

הערות – תרגיל מעשי ראשון

אתם אמורים בזמן הקרוב לקבל מייל עם ציון ופידבק על התרגיל המעשי הראשון. מי שלא מקבל מייל כזה עד שלישי בערב והגיש את התרגיל, נא לשלוח לי מייל

למרות שמדובר בתרגיל הראשון, עדיין מדובר בתרגיל מורכב, גם ביחס לתרגילים בהמשך הקורס.

חלקכם נפל בבדיקות האוטומטיות בשלבים הראשוניים, וקיבל מייל שמבקש להגיש את התרגיל מחדש. אני מבקש לא להתרגש מהציון במצב זה, כי מדובר בציון זמני עד להגשה של התרגיל מחדש. תקראו את המסמך הזה, וזה יפשט לכם את ההגשה המחודשת והגשת התרגילים הבאים בקורס.

במסמך אני ארכז את כל ההערות שהיו לי על התרגיל המעשי הראשון והוא יורכב מ:

- אופי הבדיקות האוטומטיות
- הערות קריטיות הקשורות במעבר סטטי על הקוד
- הערות נוספות

אופי הבדיקות האוטומטיות

הבדיקות האוטומטיות באו לבדוק פונקציונלית את האפליקציה. הבדיקות כללו הרצה של השרת והלקוח במתארים שונים כשכל המתארים לוו בהרצת valgrind ע"מ לוודא שלא נעשו דליפות זכרון \ חריגות בגישות לזכרון.

מתאר ראשון:

בא לבדוק פונקציונלית את התקשורת בין לקוח אחד לשרת אחד במקרה הממוצע ביותר (בדיקת Sanity לאפליקציה)

- הרצת השרת ע"י `./file_server [dir]`
- הרצת הלקוח ע"י `./file_client [dir]`
- הלקוח מקיש PUT, מקבל פלט מהשרת
- הלקוח מבצע GET file, כש-file הוא קובץ שנמצא בתיקייה בשרת ומקבל את הקובץ (נעשתה כאן השוואה בין הקובץ בשרת לקובץ אצל הלקוח אחרי הפעולה)
- הלקוח מבצע PUT file כש-file הוא קובץ שנמצא אצל הלקוח (לא אותו קובץ מהסעיף הקודם), ווידוא שהקובץ נמצא בתיקייה בשרת וזהה לאותו הקובץ בלקוח
- הלקוח מבצע QUIT

מתאר שני:

בא לבדוק את היכולת של האפליקציה להעביר קובץ בגודל 50 מגה (לא הקובץ הכי גדול שהאפליקציה אמורה לתמוך בו, אבל בהחלט יותר גדול מקובץ טקסט קטן, שאיתו נעשו הבדיקות בסעיף הקודם)

- הרצת השרת ע"י `./file_server [dir] [port]`
- הרצת הלקוח ע"י `./file_client [dir] [hostname] [port]`
- הלקוח מבצע GET file, כש-file הוא קובץ שנמצא בתיקייה בשרת ומקבל את הקובץ (נעשתה כאן השוואה בין הקובץ בשרת לקובץ אצל הלקוח אחרי הפעולה)

מתאר שלישי:

בא לבדוק עד כמה השרת robust. המטרה היא לוודא שהשרת שלכם לא תלוי בריצתו בלקוח שנמצא בו ובשגיאות שקורות לו.

זה מתאר שאף אחד לא איבד עליו נקודות אבל חשוב שתכירו אותו לתרגילים הבאים מבחינת הדרישות:

- הרצת השרת ע"י `./file_server [dir] [port]` 10 פעמים *
- הרצת הלקוח ע"י `./file_client [dir] [hostname] [port]`
- ניתוק הלקוח באופן ברוטאלי (סגירת תהליך הלקוח)

השרת אמור לתמוך במצב בו לקוח מתנתק גם באופן ברוטאלי (הודעות רשת מתקבלות על כך) והוא כן צריך להיות מסוגל לשרת לקוחות נוספים.

הערות קריטיות הקשורות במעבר על הקוד

- **בדיקת ערכי החזרה של system calls** - של `send/recv/listen` ושאר ה-`system calls` שנמצאות בשימוש בתוכנית. אני מצפה שתקראו את התיעוד (עמוד ה-man) של אותן פונקציות, כדי שתדעו דברים כמו: `recv()` יכולה להחזיר כמות בתים חיובית קטנה מכמה שציפיתם לקבל ועדיין מדובר בשימוש מוצלח בה. כמוכן שהתוכניות שלכן צריכות להתמודד עם כשלונות של הפונקציות הנ"ל בצורה הגיונית (שאותה אתם מגדירים, כך שניתן יהיה לראות שהקוד שליטתכם).
- **אחסון קבצים שלמים בזכרון** – העבודה עם חבילות שמעבירים ברשת צריכה להיות בעזרת Chunks – באפרים מוגבלים בגודל קבוע. כך למשל אם תרצו לשלוח קובץ ששוקל 50 גיגה, צד אחד כל פעם יקרא לצורך העניין 1024 בתים (או גודל אחר), ישלח ברשת והצד השני יכתוב 1024 בתים לקובץ, וכך הלאה. אתם לא יכולים להניח שניתן לאחסון כל קובץ שאתם עובדים איתו בזכרון, כשאתם מממשים שרת קבצים. כמו כן, בשיטת עבודה זו, אין שום מגבלה על גודל קובץ ולכן אין גם צורך ששרת הקבצים שלכם יעבוד רק עם קבצים מוגבלים בגודל.
- **שימוש בפונקציות לא בטוחות** - רבים מכם השתמשו בפונקציות כמו `gets` (שזרקה גם הערה בקומפילציה אך חלקכם התעלמתם), או בפונקציות כמו `strcpy`, `strcat` וכו'. אלה פונקציות לא בטוחות. שימוש בפונקציות כאלה בעולם האמיתי עלולות ליצור מצב שמשתלטים לכם בקלות על השרת וכמו כן זה נחשב סגנון לא טוב להשתמש בפונקציות כאלה. בתרגיל הזה לא הורדתי נקודות על פונקציות שהקומפילר לא העיר עליהם בקומפילציה, אבל אני ממליץ לתרגילים הבאים שתשתמשו ב: `strncat`, `strncpy` וחברותיהם..

הערות נוספות הקשורות לצורת ההגשה

כאן יפורטו הערות שונות שהיו לי שהיו קשורות לצורת ההגשה של התרגיל. אני מבקש שתקראו גם את ההערות שלא הורדו עליהן נקודות, כי לדעתי הן חשובות לא פחות מההערות עליהן הורדתי נקודות.

דברים שהורדו עליהם נקודות:

- שמות של מודולים לקמפול (התבקשתם שהמודולים המקומפלים ייקראו `file_client` ו-`file_server`).
- Warnings בזמן קומפילציה/קוד שלא מתקמפל
- היה דרוש קמפול נפרד של השרת והלקוח ב-`makefile`, שכן הם מיועדים להיות מאוחסנים במחשבים אחרים. כמוכן בלתי אפשרי ש-`main` אחד מפעיל את שניהם (באותו מודול).

- אל תדפיסו הודעות debug סתם. כשאני בודק את התרגיל, זה מפריע לראות את הודעות ה-debug שכתבתם לעצמכם בזמן הבדיקות שביצעתם, גם כי זה מסיח מההודעות האמיתיות שהאפליקציה אמור לייצר. אתם מוזמנים להשתמש בשיטה כמו [זאת](#) כדי להבטיח שההודעות יופיעו כשתרצו לדבג, ולא יופיעו בשאר המקרים.

דברים שלא הורדו עליהם נקודות:

- הודעות שהלקוח צריך לקבל כפידיבק לפעולות שהוא ביצע.
- מחזור קוד - קוד שכתבתם בלקוח עשוי להיות שימושי גם לשרת, שכן הפונקציונליות שלהם בהרבה מקרים היא דומה. תדאגו שקוד ייכתב פעם אחת באפליקציה שלכם, ובמקרה הצורך תקמפלו את השרת והלקוח עם מודולים חיצוניים משותפים, כדי להבטיח שתוכלו להשתמש בפונקציות שימושיות בשניהם. המטרה היא שכל שינוי לוגי שתצטוו לערוך, ידרוש שינויי קוד רק במקום אחד.

הדבר האחרון שיש לי לומר זאת המלצה: קונבנציות קוד אחידות

הרבה פעמים הקוד נהיה מסורבל כי הוא נהיה ארוך ושני אנשים שונים עובדים עליו בסגנון כתיבה שונה. אני ממליץ שתאמצו קונבנציות כתיבת קוד אחידות ותדאגו שהן ייאכפו בתרגיל המשותף. יש קונבנציות שונות לסביבות שונות אבל בגלל שהסביבה שאתם כותבים בה היא קוד C בלינוקס, אני ממליץ על הקונבנציות הבאות: [Linux kernel coding style](#). הלינק מכיל פירוט ארוך, אבל לדעתי הוא שווה ואף מחייב קריאה גם בשביל הניסיון שלכם לעבודה עתידית בתחום. הרבה הרגלים שם הם הרגלים שתוכלו לאמץ גם בסביבות אחרות.

בהצלחה בתרגילים בהמשך!