = seg_tmp->next_seg;
		}
	}

	page_tmp = parse_pages_table(seg_tmp->pages_table_path, seg_tmp);
	if (page_tmp == NULL)
	{
		printf("\n[Error]: Can't make real adress of [0x%.8X] because page dosen't exist
at this segment!", virtual_address);
		return virtual_address;
	}

	long page_check = BIN32(000000000, 000011111, 011111000, 000000000);
	unsigned __int32 real_page = (virtual_address & page_check);
	real_page = (real_page >> 11);
	if (real_page >= 16)
	{
		printf("\n[Error]: Can't make real adress of [0x%.8X] because page '%d' dosen't
exist!", virtual_address, real_page);
		return virtual_address;
	}
	else
	{
		while (page_tmp->next_page)
		{
			if (page_tmp->page_number == real_page)
				break;
			page_tmp = page_tmp->next_page;
		}
	}

	if (page_tmp->bit == 0)
	{
```

```
            printf("\n[Error]: Can't make real adress of [0x%.8X] because page '%d' empty!",
virtual_address, real_page);
            return virtual_address;
        }

        long page_shift = BIN32(000000000, 000000000, 000000111, 011111111);
        unsigned __int32 real_address = (virtual_address & page_shift);
        real_address += page_tmp->frame;
        return real_address;
}
```

Тести

```
1A 0 1 :
        // MOV R0 R1 //
1B 0 3 0080506F :
        // MOV R3 [0x00000079]  //
1B 1 7030B45A 4
[Error]: Can't make real adress of [0x7030B45A] because segment '897' dosen't exist!:
        // MOV [0x7030B45A] R4  //
01 2 1 :
        // ADD R2 R1  //
02 0 3 00209AE2
[Error]: Can't make real adress of [0x00209AE2] because page '19' dosen't exist!:
        // ADD R3 [0x00209AE2]  //
94 A2 :
        // JG A2  //
95 000000D4
[Error]: Can't make real adress of [0x000000D4] because page '0' empty!:
        // JG [0x000000D4]  //
80 2 0 :
        // CMP R2 R0  //
1C 1 2 30E5 :
        // MOV R2 30E5  //
```