מערכות הפעלה 31261

פתרון מבחן סופי, מועד ב', סמסטר ב' תשע"ג, 06/08/2013

שאלה 1 [10%]

א. תאר בקצרה את מנגנון הפסיקה (Interrupt) במערכת הפעלה מודרנית (באופן כללי)

ב. תן שלוש דוגמאות לשלושה סוגי פסיקות נפוצות

תשובה:

א. פסיקה (interrupt) היא מנגנון המאפשר למערכת ההפעלה לעצור תהליך במהלך עבודתו ולטפל באירועים דחופים, אירועים חריגים, או לאפשר לתהליכים אחרים להשתמש ביחידת העיבוד בכדי לחלק את זמן העיבוד בצורה הוגנת בין כל התהליכים והמשתמשים. פסיקה מתבצעת בדרך כלל על ידי סיגנל חומרה (כגון שעון או יחידת ניהול פסיקות חומרה) שמגיע למעבד (אך קיימות גם פסיקות תוכנה). מערכת ההפעלה תשמור את המצב הנוכחי של התהליך שרץ באותו רגע בטבלת התהליכים (לכל תהליך רשומת PCB מתאימה) ותעביר את השליטה לשגרת טיפול בפסיקה (interrupt handler) או בשם אחר ISR (interrupt service routine). שיגרת הטיפול בפסיקה נקבעת על פי מספר הפסיקה ומצביע מתאים בווקטור הפסיקות (interrupt vector) שמצביע על המקום בזיכרון שבו נמצאת שגרת הטיפול. וקטור הפסיקות ושגרות הפסיקות הם חלק ממערכת ההפעלה.
שמירת המצב של התהליך המושעה כוללת שמירת התוכן של כל האוגרים (registers) והמצביע לפקודה הבאה שצריכה להתבצע. כל המידע הזה (בצרוף מידע על קבצים פתוחים ועוד) שמור ברשומת ה-PCB של התהליך. לאחר הטיפול באירוע החריג או כשהגיע תורו של התהליך לרוץ שוב, מערכת ההפעלה תוכל לשחזר את כל המידע הזה מתוך רשומת ה-PCB לאוגרים של המעבד, ולחדש את פעולת תהליך מהנקודה שבה עצר.

ב. דוגמא נפוצה לפסיקת חומרה היא פסיקת מקלדת או פסיקה הנגרמת משימוש בעכבר: מערכת ההפעלה עוצרת את פעולת התהליך הנוכחי ומטפלת בקלט שהגיע מהמקלדת או העכבר. דוגמא שניה לפסיקת חומרה היא פסיקת שעון (programmable interrupt timer) המתבצעת באופן סדיר בכדי לעצור את התהליך הנוכחי לאחר שסיים את מנת הזמן שהוקצתה לו על ידי מערכת ההפעלה ולהעביר את השליטה לתהליך הבא בתור (או לטפל בעניינים אחרים). דוגמא שלישית לפסיקות חומרה היא פסיקה המגיעה ממדפסת או דיסק המוכנים לפעולה. קיימות גם פסיקות תוכנה כגון חילוק באפס (divide by zero), חריגת סגמנט (segmentation fault), חריגת דף (page fault), או גלישה נומרית (integer overflow) אשר גורמות למעבד לעצור את פעולת התהליך ולהעביר את השליטה למערכת ההפעלה.

שאלה 2 [10%]

א. תאר בקצרה מהן קריאות שרות (system calls) במערכת הפעלה מודרנית?

ב. תן שלוש דוגמאות לקריאות שרות במערכת ההפעלה Unix והסבר את תפקידיהן.

ג. תן שלוש דוגמאות לקריאות שרות במערכת ההפעלה Windows והסבר את תפקידיהן.

תשובה:

א. קריאת שרות (system call) הן פונקציות שרות בסיסיות של מערכת ההפעלה המהוות את המינשק המהותי שבין תהליך משתמש (או תהליך מערכת) ובין מערכת ההפעלה. הן כתובות לרוב בספריות C של מערכת ההפעלה, כך שכל תוכנית מחשב המעוניינת להשתמש בהן יכולה לטעון באופן דינמי או להתקמפל איתם באופן סטטי. פונקציות השרות הבסיסיות הללו מאפשרות גישה לדיסקים הקשיחים, כתיבה וקריאה של קבצים, גישה להתקנים שונים (כגון מדפסת, מסך, מקלדת, עכבר), גישה לכרטיסי רשת ולשרותי תיקשורת שונים (רשת האינטרנט, רשת אירגונית, וכדומה), איתחול וסיום תהליכים, חוטים, פתיחת וסגירת קבצים וחוצצים (pipes), יצירת ספריות קבצים, שליחת סיגנלים ופסיקות לתהליכים שונים שרצים במקביל, ועוד.

ב. דוגמאות לקריאות שרות במערכת ההפעלה Unix הן למשל:

Process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

File management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

דוגמאות לקריאות מערכת במערכת ההפעלה Windows הן למשל:

```
CreateProcess()         Create an new process
CreateFile()            Create a new file
ReadFile()              Open file for reading
WriteFile()             Open file for writng
ExitProcess()           Exit a process
GetCurrentProcessId()   Get process id
Sleep()                 Stop process (waiting queue) for a
                        Specified amount of time
CreatePipe()            Create a pipe object for inter-process
                        communication
```

שאלה 3 [5%]

הסבר בקצרה מהן תוכניות מערכת (system programs) ותן שלוש דוגמאות לתוכניות מסוג זה במערכת ההפעלה Unix, ושלוש דוגמאות במערכת ההפעלה Windows.

תשובה:

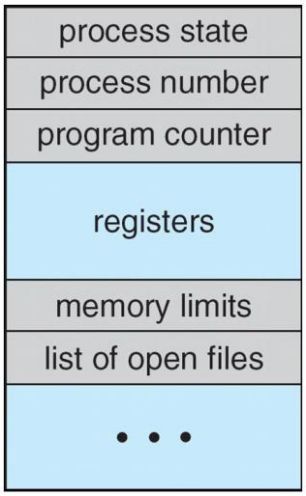
בניגוד לקריאות שרות (system calls), תוכניות שרות (system programs) הן תוכניות ביצוע רגילות (executable programs) שכל משתמש רשאי להפעיל באמצעות תפריט מערכת או באופן ידני מתוך מינשק משתמש אינטראקטיבי (command line interpreter) כמו חלון cmd.exe במערכת ההפעלה חלונות, או xterm במערכת Unix. תוכניות המערכת הן למעשה מינשק נוסף ברמה גבוהה יותר לקריאות המערכת הבסיסיות (wrappers) ומאפשרות למשתמש הרגיל (שאינו יודע לתכנת או לכתוב בשפת תיכנות כלשהי) להפעיל קריאות שרות של מערכת ההפעלה ועל ידי כך לקבל מידע חשוב על המערכת ולבצע פעולות מערכת מורכבות (כגון העברה והעתקת קבצים). בחלק מהמיקרים, תוכניות המערכת הן למעשה פקודות פנימיות של מעטפת (shell) כמו DOS או BASH. דוגמאות לתוכניות מערכת של Unix הן למשל:

```
ls      List files in current directory
cp      Copy file1 to file2
mkdir   Create a new directory
mv      Move file1 to file2
cat     Read a file and print it to screen
rm      Remove files
```

דוגמאות לתוכניות מערכת במערכת ההפעלה Windows הן:

```
dir         List files in current directory
copy        Copy file1 to file2
md          Create a new directory (make directory)
rename      Move file1 to file2
type        Read a file and print it to screen
del         Delete files
```

שאלה 4 [12%]



לפניך דיאגרמת חלקית של רשומת ה-PCB (Process Control Block) של תהליך במערכת הפעלה.

א. תן הסבר קצר לכל אחד מהשדות המופיעים בדיאגרמה

ב. תן דוגמא אחת בלבד לשדה משמעותי נוסף שאינו מופיע בדיאגרמה

תשובה:

א.

Process state	זהו מצב התהליך: חדש, רץ, מחכה, מוכן, סיים (new, running, ready, waiting, terminated)
Process number	מספר תהליך (pid). זהו המספר המזהה של התהליך אשר באמצעותו ניתן לגשת לטבלת התהליכים
Program counter	זהו מונה התוכנית המציין את מספר הפקודה הנוכחית בקוד הריצה
Registers	רשימת כל האוגרים במעבד. מדובר כמובן בשמירת התכנים של כל האוגרים מהפסיקה האחרונה של התהליך. אם התהליך במצב waiting אז החזרה שלו למצב ריצה תתבצע על ידי שיחזור כל האוגרים מרשומת ה-PCB.
Memory Limit	זהו מידע על מיקום וגודל של כל החלקים בזיכרון השייכים לתהליך. למשל איפה מתחיל ומסתיים אזור הקוד (text) של התהליך? מהם גבולות השטח של אזור ה-Data? ה-Heap? ה-Stack? וכדומה
Open Files	זהו מידע על רשימת הקבצים שנפתחו בתהליך למטרות קריאה וכתיבה

ב. הדיאגרמה הנ"ל היא כמובן חלקית וקיימים עוד מספר שדות חשובים שאינם נכללים בה. למשל כל המידע הקשור לתיזמון של התהליך (scheduling) אינו מוזכר כלל (priority, policy, nice values). שדה חשוב נוסף קשור ליומן הפעילות של התהליך (process acounting information) הכולל מידע כגון: מה היתה הפעם האחרונה שבה התהליך רץ? מהו זמן המעבד המצטבר של התהליך? מהי מנת (quanta) זמן הריצה של התהליך? על איזו ליבה התהליך רץ? מצביע לאזור בזיכרון בו שמורים האוגרים (register save area) מצביע לתהליך האב, מצביע לרשימת הילדים של התהליך, רשימת התקני קלט/פלט אשר התהליך משתמש או מחכה להם, ועוד.

שאלה 5 [8%]

בתהליך P במערכת ההפעלה Linux:

א. יש 200 חוטים (Threads) שונים A[0:200] אשר כל אחד מהם מבלה 30% מזמנו ביחידת העיבוד, ו-70% מזמנו בפעולות קלט ופלט שונות.

ב. 48 חוטים נוספים B[0:48] אשר כל אחד מהם מבלה 40% מזמנו ביחידת העיבוד, ו-60% בפעולות קלט/פלט.

ג. אף אחד מהחוטים A[100:124] לא יוכל להתחיל לפני שכל החוטים B[0:24] סיימו. כל שאר החוטים הם עצמאיים לחלוטין.

ד. במעבד שלנו יש 8 ליבות בעלות כוח חישוב זהה לחלוטין

ה. זמן הריצה של כל אחד מהחוטים A[0:200] הוא 50 שניות, ושל כל חוט B[0:48] הוא 100 שניות

בהנחה שמערכת ההפעלה היא הוגנת (Fair), ומריצה רק את התהליך P (ולא שום תהליך נוסף) בתנאים האידאליים ביותר, מהו הזמן הקצר ביותר שבו ניתן לצפות שהתהליך P יסיים את עבודתו?

תשובה:

כל אחד מהחוטים A[0:200] צורך 15 שניות של ליבת מעבד (30% של 50 שניות), וכל אחד מהחוטים B[0:48] צורך 40 שניות ליבת מעבד. על ידי שימוש במנגנוני מניעה הדדית (lock, semaphore) ניתן להבטיח שהחוטים A[100:124] ירוצו מיד לאחר החוטים B[0:24], ולכן נוכל לאחד אותם למשפחה אחת B[0:24]A[100:124] אשר כל חבר בה זקוק ל-55 שניות מעבד (40+15). יש לנו 24 חברים במשפחה, ו-8 מעבדים, ולכן מדובר בשימוש מלא של כל ליבות המעבד למשך **165** שניות (24*55/8)=165).
החוטים A[0:100], A[124:200], יצרכו (176*15/8=**330** שניות של מעבד מלא (שימוש מלא בכל הליבות)
החוטים B[24:48], יצרכו (24*40/8=**120** שניות של מעבד מלא.
יוצא שהזמן הכולל להרצת התהליך באופן שמנצל את כל הליבות בצורה אופטימלית הוא: 615 שניות

שאלה 6 [12%]

א. הסבר בקצרה את מה שקורה בתוכנית C הבאה במערכת ההפעלה Unix.

ב. מה יהיה <u>הפלט המדויק</u> של התוכנית?

ג. האם תשובתך לסעיף הקודם חייבת להיות יחידה או האם ייתכן יותר מפלט אחד אפשרי? נמק.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h> /* for pid_t */
#include <sys/wait.h>  /* for wait */

int do_work(const int a) {
    int i, b=0 ;
    printf("Entered a=%d\n", a) ;
    fflush(stdout) ;
    for (i=0 ; i<3 ; i++)
        b = a+b ;
    printf("Leaving b=%d\n", b) ;
    return b ;
}

int main() {
    pid_t pid=fork() ;
    if (pid==0) {
        do_work(10) ;
        exit(0) ;
    } else {
        waitpid(pid,NULL,0) ;
        do_work(20) ;
        fork() ;
        do_work(0) ;
    }
    return 0 ;
}
```

תשובה:

יש להתבונן בשיגרה main() בכדי להבין את מהלך התוכנית.

א. בשורה הראשונה מתבצע fork(), כלומר נוצר תהליך בן. את הערך המוחזר pid יש לנתח עכשיו בשני הקשרים שונים:
בתהליך הבן (pid=0) מתבצעת קריאה לפונקציה do_work(10) ומיד לאחר מכן סיום התהליך עם קוד יציאה 0 (exit(0)).
תהליך הבן ידפיס לכן את הפלט הבא:

```
Entered a=10
Leaving b=30
```

קישור לקובץ: fork3b.c

בתהליך האב (pid>0) מתבצעת קריאה לפונקציית המערכת waitpid המחכה עד שתהליך הבן יסיים.
מייד לאחר שתהליך הבן סיים מתבצעת קריאה לפונקציה do_work(20) אשר תגרום להדפסת הפלט הבא:

```
Entered a=20
Leaving b=60
```

בשורה הבאה מתבצע fork() בפעם השניה ונוצר תהליך ילד שני. האבא והילד השני יחדיו מבצעים do_work(0) אשר יגרום לפלט הבא:

```
Entered a=0
Leaving b=0
Entered a=0
Leaving b=0
```

לכן הסך הכולל של הפלט שנראה על המסך לאחר הפעלת התוכנית שלנו יהיה:

```
Entered a=10
Leaving b=30
Entered a=20
Leaving b=60
Entered a=0
Leaving b=0
Entered a=0
Leaving b=0
```

ג. במבט שני, הפיתרון שקיבלנו בסעיף הקודם עשוי להיות שונה משום שלאחר ה-fork() השני, האבא והבן השני רצים במקביל, ולכן אין שום וודאות לגבי סדר ההדפסות של ארבעת השורות האחרונות בפלט. אפשרויות הפלט הן:

```
Entered a=0
Leaving b=0
Entered a=0
Leaving b=0
```

```
Entered a=0
Entered a=0
Leaving b=0
Leaving b=0
```

שאלה 7 [12%]

רשום תוכנית shell קצרה בשפת C או בשפת Python אשר מקבלת מהמשתמש שם של פקודה (כמו ls או dir וכדומה) ומיד לאחר מכן מבצעת את הפקודה. התוכנית shell תחכה עד לסיום הפקודה ואז תהיה מוכנה לקבל את פקודה הבאה מהמשתמש (לולאה אינסופית).
להלן דוגמא לכיצד נראה פלט של התוכנית shell:

```
cmd> ls
    Lab  ncgi  note.html  notes.log  proj4  pyser.py
cmd> whoami
    jsmith
cmd> ps
    PID   TTY      TIME     CMD
    10927 pts/1    00:00:00 bash
    11807 pts/1    00:00:00 shell
    12508 pts/1    00:00:00 ps
```

תשובה: בשפת Python מפרש הפקודות יראה כך:

```
#  A simple command interpreter that uses Python subprocess module
#  to execute commands typed in by the user (with no arguments!)
#  Unix and Windows

import os
from subprocess import Popen, PIPE

while True:
    cmd = raw_input("cmd> ")
    p = Popen(cmd, stdin=PIPE, stdout=PIPE)
    print p.stdout.read()
```

דוגמא לפיתרון בשפת C הוא:

```
/*  shell.c
    This program is a simple command interpreter that uses execlp() to
    execute commands typed in by the user. Unix only !
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    int pid ;
    char line[81] ;

    for (;;) {
       fprintf(stderr, "cmd: " );
       pid = fork() ;             /* create a new process */

       if (pid > 0) {             /* parent process */
          waitpid(pid,NULL,0);    /* null pointer - return value not saved */
       } else if (pid == 0) {     /* child process */
          execlp(line, line, (char *) NULL);           /* execute program */
          /* some problem if execlp returns */
          fprintf(stderr, "Can't execute %s\n", line);
          exit(1);
       } else if ( pid == -1) {   /* can't create a new process */
          fprintf(stderr, "Can't fork!\n");
          exit(2);
       }
    }
}
```

קישורים לפיתרונות אלה ונוספים:

Shell1.py
Shell2.py
Shell.c

שאלה 8 [10%]

עיין היטב בתוכנית Python הבאה. בשתי השורות האחרונות נוצרים 10 חוטים שונים של הפונקציה foo.

```
from threading import Thread, Semaphore
x = y = 0
semx = Semaphore(2)
semy = Semaphore(2)

def foo():
    global x, y
    if x<y:
        semx.acquire()
        semy.release()
        x = y-1
    else:
        semx.release()
        semy.acquire()
        y = x+1

T = [Thread(target=foo) for i in range(10)]
for t in T: t.start()
```

א. האם כל 10 החוטים שנוצרו בתוכנית יסיימו את עבודתם?

ב. רשום את המספר המדויק של החוטים שיסיימו ומה יהיה הערך של y לאחר סיומם?

תשובה:

חוט עשוי להיתקע על סמפור נעול אשר אין לו סיכוי להשתחרר לעולם.

מהנתון ההתחלתי x=y=0 ברור כי הדבר הראשון שייתבצע הן השורות של משפט ה-else.

כמו-כן ברור כי לכל היותר שני חוטים יוכלו לבצע את משפט ה-else ולסיים (הסמפור semy ננעל בחוט השלישי).

ייתכנו שני תסריטים עקריים.

תסריט 1 : כל החוטים מבצעים את משפט ה-else במקביל.

תסריט 2 : חוט אחד בלבד מבצע את משפט ה-else וכל השאר את משפט if

תסריט 3 : שני חוטים או יותר מבצעים את משפט ה-else וכל השאר את משפט if

בתסריט הראשון : רק שני חוטים יסיימו את הפונקציה foo ושאר החוטים יינעלו על ידי semy=0.

בתסריט השני : חוט אחד יסיים ונעבור למצב : x=0,y=1,semx=3,semy=1.

מהמצב הזה נשאר באופן קבוע בערכים x=0,y=1, ולכן לאחר 3 חוטים שיסיימו, כל שאר החוטים יינעלו (עקב נעילה של semx). כך שבתסריט השני בדיוק 4 חוטים יסיימו ושישה אחרים יינעלו.

בתסריט השלישי : שני חוטים יסיימו במצב : x=0,y=1,semx=4,semy=0.

חוטים אחרים שינסו לבצע את משפט ה-else יינעלו (semy=0).

ולאחר מכן ארבעה חוטים נוספים יסיימו עד לנעילה של semx.

כך שבתסריט השלישי 6 חוטים מסיימים ו-4 ננעלים.

ברור כי בכל התסריטים הערך הסופי של y הוא **y=1**.

קישור לקובץ : sems.py

שאלה 9 [18%]

מערכת ניטור טמפרטורת בניין מתבססת על תהליך P שרץ באופן קבוע (על גבי מערכת Embedded Linux) ומקבל את נתוני הטמפרטורה באמצעות 80 חיישנים שונים המפוזרים על פני נקודות שונות ברחבי הבניין. כל חיישן sensor[i] רושם את הטמפרטורה שדגם לתוך תור que[i], כאשר i=0,1,2,…,79. לתהליך P יש גישה גלובלית לכל 80 התורים האלה. מטרת המערכת היא לחשב את הטמפרטורה הממוצעת בבניין averageT, הטמפרטורה המינימלית minT, והטמפרטורה מקסימלית maxT, בכל פרק זמן מחזורי של 5 שניות. החיישנים מייצרים כמויות ענק של נתוני טמפרטורה שאינן ניתנות לשליטה על ידי ליבה אחת במעבד, ונדרשות לכך שמונה ליבות.

רשום קוד Python מתאים לביצוע המשימה הזו על ידי 8 תהליכים מקבילים (למערכת יש מעבד מרובה ליבות שמאפשר זאת). המאמץ החישובי חייב להתחלק בין כל התהליכים השונים במידה מאוזנת לחלוטין.

הכוונה: הנח שכל החיישנים פועלים באופן סדיר ואחיד, אינם נתקעים ואינם נעצרים. אל תטרח למדל את החיישנים עצמם ואת הקשר שלהם לתורים que[i]: פשוט הנח שהתורים que[i] מספקים לך את הנתונים בצורה סדירה. השתמש בכל הטמפרטורות שהצטברו ב-5 השניות האחרונות מכל החיישנים. השתדל לרכז את כל החישובים הדרושים במקום אחד (סכום, ממוצע, מינימום, מקסימום).

תשובה:

```
# 80 sensor queues that supply heat data
sensor_ques = [Queue() for i in range(80)]
# temperatures from each group of 10 sensor ques will be
# channeled to one data que
data_ques = [Queue() for i in range(8)]

# get heat data from 10 sensor ques and
# put them in one dataq
def get_heat_data(ques, dataq):
    while True:
        for q in ques:
            t = q.get()  # assuming no q can get stuck ...
            dataq.put(t)

for i in range(8):
    ques = sensor_ques[10*i : 10*i+10]
    p = Process( target=get_heat_data, args=(ques, data_ques[i]) )
    p.start()

while True:
    time.sleep(5)
    temperatures = []
    for i in range(8):
        qsize = data_ques[i].qsize()
        for j in range(qsize):
            temperatures.append(data_ques[i].get())
    print ">>> Check Point <<<"
    print "Average Temperature =", sum(temperatures)/len(temperatures)
    print "Maximal Temperature =", max(temperatures)
    print "Minimal Temperature =", min(temperatures)
```

דוגמאות עובדות (בצירוף סימולציה של חיישנים) ניתן להוריד מהקישורים הבאים:

averageT.py

averageT2.py

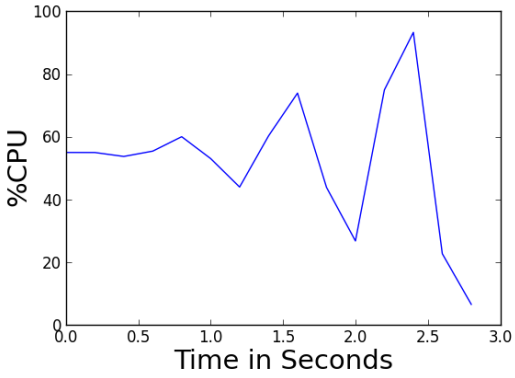
שאלת בונוס [10%]

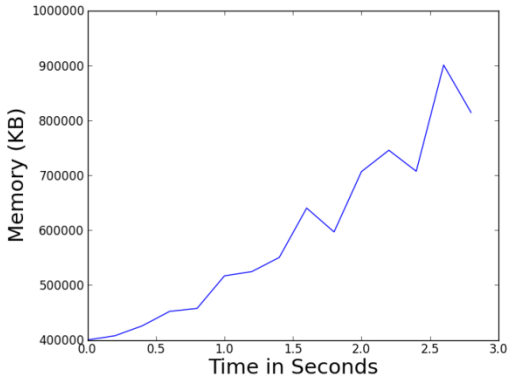
א. התוכנית sysmon.exe מדפיסה (לפלט הסטנדרטי stdout) את נתוני צריכת המעבד והזיכרון במערכת בכל 0.1 שניות בפורמט הבא:

```
cmd> sysmon.exe
time=2.1 cpu=38.6% mem=566123KB
time=2.2 cpu=42.1% mem=581703KB
time=2.3 cpu=45.2% mem=605893KB
...
```

ב. התוכנית cpu_update.exe מקבלת נתונים מהצורה "2.3,38.6%" מהקלט הסטנדרטי (stdin) ומעדכנת גרף מערכת המראה את צריכת המעבד (באחוזים) כפונקציה של זמן, כאשר t=2.3 הוא הזמן בשניות, ו- 38.6% היא צריכת המעבד באחוזים (מופרדים על ידי פסיק).

ג. התוכנית mem_update.exe מקבלת נתונים מהצורה "2.3,581703KB" מהקלט הסטנדרטי (stdin) ומעדכנת גרף מערכת המראה את צריכת הזיכרון הפיזי כפונקציה של זמן. גם כאן הזמן (t=2.3) וכמות הזיכרון מופרדים על ידי פסיק.





כתוב תוכנית בשפת C או בשפת Python שמפעילה את שלושת התוכניות הנ"ל, ולאחר מכן מעבירה מהפלט של התוכנית sysmon.exe, את המידע הדרוש לתוכניות cpu_update.exe, ו- mem_update.exe, בכדי לאפשר את עידכון הגרפים.

<u>הכוונה:</u> הנח שהגרפים נוכחים תמיד על ידי מערכת ההפעלה ומטופלים על ידיה. בכדי לפשט את התמונה, צא מהנחה ששלושת התוכניות מתחילות לפעול בזמן t=0.0 (על ידי התוכנית שתכתוב כמובן). ייתכן ותזדקק למנגנון ה-Pipe, אך אתה רשאי להשתמש בכל רעיון שנראה לך המתבסס על החומר שנלמד בקורס.

<u>תשובה:</u>

```
import time
from subprocess import Popen, PIPE

p1 = Popen(["sysmon.exe"], stdin=PIPE, stdout=PIPE)
p2 = Popen(["cpu_update.exe"], stdin=PIPE, stdout=PIPE)
p3 = Popen(["mem_update.exe"], stdin=PIPE, stdout=PIPE)

while True:
    t,c,m = p1.stdout.readline().strip().split()
    time = t.split('=')[1]
    cpu = c.split('=')[1]
    mem = m.split('=')[1]
    p2.stdin.write(time + "," + cpu + "\n")
    p3.stdin.write(time + "," + mem + "\n")
```