

תרגיל 3 – Load Balancer בסיסי

הגשה בתיקיית הבית בנובה. יש ליצור תיקיה בשם ex3 תחת התיקיה c_lab. יש להעתיק אליה את קובץ ה-README הקבוע, לתת הרשאות 755 לכל הקבצים בתוכה, ובנוסף לכך לתת לתיקיה עצמה הרשאות 777 בעזרת הפקודה (מתוך התיקיה)

`chmod 777 .`

הסטט מוודא שאלו בדיוק ההרשאות של הקבצים, וכרגיל אם הוא הדפיס משהו למסך, יש בתרגיל בעיה. תזכורת: **התרגילים נבדקים אוטומטית בעזרת מערכת לזיהוי העתקות**, כולל מול הגשות מסמסטרים קודמים.

יש לעבוד על התרגיל על מחשבי hilbert, ולא על נובה. יש להתחבר לאחד ממחשבי hilbert באופן הבא:

`ssh hilbert-XX`

כאשר XX הוא מספר דו ספרתי בין 01 ל-23.

דדליין: 16.1

לפני התרגיל – תרגלו סוקטים, לא להגשה

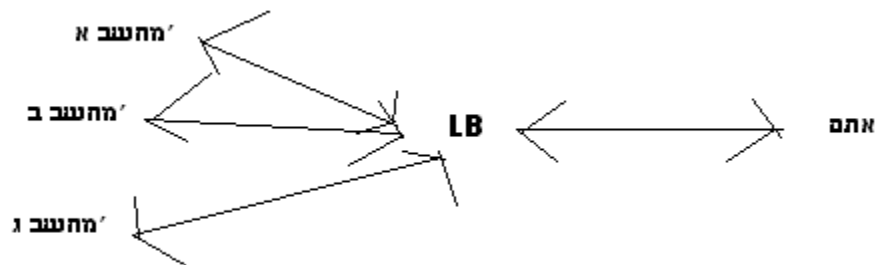
להלן מספר תרגילונים קטנים שכדאי לכם לעשות לפני שאתם צוללים לתרגיל עצמו. הם אינם להגשה, אך מומלץ מאוד לעשות אותם לפני שמתחילים את התרגיל עצמו, שעולה ברמת המורכבות משמעותית על התרגילים הקטנים.

1. התחברו לגוגל מפייתון, כמו שהדגמנו בכיתה. נסו להגיע למצב שאתם מסוגלים לעשות זאת עם מעט הצצות בדוגמאות.
2. התחברו מפייתון לפייתון אחר, כמו שהדגמנו בכיתה. גם פה, כדאי להגיע למצב שאתם מסוגלים לעשות זאת בלי להציץ כל כמה שניות בדוגמאות, אך כמובן שלא צריך להשקיע זמן בללמוד את לחש הקסמים bind-listen-accept בע"פ.
3. התחברו מ-C לפייתון שלכם. אל תדאגו אם אתם נצמדים למצגת.
4. התחברו מקוד C לקוד C אחר. אל תדאגו אם אתם נצמדים למצגת.
5. מי שמפחד מהמרת קוד סוקטים ללינוקס, מוזמן לנצל את ההזדמנות ולהמיר כבר עכשיו אחת מדוגמאות הקוד הללו ללינוקס. זה בהחלט עשוי לחסוך זמן בהמשך.

התרגיל עצמו

0. רקע ומטרה:

בתרגיל נכתוב Load Balancer בסיסי (להלן LB). LB היא תוכנית שמעבירה את התעבורה שהוא מקבל לשרתים אחרים. למשל, נניח שאתם מתחברים לגוגל ומחפשים משהו. אתם מתחברים לכתובת IP ספציפית, אבל לגוגל כידוע יש הרבה מחשבים. איך גוגל יעבירו את החישוב למחשבים השונים, ולא יאלצו לבצע את כל החישובים על המחשב אליו התחברתם? הם ישתמשו ב-LB על מנת להעביר את הבקשות שלכם למחשבים "שמאחוריו", וכך אפשר למשל לדאוג שכל חישוב מתבצע במחשב נפרד. הנה תרשים:



(התחום של load balancing הוא מן הסתם הרבה יותר מורכב מהדוגמה הפשוטה הזו, אז לא כדאי לכם להגיד שאתם מבינים ב-load balancing רק על סמך התרגיל הזה בראיונות עבודה, למשל).

בתרגיל נממש את שלושת הצדדים הבאים:

1. שרת ה-LB – יכתב על ידיכם באמצעות קוד בשפת C, ויקרא `ex3_lb`.
2. שלושת המחשבים (א', ב' וג') – יכתבו על ידיכם באמצעות שפת פייתון (אפשר גם לכתוב ב-C, אבל זה יהיה יותר מסובך), ויקראו `ex3_server`.
3. "הדפדפן שלי" – ימומש ע"י הטסטים שהכנתי, ואם תרצו להתלהב ממעשה ידיכם, ע"י דפדפן אינטרנט (כרום או פיירפוקס). אינכם צריכים לכתוב אותו בעצמכם.

1. תוכנית ה-LB (כתובה בשפת C)

- מומלץ לפתוח את קובץ הטסט האוטומטי עבור התרגיל, שיושב ב

~nimrodav/socket_ex/test.sh

ולקרא אותו תוך כדי שאתם עוקבים אחרי ההנחיות. כמובן שכרגיל, הטסט אוקף גם את ההרשאות ואת הפורמט של קובץ ה-README, וכרגיל אם הוא הדפיס דבר כלשהו למסך, צפו להפחתת נקודות.

- ה-LB עולה, תופס איזשהו פורט פנוי בעזרת bind וקורא ל-listen (מומלץ להגריל פורטים מעל 1024, ומתחת ל-64000), וכותב לקובץ server_port בתיקיה הנוכחית את מספר הפורט. הוא תופס באותו אופן פורט פנוי נוסף (אליו הדפדפן יתחבר) וכותב אותו לקובץ http_port, גם הוא בתיקיה הנוכחית. שימו לב: עבור שני הפורטים, ה-LB באמת צריך לתפוס פורט פנוי. אם הוא הגריל פורט והקריאה ל-bind נכשלה כי הפורט תפוס, הוא צריך להגריל פורט חדש, ושוב ושוב, עד שהוא תפס בהצלחה פורט. רק לאחר ששני הקבצים נכתבו, והתבצעו שתי קריאות ל-listen על שני הסוקטים הרלבנטיים, הוא עובר לחכות ש-3 השרתים יתחברו אליו. שימו לב – כל סדר פעולות אחר הוא פוטנציאל לבאג.

- בשלב הזה, הטסט האוטומטי מריץ את 3 השרתים עם ארגומנט יחיד שהוא מספר הפורט. 3 השרתים צריכים להתחבר ל-LB בפורט הזה. אם השרתים שלכם כתובים בפייתון, דאגו שהשורה הראשונה בקובץ תהיה

```
#!/usr/bin/python2.7 -tt
```

וכך הקובץ server שבעצם מכיל קוד בפייתון, יהיה ניתן להרצה בפני עצמו, כפי שהטסט מריץ אותו.

שימו לב: אם ערכתם את הקובץ בווינדוס, יש סיכוי שיש לכם את התו המרושע \r בסוף השורה, מה שגורם ללינוקס לנסות להריץ את הקובץ בעזרת התוכנית הלא קיימת python2.7\r, ואז להדפיס : No such file or directory : אני ממליץ להוסיף את השורה הזו רק כשעברתם סופית לכתוב את הקוד בלינוקס, ובכל מקרה, אם הודעת השגיאה הנ"ל מודפסת, כדאי להריץ את התוכנה dos2unix על הקובץ, מה שמסיר \r בסופי שורות.

כמו כן, הסוויץ' -tt גורם לפייתון לא להסכים להריץ את הקוד שלכם במקרה שהוא מערבב בין רווחים וטאבים, מה שמאוד מומלץ ועשוי לחסוך לכם שעות ארוכות של דיבוג מייאש.

- כעת, ה-LB מחכה שהלקוח, שמשוחק בטסט ע"י הקובץ run_test.py יתחבר אליו בפורט http_port (שהופיע במערכה הקודמת). בכל התחברות של הדפדפן, ה-LB קורא בקשת HTTP שלמה. אתם לא צריכים לדעת HTTP, רק שבקשה מסתיימת בפעם הראשונה שרצף התווים '\n\r\n' מופיע.

- ה-LB מעביר את הבקשה כולה, כולל 4 התווים בסיום, לשרת שתורו לטפל בבקשה. ה-LB מעביר את הבקשות לטיפול השרתים לפי סדר ההתחברות וחוזר חלילה, כלומר שרת 1, שרת 2, שרת 3, שרת 1, שרת 2...

- לאחר הטיפול ע"י השרת, ה LB מקבל הודעה מהשרת, קורא אותה עד שהוא מגיע לארבעת התווים "\r\n\r\n" בפעם השניה ומעביר את התשובה לדפדפן.

2. השרתים/מחשב א', ב' או ג' (כתובים בשפת פייתון)

- כשכל אחד משלושת השרתים עולה, הוא מתחבר ל-LB לפי ההנחיות, ומחכה לבקשה מה-LB.
- לאחר מכן, השרת שמקבל את הבקשה, קורא את הבקשה כולה (כזכור, קורא עד שהוא רואה את רצף התווים '\r\n\r\n'). בקשת HTTP היא בעצם רצף של טקסט. הבקשה תהיה בסגנון הדוגמה הבאה:

```
""GET /counter HTTP/1.1\r
```

```
Host: nova.cs.tau.ac.il\r
```

```
Connection: keep-alive\r
```

```
Cache-Control: max-age=0\r
```

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r
```

```
User-Agent: Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/31.0.1650.57 Safari/537.36\r
```

```
Accept-Encoding: gzip,deflate,sdch\r
```

```
Accept-Language: en-US,en;q=0.8,he;q=0.6\r\n\r\n""
```

כל מה שיש פה לא כ"כ מעניין אתכם, מלבד השורה הראשונה, שתיכף נדבר עליה. שימו לב שהשורות יהיו מופרדות ב-'\r\n'.

רק בשביל לסבר את האוזן, HTTP הוא פרוטוקול (שיטה מוסכמת להעברת מידע) טקסטואלי, שמעביר "כל דבר בשורה נפרדת", כאשר ירידת שורה היא \r\n, וסוף המידע הוא בשתי ירידות שורה צמודות.

- השרת, שכזכור כדאי לו להיות כתוב בפייתון כדי שיהיה לו קל לעבוד עם מחרוזות, מחלץ מהשורה הראשונה את המלה שמופיעה אחרי ה-GET. למלה זו נקרא בשם 'הכתובת'. בהנחה שהכתובת היא /counter, התשובה שהוא יחזיר ללקוח (או יותר נכון, ל-LB ול-LB יעביר ללקוח) היא מספר הלקוחות שביקשו את הכתובת הזו מהשרת הספציפי הזה עד עכשיו. את התשובה יש להחזיר בפורמט הבא:

```
'HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nContent-Length: 5\r\n\r\nabcde\r\n\r\n'
```

כאשר את המחרוזת abcde יש להחליף בתשובה האמיתית, כלומר מספר הלקוחות שביקשו את הכתובת הזו עד עכשיו, ואת ה-5 שמופיע אחרי ה-Content-Length: יש להחליף באורך של התשובה בתווים (בדוגמה זו השרת מחזיר את המחרוזת abcde ולכן האורך הוא 5; אם הוא היה מחזיר את המחרוזת "1024", האורך היה 4). השרת חוזר לישון עד שיקבל בקשה נוספת מה-LB.

- כעת, השרת מחזיר את ההודעה ל-LB, שכצפוי קורא אותה עד שהוא מגיע ל-'\\n\\n' בפעם השנייה (הסתכלו שוב על התשובה ו-וודאו שהבנתם למה), ואז מעביר אותה במלואה ללקוח. ה-LB חוזר לישון עד שהוא מקבל חיבור חדש מלקוח.
- אם השרת קיבל בקשת GET לכתובת שאינה /counter, השרת צריך להחזיר הודעת שגיאה מסוג 404, שמשמעותה "אין משמעות לכתובת שביקשת":

```
HTTP/1.1 404 Not Found
Content-type: text/html
Content-length: 113
```

```
<html><head><title>Not Found</title></head><body>
Sorry, the object you requested was not found.
</body></html>
```

נזכיר שוב ששורות מופרדות ב-\n.

3. הדפדפן:

את הדפדפן, למזלכם, לא צריך לכתוב. פשוט תפתחו דפדפן תקני (פיירפוקס או כרום - באקספלורר אין שום דבר תקני) והתחברו לכתובת:

http://nova.cs.tau.ac.il:http_port/counter

או

http://localhost:http_port/counter

כש-http_port מוחלף בפורט שהשרת כתב לקובץ, ותקבלו את התשובה מוצגת בדפדפן שלכם (localhost זה שם החיבה של הכתובת האהובה עלינו, 127.0.0.1).

בדיקה ע"י הדפדפן היא אופציונלית, ונועדה בעיקר בשביל שיהיה לכם משהו להתגאות בו בסוף התרגיל. הטסט האוטומטי "משחק את התפקיד" של הדפדפן בעזרת קובץ הפייתון run_test.py, ששווה לכם להעיק בו מבט במהלך העבודה. שימו לב שאין גישה לפורטים אקראיים של nova מחוץ לקמפוס, כך שתצטרכו לבצע את ההתחברות הזאת מתוך האוניברסיטה אם אתם רוצים.

שימוש ב-clang-format, clang-tidy

כל קוד ה-C שאתם מגישים במסגרת התרגיל, צריך להיות נקי מאזהרות בבדיקה של clang-tidy, ואחרי פירמוט בעזרת clang-format. הטסט האוטומטי של התרגיל בודק זאת.

אין צורך לבדוק את הקוד בעזרת valgrind, אך כמובן שיש להקפיד שלא לדלוף זכרון. סטודנטים המעוניינים בכל זאת להשתמש ב-valgrind כדי להיות בטוחים שאין בעיות גישה לזכרון בקוד, יכולים לעשות שינוי קל ב-LB כך שהוא יצא לאחר ששירת תשעה לקוחות. במצב זה, יהיה ניתן לבדוק את ה-LB עם valgrind. ניתן להוסיף ל-LB ארגומנט אופציונלי שמשנה את ההתנהגות שלו באופן הזה, כך שהוא ישירות תשעה לקוחות ואז יסיים את פעולתו, ובכך לחסוך שינויים תכופים בקוד רק לצורך בדיקות. כמובן שכל שינוי שנעשה בקוד לצורך הזה, צריך עדיין לעבור את הטסט הרגיל של התרגיל באופן חלק.

4. סדר פעולות:

הטבלה הבאה מסכמת את סדר הפעולות בתרגיל. שימו לב, אין הטבלה הנ"ל מוסיפה מידע על המידע שמופיע בסעיפים 1-3:

	הצד השולח	הצד המקבל	
1	שרתים	LB	יוצרים קשר עם ה LB
2	הדפדפן	LB	שולח הודעת HTTP ל LB
3	LB	אחד השרתים	מעביר את ההודעה לאחד השרתים (לפי סבב מעגלי).
4	אחד השרתים	LB	מחלץ מתוך הודעת HTTP את הערך שמופיע אחרי ה-GET, זוכר כמה פעמים ביקשו אותו ממנו ושולח את התוצאה בפורמט הבא: 'HTTP/1.0 200 OK\r\nContent-Type: text/html\r\nContent-Length: 5\r\n\r\nabcde\r\n\r\n' לתוכנית ה LB.
5	LB	הדפדפן	מעביר לדפדפן את התוצאה שהוא קיבל בסעיף הקודם לדפדפן.

שימו לב, הטבלה הנ"ל מתארת את תהליך התקשרות בין שלושת ה"שחקנים" בתרגיל. הדרישות המלאות בתרגיל מופיעות בסעיפים 1-3. הטבלה הינה תקציר בלבד.

הערה על bind

לאורך הסמסטרים, סטודנטים רבים התקשו לבדוק את ההתנהגות של התוכנית כשהיא נתקלת במקרה של פורט תפוס. זאת כמובן בעיה לא קלה: הרי כדי להיות בטוחים במידה סבירה שהתוכנית מתפקדת כראוי במצב הזה, היא צריכה להיבדק במצב הזה, אבל מה נעשה, נדאג שהיא בכח תנסה לתפוס פורט שהוא כבר תפוס?!

כן! יש כל מיני פתרונות למצב הכללי הזה, של לבדוק התנהגות של תוכנית בתרחיש שמשוה לא עובד חלק. בתרגיל זה, אני מציע (לא חובה) להשתמש בפתרון שמכניס את המצב הזה ישירות לתוך התוכנית: הכי פשוט שתהיה פונקציה שמגרילה מספר פורט אקראי. בסיכוי חצי, היא מגרילה פורט אקראי שאמור להיות פנוי, מעל 1024, ובסיכוי חצי, היא מחזירה פורט תפוס, למשל פורט 80 (שיהיה תפוס בנובה). אתם יכולים לחשב לבד את ההסתברות שיש באג באיזור הזה של התוכנית בהנחה שהיא רצה כבר עשר או עשרים פעם ללא תקלות...

תוכלו לקרוא עוד על הגישה הזאת פה (מאת המתכנתים של נטפליקס):

<https://medium.com/netflix-techblog/the-netflix-simian-army-16e57fbab116?>

הערה על recv

אמרנו בכיתה ונזכיר שוב: אין הבטחה שקריאה בודדת ל-recv תחזיר את כל המידע שנשלח, גם אם המידע נשלח מהצד השני בקריאה בודדת ל-send. יש לעבור על המידע ולהבין מההקשר מתי ההודעה הסתיימה. (אצלנו ההקשר הוא לרוב לספור כמה פעמים ראינו \חל\חל\). למעשה, run_test.py מתעקש ושולח את המידע בשתי קריאות שונות ל-send, כשביניהן השהיה, על מנת לבדוק את ההתנהגות של הקוד שלכם במצב הזה. מומלץ לקרוא את run_test.py ולוודא שהבנתם את הפסקה האחרונה.

מהסיבות הנ"ל, מומלץ מאוד להריץ את הטסט 10-20 פעמים על מנת להשתכנע שאין בתרגיל באגים, או לפחות שההסתברות לבאג בהרצה ספציפית היא נמוכה.

BIND_RECV הקובץ

לנוחותכם, ועל מנת להקל על שיתוף הפעולה בין שותפים, אנא צרו קובץ בשם BIND_RECV בתיקיית ההגשה, אשר מכיל שורה אחת ובה רצף התווים BIND_RECV. כך תוכלו להיות בטוחים שלפחות אחד מהשותפים נתן את דעתו על הצורך לבצע קריאות חוזרות לשתי הפונקציות הללו, ושלא תאבדו נקודות סתם. הטסט האוטומטי של התרגיל אוכף זאת.

5. הערות נוספות ובאגים פופולריים שכדאי להימנע מהם:

1. למען הסר ספק: אין בתרגיל מיקבול. ה-LB מחכה ללקוח, מסיים לטפל בו, ואז מחכה ללקוח חדש. אתם בטח תוהים מה בעצם השגנו אם אמרתי מראש שרוצים לבצע כמה חישובים במקביל על כמה מחשבים, ואז בסוף ה-LB מטפל בכל חישוב בנפרד – התשובה היא שנהוג לפתור את הבעיה הזאת בעזרת הפונקציה `select`, שאולי נספיק ללמוד הסמסטר ואולי לא.
2. שימו לב שה-LB לא צריך "לדעת HTTP", כלומר הוא לא צריך להסתכל בתוכן של הבקשות או התשובות, אלא רק לקרוא עד `\r\n\r\n`. חבל לכתוב קוד מסובך יותר.
3. המלצה לצורת עבודה: התחילו מלכתוב את השרת, כך שיחכה דווקא בפורט 80 (זה אפשרי רק על המחשב שלכם, לא על נובה), ואז התחברו מדפדפן לאותה כתובת, וודאו שהעסק עובד. מן הסתם לא כדאי לכתוב את השרת במכה אחת, אלא למשל לכתוב קודם שרת שמקבל בקשה ומדפיס אותה וזהו, ואז לחלץ מהבקשה את הכתובת המבוקשת ולהדפיס גם אותה, ואז להחזיר תשובה קבועה (אפילו הדוגמה עם ה-`abcde` תעבוד לצורך הבדיקות), ואז לממש את התשובה האמיתית.
4. אם אתם מריצים ידנית רק חלק מהפקודות מהטסט לצרכי דיבאג, הקפידו להריץ את כל פקודות ה-`kill` שבסוף הטסט לפני שאתם מתנתקים מנובה, ואין צורך להריץ את הפקודות `disown` במצב הזה. אם לא תקפידו לעשות זאת, התוכנות שלכם ימשיכו לרוץ ברקע ולתפוס משאבים יקרים על נובה, ותתחילו לקבל המון מיילים אוטומטיים על כך.
5. זה בד"כ באג לקרוא פעמיים ל-`srand` באותה תוכנית – רצף התוצאות בקריאות ל-`rand` הוא דטרמיניסטי בהנתן הארגומנט של `srand`. כלומר אם קראנו ל-`srand`, ואז קראנו כמה פעמים ל-`rand`, ואז קראנו שוב ל-`srand` עם אותו ארגומנט, אז בקריאות הבאות ל-`rand` נקבל את אותן תוצאות כמו ברצף הקריאות הראשון ל-`rand`. ל-`srand` קוראים פעם אחת בתחילת התוכנית, ומשם קוראים רק ל-`rand`. מי שעושה אחרת, מסתכן בבאג שמאוד לא נעים לדבג.
6. מומלץ להשתמש באופציה `SO_REUSEADDR` על סוקטים לפני הקריאה ל-`bind`. אופציה זו מאפשרת לכם לעשות `bind` שוב ושוב על אותו פורט ביותר קלות. קיראו על האופציה הזו ב-`man` או במקומות אחרים, יותר קריאים.
7. באג שחזר בשנים קודמות הוא קריאה ל-`recv` עם ארגומנט אורך שהוא `strlen` של הבאפר אליו רוצים לקבל את המידע. ברגע שאתם כותבים קוד כזה, כמעט בוודאות יש לכם באג, ונסביר למה: `strlen` מקבלת כתובת, ומתחילה לסרוק את הזיכרון החל ממנה, תו אחרי תו, עד שהיא מגיעה לתו `\0` הראשון. לעומת זאת, מה שאתם רוצים להעביר ל-`recv` הוא גודל הזיכרון המוקצה. `strlen` יכולה להחזיר גודל קטן יותר אם במקרה יש תו 0 באמצע הזיכרון המוקצה, או גודל גדול יותר אם אין תו 0 בזיכרון המוקצה כלל, ואז היא תמשיך לסרוק את כל הזיכרון עד שהיא מוצאת תו 0 (תיאורטית היא יכולה להחזיר גדלים אדירים אם במקרה לא היה שום תו 0 באיזור, בפועל כמעט תמיד יהיה אחד כזה מתישהו והיא סתם תחזיר גודל שגוי).

בהצלחה!