תרגיל 5 – Virtual Memory Management

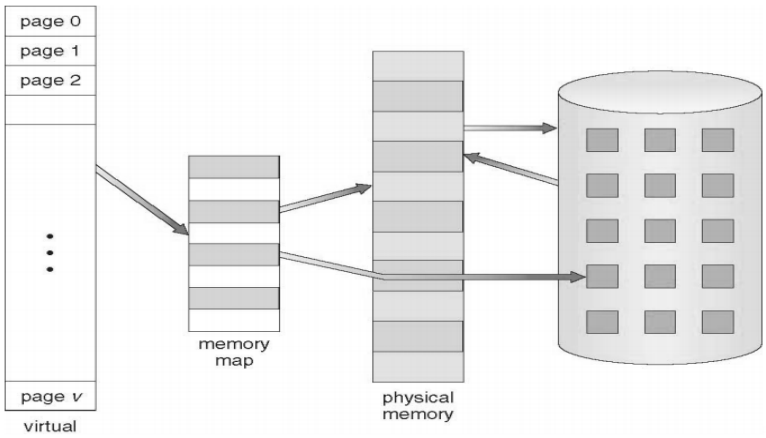
הגשה :12.6.2022

<u>תיאור המערכת</u>

סימולציה זוהי סביבת הדמיה בתוכנה לאירועים ופעולות הקורים במערכת אמתית (חמרה או תכנה.)

בתרגיל זה נממש סימולציה של גישות המעבד לזיכרון. אנו משתמשים במנגנון ה-paging המאפשר להריץ תוכניות כאשר רק חלק ממנה מצוי בזיכרון. זיכרון התכנית (נקרא גם זיכרון וירטואלי) מחולק לדפים אשר נטענים לזיכרון הראשי על פי הצורך.

אנחנו נממש זיכרון ווירטואלי של מחשב עם עד שתי תוכניות שיכולות לרוץ במקביל. מערכת מיפוי הזיכרון בתרגיל מודגמת באיור שלהלן:



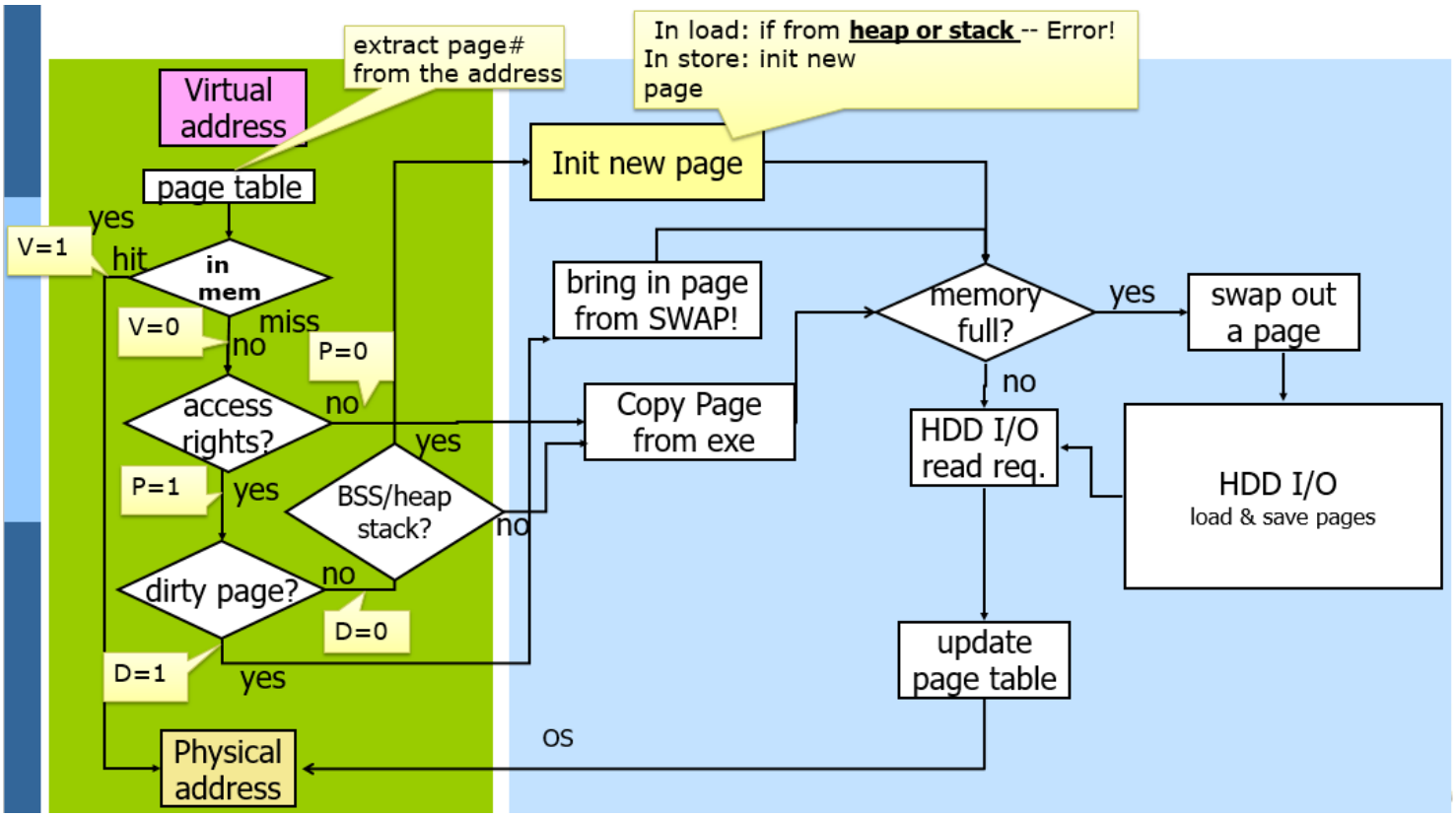
פונקציית ה-main של התרגיל תהיה מורכבת מרצף פקודות load ו-store (אקראיות), **פונקציית אלו מדמות את קריאה\כתיבה של המעבד [פעולות המעבד מורכבת מפעולות קריאה , כתיבה וחישוב\עיבוד – אנו נתרכז רק בקריאה וכתיבה לצורך דימוי פעולות הזיכרון במחשב].** אפשר לראות דוגמא לmain כזה בסוף התרגיל. החלק הארי של התרגיל יעסוק במימוש הפקודות load ו-store דרך טבלת דפים הממפה את הדפים הלוגיים לפיזיים

התרגיל ייכתב ב c++ . מעבר לconstructor ו destructor, הפונקציות העיקריות הן -

<u>הפונקציה load:</u>
מקבלת כתובת לוגית אליה יש לגשת לצורך קריאה של נתון. בעיקרה, הפונקציה דואגת **<u>שהדף</u>** הרלוונטי של התהליך המבוקש תהיה בזיכרון הראשי.

<u>הפונקציה store:</u>
מקבלת כתובת אליה יש לגשת לצורך כתיבה של נתון. בדומה לload, יש לדאוג הדף של הכתובת המדוברת עבור התהליך הנתון יהיה בזיכרון. .

תרשים הזרימה של התוכנית, כפי שראינו בשיעור (המדגים את ההתנהגות מנקודת מבט של תהליך אחד):



הסבר התרשים, ההסבר רלוונטי עבור תהליך אחד (ובאינדוקציה עבור שני תהליכים ☺).

- עבור כל כתובת שנקבל ראשית יש לבצע המרה מכתובת וירטואלית לכתובת פיזית:
  - כתובת וירטואלית מורכבת ממספר ה-page + ההיסט בתוך הדף, ולכן בהינתן כתובת וירטואלית נרצה לזהות קודם כל לאיזה page שייכת הכתובת ומהו ההיסט.
  - טבלת הדפים תחזיר לנו עבור ה-page שמצאנו מהו ה-frame המתאים
- אם הדף כבר נמצא בזיכרון הראשי, לבדוק בטבלת הדפים (הספציפית של התהליך) וכך נוכל לגשת ל- frame המתאים בזיכרון הראשי ולהתקדם בו על פי ההיסט לצורך קריאה או כתיבה.
- אם הדף לא נמצא בזיכרון הראשי, יש להביא אותו מהמקום המתאים. כאן יתכנו מספר אפשרויות:
  - יש לבדוק האם זה דף בעל הרשאות קריאה בלבד (text) או כשניתן לכתיבה (data+bss+ stack +heap)
    - במקרה שאין הרשאות כתיבה
      - ומדובר <u>בפעולת load</u>, אזי הדף נמצא בקובץ הexecutable. (מדוע ?)
      - אם מדובר <u>בפעולת store</u>, הפונקציה תדפיס הודעה שגיאה מתאימה ותחזור (התכנית תמשיך).
    - במקרה שיש הרשאות כתיבה, כלומר מדובר בדף מסוג data, heap, stack או bss אז יש לבדוק:
      - אם הדף "dirty" – כלומר נכתבו עליו דברים ונעשו בו שינויים (במילים אחרות, הייתה כבר קריאה עם store לדף זה)
        - אם כן אז אזי הדף נמצא בקובץ הswap.
        - אם לא, יש לבדוק האם מדובר בדף של heap, stack, bss או data
          - אם heap, stack, bss נקצה דף חדש וריק – דף שכולו 0. (כאן מדמים מצב של malloc או קריאה לפונקציה)
          - אחרת, כתובת data נקרא מהקובץ
    - בהסתייגות אחת – יש להבדיל בין heap, stack ו bss במקרה של load
      - אם נגשנו לדף מסוג heap, stack, לא ניתן לבצע ממנו פעם ראשונה load – אחרת זאת תהיה שגיאה! במילים אחרות , כאשר עושים load דף מסוג heap, stack, חייב להיות במצב V=1 או D=1 – אחרת זאת תהיה שגיאה! (שגיאה זו שקולה למצב בו נבצע קריאה מהזיכרון לפני שעשינו malloc)

מבני הנתונים של המערכת

.1

```
#define MEMORY_SIZE 200
extern char main_memory[MEMORY_SIZE];
```

זיכרון של 200 תווים

.2

class sim_mem : המחלקה הראשית של התכנית. יוגדר בקובץ ה-h של התכנית שלכם. מכיל את השדות הבאים:

- swap_fd – ה-file descriptor של קובץ ההחלפה, אליו נפנה דפים מהזיכרון הראשי (מתקבל על ידי open של הקובץ). קובץ כזה קיים רק אחד במערכת גם במקרה של שני תהלכים.
- program_fd – מערך בגודל של 2. ה-file descriptor של קובץ התכנית אותה אנו "מריצים" במערכת ובה יושבים הנתונים (מתקבל על ידי open של הקובץ).. במקרה של תהליך בודד נישתמש רק בתא הראשון של המערך...
- .... שאר השדות ראה הסבר בחתימה של ה constructor (בעמוד הבא)
- page_table — מצביע כפול שיכול לשמשת להקצעת מערך של עד שני טבלאות דפים של המערכת (עבור שני התהליכים שניתן להריץ במערכת).

```
class sim_mem {
    int swapfile_fd;                                    //swap file fd
    int program_fd[2];                                     //executable file fd
    int text_size;
    int data_size;
    int bss_size;
    int heap_stack_size;
```

```
    int num_of_pages;
    int page_size;
    int num_of_proc;

    page_descriptor **page_table;       //pointer to page table

    public:
         sim_mem::sim_mem(char exe_file_name1[], char swap_file_name2[], int text_size,
                          int data_size, int bss_size, int heap_stack_size,
                          int num_of_pages, int page_size, int num_of_process)
        ~sim_mem();

        char load(int process_id, int address);
        void store(int process_id, int address, char value);

        void print_memory();
        void print_swap ();
        void print_page_table();
```

את טבלת הדפים (page_table) נממש כמערך מטיפוס `page_descriptor` בגודל NUM_OF_PAGES כדלקמן:

```
typedef struct page_descriptor
{
      int V; // valid
      int D; // dirty
      int P; // permission
      int frame; //the number of a frame if in case it is page-mapped
      int swap_index; // where the page is located in the swap file.

} page_descriptor;
```

<u>הפונקציות שאתם נדרשים לכתוב</u>

1) ctor – מאתחלת את סביבת העבודה

```
sim_mem::sim_mem(char exe_file_name1[],char exe_file_name2[], char swap_file_name[], int
text_size,  int data_size, int bss_size, int heap_stack_size, int num_of_pages, int
page_size, int num_of_process)
```

הפונקציה מקבלת

1. exe_file_name1 - מחרוזת - שם קובץ executable של תהליך ראשון להרצה
2. exe_file_name2 - מחרוזת - שם קובץ executable של תהליך שני. (במקרה של תהליך אחד תשלח כאן מחרוזת ריקה)
3. swap_file_name -מחרוזת - שם קובץ swap
4. text_size גודל האזור הטקסט של התוכנית בבתים
5. data_size - גודל איזור הdata של התוכנית בבתים
6. bss_size - גודל איזור הbss של התוכנית בבתים
7. heap_stack_size - גודל איזור הheap וה stack של התוכנית בבתים
8. num_of_pages – מספר הדפים בזיכרון הוירטואלי
9. page_size - גודל דף (וגם frame) במערכת
10. -int num_of_proces , - מספר התהליכים במערכת יכול להיות אחד או שנים

על הפונקציה:

1. לאתחל את כל מבני הנתונים של המערכת
2. לפתוח את הקבצים exe_file_name ו exe_file_name1 <-- שימו לב! במקרה של 2 תהליכים יש לפתוח שני קבצים ו exe_file_name2
3. לאתחל את הזיכרון הראשי באפסים
ולהקצות ולאתחל את מערך ה page_table (לדוגמא: frame = -1, valid = 0). במקרה של שתי תהליכים לאתחל מערך בגודל שנים
4. את קובץ הSWAP יש לאתחל ב0 לפי גודל הזיכרון הלוגי (num_of_pages-text_pages)*page_size תווים)

ניתן להניח כי הגדלים המתקבלים הם תקינים: מספרים חיוביים שסכומם קטן מגודל מרחב הזיכרון. לא ניתן להניח ששם הקובץ אינו NULL, שהקובץ קיים, או שיש לכם הרשאות אליו (ראו בהמשך מה לעשות במקרה של שגיאה).

- אם הקובץ SWAP לא קיים יש ליצור אותו.
- אם הexecutable לא קיים – יש לצאת מהתוכנית (הדפסת שגיאה וexit)

2) load – מנסה לגשת לכתובת מסוימת לצורך קריאה

```
char sim_mem::load(int process_id,  int address)
```

המטרה: לדאוג לכך שהדף המתאים (לכתובת address) של התהליך המתאים (תהליך אחד או שנים) אכן יישב בזיכרון הראשי (ע"פ התרשים לעיל) ואז יש לגשת אל הכתובת הפיזית בזיכרון ולהחזיר את התו היושב בכתובת זו.
במקרה של שגיאה יש להחזיר 0\

שימו לב! לא טוענים לזיכרון הראשי רק את הכתובת המבוקשת (במקרה שלנו תו יחיד) אלא דף שלם כלומר PAGE_SIZE תווים.

3) store – מנסה לגשת לכתובת מסוימת לצורך כתיבה

```
void sim_mem::store(int process_id, int address, char value)
```

המטרה: לדאוג לכך שהדף המתאים (לכתובת address) של התהליך המתאים (תהליך אחד או שנים) אכן יועתק לזיכרון הראשי. אחרי שהדף הועתק לגשת לכתובת הפיזית ולאחסן את התו value בכתובת זו.

4) print_memory – תדפיס את תכולת הזיכרון הראשי – מימוש ניתן למצוא בסוף הקובץ

```
void sim_mem::print_memory();
```

הפונקציה מדפיסה את תוכן הזיכרון הראשי.

5) הדפסת הsawp – מימוש ניתן למצוא בסוף הקובץ

```
void sim_mem::print_swap();
```

הפונקציה מדפיסה את תוכן קובץ הswap.

6) הדפסת הpage_table – מימוש ניתן למצוא בסוף הקובץ

```
void sim_mem::print_page_table();
```

7) destructor

```
void  sim_mem::~sim_mem();
```

סוגרת את הקבצים הפתוחים, משחררת את הזיכרון

ניתן ורצוי להוסיף פונקציות עזר כרצונכם במקרה הצורך.

דוגמה אפשרית לתוכנית הראשית:

```
char main_memory[MEMORY_SIZE];

int main()
{
    char val;
    sim_mem mem_sm("exec_file", "",  "swap_file" ,25, 50, 25,25, 25, 5,1);
    mem_sm.store(1, 98,'X');
    val = mem_sm.load (1, 98);
    mem_sm.print_memory();
    mem_sm.print_swap();
}
```

הערות נוספות על המימוש:

- באתחול המערכת אין לטעון אף אחד מן הדפים מהקובץ לזיכרון. טעינת הדפים תתבצע באופן "lazy", כלומר על פי דרישה **(demand paging).**
- **שימו לב, ש**על דפי הtext לא ניתן לכתוב! ולכן אף פעם יהיה צורך "לגבות שינויים בהם " כלומר , לא יהיה צורך לכתוב אותם ל-swap, ותמיד ניתן יהיה לקרוא אותם מחדש מקובץ הexecutable במידת הצורך.
- דפים נכתבים לswap לפי first-fit **במקום הראשון הפנוי. כאשר דף נטען מה swap לזיכרון. יש למחוק (לאפס) את התוכן שלו בקובץ ה swap. (לאפסים). [[שאלה למחשבה: זה באמת חכם לאפס ?]]**
- בחירת דף לפנוי מהזיכרון כאשר הזיכרון מלא -- **בעזרת אלג' FIFO !**
- אסור לשנות את החתימות של הפונקציות
- **נקודה למחשבה ? איך נדע איזה פריים פנוי בזיכרון (רמז: כנראה צריך מערך נוסף?!?)**
- המערכת שלנו מריצה תהליך בודד או שנים. תהליך(ים) אלו ""ייטענו"" למערכת מתוך קובץ executable ששמו מועבר כארגומנט לתוכנית ונשמר במשתנה מהטיפוס sim_mem. (יכולים כאמור להיות עד שני executable)
- **מטעמי פשטות קובץ זה לא יהיה קובץ executable סטנדרטי המכיל פקודות מכונה ממשיות – אבל אנו נתייחס כאילו הוא כזה (שימו לב שאתם מבינים מה המשפט הזה אומר, הבנתו היא קריטית להבנת התרגיל)**

הערות כלליות:

יש לבדוק את ערכי החזרה של כל הפונקציות שעשויות להיכשל - פתיחה/קריאה/כתיבה לקבצים,
עבור כל שגיאה כנ"ל שנתקלים בה במהלך הריצה יש להדפיס הודעת שגיאה מתאימה (מומלץ להיעזר ב-perror) ולהחזיר מהפונקציה את ערך השגיאה שלה.
במקרה של חזרה לא מתוכננת כזו מפונקציה – אל תשכחו לשחרר את המשאבים שהקציתם עד כה במהלך קריאה זו.

הנחיות הגשה:

יש להגיש את הקבצים הבאים:
sim_mem.h – חתימות הפונקציות אשר לעיל
sim_mem.cpp – המימושים של הפונקציות.
main.cpp – תוכנית ראשית קטנה (צרו כמה קריאות שמירה\קריאה לזכרון)
בדקו מה קורה שהזיכרון מתמלא, שומרים וכותבים לאותו מקום וכו'
README – הסבר על המערכת, הקבצים, הפונקציות ואופן הקמפול וההרצה

בהצלחה!

```
/*****************************************************************************/
void sim_mem::print_memory() {
      int i;
      printf("\n Physical memory\n");
      for(i = 0; i < MEMORY_SIZE; i++) {
            printf("[%c]\n", main_memory[i]);
      }
}

/*****************************************************************************/

void sim_mem::print_swap() {
      char* str = malloc(this->page_size *sizeof(char));
      int i;
      printf("\n Swap memory\n");
      lseek(swapfile_fd, 0, SEEK_SET); // go to the start of the file
      while(read(swapfile_fd, str, this->page_size) == this->page_size) {
            for(i = 0; i < page_size; i++) {
                  printf("%d - [%c]\t", i, str[i]);
            }

            printf("\n");
      }
}

/*****************************************************************************/
void sim_mem::print_page_table() {
      int i;
      for (int j = 0; j < num_of_proc; j++) {
          printf("\n page table of process: %d \n", j);
          printf("Valid\t Dirty\t Permission \t Frame\t Swap index\n");
          for(i = 0; i < num_of_pages; i++) {
            printf("[%d]\t[%d]\t[%d]\t[%d]\t[%d]\n",
                        page_table[j][i].V,
                        page_table[j][i].D,
                        page_table[j][i].P,
                        page_table[j][i].frame ,
                        page_table[j][i].swap_index);
          }
      }

}
```