מערכות הפעלה 31261

פתרון מבחן סופי, מועד א', סמסטר א' תשע"ד, 30/01/2014

הוראות לנבחן: משך הבחינה שלוש שעות. חומרי העזר המותרים הם השקפים של הקורס שנמסרו באתר הקורס בלבד. שימוש במחשבונים ומכשירים אלקטרוניים כלשהם אסור בהחלט. רשום את תשובותיך במחברת במצורפת לבחינה. ציין בבירור את מספר השאלה במחברת והשתדל להיות **קצר ועניני** (תשובות ארוכות מדי או נסיונות לתת מספר תשובות אפשריות ייפסלו את תשובתך!). הקפד על כתב יד ברור ומסודר, ומחק את כל חומר הטיוטא. השימוש בשפה האנגלית מותר. השאלון מכיל 11 שאלות בשווי של 106 נקודות, ונפרש על פני 3 עמודים.
בהצלחה !

הערה חשובה: הפתרונות כאן ניתנים באריכות רבה יותר מהנדרש בכדי לספק את כל הפרטים וכל ההסברים להבהרת התשובות. במספר מקרים ניתנו תשובות אפשריות נוספות.
בפועל, התשובות הנדרשות מהתלמיד אמורות להיות יותר קצרות וענייניות!

שאלה1 [12%]

א. הסבר בקצרה מהי תוכנית מערכת (system program) ותן שלוש דוגמאות לתוכניות כאלה במערכת ההפעלה Linux וגם במערכת ההפעלה Windows.

ב. הסבר כיצד תוכניות מערכת מופעלות על ידי מערכת ההפעלה? תן תאור כללי וקצר של המנגנון שבאמצעותו תוכניות מערכת מופעלות, ותן שתי דוגמאות שונות להפעלה של תוכנית מערכת במערכת ההפעלה Linux באמצעות קוד C או קוד Python.

תשובה:

א. בניגוד לקריאות מערכת (system calls), תוכניות מערכת (system programs) הן תוכניות ביצוע רגילות (executable programs) שכל משתמש רשאי להפעיל באמצעות תפריט גרפי 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. דוגמאות לתוכניות מערכת של Unix:

```
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
```

ב. במערכת ההפעלה Linux מנגנון ההפעלה של תוכנית מערכת P מתבצע באמצעות שילוב של קריאת המערכת fork() ואחת מקריאות המערכת: execl(), execv(), execlp(), execvp().
הפונקציה fork() משכפלת את התהליך הנוכחי לשני עותקים זהים: תהליך אב, ותהליך בן. בשלב הבא, אחת מפונקציות ה-exec מחליפה את תהליך הבן (או תהליך האב) בתוכנית מערכת שנשלחה אליה כארגומנט, ובכך נוצר תהליך חדש שמפעיל את תוכנית המערכת P.
שתי דוגמאות פשוטות:

```
pid = fork();
if (pid == 0) {
   // child process starts the system program
   char *argv[] = {"ls", "-l", "/usr/src", (char*)NULL};
   execv("/usr/bin/ls", argv)
} else {
   // parent process continues here
}
```

```
pid = fork();
if (pid == 0) {
   // child process starts the system program
   execl("/usr/bin/ls", "ls", "-l", "/usr/src", (char*)NULL)
} else {
   // parent process continues here
}
```

בשפת Python הדוגמאות יראו כך:

```
pid = os.fork();
if pid == 0:
   # child process starts the system program
   argv = ["ls", "-l", "/usr/src"]
   os.execv("/usr/bin/ls", argv)
else:
   # parent process continues here
```

```
pid = os.fork();
if pid == 0:
   # child process starts the system program
   os.execl("/usr/bin/ls", "ls", "-l", "/usr/src")
else:
   # parent process continues here
```

שאלה 2 [12%]

תאר ארבעה תפקידים מרכזיים של מערכת ההפעלה הקשורים לניהול **מערכות ההגנה והבטיחות** של מערכת המחשב (Protection and Security) ותאר את סוגי הבעיות שאיתם מערכת ההפעלה צריכה להתמודד.

תשובה:

א. **הגנה על משאבי המערכת (Protection):** על מערכת ההפעלה להגן על משאבי מערכת המחשבת מפני נזקים כגון שימוש לא מבוקר ביחידת העיבוד (נסיון לנצל את המעבד יתר על המידה), כונני דיסקים (נסיון לכתוב על שטחים גדולים ממה שיש, נסיון של תהליך אחד להשתלט על הדיסק לפרק זמן ארוך מדי) וכו' – מושג באמצעות הבחנה בין שני מצבי עיבוד (kernel mode, user mode)

בטיחות (Security) מתחלק למספר תפקידים עקריים:

ב. **אימות משתמשים (user authentication):** על מערכת ההפעלה לזהות כל גורם המבקש להשתמש במערכת ולא לאפשר לו שימוש במערכת במידה והזיהוי שלו נכשל. גורמים שזוהו בהצלחה יוכלו להשתמש במערכת בהתאם לרמת הייחוס (privillege) שלהם (user, administrator, וכו').

ג. **הגנה על משאבי המערכת מפני גורמים שאין להם את ההרשאות המתאימות:** משתמש x לא יוכלו להסיר, לכתוב או אפילו לקרוא קובץ השייך למשתמש y, אם המשתמש y לא נתן הרשאה מתאימה לכך. משתמש לא יוכל לנצל שטח דיסק גדול מזה שהוקצה לו על ידי המערכת.

ד. **הגנה על מערכת הזיכרון** מפני פלישה של תהליכים הפועלים במקביל אחד לשטח הזיכרון של חברו. בפרט, מערכת ההפעלה חייבת להגן מפני כל נסיון של תהליך משתמש לחדור ולזהם את שטח הזיכרון שלה עצמה.

ה. **הגנה על המערכת מפני נסיונות תקיפה מבחוץ:** נסיונות למנוע פעילות תקינה של המערכת באמצעות התקפות תקשורת בלתי פוסקות, החדרת סוסים טרויאניים ונסיון להשתלט על יחידת העיבוד ומערכת ההפעלה עצמה, וכדומה

שאלה 3 [10%]

בנסיבות רבות יש צורך בהעתקת נתונים ממערכת הזיכרון (RAM) למערכת הקבצים (דיסק קשיח) ובחזרה.

א. נסה להסביר מדוע זה לא כדאי להטיל משימה כזו על יחידת העיבוד המרכזית (CPU), ומהן הבעיות העיקריות שבהטלת משימה כזו על ה-CPU?

ב. הסבר מהי השיטה המקובלת להתגבר על הבעיות שתיארת בסעיף הקודם על ידי שילוב נכון בין מערכת הפעלה וחומרה מתאימה לשם העברה יעילה של הנתונים מה-RAM לדיסק (בשני הכוונים).

תשובה:

א. העתקת נתונים מדיסק לזיכרון או מהזיכרון לדיסק, היא פעולה פשוטה ואיטית מאוד, ולכן העסקת יחידת העיבוד בפעולה כזו היא בזבוז אדיר של המשאב הכי חשוב במערכת. בכל זמן נתון ישנו תור של מאות או אלפי תהליכים שונים הזקוקים ליחידת העיבוד המרכזית לשם ביצוע מטלות הרבה יותר חשובות ומורכבות, בעדיפות גבוהה יותר מהעתקת נתונים פשוטה.

ב. צורך זה הוביל להוספת בקר ה-DMA על לוח המחשב אשר עליו מוטלת מלאכת ההעתקה של נתונים מהזיכרון לדיסק ובחזרה. רכיב ה-DMA משמש כעין "סוכן משנה" (או "קבלן משנה") של יחידת העיבוד המרכזית. בכל פעם שתהליך מסוים צריך לקרוא נתונים מהדיסק למערכת הזיכרון, או לכתוב נתונים מהזיכרון לדיסק, מערכת ההפעלה מעבירה את המשימה הזו לבקר ה-DMA, התהליך עובר למצב המתנה (waiting) ומפנה את המעבד עבור תהליכים אחרים שזקוקים לו בדחיפות. כאשר רכיב ה-DMA מסיים את משימת ה-I/O הוא מודיע על כך (באמצעות פסיקה מתאימה) למעבד ואז מערכת ההפעלה עשויה להחזיר את התהליך הממתין (לכשיגיע תורו) לשוב להמשך פעולה על המעבד.

שאלה 4 [10%]

בחברה ידועה מאוד לייצור רכיבים אלקטרוניים, צוות פיתוח תוכנה נתקל בבעייה של העברת נתונים מתהליך יצרן P לתהליך צרכן C. בכדי להימנע משינויים לא פשוטים הדרושים בקוד של שתי התוכנות המתאימות (כמו צורך להשתמש ב-Pipe או מנגנון דומה לשיתוף זיכרון), חבר בכיר בצוות הציע שהתוכנית P תכתוב את כל הנתונים לקובץ זמני data.txt, ולאחר שתסיים לכתוב את הנתונים, התוכנית C תקרא אותם מהקובץ data.txt. מהם לדעתך הבעיות הצפויות לחברה עקב קבלת ההצעה כזו? מנה לפחות שלושה חסרונות משמעותיים שיש בשימוש ברעיון זה לעומת שימוש ב-Pipe.

תשובה:

ייתכנו מספר בעיות לא פשוטות ומגוונות עם הסידור הזה:

א. בזמן שהיצרן P כותב לקובץ data.txt, הצרכן C מתבטל לחלוטין ומחכה עד לסיום הכתיבה של הקובץ. התסריט הרע הוא שבכדי ליצור את הנתונים לקובץ data.txt, היצרן עשוי להידרש לבצע חישובים מורכבים וארוכים, שעשויים לעכב עוד יותר את כתיבת הקובץ (שבמהלכם הצרכן נדרש לחכות בבטלה מוחלטת ובסבלנות עד לסיום כתיבת הקובץ). זה כמובן פוגע במקביליות ומאט את פעולת המערכת. בשימוש ב-Pipe, הצרכן C יכול לקרוא את הנתונים באותו הזמן שהיצרן רושם אותם ולבצע את עבודתו במקביל לעבודת היצרן, ובכך להאיץ את פעולת המערכת.

ב. באופן דומה, בזמן שהצרכן קורא את הנתונים מהקובץ, היצרן יושב בבטלה מוחלטת ומחכה עד לסיום עבודת הצרכן (שעשוי גם להתעכב אם הוא גם מעבד אותם ומבצע חישובים ארוכים!). המשמעות המעשית של שני הסעיפים האחרונים היא בפשטות: בשימוש בקובץ היצרן והצרכן פועלים באופן טורי, בעוד שעם Pipe הם פועלים באופן מקבילי! (מה שעלול להתבטא בזמן עבודה כפול ובאי-ניצול משאבים).

ג. כתיבה לקובץ על דיסק היא איטית באופן משמעותי מאשר כתיבה ל-Pipe בזיכרון (בפרט אם ה-Pipe שוכן בזיכרון מטמון כגון L1 או L2). וזה כמובן נכון גם עבור קריאת הקובץ לאחר מכן על ידי הצרכן – קריאה מהזיכרון מהירה מאוד באופן משמעותי מקריאה מדיסק.

ד. העניין מתחיל להיות קריטי אם כמות הנתונים שהיצרן P צריך לכתוב גדולה מאוד. ייתכן והשטח הפנוי על הדיסק אינו מספיק, וייתכנו בעיות דיסק אחרות שעשויות להכשיל את המערכת (מחיקת או שינוי הקובץ על ידי משתמש אחר, נעילת הדיסק על ידי תוכנת הגנה או תוכנת תחזוקה שוטפת, וכדומה). בכתיבה ל-Pipe לעומת זאת, אין סכנה כזו מאחר ושטח הזיכרון של ה-Pipe הוא קבוע (בדרך כלל 4K !), ובכל פעם שהוא מתמלא, מערכת ההפעלה עוצרת את היצרן עד שהצרכן פינה חלק מסוים מהנתונים מהצינור. באופן כזה, שטח העבודה הנדרש על ידי שימוש ב-Pipe קטן באופן משמעותי מאוד מהשטח הנדרש על ידי עבודה עם קובץ.

ה. קיימת גם בעייה לא פשוטה בתיאום בין היצרן והצרכן לגבי איך להודיע על סיום כתיבת הקובץ? ואיך היצרן ידע מתי הצרכן סיים את הקריאה? (ובכך יאפשר לצרכן לכתוב את המנה הבאה) כך שהפיתרון של העברת נתונים דרך קובץ אינו משחרר את היצרן והצרכן מהצורך לתכנן מנגנון תיאום שעשוי להיות לא פחות פשוט ממנגנון של Pipe.

ו. הבעיות מסתבכות יותר אם היצרן צריך לשרת כמה צרכנים בו-זמנית. במקרה כזה עבודה דרך קבצים הופכת להיות מסובכת מאוד (איך שני תהליכים שונים יכולים לשתף קובץ פתוח משותף???) והבעיות מתארכות יותר ויותר ...

שאלה 5 [8%]

א. תאר בקצרה את מנגנון ההרשאות לקריאה, כתיבה, והפעלה של קבצים במערכת ההפעלה Unix.

ב. למשתמש u12345 יש קובץ בשם top_tasks.py בחשבון ה-Unix שלו. רשום פקודת Unix מתאימה שהוא צריך להפעיל על הקובץ, אשר תאפשר לכל שאר המשתמשים להפעיל את הקובץ אך תחסום אותם מפני אפשרות לקרוא או לכתוב לקובץ זה.

תשובה:

א. לכל קובץ (רגיל או ספריה) יש 9 סיביות הגנה עבור אפשרות קריאה, כתיבה, או הפעלה: שלוש סיביות עבור המשתמש עצמו (שמסוגל לנעול את הקובץ לקריאה, כתיבה, או הפעלה אפילו עבור עצמו), שלוש סיביות עבור כל המשתמשים בקבוצה אליה הוא משתייך (כל משתמש משתייך לקבוצה מסוימת), ושלוש סיביות עבור כל שאר המשתמשים. רק בעל הקובץ בעצמו מסוגל לשנות את כל אחת מתשע הסיביות הללו. הפקודה העיקרית שבאמצעותה ניתן לשלוט על מצב הסיביות היא chmod.

Permissions

user			group			other		
r	w	x	r	w	x	r	w	x

ב. הפקודה שנדרש לבצע כאן היא: `chmod go+x, go-rw top_tasks.py`
תשובה אלטרנטיבית שניה: `chmod g=x,o=x top_tasks.py`
תשובה אלטרנטיבית שלישית קצרה: `chmod go=x top_tasks.py`

מספר תלמידים רשמו את התשובה: `chmod 711 top_tasks.py`
למרות שפקודה זו מבצעת את הנדרש לגבי group ולגבי other היא גם כופה את הסיביות 111 על ה-user, מה שאינו מתבקש בשאלה (ייתכן למשל שהמשתמש בחר בסיביות 101 עבור עצמו ואינו מעוניין בשינוי ל-111)

שאלה 6 [6%]

תאר במשפט אחד מה בדיוק מתבצע על ידי תוכנית Python הבאה:

```
import os, time

dir = "C:/BRAUDE/OS"
t = time.time()
for path, dirs, files in os.walk(dir):
    for f in files+dirs:
        p = os.path.join(path,f)
        if os.path.getmtime(p) > t-60:
            print p
```

הערה: הפונקציה **os.path.getmtime** מחזירה את זמן השינוי האחרון של קובץ (בשניות משנת 1970).

תשובה:

התוכנית מדפיסה את כל הקבצים והתיקיות הכלולים בתיקיה C:/BRAUDE/OS (רקורסיבית), אשר השתנו בדקה האחרונה.

שאלה 7 [5% + 5% bonus]

תהליך A במערכת Linux צריך להוליד שני תהליכים חדשים B1 ו-B2. התהליך B1 צריך למסור את ההודעה "Hello Brother" לתהליך B2. תן הסבר מילולי (אין צורך בקוד) לפעולות הנדרשות להשגת מטרה זו באמצעות קריאות מערכת כמו fork(), pipe(), read(), write(), וכדומה.

שאלת בונוס: תוספת בונוס של 5 נקודות יינתנו לכל מי שירשום תוכנית Python (או C) שתבצע את התסריט הנ"ל במלואו ובמדויק.

הערה: התהליכים B1, B2 הם ילדים של התהליך A, אך ביניהם אין שום קשר של אבא/בן.

תשובה:

```
import os
rside, wside = os.pipe()
pid1 = os.fork()

if pid1 == 0:  # Child_1 process (B1)
    # Child_1 sends a message to his brother (Child_2)
    os.write(wside, "Hello Brother!")
else:  # Parent process
    pid2 = os.fork()
    if pid2 == 0:  # Child_2 process (B2)
        msg = os.read(rside, 100)
        print "Message from my brother:", msg
```

א. תהליך האב A מוליד את תהליך הבן B1 על ידי ביצוע הפקודה pid1=os.fork(). לאחר מכן יש לנו שני תהליכים שרצים במקביל: A, B1

ב. התהליך B1 (pid1==0) כותב לצינור את המשפט "Hello Brother" ומסיים.

ג. תהליך האב מבצע os.fork() נוסף ומוליד ילד שני B2 ומסיים.
ד. הילד השני B2 קורא את ההודעה מהצינור ומדפיס אותה.
יש לזכור שמאחר שהצינור pipe נוצר בשלב מוקדם, הרי שהוא משותף לכל שלושת התהליכים! זה מאפשר לבן B1 למסור הודעה לבן B2 מעל גבי אותו צינור.

שאלה 8 [10%]

הפלט של הפקודה **/usr/bin/ping** במערכת ההפעלה **Linux** נראה כך:

```
Linux> /usr/bin/ping -c 7 www.twitter.com
PING twitter.com (199.16.156.70) 56(84) bytes of data.
64 bytes from 199.16.156.70: icmp_seq=1 ttl=55 time=165.1 ms
64 bytes from 199.16.156.70: icmp_seq=2 ttl=55 time=181.4 ms
64 bytes from 199.16.156.70: icmp_seq=3 ttl=55 time=132.3 ms
64 bytes from 199.16.156.70: icmp_seq=4 ttl=55 time=207.0 ms
Request timed out
64 bytes from 199.16.156.70: icmp_seq=6 ttl=55 time=171.8 ms
64 bytes from 199.16.156.70: icmp_seq=7 ttl=55 time=152.0 ms
```

התוכנית /usr/bin/ping שולחת חבילה של 64 בתים לאתר המבוקש ומודדת את הזמן שלוקח לחבילה להגיע ליעד ולחזור (עם אישור קבלה) למקור. זמן זה נקרא זמן RTT (Round Trip Time).
על בסיס צורת הפלט הזו:

א. רשום פונקציית Python, בשם **max_trip_time(site, n)** המקבלת שם של אתר אינטרנט (site) ומספר שלם n ומחשבת את זמן ה-RTT המקסימלי עבור n דגימות.
ב. עשה שימוש בפונקציה os.fork() בכדי להפעיל את הפונקציה **max_trip_time(site, 10)** במקביל עבור שני האתרים: www.youtube.com, www.twitter.com , ומדפיסה את התוצאה למסך. רשום תוכנית Python קצרה שמבצעת תסריט זה.

תשובה:
התוכנית הבאה מסבירה הכל בצורה ברורה

```
from subprocess import Popen, PIPE
import os, time

def max_trip_time(website, n=8):
    cmd = ['ping', '-c', str(n), website]
    p = Popen(cmd, stdout=PIPE)
    max_time = -1
    counter = 0
    while not p.stdout.closed:
        line = p.stdout.readline().strip()
        counter += 1
        if counter == n+1: break
        if not "64 bytes from" in line:
            continue
        fields = line.split()
        for field in fields:
            if "time=" in field:
                t = float(field.strip('time='))
                if t > max_time:
                    max_time = t

    p.terminate()
    return max_time

site1 = 'www.youtube.com'
site2 = 'www.twitter.com'

pid = os.fork()

if pid == 0:     # Child process
    print site1, max_trip_time(site1, 5)
else:
    print site2, max_trip_time(site2, 5)
```

שאלה 9 [8%]

עיין היטב בקוד C הבא וענה על השאלות הבאות:

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

void main()
{
    int x=5;
    if (fork() == 0) {
        printf("pid=%d\n", getpid());
        exit(0);
    } else {
        while (x++)
            x = x-1;
    }
}
```

א. כמה תהליכים נוצרים לאחר הפעלת התוכנית? ומהו זמן הריצה (המוערך) הצפוי של כל אחד מהם?

ב. מה בדיוק מתבצע על ידי התהליכים שמנית בסעיף הקודם?

ג. באיזה מצבי תהליך הם עשויים להימצא עד לסיומם? תן תסריטים מתאימים לכל המצבים האפשריים.

תשובה:

א. מאחר ומתבצעת קריאה אחת ל-fork הרי שמיד לאחר מכן התהליך מתפצל לשניים: אב ובן.

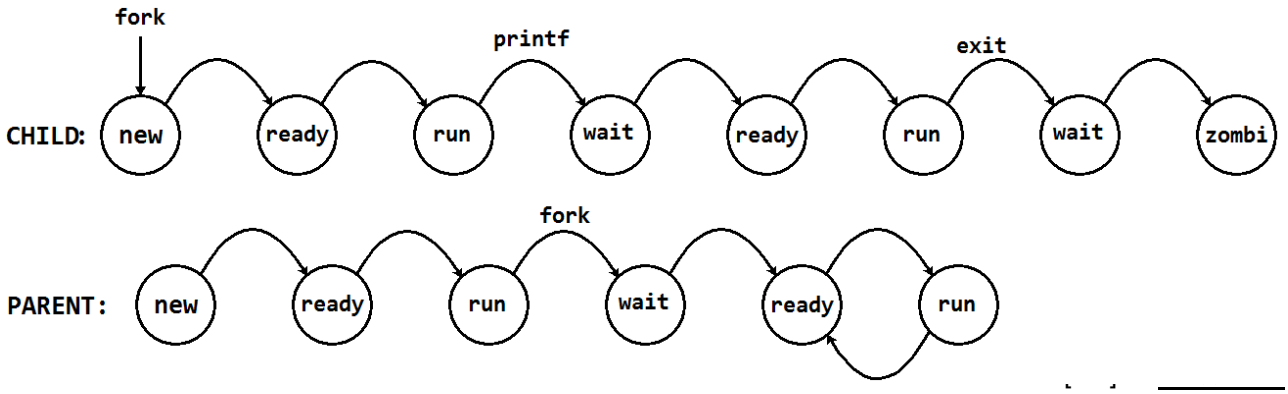
ב. תהליך הבן מדפיס את מספר התהליך שלו ומבקש לסיים על ידי קריאה ל-exit(0) (מערכת ההפעלה אינה חייבת לכבד בקשה זו אם תהליך האב עדין חי). תהליך האב נכנס ללולאה אינסופית (שבה x "מתנדנד" בין הערך 5 לערך 6 ללא עצירה)

ג. **תהליך הבן:**

a. מתחיל במצב ready (כמו כל תהליך שמתחיל)
b. עובר למצב running (זה המצב היחיד שבא אחרי ready!)
c. ומייד לאחר מכן עובר למצב waiting מאחר וביצע פעולת I/O (הדפסה למסך).
d. לאחר ההדפסה, התהליך עובר שוב למצב ready (מאחר שכרגע יצא ממצב waiting)
e. עובר למצב running וניגש להפעיל את הקריאה exit(0)
f. לאחר הפעלת exit(0), התהליך עובר למצב waiting (כי exit היא קריאת מערכת!)
g. ואז מייד עובר למצב zombie (שנקרא גם terminated או defunct), מאחר והאבא שלו נמצא בלולאה אינסופית ואינו צפוי לסיים בקרוב, אך לא ביצע wait עבור הבן היחיד שלו! עקב כך הבן לא יוכל לסיים כל עוד האב לא סיים.

תהליך האב:

a. תהליך האב מתחיל במצב ready כמו כל תהליך נורמלי
b. עובר למצב running (זה המצב היחיד שבא אחרי ready!)
c. ולאחר ביצוע fork נכנס למצב waiting (כי fork היא קריאת מערכת שמעבירה שליטה למערכת ההפעלה הולכת לשכפל את התהליך!)
d. לאחר ה-fork התהליך עובר שוב למצב ready (כרגע יצא מ-waiting)
e. ואז נכנס למצב running תוך כדי לולאה אינסופית (שבה הוא מזיז את x בין 5 ל-6)
f. בכל פעם שמנת המעבד שלו מסתיימת (15 עד 60 מילי-שניות) תהליך האב עובר למצב ready ולאחר פרק זמן קצר חוזר שוב למצב running (וכך הוא מתנדנד לו בין שני מצבים אלה עד שמישהו יעצור אותו ...)



לתהליך P יש 8 חוטים (threads) עצמאיים לחלוטין, שבדרך כלל רצים כולם במקביל באופן יעיל על מחשב בעל 8 ליבות ועקב כך התהליך P רץ במהירות גדולה פי כמה וכמה מאשר ללא חוטים. עקב קיצוצים בתקציבי מיחשוב

וציוד, האירגון נאלץ לעבור למחשבים בעלי ליבה אחת בלבד. מנהל יחידת המיחשוב חשב שבתנאים של ליבה יחידה, כבר אין טעם בעבודה עם חוטים ועדיף לעבור לתיכנות פשוט ללא חוטים וללא תיכנות מקבילי.
האם ההנחה הזו של המנהל נכונה? או שהיא מוטעית? האם לתיכנות מונחה חוטים יש עדיין יתרונות גם בסביבה מחשב בעל ליבה אחת? אם כן, מנה את היתרונות השונים שיש לדעתך וספק דוגמא מוחשית ברורה שמוכיחה את הייתרונות שמנית. אם אתה סבור שהנחת המנהל נכונה, ספק נימוק ברור מדוע לדעתך אין טעם לתיכנות מונחה חוטים במחשב בעל ליבה אחת?

<u>תשובה:</u>

הנחת המנהל שגויה לחלוטין. גם עם ליבה אחת, לתיכנות באמצעות חוטים יש ייתרונות רבים וקריטיים!!

א. מרבית התהליכים מבלים את מרבית זמנם בציפייה בכתיבה או קריאת נתונים מהדיסק, תגובה מהמשתמש, או בכתיבה/קריאה נתונים מהרשת. השימוש ביחידת העיבוד הוא בדרך כלל קטן מאוד (פחות מאחוז אחד של הזמן). חלוקת העבודה לחוטים מאפשרת הרצה של משימות על המעבד בזמן שמשימות אחרות מחכות לקלט ופלט. בזמן בו חוט אחד מחכה לקלט, חוט אחר יישתמש במעבד, ובכך עבודת התהליך תתקצר באופן משמעותי גם על ליבה אחת.

ב. גם במערכת מחשב מיושנת בעלת ליבה אחת, יש מידה מסוימת של מקביליות ברמת החומרה (למשל רכיב ה-DMA מסוגל לבצע פעולות העברת נתונים מהזיכרון לדיסק ובחזרה באופן עצמאי ובלתי תלוי במעבד שיכול לבצע משימה אחרת לגמרי באותו זמן, ובמערכות מסוימות קיימים 4 או 8 רכיבי DMA שונים!). ריבוי חוטים עשוי לנצל את הייתרון הזה בכדי להאיץ את פעולות הקלט והפלט באופן משמעותי גם כשברקע יש לנו רק ליבה אחת.

ג. המבנה הלוגי הטבעי של פתרונות תוכנה מורכב בדרך כלל ממספר משימות עצמאיות שיכולות להתבצע במקביל, אך צורת התיכנות המסורתית אינה מאפשרת זאת ומריצה אותם באופן סידרתי אחת אחר השניה ובכך מתבזבז זמן יקר ומשאבים לא מנוצלים בכל פעם שמשימה מסוימת "נתקעת" לזמן ארוך בציפיה לקלט וכל שאר המשימות מחכות לה בתור עד שתסיים.

ד. השיטה הכי נפוצה לפשט את תהליך התיכנון של תוכנה היא על ידי עקרון "הפרד ומשול" (divide and conquer). כלומר, מבנה פשוט וקל לתיחזוק של תוכנה מתקבל באופן טבעי על ידי חלוקתה למספר חלקים שהקשר ביניהם פשוט ככל האפשר. כלומר גם תהליך כתיבת תוכנה תומך באופן מובהק בריבוי חוטים.

ה. הדוגמא הכי מובהקת שפגשנו במהלך הקורס קשורה לתוכנת התיקשורת ping. פקודה זו מקבלת שם של אתר ברשת ודוגמת את הזמן הממוצע הדרוש בכדי להגיע אליו ולחזור. צריכת ה-CPU במקרה זה היא מזערית ביותר – מרבית הזמן מתבזבז על המתנה לחבילת התיקשורת שיצאה לאתר הרחוק ולחזרתה. בזמן ההמתנה הממושך מאוד (במונחים של זמן מעבד) הצלחנו להשחיל עוד כמה עשרות חוטים נוספים של הפקודה ping ולקבל הספק זהה של דגימות ping פי 20 או 30 באותו פרק זמן על ידי חוטים (גם על מחשב בעל ליבה אחת!). מה שמפריך לחלוטין את הנחת המנהל למעלה.
בכדי לחדד : נניח שפעולת ping שגרתית משתמשת ב-1ms מעבד ומחכה 100ms לחבילה. בתיכנות טורי רגיל, ביצוע של 50 פעולות ping היה מצריך מעל 5 שניות (ליבה אחת). אם לעומת זאת נשתמש ב-50 חוטים שונים, אפילו על ליבה אחת, כל פעולת ping תשתמש במעבד 1ms ותעבור מייד למצב המתנה (waiting) ובכך המעבד יתפנה לפעולת ה-ping הבאה בזמן שהפעולה הקודמת מחכה בתור לחבילה שתשוב. בצורה זו ניתן להריץ את כל החוטים ב-50ms וכל החוטים ימתינו במקביל עוד 100ms, כך שבאופן תאורטי, כל 50 פעולות ה-ping ירוצו ב-150ms. זה כמובן פי 30 מאשר בתיכנות טורי!

ו. החלטת המנהל עשויה להיות נכונה במקרה קיצוני אחד שבו כל החוטים מבצעים פעולות חישוב בלבד (כלומר שימוש של 100% על המעבד). במקרה הקיצוני כזה, ההפרדה של התהליך ל-8 חוטים עשויה להאט מעט (אבל לא ממש) את עבודת התהליך. אך כאמור, זהו מקרה יחסית נדיר, מאחר ובמרבית המקרים חוטים מבלים זמן ארוך בפעולות I/O.

שאלה 11 [12%]

עיין היטב בתוכנית C הבאה xfactor.c:

```
#include <dirent.h>
#include <stdio.h>

static int xfactor(const char *path) {
    DIR *dp ;
    struct dirent *de ;
    dp = opendir(path) ;
    int x = 0 ;
    while ((de = readdir(dp)) != NULL)
        x++ ;

    (void) closedir(dp);
    return x ;
}

void main(int argc, char *argv[]) {
    int x ;
    x = xfactor(argv[1]) ;
    printf("%d\n", x) ;
}
```

א. הסבר מהו הארגומנט שהפונקציה xfactor מקבלת ומה בדיוק מתבצע בגוף הפונקציה?

ב. הסבר את המשמעות של המשתנים path, dp, de.

ג. האם הפונקציה xfactor היא קריאת מערכת (system call)? הסבר את תשובתך.

ד. לאחר קומפילציה של קוד זה, תתקבל תוכנית בשם xfactor.
תן דוגמא פשוטה להפעלת התוכנית xfactor מתוך שורת הפקודה (command line) ודוגמא אפשרית לתוצאה של הפעלה כזו.

תשובה:

א. הארגומנט path שהתוכנית מקבלת הוא מצביע (pointer) למחרוזת המייצגת מסלול של תיקיה (כמו: "/home/samyz/os/proj5"). הפונקציה פותחת את התיקיה (כמו פתיחת קובץ רגיל). התכולה של תיקיה היא רשימה של הקבצים הכלולים בה. הפונקציה xfactor עוברת על כל הרשומות בספריה ומדווחת כמה קבצים כלולים בתיקיה.

ב. כאמור המשתנה path מציין מסלול לתיקיה (כמחרוזת טקסט), המשתנה dp הוא מצביע למבנה נתונים של תיקיה (directory pointer). מבנה נתונים של תיקיה מורכב מרשימה של רשומות שכל אחת מהם היא מצביע לקובץ רגיל או לתיקיה נוספת. המשתנה de מציין מצביע לרשומה כזו (directory entry).

ג. ברור שהפונקציה xfactor **<u>אינה</u>** קריאת מערכת מאחר וכרגע בנינו אותה והיא אינה כלולה בגרעין מערכת ההפעלה!

ד. לאחר קומפילציה נוכל להריץ את התוכנית xfactor באופן הבא: שמשמעותו שבתיקיה /home/samyz/os/proj5 ישנם 17 קבצים ותיקיות

```
~u12345> xfactor /home/samyz/os/proj5
17
```

הערה: תוכנית זו היא וריאציה קלה על תרגיל 5 בפרוייקט 5 (dirtest.c, listfiles).