מערכות הפעלה 31261

פתרון מבחן סופי, מועד א', סמסטר ב' תשע"ג, 14/07/2013

שאלה 1 [10%]

א. תאר בקצרה מהו תהליך חישוב במערכת מחשב?
ב. מהם המרכיבים השונים מהם הוא מורכב? פרט מהם ההבדלים העקריים בין data ובין Heap.
ג. מהם המצבים השונים בהם עשוי תהליך להיות? הסבר בקצרה את המאפיין העיקרי של כל מצב.

תשובה:

א. תהליך הוא תוכנית מחשב במצב ריצה על מכונת חישוב עם גישה לכל התקני החומרה.
ב. תהליך במערכת ההפעלה מתאפיין על ידי שטח זיכרון המורכב מארבעה חלקים עקריים: Code, Data, Heap, Stack, ומבנה נתונים (PCB=Process Control Block) שמנוהל על ידי מערכת ההפעלה באמצעות טבלת תהליכים כללית. מערכת ההפעלה מנהלת גם את רשימת הקבצים הפתוחים שהתהליך משתמש בהם, וגם את ערוצי התיקשורת שהתהליך עשוי לפתוח במהלך עבודתו (TCP/IP sockets)
ג. תהליך עשוי להימצא באחד מתוך 5 מצבים: new, ready, running, waiting, terminated
New – המצב ההתחלתי של תהליך בזמן יצירתו על ידי מערכת ההפעלה
Ready – התהליך נמצא בזיכרון (על כל ארבעת חלקיו) ורשומת ה-PCB נמצאת בטבלת התהליכים, אך התהליך אינו רץ, אך מוכן לריצה, ומחכה בתור לרוץ
Running – התהליך רץ על המעבד (או על אחת הליבות) ושולט על האוגרים (registers) וניגש לזיכרון ככל שנדרש.
Waiting – התהליך הפסיק לרוץ עקב הצורך לחכות לסיומה של פעולת קלט או פלט (שלרוב דורשת זמן רב יחסית לזמן מעבד יקר, ולכן מערכת ההפעלה הורידה אותו מהמעבד באמצעות פסיקה). התהליך אינו מוכן לריצה עד לסיומה של פעולת ה-I/O (או כל תנאי אחר שהביא לעצירתו).
Terminated – במערכות הפעלה מסוימות, תהליך עובר למצב terminated (או zombie, או defunct) לאחר שסיים את עבודתו. זה מתבטא בכך שמערכת ההפעלה שומרת את רשומת ה-PCB שלו, קוד יציאה, אופן הסיום, ועוד פרטים דומים, עד לשלב שבו הם אינם דרושים יותר (בדרך כלל על ידי תהליך אב או תהליכים ששיתפו פעולה שמצפים לקבל מידע).

שאלה 2 [10%]

א. תאר בקצרה מהו חוט (Thread) ומהם ההבדלים העקריים שבין חוט לתהליך?
ב. מנה את כל הדרכים שבהם חוטים שונים (באותו תהליך) יכולים לשתף מידע?
ג. מנה לפחות שלושה דרכים שבאמצעותם תהליכים שונים עשויים לשתף מידע

תשובה:

א. תהליך במערכת הפעלה מודרנית עשוי להתחלק למספר משימות עצמאיות הרצות במקביל באופן כמעט בלתי-תלוי. כל משימה מסוג זה נראת חוט (Thread). באופן מסורתי, משימות שונות במסגרת תהליך מתבצעות באופן סדרתי אחת לאחר השניה על פי סדר ברור: המשימה הבאה לא תתחיל לפני שהקודמת סיימה. חוטים לעומת זאת מייצגים משימות מוגדרות היטב המתבצעות במקביל: מערכת ההפעלה מקצה לכל חוט זמן פעולה קצר (בין 20 ל-60 מילי-שניות) ועוברת בין כל החוטים השונים על פי תור סיבובי (Round-Robin), באופן כזה שנראה כי החוטים רצים במקביל ומקבלים זמן מעבד (ומשאבים אחרים) שווה.
ב. הייתרון המשמעותי של ריבוי חוטים על פני ריבוי תהליכים הוא שכל החוטים של אותו תהליך משתתפים בכל חלקי הזיכרון של התהליך מלבד ב-Stack. לכל החוטים יש גישה לאותו קוד תוכנה (אזור ה-Code של התהליך). כל החוטים שותפים באזורי ה-Data וה-Heap. כל שני חוטים מסוגלים לשתף ביניהם מידע באמצעות כתיבה וקריאה ישירה מתוך אזורי ה-Data וה-Heap (וזה כמובן כולל משתנים גלובליים וסטטיים). בנוסף לזאת, כל החוטים באותו תהליך שותפים גם בטבלת הקבצים הפתוחים וערוצי התיקשורת של התהליך (sockets). כלומר, כל החוטים יוכלו לקרוא או לכתוב לאותו קובץ במקביל. באותה מידה, כל

החוטים יוכלו להתחבר לשרת אינטרנט, ולקרוא ולכתוב אליו במקביל (על אותן ערוץ). בניגוד לתהליכים, קל מאוד להתחיל ולסיים חוטים: הם צורכים משאבים מועטים ביחס לתהליכים, והניהול שלהם על ידי מערכת ההפעלה (או על ידי ספריות מערכת) הוא יותר קל ונוח.

ג. לתהליכים שונים (Processes) לעומת זאת אין שום שטחי זיכרון משותפים שבאמצעותם יוכלו לשתף מידע. בכדי ששני תהליכים שונים יוכלו לשתף מידע עליהם להשתמש באופן מוצהר באחד מהמנגנונים הבאים:

a. Shared Memory: במערכות הפעלה מסוימות קיימות ספריות מערכת שבאמצעותן שני תהליכים יוכלו להשתתף בקטע זיכרון לשם החלפת מידע.

b. Pipes: במערכות ההפעלה Unix וגם Windows קיים מנגנון Pipe שהוא למעשה שטח זיכרון שנמצא ברשות מערכת ההפעלה, ושני תהליכים שמשתפים פעולה ביניהם, יוכלו להשתמש בו לשם החלפת מידע ביניהם. בדרך כלל Pipe נוצר באמצעות קריאת מערכת ()pipe של תהליך אב שמייד לאחר מכן יוצר תהליך בן אשר יורש את ה-Pipe מהאבא.באופן כזה מקבלים שני תהליכים שונים המשתפים ביניהם אובייקט Pipe. לרוב קיימת מוסכמה שתהליך אחד הוא כותב בלבד, והתהליך השני קורא בלבד, ובכך פעולת שיתוף המידע נעשית יותר פשוטה לניהול (קשה מאוד לנהל מצבים בהם לשני התהליכים יש רשות כתיבה וקריאה בו-זמנית).

c. Sockets: שיטת ה-Pipe מוגבלת רק לתהליכים שרצים על אותה מכונה. בכדי ששני תהליכים הנמצאים בשתי מכונות שונות (ואולי גם בשתי יבשות שונות) יוכלו לשתף מידע, עליהם בדרך כלל להשתמש בפרוטוקולי תקשורת של האינטרנט (TCP/IP). המנגנון הבסיסי עליו מושתתות כמעט כל טכנולוגיות התיקשורת באינטרנט הוא מנגנון ה-Socket. דוגמא מובהקת היא שיתוף המידע שבין דפדפן Chrome שמותקן על מחשב אישי קטן ובין שרת אינטרנט מרוחק שמארח את האתר ynet (שבדרך כלל נמצא בגרמניה).

שאלה 3 [5%]

תאר בקצרה (ובאופן סכימטי בלבד) כיצד מערכת ההפעלה מבררת את המידע הבא עבור תהליך נתון

א. מהי הכתובת ההתחלתית של הקוד שצריך להריץ?

ב. מהי הכתובת הראשונה בזיכרון Heap פנוי?

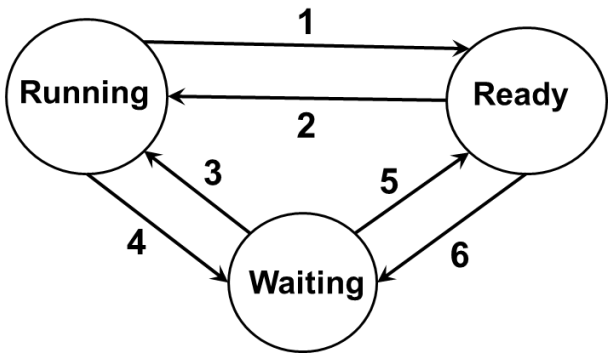
ג. הערך ההתחלתי של אוגר המחסנית (Stack Pointer)?

תשובה:

התשובה לכל השאלות האלה במילה אחת היא: PCB. מערכת ההפעלה מקצה רשומת Process Control Block לכל תהליך במערכת. רשומה זו כוללת את כל המידע הנחוץ עבור מערכת ההפעלה בכדי לגשת לכל חלקי הזיכרון של התהליך, זמני הריצה בעבר, מצב נוכחי, רמת תיעדוף (Priority), וכדומה. השדה mm (Memory Managemnt Info) שמופיע ב-PCB כולל מידע לגבי הזיכרון של התהליך.

שאלה 4 [18%]

לפניך דיאגרמת מצבים בהם עשוי להימצא תהליך הרץ במערכת הפעלה מרובת תהליכים. כל חץ מסמן מעבר ממצב אחד למצב שני. תאר בקצרה את התנאים שעשויים לגרום לתהליך לעבור ממצב אחד לשני (אם המעבר אפשרי) ורשום את תשובותיך בהתאם למספר החץ בדיאגרמה. במידת האפשר, תן דוגמאות לתנאים כאלה.



רשום את תשובתך במחברת הבחינה בפורמט הבא:

חץ 1: תאור התנאים למעבר ...

חץ 2: תאור התנאים למעבר ...

חץ 3: תאור התנאים למעבר ...

חץ 4: תאור התנאים למעבר ...

חץ 5: תאור התנאים למעבר ...

חץ 6: תאור התנאים למעבר ...

תשובה:

חץ 1: בדרך כלל המעבר ממצב Running למצב Ready מתבצע לאחר שהתהליך סיים לנצל את מנת הזמן הקבועה שמוקצית לו לריצה (quanta) על ידי מערכת ההפעלה. התהליך מופסק באמצעות פסיקת Timer ומועבר לתור התהליכים Ready, מאחר שבדרך כלל, התהליך הופסק באמצע עבודתו ומוכן להמשיך לעבוד באופן מיידי.

במצבים נדירים מאוד, מערכת ההפעלה תפסיק תהליך באמצע עבודתו בכדי לפנות מקום בזיכרון לתהליכים חדשים.

חץ 2: המעבר ממצב Ready למצב Running מתבצע כאשר מגיע תורו של תהליך המוכן לריצה לעבור למצב ריצה. המתזמן קצר הטווח (short time scheduler) מחליט בכל רגע נתון איזה מבין התהליכים שנמצאים במצב Ready יעבור למעבד למצב Running.

חץ 3: מעבר כזה אינו אפשרי. בכדי לעבור למצב Running חובה על תהליך להיות קודם כל במצב Ready.

חץ 4: המעבר ממצב Running למצב Waiting קורה כאשר תהליך צריך לבצע פעולת כתיבה או קריאה מדיסק או מרשת. במצבים נדירים, מערכת ההפעלה עשויה להוריד תהליך גדול מאוד שרץ במשך זמן רב מאוד או שרמת פעילותו מאוד נמוכה לדיסק בכדי לתת לתהליכים קטנים וקצרים יותר הזדמנות לרוץ.

חץ 5: המעבר ממצב Waiting למצב Ready מתרחש לאחר סיומה של פעולת קלט/פלט שהתהליך חיכה לה.

חץ 6: מעבר כזה אינו אפשרי! תהליך שנמצא במצב של Ready יוכל לעבור רק למצב Running בלבד.

שאלה 5 [5%]

בתהליך P במערכת ההפעלה Linux יש 10 חוטים (Threads) שונים. שישה חוטים מבלים 70% מזמנם בכתיבה לדיסק ו-30% מזמנם ביחידת העיבוד (CPU). ארבעת החוטים הנותרים מבלים 20% מזמנם באיסוף חבילות תיקשורת (מול כרטיס התיקשורת) ו-80% מהזמן בעיבוד CPU. בהנחה שמערכת ההפעלה שלנו היא הוגנת (Fair), מהו הזמן הכולל שבו התהליך P מבלה ביחידת העיבוד המרכזית?

תשובה:

בהנחה שמערכת ההפעלה מקצה מנת זמן שווה לכל החוטים (Fair OS) הרי שהזמן הכולל שבו כל החוטים יחדיו מבלים ביחידת העיבוד הוא:

$$(6 * 30\% + 4 * 80\%) / 10 = 50\%$$

שאלה 6 [10%]

מה יתרחש לאחר שנריץ את התוכנית C הבאה במערכת ההפעלה Unix. פרט בקצרה את כל השלבים בדרך.

```
#include <unistd.h>
#include <stdio.h>
int main()
{
    int pid, pip[2] ;
    char instring[7] ;

    pipe(pip) ;

    pid = fork() ;
    if (pid == 0) {
        write(pip[1], "Hi Mom!", 7) ;
    } else {
        read(pip[0], instring, 7) ;
        printf("%s\n", instring) ;
    }
}
```

תשובה:

א. התוכנית יוצרת אובייקט Pipe (בשם pip) שמיוצג למעשה על ידי שני מספרים שלמים (pip[0] ו-pip[1]) שהם למעשה שני הקצוות של "הצינור" (file descriptors). בכל אחד מהם ניתן לכתוב או לקרוא מידע בין שני תהליכים שונים.

ב. מתבצע fork() : התהליך משכפל את עצמו ובכך נוצר תהליך חדש (שנקרא תהליך הבן)

ג. תהליך הבן, מהיותו שיכפול מדויק של תהליך האב, כולל מצביע לאותו ה-Pipe שנבנה בשלב א' (ה-Pipe עצמו הוא אובייקט ששייך למערכת ההפעלה)

ד. קריאת המערכת fork() חוזרת בשני תהליכים שונים: בתהליך האב היא מחזירה את מספרו של תהליך הבן pid (process id), ובתהליך הבן, הפונקציה fork() תחזיר 0.

ה. כאמור, בשלב זה יש לנו שני תהליכים שרצים במקביל. תהליך הבן שקיבל ערך pid=0, כותב לקצה pip[1] את המשפט "Hi Mom!" ומסיים. במקביל, תהליך האב שקיבל ערך pid שונה מאפס, מנסה לקרוא את הקצה pip[0]. הוא עשוי לחכות מעט עד שהמידע יגיע. לאחר קבלת המידע לתוך החוצץ instring, תהליך האב מדפיס אותו ומסיים. כמובן תהליך האב ידפיס את המשפט "Hi Mom!" (למרות שלמעשה הוא האבא...)

שאלה 7 [10%]

רשום תוכנית קצרה בשפת C או בשפת Python שמשתמשת בקריאות מערכת (כמו: fork, waitpid, exit, kill), ומבצעת את הדברים הבאים:

א. תהליך אב מוליד תהליך בן ומדפיס את מספר התהליך שלו (pid)
ב. תהליך הבן מוליד תהליך נכד והולך לישון 120 שניות
ג. תהליך האב הולך לישון 60 שניות ואז מסיים את כל התהליכים ומסיים בעצמו

תשובה:

לשם ההמחשה כללנו שורות הדפסה נוספות מעבר לנדרש, ולכן ניתן לקצר. נסה להריץ על Linux.
בכדי להקל על הזיהוי, קוד היציאה של האבא הוא 0, של הבן הוא 1, ושל הנכד 2 (זה לא התבקש בשאלה, אבל זה עוזר להבין את התשובה):

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>

int main()
{
    int pid1, pid2 ;

    pid1 = fork() ;
    if (pid1 == 0) {
        printf("I am the child\n") ;
        pid2 = fork() ;
        if (pid2 == 0) {
            printf("I am the grandchild\n") ;
            exit(2) ;  /* grandchild exits immediately after printf */
        } else {
            printf("I am still the child\n") ;
            sleep(120) ;
            /* next line should not happen! Child process dies before */
            printf("I am still the child - normal exit – not to be seen\n") ;
            exit(1) ;
        }
    } else {
        printf("I am the father\n") ;
        printf("child pid = %d\n", pid1) ;
        sleep(60) ;
        kill(pid1, SIGKILL) ;
        /* no need to kill pid2 which exited long ago */
        exit(0) ;
    }
}
```

ניתן להוריד את הקובץ מתוך הקישור: family.c

בשפת Python הפיתרון יראה כך:

```
import os, sys, time, signal   # Unix only! Will not run on Windows

pid1 = os.fork() ;
if pid1 == 0:
    print "I am the child: pid =", os.getpid()
    pid2 = os.fork()
    if pid2 == 0:
        print "I am the grandchild: pid =", os.getpid()
        sys.exit(2)   # grandchild exits immediately after print
    else:
        print "I am still the child"
        time.sleep(120)
        # this should not happen! child already killed by parent
        print "I am still the child - normal exit - shouldn't see this!"
        sys.exit(1)
else:
    print "I am the father: pid =", os.getpid()
    print "child pid =", pid1
    time.sleep(60) ;
    os.kill(pid1, signal.SIGKILL)
    # no need to kill pid2 which exited long ago
    sys.exit(0) ;
```

אפשר להוריד את התוכנית מהקישור הבא: family.py

שאלה 8 [15%]

נתונים לנו שני וקטורים u,v באורך 8000 כל אחד. המכפלה של שני וקטורים מוגדרת להיות סכום מכפלות האיברים המתאימים: $u * v = \sum_{i=0}^{n} u[i] * v[i]$, כאשר n מציין את אורך הוקטור (במקרה שלנו: n=8000).
כתוב קוד של Python לביצוע המכפלה במקביל על ידי 8 תהליכים שונים. המאמץ החישובי חייב להתחלק במידה שווה בין כל 8 התהליכים.

הכוונה: הנח שווקטור הוא רשימה (list) או סדרה (tuple) פשוטה ב-Python. הנח שברשותך מעבד מרובה ליבות (ככל שיידרשו). מדובר בתהליכים (Processes) ולא בחוטים (Threads) לכן אתה עשוי להזדקק לתור (Queue) אחד או יותר בכדי לשתף מידע בין התהליכים השונים.

תשובה:

```
def _product(u, v, que):
    result = 0.0
    for i in range(len(u)): result += u[i]*v[i]
    que.put(result)

def parallel_product(u,v):
    que = Queue()
    P = [Process (
            target = _product,
            args = (u[1000*i:1000*(i+1)], v[1000*i:1000*(i+1)], que)
         ) for i in range(8)]
    for p in P: p.start()
    for p in P: p.join()
    result = 0.0
    for i in range(8): result += que.get()
    return result
```

שאלה 9 [7%]

עיין היטב בתוכנית Python הבאה:

```
a = 2 ; b = 4 ; c = 6

def foo():
    global a, b, c
    b = 2*a + 1
    c = b - 1

def bar():
    global a, b, c
    a = 2*c - 1
    b = b + 1

t1 = Thread(target=foo) ; t2 = Thread(target=bar)
t1.start() ; t2.start()
t1.join() ; t2.join()
sum = a+b+c
print sum
```

מהן כל התוצאות האפשריות של הפלט (sum) של התוכנית? תן נימוק קצר לתשובתך.

תשובה:

הערכים האפשריים הם: 17, 21, 22, 56, 57, 58

הסבר: קיימים שישה אפשרויות ריצה של 4 החישובים הנ"ל כפי שמודגם בדיאגרמה הבאה.

אם נעבור על כל האפשרויות נקבל את הסכומים האלה כתוצאות אפשריות.

```
A:    b = 2*a + 1
B:    c = b - 1

C:    a = 2*c - 1
D:    b = b + 1

Possible runs:  ABCD, ACBD, ACDB, CDAB, CADB, CABD
```

למשל עבור הריצה CADB החישוב הוא:

```
a = 2 ; b = 4 ; c = 6
C:    a = 2*c – 1 = 11
A:    b = 2*a + 1 = 23
D:    b = b + 1   = 24
B:    c = b – 1   = 23
sum = a+b+c = 11+24+23 = 58
```

שאלה 10 [10%]

בפרוייקט תוכנה רפואית, יש שלושה חוטים t1, t2, t3, של תהליך מסוים הניגשים לארבעה קבצים file1, file2, file3, file4, למטרות כתיבה בתדירות גבוהה מאוד. פונקציית הכתיבה שכל חוט עשוי להפעיל בכל זמן נתון היא פשוטה:

```
def lwrite(file,line):
    f = open(file, "a")
    f.write(line)
    f.close()
```

והיא נמשכת כמובן פרק זמן קצר מאוד. אך למרות זאת, מספר שבועות לאחר שיחרור המוצר, הלקוחות נתקלו במצבים בהם אותו הקובץ נפתח במקביל על ידי שני חוטים ותוצאת הכתיבה יצרה שיבושים (שמהווים סכנה לבריאות המטופלים). בכדי להתגבר על הבעייה, יש לתקן את הפונקציה lwrite() על ידי שימוש במנגנוני מניעה הדדית (Lock או Semaphore) בכדי להבטיח את התנאים הבאים:

א. כל חוט יוכל במוקדם או במאוחר לגשת לכל קובץ שייבחר (מניעת מצב "הרעבה")

ב. בכל רגע נתון יהיה לכל היותר חוט אחד שכותב לקובץ נתון (כלומר: שני חוטים לא יוכלו לכתוב לאותו קובץ בו-זמנית)

ג. בכל רגע נתון לא יהיה ניתן לכתוב ליותר משני קבצים במקביל (כלומר: לא יכתבו שלושה קבצים במקביל, גם לא על ידי שלושה חוטים שונים!), אך לא תמנע כתיבה לשני קבצים על ידי שני חוטים.

תקן את הפונקציה lwrite() בכדי שדרישות אלה ייתמלאו באופן סביר ולא בזבזני.

תשובה:

ניתן לפתור את הבעייה בכמה דרכים. נציג דרך אחת אפשרית:

נגדיר ארבעה מנעולים lock[file] עבור כל אחד מהקבצים (כלומר lock הוא מילון שהמפתחות שלו הם ארבעת הקבצים, והערכים הם מנעולים). בנוסף למנעולים, נגדיר Semaphore בשם sem שיאותחל לערך 2. חוט יוכל להפעיל את הפונקציה lwrite(file,line) יהיה חייב לנעול את lock[file] ולאחר מכן לנעול את הסמפור sem. הנעילה הראשונה תמנע משני חוטים לכתוב לאותו הקובץ, והנעילה השניה תמנע משלושה חוטים שונים לכתוב לשלושה קבצים שונים בו-זמנית.

```
files = ["file1.log", "file2.log", "file3.log", "file4.log"]
sem = Semaphore(2)
lock = {}
for file in files:
    lock[file] = Lock()

def lwrite(file,line):
    lock[file].acquire()
    sem.acquire()
    f = open(file, "a")
    f.write(line)
    f.close()
    sem.release()
    lock[file].release()
```

שאלת בונוס [10%]

כתוב פונקציית Python בשם dirsize(dir) המקבלת ספרייה dir, ומחזירה את גודל הכולל של כל הקבצים הרגילים בספריה (רקורסיבית).

הכוונה: יש לעשות שימוש בפונקציות ספריה: os.path.isfile, os.path.listdir, os.path.isdir

אין להשתמש בפונקציה os.walk.

תשובה:

דוגמא זו ניתנה באחד משיעורי התרגיל ואף גילינו שזמן הריצה על ספריות גדולות ב-Windows הוא סביר מאוד:

```
import os

def dirsize(dir):
    size = 0
    for f in os.listdir(dir):
        file = dir + os.sep + f
        if os.path.isfile(file):
            size += os.path.getsize(file)
        elif os.path.isdir(file):
            size += dirsize(file)

    return size
```