תרגיל בית מס. 1

מטרת התרגיל: למדוד כמה זמן לוקח לקרוא קובץ בצורה סדרתית לעומת צורה אקראית.

לשם כך נכתוב תכנית המייצרת קובץ (שאת גודלו נקבע כפרמטר לתכנית) , ואחר כך קוראת אותו פעמיים:

1. באופן סדרתי: בלוק אחרי בלוק לפי סדרם בקובץ.
2. באופן אקראי: בכל פעם בוחרים מספר בלוק באמצעות מחולל מספרים אקראיים ומבקשים לקרוא אותו.

בסך הכל קוראים את אותו מספר בלוקים (כמספר הבלוקים שבקובץ) בשתי השיטות. בשני המקרים גם דוגמים את השעון לפני הקריאה ואחריה כדי לדעת כמה זמן נדרש לבצע את הקריאה כולה.

את התכנית נריץ מספר פעמים, עם גודלי קבצים שונים, ונראה כיצד גודל הקובץ משפיע על ההבדל בזמן הדרוש לקריאה סדרתית לעומת קריאה אקראית.

וכעת לפרטים של המימוש. נזדקק למספר פונקציות שלא נזכרו בהרצאה. היעזרו בפקודה man (קיצור של manual) שאותה אפשר לתשאל לגבי כל פקודה או פונקציה של מערכת ההפעלה או הספריות השונות. למשל,

```
$ man random
```

תחזיר דף שלם המסביר כיצד יש להשתמש בפונקציה random(), מהם ה-header files הדרושים לה, אילו שגיאות היא עלולה להפיק ומהם הבאגים הידועים במימושה. למדו לנצל את התיעוד הזה.

א. העברת פרמטרים לתכנית

(את זה אתם כבר אמורים לדעת, וזה כאן רק לתזכורת.)

ב-C יש מנגנון מובנה להעברת פרמטרים ל-main(), אותה יש להגדיר (תמיד!!) כך:

```
int main( int argc, char *argv[] )
```

הפרמטרים, שכולם מחרוזות טקסט, יופיעו במערך המחרוזות argv לפי הסדר בו הם נכתבו בשורת הפקודה. מספרם יהיה נתון ב-argc. יש לזכור שגם פרמטרים מספריים מועברים לתכנית כמחרוזת טקסט, ולכן, על מנת להשתמש בהם בתכנית כמספרים יש להמיר אותם, אפשר באמצעות פונקציות ההמרה המובנות (כולן מוגדרות ב-stdlib.h):

- atoi() – הופכת מחרוזת מספרית ל-int.
- atol() – הופכת מחרוזת מספרית ל-long.
- atof() – הופכת מחרוזת מספרית ל-double.

ב. יצירת קובץ

את הפרטים ליצירת קובץ למדנו בהרצאה. אני ממליץ ליצור מערך בגודל בלוק ולכתוב אותו שוב ושוב לקובץ חדש. (גודל בלוק הוא 512 בתים ברוב המימושים, אבל בדקו את הפורמט של הכונן עליו אתם מתכוונים לעבוד.) <u>מספר הבלוקים בקובץ הוא פרמטר לתכנית.</u>

זוכרים את lazy write? הקפידו לקרוא ל-fsync() על ה-fd של הקובץ שזה עתה כתבתם, כדי להבטיח של כל הבלוקים שאת כתיבתם דרשתם אכן נרשמו פיזית לכונן. כדאי גם לסגור את

הקובץ (עם close()) כדי שאת קריאתו תתחילו אחרי open() חדש, מה שמבטיח שתתחילו לקרוא מההתחלה (ולא מהמקום בו סיימתם לכתוב...)

ג. קריאה סדרתית של הקובץ

זה קל... פשוט השתמשו בפונקציות שלמדנו בהרצאה כדי לפתוח את הקובץ מחדש ולקרוא אותו, בלוק אחרי בלוק, עד סופו. יש למדוד את הזמן שנדרש לכך (ראו בהמשך כיצד אפשר למדוד זמן).

ד. קריאה אקראית

זה כבר יותר מורכב... כאן עלינו לבחור מספר בלוק אקראי, ולדאוג שזה יהיה מספר הבלוק שהקריאה הבאה ל-read() תקרא. לשם כך עלינו להשתמש בשתי פונקציות עליהן לא דיברנו עד כה.

a. random()
b. lseek()

random() היא מחולל מספרים אקראיים[1]. בכל פעם שהיא נקראת היא מחזירה מספר בין 0 ל-2^{31} - 1 (=2147483647). עליכם להפוך את המספר הזה למספר שלם בטווח [0,N-1] כאשר N הוא מספר הבלוקים בקובץ.

lseek() היא פונקציה שיכולה לשנות את המקום הנוכחי בקובץ שעליו היא מופעלת. לפונקציה שלושה פרמטרים:

```
off_t  lseek( int fd,  off_t offset, int flag )
```

off_t הוא טיפוס (typedef) המוגדר ב-unistd.h, ומציין מספר שלם גדול דיו כדי לציין את מספר הבתים בקובץ הגדול ביותר האפשרי. (int אינו מתאים למטרה זו – בהחלט יתכן קובץ גדול מ-4 GiB.) הפרמטר הראשון הוא ה-fd של הקובץ הפתוח שאת מיקומו הנוכחי יש לשנות.
הפרמטר השני הוא המקום (כמספר בייט) אותו נבקש לקבוע כמקום נוכחי (ממנו תתחיל הקריאה או הכתיבה הבאה). הפרמטר השלישי קובע יחסית למה מתייחס הפרמטר השני:

i. SEEK_SET – יחסית לתחילת הקובץ
ii. SEEK_CUR – יחסית למקום הנוכחי
iii. SEEK_END – יחסית לסוף הקובץ

במקרה שלנו טוב להשתמש ב-SEEK_SET, אבל יש לזכור שהמיקום נקבע כמספר בייט, ואנו רוצים מספר בלוק: לפיכך יש לכפול את מספר הבלוק הרצוי בגודל הבלוק (שהוא, כאמור, בדרך כלל 512, אבל תבדקו את הכונן שלכם.)

למרות שהקריאה אקראית, מספר הבלוקים שעליכם לקרוא הוא בדיוק מספר הבלוקים שבקובץ, וזאת כדי שמספר הבלוקים שייקראו בשתי השיטות יהיה שווה.

ה. מדידת זמן ביצוע

לצורך מדידת זמן הביצוע נשתמש בפונקציה נוספת, gettimeofday(), שמחזירה את השעה ברגע הקריאה. הזמן ב-Unix מצויין בשני שדות:

[1] המחשב פועל על פי אלגוריתם, ולכן אין ביכולתו לייצר שום דבר אקראי. אם נריץ את אותו המחולל פעם נוספת, נקבל בדיוק את אותה סדרת מספרים "אקראיים". לכן הם נקראים pseudorandom generators, מהמילה היוונית ψευδο (פְּסֶבְדוֹ), שמשמעה "שקר".

1. מספר השניות שחלפו מאז חצות של ה-1 בינואר, 1970
2. מספר המיקרושניות (מליוניות של שניה) שחלפו מאז תחילת השניה האחרונה.

כיון הערך בשדה הראשון יכול לגדול לאין שיעור (אם נחכה מספיק זמן... הוא יחרוג מקיבול של int בשנת 2038, לא עתיד כזה רחוק 😉), גם לשדות אלה יש typedef ספציפי. הפונקציה gettimeofday(), אם כן, מחזירה תשובה בתוך struct המוגדר כך (ב-sys/time.h):

```
struct timeval {
        time_t        tv_sec;        /* seconds */
        suseconds_t   tv_usec;       /* microseconds */
    };
```

הקריאה לפונקציה נעשית כך:

int gettimeofday(struct timeval *tz, NULL ופרמטר נוסף שצריך להיות)

אם הקריאה לא מחזירה 1-, אזי המבנה tz מכיל את השעה הנוכחית בשניות ומיקרושניות. שמרו את הערכים הללו, וחסרו אותם מהערכים שיתקבלו בקריאה נוספת לפונקציה אחרי ביצוע הפעולה שאת משכה תבקשו למדוד. (שימו לב שההפרש במיקרושניות עשוי להיות שלילי...)

ו. הרצה

כעת, משהכל מוכן, אפשר להריץ ולבצע מדידות.

הריצו את התכנית מספר פעמים עם כל ערך של פרמטר וחשבו את הממוצע של המדידות שקיבלתם עבור כל ערך. בדקו את הזמנים עבור קבצים בגודל של 1000, 10,000, 100,000, 1,000,000 ו-10,000,000 בלוקים. (הקובץ האחרון יהיה בגודל של 5GiB – לא מעל היכולת של מחשבים היום).

ז. הערות

1. נצלו את התיעוד המובנה בתוך Unix: הפקודה man (קיצור של Manual) מספקת הסברים על כל פקודה, קריאת מערכת, פונקצית ספריה או סתם נושא שקשור בבצוע מטלות ב-Unix. פשוט קראו ל-"man <שם>" על מנת לקבל הסבר על המטרה ודרך השימוש ב-<שם>.
2. במעבדות ראיתי שאנשים לא תמיד מעבירים פרמטרים ל-main שהם כותבים, ומעדיפים לקבוע בקוד פרמטרים (לפעמים באמצעות #define). לעיתים רחוקות זו ההתנהגות המבוקשת, ולעולם לא בקורס הזה. אנא העבירו את הפרמטרים לתכניות שאתם כותבים משורת הפקודה, תוך ניצול המנגנון הבנוי של main(int argc, char *argv[ ]) לשם כך.

את התרגיל יש לבצע ביחידים (לא בזוגות!) בבית על מחשב אישי או על מחשב במעבדה במכללה. יש לקבץ את קוד התכנית ואת הפלט שלה באמצעות ZIP או RAR. שם הקובץ שתיצרו צריך להיות בפורמט הבא:

Israel_Israeli_123456789_Ex1.zip

כלומר: השם באנגלית, מספר הזהות, והמחרוזת "Ex1" עם קו תחתון ביניהם. את הקובץ (היחיד) יש להעלות באמצעות moodle לאתר הקורס. נא ציינו, בנוסף, את שמכם (בעברית) ואת מספר הזהות שלכם בגוף ההגשה בקובץ ReadMe.txt. את התרגיל יש להגיש עד

בהצלחה!