הטכניון - מכון טכנולוגי לישראל
הפקולטה להנדסת חשמל

046209 - מבנה מערכות הפעלה

תכנון מערכת לגישה ותרגום זכרון וירטואלי

2017

תוכן עניינים

Table of Contents

1 מטרות התרגיל:

חידוד ההבנה של מנגנון הזכרון הוירטואלי תוך התנסות והיכרות עם המנגנונים השונים הקיימים במערכות מסוג זה באמצעות מימוש מערכת שמדמה זיכרון וירטואלי שכוללת מערכת תרגום כתובות (שבדרך כלל מבוצעת על ידי ה MMU בחומרה) ומערכת ניהול זיכרון (שבדרך כלל מבוצעת על ידי מערכת ההפעלה).

:MMU
אנחנו נממש את תהליך התרגום בתוכנה כדי להבין אותו באופן ייסודי כשבמרכזו נמצא תהליך ה Page Walk, על מנגנון ה TLB החלטנו לוותר לטובת פישוט התרגיל.
זיכרון וירטואלי:
נממש את כל טבלאות התרגום, ממשק זיכרון פיזי וממשק ל Swap Device.

על מנת לבדוק את נכונות המימוש, נממש תוכנית פשוטה שתשתמש במערכת זו, במקום בזיכרון הרגיל.
על מנת לעשות זאת נממש אובייקט חדש שייתנהג כמצביע מיוחד, נקרא לו OurPointer. בכל גישה לזיכרון דרך המצביע OurPointer, נפעיל את מנגנון התרגום אשר יבצע Page Walk בטבלת התרגום, יבצע SWAP אם צריך, ובסופו של תהליך יאפשר גישה לדף עם המידע.
אנו נדמה מערכת עם 256KB של זיכרון פיזי ו 4GB של זיכרון וירטואלי.

2 תכנון מבנה המערכת והכתובת

בתרגול ראינו את מבנה הכתובת במערכת ההפעלה Linux, אנו נממש מערכת זכרון המשתמשת במבנה הזכרון אשר נלמד בתרגול עבור מערכת 32-bit.
מספר הביטים בכתובת : 32

מבנה הכתובת :
Directory : 10 ביטים
Table : 10 ביטים
Offset : 12 ביטים
המחלקה של הטבלה (PageTable) תכיל מצביע(זהו המקביל לרגיסטר cr3 עליו למדנו) למערך של 1024 Page Directory Entries וכל Page Directory Entry יצביע למערך של 1024 Page Table Entries.

גודל מרחב הזכרון הוירטואלי : $2^{32}Bytes = 4GB$

מרחב הזכרון הזכרון הפיזי שלנו יוגדר על ידי 18 ביטים
מספר הדפים הנכנסים בזכרון הפיזי שלנו בזמן נתון: $2^{18} / 2^{12} = 2^6 = 64$
גודל מרחב הזכרון : $2^{18}Bytes = 256KB$

3 מבנה המערכת

המערכת שלנו תהיה בנויה ממספר מחלקות:
1. OurPointer – מחלקה שתשמש כמצביע שבו אנו משתמשים, בתרגיל זה נממש תמיכה בהצבעה למשתנים מסוג int בלבד.
2. PageTableEntry – כניסה בטבלה הפנימית
3. PageDirectoryEntry – כניסה בטבלה החיצונית
4. PageTable – טבלת הדפים.
5. SwapDevice
6. PhysMem – מחלקה עם Instance בודד שתספק ל VirtualMemory גישה לזכרון הפיזי.
7. VirtualMemory – המחלקה שתעטוף את כל הניהול של טבלת הדפים והזכרון הפיזי.

אתם תממשו כל אחת מבין מחלקות אלו במהלך תרגיל זה מלבד אחת – PhysMem אשר מימושה יסופק לכם במסמך זה. אנו נספק לכם קוד ראשוני למחלקות שתצטרכו לממש, באפשרותכם בכל שלב להוסיף פונקציות ומשתנים חברים למחלקות למען תקינות ועבודה נכונה של המערכת שלכם.

מחלקת PhysMem זו תכיל 256KB של זכרון אשר יהוו "הזכרון הפיזי" שלנו. המחלקה תכיל 2^6 מסגרות וכל אחת מהן תוכל להכיל את התוכן של אחד מתוך 2^{20} דפים של זכרון וירטואלי במרחב ה 32 ביט של זכרון וירטואלי.
הגישה לזכרון הפיזי שלנו תתרחש בצורה הבאה:

```
PhysMem::Access().GetFrame(FrameNumber) – Returns an int* pointer which is a
pointer to the base of the frame.
```

דוגמה:

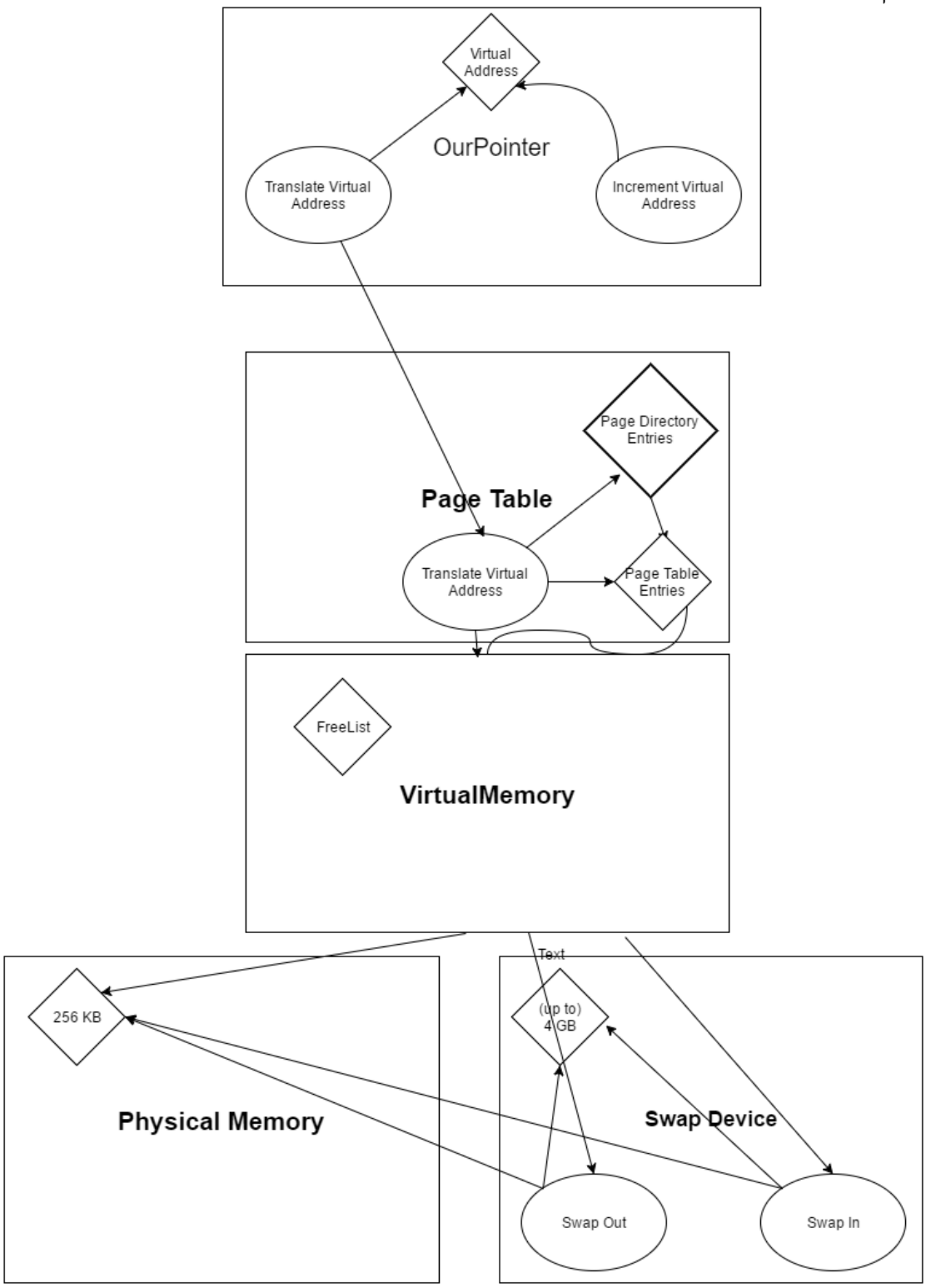
```
PhysMem::Access().GetFrame(1)[5]
PhysMem::Access().GetFrame(0)[6]
PhysMem::Access().GetFrame(0)[0]
PhysMem::Access().GetFrame(63)[1023] – The largest values we can access.
```

מימוש:

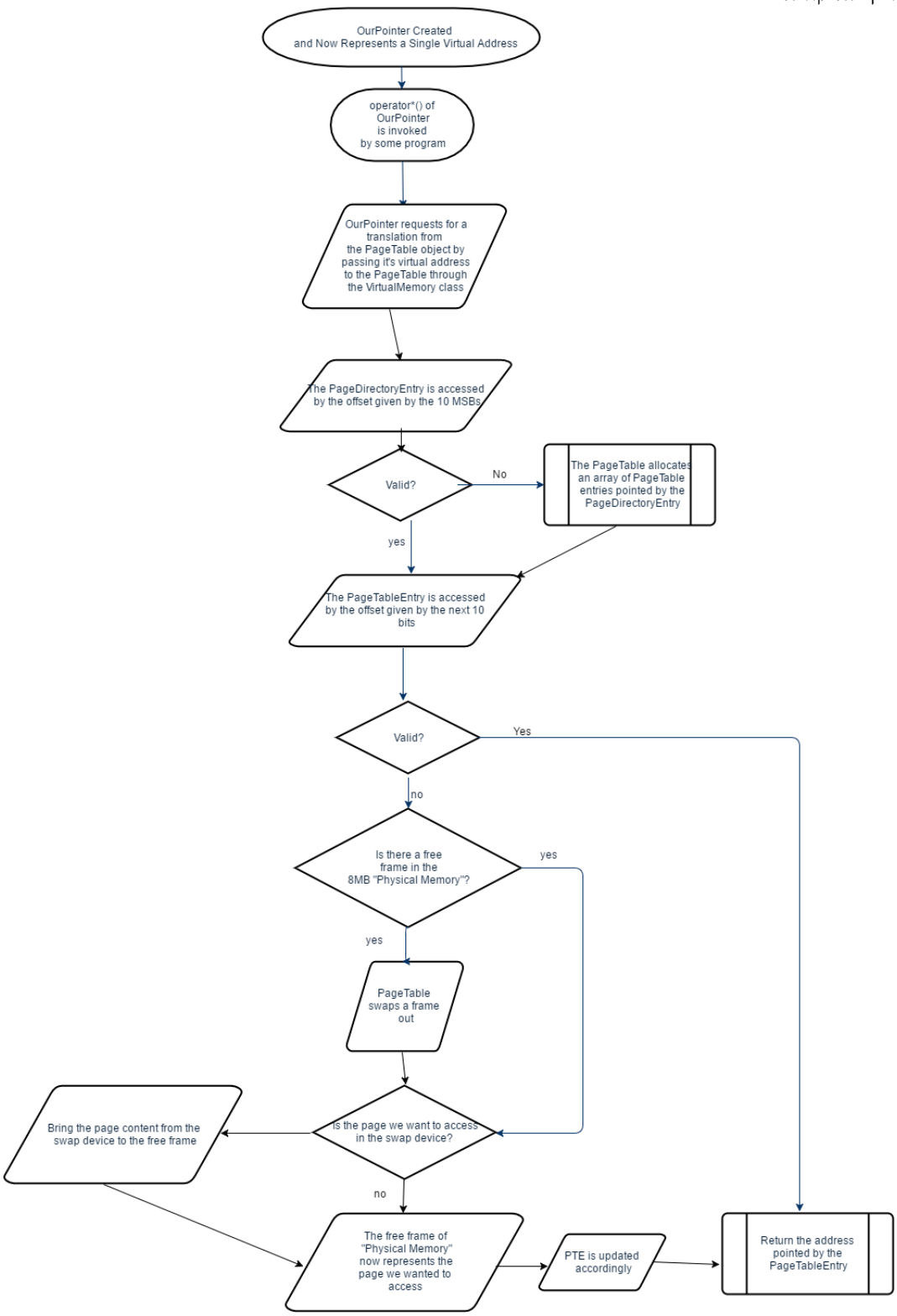
```
#pragma once

#include <stdlib.h>
#include <iostream>

using namespace std;

#define PHYSMEMSZ 262144

class PhysMem {
friend class VirtualMemory;
public:
	static PhysMem& Access();
	private:
		PhysMem() {}
		static int* physMem;

	public:
		PhysMem(PhysMem const&) = delete;
		void operator=(PhysMem const&) = delete;
		int* GetFrame(int frameNumber);
};
```

```
#include "PhysMem.h"

int* PhysMem::physMem;

PhysMem & PhysMem::Access() {
	static PhysMem single;
	if (physMem == NULL) {
		physMem = (int*)malloc(PHYSMEMSZ);
	}
	return single;
}

int* PhysMem::GetFrame(int frameNumber) {
	if (frameNumber < 0 || frameNumber >= 64)
		throw "Invalid Frame Number";
	return &(physMem[1024 * frameNumber]);
}
```

נוכל להסתכל על מערכת זו בצורה הבאה:
אליפסה – מתודה
מלבן – מודול
מעוין – מידע



כל התממשקות חיצונית עם המערכת תתרחש דרך המחלקה OurPointer, ניתן לקדם את הכתובת הוירטואלית עליו OurPointer מצביע או לתרגם את הכתובת הוירטואלית שהוא מכיל ולקבל גישה ישירה למשתנה עליו הוא מצביע.
כאשר נרצה לגשת לאזור בזכרון שמיוצג על ידי OurPointer (כתובתו הוירטואלית מצויה במשתנה בתוך OurPointer), המחלקה OurPointer תחזיר לנו רפרנס של טיפוס int שמקורו באזור מסויים ב "Physical Memory".
איך זה קורה?



3.1 השוואה למערכת שלמדנו בהרצאה ובתרגול

ישנם מספר הבדלים בין המערכת שלמדנו עליה למערכת זו, נציין את החמישה המהותיים ביותר.

- "הזכרון הפיזי" שלנו הוא בעצם זכרון וירטואלי, אך מכיוון והמקום שהוא מכיל הוא מוגבל, אנו מחלקים אותו למסגרות וכל מסגרת מייצגת כתובת וירטואלית אז מצב זה מדמה מערכת עם זכרון פיזי וזכרון וירטואלי.
- Swap Out מתרחש כשיש צורך להשתמש במסגרת ולא כפי שלמדנו שהמערכת מבצעת Swap Out כדי לשמור על אחוז מסויים של מסגרות פנויות בזכרון בכל רגע נתון.
- הטבלאות עצמן לא נשמרות "בזכרון הפיזי" שלנו אלא בזכרון של המחשב, בחרנו פתרון זה לשם פשטות כדי שלא נצטרך לנהל גם את המקום שהטבלאות תופסות.
- המערכת שלנו תומכת רק במשתנים מסוג int.
- אין TLB במערכת זו.

3.2 המצביעים איתם נעבוד בתכנית

כל מבני הנתונים והמחלקות אשר תידרשו לממש במהלך התרגיל מלבד תכנית הבדיקה עצמה ישתמשו במצביעים רגילים(ולא OurPointer אשר תממשו בתרגיל) מאחר ולצורך פשטות נניח בתרגיל זה כי מערכת ניהול ה Virtual Memory נשמרת בזכרון אחר אשר לא מנוהל על ידה.

---

4 VirtualMemory

מחלקה זו תשמש כמנהלת של שאר המערכות, דרכה נבצע OurMalloc שתהיה הוריאציה שלנו ל malloc ותחזיר מצביע מסוג OurPointer. מאחורי הקלעים Virtual Memory תנהל רשימה של מסגרות פנויות ותקצה אותן עבור טבלת דפים.
נשאר לכם לממש Free List עבור המסגרות הפנויות ואת בנאי המחלקה שיאתחל את אותה הרשימה, אשר צריכה להיות מנוהלת עם מדיניות FIFO לפי זמן ההקצאה (זמן הגישה האחרון לא ישפיע על מדיניות הפינוי כלל, רק הזמן שבו הבאנו את הדף לזכרון הפיזי בפעם הראשונה, זמן ההקצאה!), ניתן להשתמש ב Queue לטובת תחזוקת הרשימה, מדיניות FIFO היא חובה פה כדי לעבור את הטסטים כמו שצריך, בפרט יש לאתחל את הרשימה כך שאם נפעיל את GetFreeFrame() 64 פעמים נקבל בהתחלה מצביע למסגרת הראשונה, בפעם השנייה לשנייה וכך הלאה עד המסגרת ה 64 שנקבל בקריאה ה 64 כפי שהתקבל ממחלקת PhysMem כהתנהגות של תור פשוט, לאחר מכן זה יהיה עם מדיניות FIFO רגילה לפי סדר ההכנסה ל FreeList.

```
#pragma once
#include <queue>
#include "ourpointer.h"
#include "PhysMem.h"
#include "PageTable.h"
#define PAGESIZE 4096
#define VIRTMEMSIZE 4294967296
#define NUMOFFRAMES 64

using namespace std;

class VirtualMemory {
friend class PageTable;
public:
	VirtualMemory(); //Initialize freeFramesList to contain all 64 frame
pointers as given by PhysMem Class, initialize the PageTable, give the
pageTable a pointer to this object so it can utilize GetFreeFrame and
ReleaseFrame
	~VirtualMemory();
	int* GetFreeFrame(); //Remove one item from the freeFrameList and
return it – suggestion, use memset(framePtr, 0, PAGESIZE) before return,
might help debugging!
	void ReleaseFrame(int* framePointer); //releases the frame pointed by
the framePointer, make sure you only use this function with a pointer to the
beginning of the Frame! it should be the same pointer as held in the PTE.
	OurPointer OurMalloc(size_t size) { //allocates a pointer, we added the
code for your convenience
		if (allocated + size >= (VIRTMEMSIZE >> 2)) {
			throw "We are limited to 4294967296 bytes with our 32 bit
address size";
		}

		OurPointer ptr(allocated, this);
		allocated += size;
		return ptr;
	}

	int* GetPage(unsigned int adr) { return pageTable.GetPage(adr); }
private:
	size_t allocated; //The number of ints already allocated, ((allocated *
4) = (number of bytes already allocated)), this can also be used as the next
address to be allocated.
	queue<int*> freeFramesList;
	PageTable pageTable;
};
```

Implementing The Translation System 5

Page Table Entry 5.1

בחלק זה נממש Class בשם PageTableEntry.
מטרת מחלקה זו הינה לשמור את הכתובת הפיזית שבה מוחזק הדף ומידע שימושי נוסף עליו, השימוש במחלקה זו יתבצע באופן פנימי וממשקו לא יהיה חשוף החוצה לטובת שימוש במערכת.

המשיכו לממש את המחלקה תוך תמיכה בארבעת הפונקציות ששמנו ב header הבא:
(זוהי גרסה מפושטת של ה PTE, הרבה מהדגלים אשר למדנו לא קיימים פה ועל כן ויתרנו על פונקציונליות מסויימת מזאת שנלמדה)
קובץ h:

```
#pragma once

class PageTableEntry
{
public:
	//Your Constructor (and Destructor if you need one) should go here
	int* get_page_address(); //Pointer to beginning of frame
	void set_page_address(int* adr); //Set the pointer to a frame

	bool is_valid(); //Returns whether the entry is valid
	void set_valid(bool valid); //Allows to set whether the entry is valid


private:
	//Fill the class with the necessary member variables
};
```

5.2 Page Directory Entry

בחלק זה תממשו את המחלקה שהאובייקטים שלה יהיו ה Page Directory Entries.
חלק זה יהיה פחות מודרך מכיוון והוא דומה מאד לסעיף הקודם.
דגשים, טיפים ושאלה לפני המימוש:

- ניתן להשתמש בירושה.
- ניתן לחזור לחלק זה לאחר הסעיף שבו נממש את טבלת הדפים.
- במידה ואתם צריכים לאכסן באובייקט זה טבלה אחרת יש לממשה באמצעות מערך של איבריה וניתן להקצותו עם malloc פשוט, זה אומר שטבלאות אלו לא "גונבות לנו" מקום מתוך ה 256 קילו בייט שלנו.

5.3 Page Table

בחלק זה נתחיל לממש Class בשם Page Table.
מטרת מחלקה זו היא לתרגם כתובות וירטואליות ל"פיזיות", בפועל – לספק מצביע לבסיס של מסגרת שהדף שחיפשנו נמצא בתוכה.

```
int* GetPage(unsigned int adr);
```

לבינתיים נניח כי לא נשתמש ביותר דפים מהכמות ש"הזכרון הפיזי" יכול להכיל ועל כן אין צורך להשתמש ב Swap Device או לבחור מדיניות פינוי דפים.
דגשים, טיפים ושאלות לפני שמתחילים לממש:

- במידה ואתם צריכים לאכסן באובייקט זה טבלה יש לממשה בעזרת מערך פשוט של איבריה בדומה לסעיף הקודם.
- מה התהליך שמתרחש בפעם הראשונה שאנו ניגשים לדף מסויים? ניגשים ל Page Directory Entry לפי ה10 ביטים העליונים, מגלים כי הוא לא מצביע לשום מקום, מקצים Page Table, מזינים ל Page Table Entry הרלוונטי valid = 1 ומעדכנים את הכתובת שלו להצביע על כתובת של מסגרת פנויה שנקבל ממחלקת VirtualMemory, מחזירים את הכתובת המוצבעת על ידי ה PTE.
- מה התהליך שמתרחש כשאנו ניגשים לדף שכבר ניגשנו אליו? ניגשים ל Page Directory Entry לפי הביטים המתאימים בכתובת, ניגשים ל Page Table Entry לפי הביטים המתאימים, מקבלים מצביע ומחזירים אותו במידה והוא Valid (והוא יהיה Valid לבינתיים כי הנחנו שלא נשתמש ביותר מ256KB שהם גודל הזכרון הפיזי שלנו)
- על התכנית לייצר קובץ בשם log.csv אשר פרטיו מפורטים בסעיף האחרון במסמך זה, שדות אשר לא רלוונטיים עדין בחלק זה מכיוון והרכיב שהם תלויים בו טרם מומש צריכים להכיל 1-.

קובץ h:

```
#pragma once

#include “PageDirectoryEntry.h”

class VirtualMemory; //You will probably want to include this in PageTable.cpp
class PageTable
{
public:
	//Your Constructor (and Destructor if you need one) should go here
	int* GetPage (unsigned int adr);



private:
	//Fill the class with the necessary member variables
};
```

6 OurPointer

בסעיף זה נגדיר את ה Class שיהיה ה Pointer שלנו. מכיוון ובתרגיל זה נלמד על מימוש מערכת התרגום אז לא נרצה להשתמש במערכת התרגום הקיימת - אנו נשתמש במערכת תרגום שאנו ניצור, לכן גם נאלץ לממש מחלקה חדשה של מצביעים. מחלקה זו תתמוך רק במצביעים מסוג int לשם פשטות.
כדי לקבל מצביע שימושי תצטרכו לבצע Overload לפחות לאופרטורים הבאים:

- operator*()
- operator++()
- operator--()

מדוע צריך לגשת לטבלת הדפים בכל שימוש ב Pointer/Dereference Operator (*): המצביע יכיל כתובת וירטואלית, כדי להשתמש במידע עליו הוא מצביע יש צורך לתרגם כתובת זו למרחב הפיזי.

מדוע נשמור אך ורק את הכתובת הוירטואלית ולא כתובת פיזית במחלקה זו: זה המצביע שלנו אשר מצביע לכתובת במרחב הכתובות שלנו, באחריות מערכת התרגום למצוא ולנהל את המיקום של הכתובת ב"מרחב הפיזי" או אפילו מחוץ לה(Swap Device).

כדי לחסוך התעסקות בטפל מצורף מימוש של קובץ header בסיסי, השלימו אותו כראות עיניכם, שימו לב כי אין להשתמש במצביעים, מלבד המצביע ל VirtualMemory(ניתן גם להשאיר את הקובץ בדיוק כפי שמופיע פה):

```
#pragma once

#include "PageTable.h"

class OurPointer {
public:
	OurPointer(int adr, VirtualMemory* vrtlMem); //Constructor
	~OurPointer(); //Destructor
	int& operator*(); //Overload operator*
	OurPointer& operator++(); //Overload ++operator
	OurPointer operator++(int); //Overload operator++
	OurPointer& operator--(); //Overload operator--
	OurPointer operator--(int); //Overload --operator

private:
	unsigned int _adr; //the virtual address
	VirtualMemory* _vrtlMem; //for requesting translations
};
```

ממשו את OurPointer בצורה הבאה – בתוך OurPointer תישמר הכתובת הוירטואלית ובכל פעם שנרצה לכתוב או לקרוא את הערך עליו הוא מצביע נשתמש "באופרטור *" אשר יבצע שימוש ב PageTable לתרגום הכתובת עליו הוא מצביע.
כאשר נשתמש "באופרטור ++"של OurPointer נוסיף לכתובת הוירטואלית את המספר הבתים המתאים לפי המשתנה עליו אנו מצביעים, מומלץ להשתמש בפונקציה ()sizeof (למרות שבמקרה זה אנו בוחנים מקרה פשוט שבו המחלקה שלנו תומכת רק במשתנה מסוג יחיד ועל כן גם גודלו יחיד – 4 בתים).
הבנאי של OurPointer יקבל את הכתובת הוירטואלית עליו הוא מצביע ומצביע לאובייקט VirtualMemory.

7 שימוש ב OurPointer

בחלק זה נשתמש בטבלה שיצרנו, בעיקרון אין מגבלה על אופן השימוש בטבלה, אך אנו נבצע אפשרות מסויימת, מכפלת מטריצה בוקטור.

דגשים:

- אין לבצע malloc או להקצות זכרון מכל סוג שהוא (גם אסור להקצות מערכים).
- ניתן להקצות OurPointer יחיד בצורה סטטית לכל מבנה נתונים שנשתמש בו (אחד לכל מטריצה), אין להשתמש במצביעים שאינם מסוג OurPointer.
- ניתן להשתמש ב OurPointer גם עבור האינדקסים של הלולאות אך אין צורך.

שאלות(לא צריך לענות):

- לכמה זכרון התכנית שלנו מוגבלת בשלב זה של התרגיל? מה יקרה אם נחרוג מכמות זו? האם שימוש בזכרון וירטואלי במצב הנוכחי מהווה יתרון?
- האם הוספת תמיכה ב Swap Device תאפשר לנו להשתמש ביותר זכרון? בכמה זכרון נוכל להשתמש במקרה זה?

בסעיף זה כתבנו תכנית המכפילה מטריצה בגודל 50 על 50 בוקטור באורך 50 ומדפיסה את המטריצה, את הוקטור ואת וקטור התוצאה.

- דוגמה ל Output עבור מטריצה 3 על 3:

```
Matrix:
1 2 3
4 5 6
7 8 9
Vector:
10 11 12
Multiplication:
68 167 266
```

```
#include <iostream>
#include <fstream>
#include “VirtualMemory.h”

#define VECSIZE 50

using namespace std;

int main(){
    VirtualMemory vrtlMem;

    OurPointer matBase = vrtlMem.OurMalloc(VECSIZE*VECSIZE);
    OurPointer vecBase = vrtlMem.OurMalloc(VECSIZE);
    OurPointer resBase = vrtlMem.OurMalloc(VECSIZE);

    OurPointer mat = matBase;
    OurPointer vec = vecBase;
    OurPointer res = resBase;

    ofstream matricesFile;
    matricesFile.open("matrices.txt");

    srand(1);
    for (int i = 0; i < VECSIZE * VECSIZE; ++i) {
        *(mat++) = rand() % 100;
    }
    mat = matBase; for (int i = 0; i < VECSIZE; ++i) {
        *(vec++) = rand() % 20000;
        *(res++) = 0;
    }
    vec = vecBase;
    res = resBase;
    for (int row = 0; row < VECSIZE; ++row) {
        for (int col = 0; col < VECSIZE; ++col) {
            //*res += (*(mat++)) * (*(vec++)); //it works, but accesses to operator*
might happen in an unspecified order
            int mat_value = *(mat++);  //our fix for log
            int vec_value = *(vec++); //log file consistency
            *res += mat_value * vec_value;
        }
        vec = vecBase;
        ++res;
    }
    mat = matBase;
    vec = vecBase;
    res = resBase;

    matricesFile << "Matrix:" << endl;
    for (int row = 0; row < VECSIZE; ++row) {
        for (int col = 0; col < VECSIZE; ++col)
            matricesFile << *(mat++) << " ";
        matricesFile << endl;
    }

    matricesFile << "Vector:" << endl;
    for (int col = 0; col < VECSIZE; ++col)
        matricesFile << *(vec++) << " ";
    matricesFile << endl;

    matricesFile << "Result:" << endl;
    for (int col = 0; col < VECSIZE; ++col)
        matricesFile << *(res++) << " ";
    matricesFile << endl;
}
```

מומלץ לבדוק כי התכנית מבצעת מכפלה תקינה, סיפקנו עבורכם קופץ output של תכנית זו לדוגמה, בנוסף גם סיפקנו log.csv שהמערכת שכתבתם אמורה לייצר בעת ריצת תכנית זו:

- Testprog1.matrices.txt
- Testprog1.log.csv

מומלץ לבדוק כי התכנית שלכם אכן חישבה והדפיסה תוצאות נכונות מבחינת matrices.txt.
בנוסף תבצעו diff עם קובץ ה csv הזה וה log file שלכם(מכיוון ומדובר על פעולה שמבוססת על rand, למרות שה seed זהה ייתכן ונקבל מטריצות שונות על מערכות שונות, לכן אם diff על matrices.txt לא יספיק, ניתן לבדוק את התקינות הקובץ גם ידנית, הוספנו נספח בסוף שמראה דוגמה כיצד ניתן לבצע זאת).

8 הרחבת הטבלה

8.1 *Swap Device*

כעת נרצה לפתור את המגבלות לגבי גודל הזכרון במימוש הנוכחי נבצע זאת באמצעות מנגנון החלפת דפים.

נממש Swap Device פשוט בצורה הבאה:

```
#pragma once

#include <unordered_map>
#include <vector>
#include <string.h>
#include <iostream>

#define PAGESIZE 4096
class SwapDevice
{
    public:
        SwapDevice() : _size(0) {}
        void WriteFrameToSwapDevice(int pageNumber, int* pageOut); //Write
the content of page to the swap device, "pageOut" is the frame base
pointer where the page is now allocated
        int ReadFrameFromSwapDevice(int pageNumber, int* pageIn); //Put
the content of the page in "page". " pageIn " is the frame base pointer to
where the page is about to be allocated, returns -1 if page is not stored
in the swap device.
    private:
        std::unordered_map<int, int*> _data;
        size_t _size;
};
```

```
#include "SwapDevice.h"

using namespace std;

void SwapDevice::WriteFrameToSwapDevice(int pageNumber, int* pageOut) {
    if (_data[pageNumber] == NULL) {
        if (++_size > 1048576) {
            cerr << "The swap device seems to be using too much space,
worth investigating" << endl;
        }
        _data[pageNumber] = (int*)malloc(PAGESIZE);
    }
    memcpy(_data[pageNumber], pageOut, PAGESIZE);
}

int SwapDevice::ReadFrameFromSwapDevice(int pageNumber, int* pageIn) {
    if (_data[pageNumber] == NULL) {
        return -1;
    }
    memcpy(pageIn, _data[pageNumber], PAGESIZE);
    return 0;
}
```

שנו את המחלקות השונות כך שיתאפשר לבצע Swap In ו Swap Out, לאפשר גישה לדפים שלא נמצאים בזכרון ולהשתמש בזכרון יותר גדול מאשר הזכרון הפיזי שהגדרנו.

שימו לב:

- כאשר ננסה לגשת לדף אשר לא קיים בזכרון הפיזי נקבל Page Fault ומחלקת טבלת הדפים תצטרך לבקש מסגרת פנויה ממחלקת VirtualMemory, במידה ולא קיימת אחת פנויה נאלץ לשחרר מסגרת אחרת, בסעיף הבא נדבר על מדיניות פינוי.
- צריך לטפל במקרה שבו כתובת נקראת בפעם הראשונה, ניתן לבצע שינוי בטבלת הדפים ולהוסיף דגל ב PTE דוגמה, בכל מקרה עבור מקרה זה יש להדפיס לקובץ ה Log כי לא התבצעה פעולה ב Swap Device.
- עליכם לאתחל את SwapDevice בתוך המחלקה VirtualMemory אשר מנהלת את מרחב הזכרון.

8.1.1 מדיניות פינוי

כעת נוסיף תמיכה במדיניות הפינוי FIFO, הוסיפו למחלקת ה queue Page Table אשר יאפשר לעקוב אחרי הסדר שבו הקצנו את המסגרות לדפים.
דאגו כי בעת פינוי הדף נפנה את הדף שהובא לזכרון הפיזי הכי מוקדם, אין הגבלה לאופן המימוש, מומלץ להשתמש ב queue.

- נסו את תכנית הבדיקה שכתבנו בשנית, הגדילו את המטריצה כך שלא תיכנס כולה בזכרון הפיזי (לדוגמה, מטריצה בגודל 300 X 300), שמרו עותק של התכנית המקורית בקובץ testProg1.cpp, לתכנית החדשה קראו testProg2.cpp
  - ודאו כי גם testProg1 עדיין פועלת כראוי אפילו שלא אמור להתבצע Swap לאף דף ועל כן לא אמור להתרחש בשינוי בתפקוד testProg1.
  - בדקו כי הקבצים testprog2.matrices.txt ו testprog2.log.csv תואמים, יש לבצע diff עם testprog2.log.csv ולתקן את התכנית בהתאם.

9 Log File

עליכם לבצע כתיבה ל Log File בכל פעם שמתבצע תרגום לכתובת.

- פורמט הקובץ יהיה CSV, מה שאומר שהשדות השונים יופרדו על ידי ','.
- הקובץ יכיל 7 שדות:
  - Page Number – מספר הדף שאליו אנו מנסים לגשת. (הכתובת חלקי 4096)
  - Virtual Address – הכתובת הוירטואלית אליה ניסינו לגשת.
  - Physical Address – הכתובת הפיזית שתורגמה, שימו לב כי אם לא תחסרו את ערך המצביע של ה Frame הראשון מכל הכתובות האלו תקבלו מספרים שונים בשדה זה.
  - Page Fault – האם התרחש Page Fault בתהליך התרגום?
  - Swap – האם היה שימוש ב Swap Device?
  - Evicted – במידה והתבצע Swap מה מספר הדף שביצענו לו Swap Out
  - Allocated Page Table Entries – האם הוקצה מקום עבור Page Table Entries חדשים(גישה ל Page Directory Entry חדש).
- בדקו מול 2 קבצי ה csv לדוגמה שסיפקנו לכם כי אתם מדפיסים כנדרש.

10 הגשה

יש להגיש את כל הקבצים הדרושים לטובת קימפול התכנית כולל Makefile כולל החלקים המסופקים לכם, מצורפת רשימה של קבצים שכנראה יופיעו אצלכם, כדאי לבדוק :

PageTable.h
PageTable.cpp
SwapDevice.h
SwapDevice.cpp
OurPointer.h
OurPointer.cpp
PageDirectoryEntry.cpp
PageDirectoryEntry.h
PageTableEntry.cpp
PageTableEntry.h
VirtualMemory.cpp
VirtualMemory.h
PhysMem.cpp
PhysMem.h

על ה Makefile לקמפל את התכנית שלכם כאשר פונקציית ה main נמצאת בקובץ main.cpp וקובץ ההרצה שיווצר יקרא "main".

מומלץ להריץ טסטים נוספים שיבדקו גם גישה יותר "רנדומלית" ולא מסודרת לזכרון שתאפשר לבדוק את כל מרכיבי המערכת.

11 נספח – בדיקה ידנית של נכונות ההכפלה

בנספח זה נציג רעיון אפשרי לטובת ביצוע בדיקה ידנית כי כפל המטריצות התבצע כראוי בעזרת קוד מתלב קצר והתוכן של matrices.txt:
ראשית צריך לפתוח מתלב, למי שאין גישה למתלב ניתן לבצע זאת בקלות בעזרת ממשק אינטרנטי, חיפוש בגוגל "matlab online" מספק תוצאות טובות.

נדגים באמצעות מטריצה 3x3:
קובץ ה matrices.txt:

Matrix:
41 67 34
0 69 24
78 58 62
Vector:
4464 5705 8145
Result:
842189 589125 1184072

נבצע copy לשורות 2 עד 4 ונכתוב במתלב:;mat = ["ctrl + v"]
לאחר מכן נבצע copy לשורה 6 ונכתוב במתלב: vec = ["ctrl + v"];
לבסוף נבצע copy לשורה האחרונה ונכתוב במתלב expected_res = ["ctrl + v"];
שימו לב כי עליכם להקליד את הסוגריים המרובעים, לאחר מכן לבצע הדבק ולהקליד עוד סוגריים מרובעים.

כעת כל מה שנותר לעשות הוא:
res = mat * vec.';
isequal(expected_res,res.')

ולוודא שהפונקציה החזירה 1.

זה נראה כך:

```
octave:13> res = mat * vec.';
octave:14> isequal(expected_res,res.')
ans =  1
octave:15> mat = [41 67 34
0 69 24
78 58 62 ];
octave:16> vec = [4464 5705 8145 ];
octave:17> expected_res = [842189 589125 1184072];
octave:18> isequal(expected_res,res.')
ans =  1
```