מערכות שרת לקוח ותכנות מקבילי 31666

פתרון מבחן סופי מועד א, סמסטר ב' תשע"ג, 12/07/2013

שאלה1 [10%]

תאר בקצרה שלושה מאפיינים עקריים של מערכת הפעלה מבוזרת (distributed computing system)

פתרון:

א. מערכת מבוזרת היא למעשה מערכת תוכנה הבנויה מעל רשת מחשבים

ב. מערכת התוכנה **מסתירה את ריבוי המחשבים** ברשת מהמשתמש

ג. מנקודת המבט של המשתמש, המערכת המבוזרת נראית כמו מחשב יחיד

ד. מערכת הקבצים במערכת מבוזרת פרושה על פני מספר גדול של דיסקים השייכים למחשבים שונים ומרוחקים גאוגרפית. במקרים מיוחדים (Hadoop), קובץ יחיד עשוי להיות פרוש על פני כמה דיסקים מרוחקים גאוגרפית.

ה. **יתירות (redundancy)**: כשל זמני של דיסק או אפילו מכונה שלמה במערכת אינו משבית את המערכת המבוזרת מאחר והתפקידים של הדיסק או המעבד שנכשל עוברים למחשבים ודיסקים אחרים. קבצים ניתנים לשיחזור גם לאחר נפילה או אובדן סופי של דיסקים בהם חלקי הקובץ שכנו (בדומה למערכת RAID)

ו. משימה אחת עשויה להתפרק למספר רב של תהליכים, תתי-תהליכים (או חוטים) שרצים במקומות מרוחקים (אפילו שתי יבשות שונות), ושמתקשרים ביניהם באמצעות פרוטוקול TCP/IP או פרוטוקולים אחרים. כל המורכבות הזו שקופה עבור המשתמש שאינו מודע כלל לקיומה.

שאלה 2 [10%]

מנה בקצרה את ההבדלים העקריים שבין פרוטוקול TCP ופרוטוקול UDP

פתרון:

פרוטוקול TCP יש את התכונות הבאות:

א. מונחה תיקשורת (connection oriented): שני הצדדים חייבים קודם להקים קו תיקשורת ביניהם לפני שיוכלו להחליף חבילות, וחייבים לסגור את הקו לאחר סיום ההתקשרות.

ב. משלוח חבילות אמין מהשולח לנמען (שום חבילה לא תלך לאיבוד ללא ידיעה מלאה של שני הצדדים)

ג. בדיקת שגיאות באמצעות איזון קצב שליחה קבלה (אלגוריתם חלון זז) ובדיקת checksum.

ד. סידרור (sequencing) של התוכן על פני החבילות המרובות

ה. עקב כל הנ"ל, פרוטוקול TCP איטי ביחס לפרוטוקול UDP.

ו. ראש חבילה = 20 בתים מינימום

פרוטוקול UDP יש את התכונות הבאות:

א. משתמש ישירות בחבילות IP ללא עלויות פתיחה, אחזקה, וסגירה של קו תיקשורת

ב. מהיר מאוד ויעיל עבור שידור בו-זמני למספר רב של נמענים

ג. יש רק בדיקת checksum מינימלית ללא וידוא הגעה או קבלת אישור מהנמען

ד. אין סידרור (sequencing) חבילות.

ה. מהירות שליחה ויעילות גבוהה בהרבה מ-TCP (אך לא אמינה)

ו. ראש חבילה = 8 בתים בדיוק

הערה: זהו רק אופן אחד שבו ניתן לרשום את ההבדלים בין שני הפרוטוקולים, וייתכן שיש עוד הבדלים שוליים שלא נכללו. ניקוד מלא יינתן עבור ההבדלים העקריים המופיעים ברשימה.

שאלה 3 [16%]

נתונה חבילת IP הבאה (IP datagram) בצורה hexadecimal.

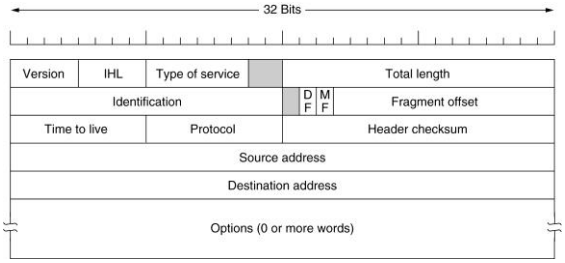
```
45 00 00 1c
92 cc 80 93
38 06 00 00
92 95 ba 14
a9 7c 15 95
2a 71 9f 57
b0 49 81 c6
```

א. בדוק האם חבילת IP זו מהווה חלק (fragment) מתוך חבילת IP שנשברה לחלקים (fragments) במהלך הדרך?

ב. אם תשובתך לשאלה הקודמת חיובית, חשב את גודל חבילת האם בבתים. אם לא, הסבר מדוע לא תוכל לחשב את גודל חבילת האם? תן הסבר קצר לתשובותיך (במחברת)

ג. רשום פונקציית Python קצרה בשם isFragment(dtgrm) המקבלת חבילת IP בצורה hexadecimal (מחרוזת רציפה ללא תווים לבנים) ומחזירה ערך True או False, בהתאם לכך אם החבילה קטועה או לא. לדוגמא: isFragment(dtgrm) = False

פתרון:



יש להתבונן ב-13 הסיביות האחרונות בשדה 0x8093

bin(0x8093) = 1000000010010011

לכן: fragment offset =0b10010011= 147

לכן מדובר בקטע של חבילה גדולה יותר שנשברה לחלקים.

השדה MF (More Fragments) הוא 0. כלומר זהו הקטע האחרון בשרשרת. גודל הקטע הנוכחי הוא 0x1c, כלומר 28 בתים. יש להוריד את גודל הראש (header) שהוא 20 בתים.

הגודל הכולל של כל הקטעים הקודמים הוא

147*8=1176 בתים, ולכן הגודל הכולל של חבילת האם הוא: 1184 בתים (1176+8).

```
def isFragment(dtgrm):
    binform = bin(int(dtgrm[12:16],16))
    MF = bool(binform[2])
    frag_offset = int(binform[3:16],2)
    if MF or frag_offset:
        return True
    else:
        return False
```

שאלה 4 [8%]

כשמריצים את התוכנית foo.py על ליבה אחת היא מבלה 60% מהזמן בחישובים מתמטיים שניתנים למיקבול באופן מלא, ואת יתר 40% מהזמן היא מבלה בקליטת נתונים מערוץ רשת שמנפיק אותם באופן אחיד וקבוע. זמן ריצה ממוצע על ליבה אחת הוא 50 דקות. מה יהיה זמן הריצה הטוב ביותר שניתן להשיג על ידי הרצת התוכנית באופן מקבילי מאוזן על ארבעה ליבות?

פתרון:

עקב עמימות בניסוח השאלה, ניתן להבין אותה בכמה דרכים שונות אשר חלקן התקבלו כתשובה כמעט מלאה. מתוך 50 דקות, זמן המעבד (ליבה אחת) הוא 30 דקות, ולכן על 4 ליבות במקביל ניתן לבצע את חישובי המעבד ל-7.5 דקות. אבל התוכנית זקוקה לנתוני הרשת שמונפקים בקצב קבוע במשך 20 דקות (40% של 50 דקות).

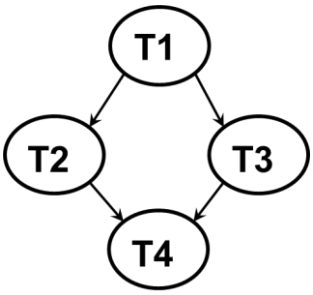
המשמעות של 20 הדקות הנוספות הוא שמידי פעם התהליך נאלץ לעצור (מצב waiting) ולחכות להגעת נתונים מהרשת. במצב של ליבה אחת, זה מצריך 50 דקות לסיום התהליך, כאשר ב-20 דקות מתוכם המעבד מובטל לחלוטין ומחכה לנתונים. כשבידנו 4 ליבות, זמן הבטלה יהיה עדיין 20 דקות, אך ניתן לפרישה בין כל 4 הליבות (5 דקות בטלה לליבה), או על ליבה אחת שתקלוט את כל הנתונים במשך 20 דקות הדרושות ושלושת הליבות האחרות יבצעו את החישובים במקביל. זמן העיבוד, 30 דקות, יתחלק באופן שווה בין 4 ליבות – 7.5 דקות לליבה, או בין שלוש ליבות (10 דקות לליבה) לכן זמן הריצה הטוב ביותר שנוכל לצפות לו במקביל על 4 ליבות הוא 20 דקות. כך שפחות מ-20 דקות לא ניתן להשיג, אך באופן תאורטי בתנאים אידאליים, 20 דקות עשוי להיות מספיק בהחלט. למרות זאת יתקבלו באופן כמעט מלא גם תשובות של 27.5 דקות ו-12.5 דקות, עקב עמימות שמאפשרת להבין את הבעייה באופן שונה מהמצופה.

בכדי להשתכנע בנכונות הטיעון המופשט כדאי להיעזר בדוגמא הקונקרטית הבאה: נניח כי נתוני הרשת הם סדרה של 20 מספרים (נניח טמפרטורות) שמגיעים בקצב של אחד לדקה. נניח גם כי לאחר הגעת מספר n מהרשת מתבצע חישוב foo(n) שצורך 90 שניות. כלומר נדרשים 30 דקות בכדי לבצע 20 חישובי foo(n), ו- 20 דקות נוספות שצריך לחכות לקליטת כל 20 המספרים (התהליך הוא פשוט: מחכים דקה להגעת מספר n ולאחר מכן מבצעים חישוב foo(n) במשך דקה וחצי). יוצא לכן שדוגמא זו מצייתת לתנאים של הבעייה שלנו: 30 דקות חישוב ו-20 דקות ציפייה לקליטת נתונים.

ננסה לחשוב עכשיו מה יקרה עם 4 ליבות? על פי הנתון, אפשר לבצע כל חישוב foo(n) במקביל. נוכל למשל ליעד ליבה אחת לקליטת 20 המספרים (מה שלוקח 20 דקות) ואת שלושת הליבות האחרות לבצע את החישובים foo(n). כל ליבה תבצע בין 6 ל-7 חישובים, מה שיקח בין 9 ל-10.5 דקות, וזה ניכנס לתקציב של 20 דקות לליבה. דוגמא זו אינה אופטימלית כי החישובים עבור הנתון 19 והנתון ה-20 ייגלשו מעבר ל-20 דקות (21.5 דקות ליתר דיוק) אך כמובן ניתן למצוא דוגמאות שבהן 20 דקות יספיקו בהחלט (למשל אם כל החישובים מתבצעים רק עבור 15 הנתונים הראשונים, 2 דקות לנתון, בעוד שחמשת הנתונים האחרונים נדרשים רק לתיעוד ואינם דורשים חישוב כלשהו). הרעיון הוא שהמיספר 20 דקות הוא מינימום מוחלט שלא ניתן לרדת מתחתיו, למרות שמבחינה מעשית תמיד יש תקורה מסוימת בנוסף לכך.

ניתן לצפות בסימולציה של Python למצב כזה בקישור: problem4.py

שאלה 5 [10%]



לפניך גרף המתאר את התלויות שבין ארבעה חוטים שונים: T1, T2, T3, T4 המשתייכים לאותו תהליך. חץ בין חוט T1 לחוט T2 מציין שחוט T1 חייב להסתיים בכדי שחוט T2 יוכל להתחיל. הנח שארבעת החוטים עשויים להתחיל בכל סדר שהוא. השתמש בסמפורים (Semaphores) בכדי לכפות באופן מוחלט את מערכת התלויות הללו.

א. עליך להגדיר מספר סמפורים ולאתחל אותם לערכים המתאימים

ב. עליך לקבוע מתי לנעול ומתי לשחרר את כל אחד מהסמפורים שהגדרת בסעיף הקודם

ג. את פעולות הנעילה והשחרור תוכל להוסיף רק בתחילתו או סופו של חוט. תאר את תשובתך באופן הבא: "לפני חוט Tx יש להוסיף שורת קוד line1 ולאחר חוט Ty יש להוסיף שורת קוד line2, וכו'..."

פתרון:

```
sem2a = Semaphore(0)
sem3a = Semaphore(0)
sem2b = Semaphore(0)
sem3b = Semaphore(0)

# After T1:
sem2a.release() ; sem3a.release()
# Before T2:
sem2a.acquire()
# Before T3:
sem3a.acquire()
# After T2:
sem2b.release()
# After T3:
sem3b.release()
# Before T4:
sem2b.acquire() ; sem3b.acquire()
```

שאלה 6 [15%]

נתונה פונקציה packet_generator(que) המייצרת 8000 חבילות TCP בכל שניה ומכניסה אותם לתוך התור que (Queue). נתונה פונקציה packet_sender(que) המקבלת חבילת TCP מתוך התור que, ומשגרת אותה לרשת האינטרנט באמצעות כרטיס רשת בקצב של אלפית שניה לכל חבילה.
במחשב שלנו יש 8 כרטיסי רשת ומעבד בעל 9 ליבות.

א. רשום תוכנית Python שמריצה 9 תהליכים במקביל: תהליך אחד של packet_generator,ו- 8 תהליכים של packet_sender, המשתפים פעולה באמצעות תור יחיד que.

ב. אם נרצה להגביל את כמות המשלוחים הכללי למיליון חבילות ולעצור: איזה שינוי בקוד נצטרך לבצע? רשום רק את הרעיון המילולי – אין צורך לרשום קוד.

פתרון:

א.

```
from multiprocessing import Process, Queue
que = Queue()
# Launch the packet generator:
g = Process(target=packet_generator, args=(que,))
g.start()
# Launch 8 senders:
senders = [Process(target=packet_sender, args=(que,)) for i in range(8)]
for s in senders: s.start()
```

ב. ניתן לעשות זאת במספר אופנים.

רעיון 1: להוסיף סמפור sem שמאותחל ל-1000000, ובכל פעם ש-packet_sender שולח חבילה לבצע sem.acquire(). בנוסף לסמפור, יש צורך חוט מוניטור (Monitor thread) שמסיים את כל השולחים לאחר שהסמפור ננעל.

רעיון 2: אפשר להוסיף תור חדש בשם q2 שתפקידו היחיד הוא לספור את החבילות שנשלחו (על ידי כל השולחים). הגישה לתור זה צריכה להיות רק מתוך packet_sender():

```
# initialization (at global scope):
q2 = Queue()
q2.put(0)

def packet_sender(que, q2):       # q2 must be added as a second argument
   #... some code ...
   # Add this code to packet_sender body before sending a packet:
   counter = q2.get()           # pop the counter from q2
   if counter == 1000000:       # stop when you reach 1000000 packets
       q2.put(counter)          # put back counter so other processes see it
       return                   # Exit the packet_sender (ends process)
   counter += 1
   q2.put(counter)
   #... Now we can send the packet ...
   #... code ... code ... to send the packet
```

ניתן למצוא דוגמאות מלאות בקישורים הבאים: packgen1.py packgen2.py

שאלה 7 [15%]

במערכת ניטור טמפרטורת חדר יש תהליך P שרץ באופן קבוע וצריך לאגור בקובץ מרכזי את כל הטמפרטורות המדווחות על ידי 10 חיישנים שונים בנקודות שונות בחדר. החיישנים השונים דוגמים את הטמפרטורה בפרקי זמן לא סדירים, ובמקרים מסוימים עשויים להפסיק לפעול באופן אקראי. כל חיישן sensor[i] רושם את הטמפרטורה שדגם לתוך תור que[i], כאשר i=0,1,2,…,9. לתהליך P יש גישה גלובלית לכל 10 התורים האלה.

רשום קוד Python אשר על התהליך P להריץ בכדי לאגור את כל הטמפרטורות הנ"ל לקובץ יחיד temper.log, על פי כל סדר שתבחר לנכון אך בלבד שלא נאבד אף טמפרטורה מאף אחד מהחיישנים.

פתרון:

רעיון התחלתי: לעבור בלולאה על כל התורים que[i] ולשלוף מהם את הטמפרטורות לקובץ temper.log. הקושי היחיד שקיים כאן זו הוא החשש שחיישן שיצא משימוש ייתקע את כל הלולאה, למרות ששאר החיישנים ממשיכים לעבוד ולייצר טמפרטורות.

בכדי להתגבר על בעייה זו אנו נאלצים להוסיף תור חדש main_que אשר אליו ננקז את כל הטמפרטורות מכל 10 התורים באמצעות 10 חוטים בלתי-תלויים: כל חוט T[i] יטפל בתור que[i] ויעתיק את הטמפרטורות שלו לתור main_que. אם חיישן מסויים sensor[i] ייתקע או יפסיק לעבוד, זה ייתקע את החוט T[i] בלבד, כל שאר החוטים האחרים ימשיכו לעבוד כרגיל באופן בלתי תלוי:

```
# We need an additional Queue:
main_que = Queue()

# This is the function that moves temperatures from que[i] to main_que
def reader(q):
    while True:
        temperature = q.get()
        main_que.put(temperature)

# Start 10 reader threads for each que[i]
readers = [Thread(target=reader, args=(que[i],)) for i in range(10)]
for r in readers: r.start()

# Read temperatures from main_que and write them in log file
f = open("temper.log", "w")
while True:
    temperature = main_que.get()
    f.write(str(temperature) + '\n')
```

דוגמא מלאה (כולל סימולציה של כל החיישנים) ניתן לראות בקישור הבא: tempsens1.py

פיתרון נוסף לבעייה (אך בזבזני יותר) ניתן למצוא בקישור: tempsens2.py

שאלה 8 [16%]

התוכנית ichat.exe מקבלת מספר טלפון סלולרי כארגומנט יחיד (command line) ויוצרת קשר עם בעל הטלפון הסלולרי המאפשר שיחת טקסט אינטראקטיבית (chat) בין המשתמש ובין בעל הטלפון. לדוגמא:

```
C:\Braude> ichat.exe 054-9138812
* Connecting
OK connection established
SEND: Hi Dan are you awake?
PEER: Now I am ...
SEND: sorry for bothering you :-(
PEER: no problem, that's fine
SEND: Bye!
PEER: Good night
SEND: .
DONE Connection closed
C:\Braude> ichat.exe 054-9138812
* Connecting
OK connection established
SEND: Hi again Dan, forgot to say good night
ERR connection closed unexpectedly
C:\Braude>
```

א. על בסיס דוגמא זו בלבד, תאר בקצרה את פרוטוקול התקשורת של התוכנית ichat.exe.
הנח שכל ההודעות הן באורך של שורה אחת בלבד.

ב. השתמש במנגנון ה-Pipe מתוך חבילת ה-subprocess של Python בכדי לכתוב פונקצית Python בשם ichat_test()

i. שמפעילה את התוכנית ichat.exe באופן אוטומטי 100 פעמים למספר הטלפון של Dan
ii. שולחת את ההודעה "Good night" (בכל אחת מ-100 הפעמים)
iii. קולטת ומדפיסה את תגובתו של Dan להודעה זו (תגובה רגילה או ניתוק)
iv. מסיימת את התהליך על ידי ניתוק רגיל או kill
v. בין כל שני התחברויות התוכנית יושנת למשך זמן אקראי שבין 10 ל-20 שניות
vi. הנח ש-Dan זמין ועונה לכל 100 ההתחברויות, אך עשוי לנתק או להגיב לכל אחת מהן

פתרון:

א. תאור פרוטוקול:

```
*       Message code: ichat is printing info on what it is
        doing (like connecting)

OK      Success code: ichat infors on success of an operation
        like successful connection or possibly other actions

SEND:   Sender prompt for sending a one line message

PEER:   Receiver code which prefixes each message he sends

SEND: .    End connection code

DONE    Connection closed code

ERR     Error code (exceptional event happened)
```

ב. אוטומציה של 100 התקשרויות כפי שמתואר בבעייה באופן הכי מדויק יידרשו בדיקות רבות, אך לא תהיה על כך הקפדה חריגה בבדיקת תשובת התלמיד (כל עוד הרעיונות העקריים נוכחים בתשובת התלמיד):

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

def ichat_test():
    for i in range(100):
        p = Popen(["ichat.exe", "054-9138812"], stdin=PIPE, stdout=PIPE)
        line1 = p.stdout.readline()
        if not line1[0] == "*":
            p.terminate()
            continue
        line2 = p.stdout.readline()
        if not line2[0:2] == "OK":
            p.terminate()
        s = p.stdout.read(5)    # read the "SEND:" code
        if not s == "SEND:":
            p.terminate()
            continue()
        p.stdin.write("Good night\n")
        line3 = p.stdout.readline()
        if line3[0:5] == "PEER:":  # check if peer responded
            print line3
            p.stdin.write(".")
        p.terminate()
        t = random.uniform(10,20)
        time.sleep(t)
```