סיכום – תכנות מתקדם:

redirect – אחראית להעביר קלט ופלט מהאמצעים הסטנדרטים לאן שנבחר:

1. stdout – באופן דפולטיבי הולך למסך.
2. stderr - באופן דפולטיבי הולך למסך.
3. Stdin – הקלט שהוא מקבל באופן דפולטיבי הוא מהמקלדת.

מה הכוונה באופן דפולטיבי הולך למסך? לכל תוכנית בלינוקס קיימת טבלה המכילה מידע שרלוונטי לתוכנית, חלק מהמידע הרלוונטי הזה היא באיזה קבצים יש שימוש על ידי התוכנית, לכל תוכנית קיים File Descriptor המכיל integer שערכיו הם stdin-0 או stdout-1 או stderr-2 ,

דוגמה: הפקודה ls מראה את כל הקבצים והתיקיות הנמצאות בתיקייה ספציפית ולכן התוכנית הזאת משתמשת בהוצאת למסך כלומר stdout,

נחזור לפקודה redirect, המטרה היא לשנות את הערך שמחזיק ה-(file descriptor) fb, כלומר במקום שפקודה מסוימת תדפיס למסך אולי נרצה נשמור את הפלט בקובץ או להעביר אותו כקלט לתוכנית אחרת,

נניח ונרצה לשמור את הפלט של הפקודה ls -l בקובץ שנקרא ls_out, נבצע:

```
~$ ls -l > ls_out
```

בעצם ביצענו redirect לפלט של התוכנית, אפשרות נוספת היא במקום לשמור את הפלט לקובץ נוכל להוסיף את הפלט לסוף הקובץ קיים ע"י:

```
~$ ls -l >> ls_out
```

בצורה דפולטיבית הפקודה redirect בעזרת הסימון משתמשת בערך דפולטיבי של 1 כלומר stdout, נוכל לבצע את הסימון <2 ואז במקום להשתמש ב-stdout נשמש ב-stderr:

```
~$ ls -l 2> ls_out
```

דוגמה: נניח את התוכנית הפשוטה הבאה (כתובה ב-C):

```c
#include <stdio.h>
int main(void) {
        printf("STDOUT\n");
        perror("STDERR\n");
        return 0;
}
```

וכעת נבצע את הפקודה הבאה:

```
kfir@kfir-VirtualBox:~/Desktop$ 2>&1 ./main > ls_out
STDERR
: Success
```

kfir.goldfarb@msmail.ariel.ac.il

נוכל לראות שרק ה-stderr מופיע על הסמך אבל ה-stdout מופיע בתוך הקובץ ls_out:

```
1 STDOUT
```

ראינו שחץ ימינה > מוביל אל stdout, אז חץ שמאלה < מוביל אל stdin, למשל:

```
kfir@kfir-VirtualBox:~/Desktop$ cat < ls_out
STDOUT
```

הפקודה cat מדפיסה בצורה פשוטה ל-terminal שעליו אנחנו עובדים כעת.

sort:

נניח וקיים לנו הקובץ fruit.txt המכיל שמות של פירות כך:

```
kfir@kfir-VirtualBox:~/Desktop$ cat fruit.txt
Peach
Apple
apple
Grapes
Apple
Banana
Mango
Avocado
Tometo
Watermelon
Orange
```

כמו שאנחנו מציגים בעזרת cat למסך שלנו (terminal) נרצה להציג בצורה ממיינת:

```
kfir@kfir-VirtualBox:~/Desktop$ sort fruit.txt
apple
Apple
Apple
Avocado
Banana
Grapes
Mango
Orange
Peach
Tometo
Watermelon
```

pipeline:

נרצה להפוך פלט של תוכנית אחת לקלט של תוכנית אחרת בעזרת הסימון |,

```
kfir@kfir-VirtualBox:~/Desktop$ sort fruit.txt | uniq
apple
Apple
Avocado
Banana
Grapes
Mango
Orange
Peach
Tometo
Watermelon
```

כמו שאנחנו רואים uniq היא תוכנית המקבלת טקסט ומוחקת מימנו מילים/שורות שהם אותו דבר.

word count:

סופרת שורות בטקסט, למשל כדי לדעת כמה קבצים יש בתיקייה הנוכחית נוכל לבצע:

```
kfir@kfir-VirtualBox:~/Desktop$ ls -l | wc -l
6
```

התוספת של l- בסוף ה-wc היא לספור את מספר השורות בטקסט הנכנס אליו כקלט, ואם נמחק את l- אז בצורה דפולטיבית wc ידפיס כמה שורות, מילים ותווים יש בטקסט המתקבל:

```
kfir@kfir-VirtualBox:~/Desktop$ ls -l | wc
      6      47     258
```

wild cards:

המטרה של wild cards היא לבצע פעולות על מספר קבצים בבת אחת, נראה מספר דוגמאות לפעולות בסיסיות בעזרת הפקודה ls, פעולות אלו עובדות גם עם הפעולות (remove) rm, (copy) cp ועוד, הפקודה ls כמו שאמרנו מראה את כל הקבצים בתיקייה:

```
kfir@kfir-VirtualBox:~/Documents$ ls
doc          doc2.txt  file        history        some_folder_14
doc1.txt     doc3.txt  folder_12   README.md
doc_22.txt   empty     folder_199  some_folder_12
```

מה אם נרצה לראות רק את הכל הקבצים אשר שמם מתחיל במילה doc?

```
kfir@kfir-VirtualBox:~/Documents$ ls doc*
doc  doc1.txt  doc_22.txt  doc2.txt  doc3.txt
```

kfir.goldfarb@msmail.ariel.ac.il

נשים לב שיש לנו 3 קבצים טקסט בעלי פורמט דומה (doc1.txt, doc2.txt, doc3.txt) ולכן כדי להציג רק את הפורמט הזה נוכל להשתמש בסימן שאלה כך:

```
kfir@kfir-VirtualBox:~/Documents$ ls doc?.txt
doc1.txt  doc2.txt  doc3.txt
```

נניח ויש לנו את הקבצים בפורמט הבא (כמו בפורמט הקודם):

```
kfir@kfir-VirtualBox:~/Documents$ ls doc?.txt
doc1.txt  doc2.txt  doc3.txt  doc4.txt  doc5.txt  doc6.txt
```

נניח ונרצה לראות רק לפי תווך מסוים, נשתמש ב-[] כך:

```
kfir@kfir-VirtualBox:~/Documents$ ls doc[3-5].txt
doc3.txt  doc4.txt  doc5.txt
```

כלומר הגדרנו תווך של תצוגה רק בין 3 ל-5.

נוכל גם לקחת מהפורמט קבצים ספציפיים (אפשר להשתמש גם בלי פסיקים):

```
kfir@kfir-VirtualBox:~/Documents$ ls doc[1,2,5,6].txt
doc1.txt  doc2.txt  doc5.txt  doc6.txt
```

אפשר גם בעזרת סימן קריאה להביא תוצאות בלי קבצים ספציפיים לפי פורמט מסוים כלומר:

```
kfir@kfir-VirtualBox:~/Documents$ ls doc[!2].txt
doc1.txt  doc3.txt  doc4.txt  doc5.txt  doc6.txt
```

נניח וקיימים לנו 12 קבצים בשני פורמטים שונים (txt, doc):

```
kfir@kfir-VirtualBox:~/Documents$ ls doc?.txt
doc1.txt  doc2.txt  doc3.txt  doc4.txt  doc5.txt  doc6.txt
kfir@kfir-VirtualBox:~/Documents$ ls doc?.doc
doc1.doc  doc2.doc  doc3.doc  doc4.doc  doc5.doc  doc6.doc
```

מה אם נרצה להציג קבצים שהם משני הפורמטים הנ"ל? (נשתמש ב-{}):

```
kfir@kfir-VirtualBox:~/Documents$ ls {doc*.txt,doc*.doc}
doc1.doc  doc_22.txt  doc2.txt  doc3.txt  doc4.txt  doc5.txt  doc6.txt
doc1.txt  doc2.doc    doc3.doc  doc4.doc  doc5.doc  doc6.doc
```

kfir.goldfarb@msmail.ariel.ac.il

who & whomai?

הפקודה who מדפיסה מי מחובר למחשב ומתי הוא התחבר:

```
kfir@kfir-VirtualBox:~/Documents$ who
kfir     :0           2022-04-11 10:21 (:0)
```

הדפסה של אותו דבר רק בצורה של טבלה:

```
kfir@kfir-VirtualBox:~/Documents$ who -H
NAME     LINE         TIME             COMMENT
kfir     :0           2022-04-11 10:21 (:0)
```

הפקודה whoami נותנת רק את השם של המשתמש שמחובר למחשב:

```
kfir@kfir-VirtualBox:~/Documents$ whoami
kfir
```

מתי לאחרונה המחשב ביצע boot:

```
kfir@kfir-VirtualBox:~/Documents$ who -b
         system boot  2022-04-11 10:20
```

באיזה מצב ריצה המחשב נמצא:

```
kfir@kfir-VirtualBox:~/Documents$ who -r
         run-level 5  2022-04-11 10:20
```

(run-level 5 הוא המצב הסטנדרטי שאומר שמספר משתמשים יכולים להתחבר למערכת הפעלה)

הערה: נשים לב שהזמן האחרון שבוא בוצע boot וגם המצב ריצה של המחשב נותנים אותו זמן כי לאחר שעשינו boot למחשב המחשב נכנס למצב ריצה הזה.

מספר המשתמשים המחוברים כרגע למחשב ושמותיהם:

```
kfir@kfir-VirtualBox:~/Documents$ who -q
kfir
# users=1
```

שאלות ותשובות כלליות לתכנות מתקדם:

1. מהו buffering IO?

buffer הוא אזור מוגדר בזיכרון (בדרך כלל בגודל 4K) השומר נתונים הניתנים להעברה בין שני התקנים (devices) שונים או בין התקן ואפליקציה.

ה-buffer מתמודד במהירות בין העברת מידע (data stream) בין ה-.consumer producer

ה-buffer מיוצר בזיכרון הראשי כדי להגדיל את הבתים (Bytes) המתקבלים.

לאחר שנתונים מתקבלים ב-buffer הם מועברים ל-disk בפעולה אחת (לא באופן מידי), כדי שהמודם יקבל עוד נתונים הוא נדרש לייצר עוד buffer חדש, כאשר ה-buffer הראשון מתחיל להתמלא הוא מתחיל להעביר נתונים לדיסק, במקביל ה-modem מתחיל להתמלא, כאשר שני ה-buffer-ים מסיימים את המשימה שלהם, אז ה-modem עובר שוב ל-buffer הראשון והנתונים ב-buffer השני עוברים לדיסק.

ה-buffer מספק מספר ורציות של עצמו אשר מתאימות להתקנים שונים בעלי קיבולת זיכרון שונה.

המטרה של ה-buffer היא להפחית את הקריאות למערכת.

בקריאה מהדיסק: בכל פעם שה-buffer ריק וגם בפעם הראשונה, מתבצעת קריאה בגודל 4K מהדיסק ל-buffer והאפליקציה קוראת בתים מה-buffer עד שמתרוקן.

בכתיבה לדיסק: האפליקציה כותבת בתים ל-buffer, כאשר מתמלא הנתונים ב-buffer מועברים לדיסק.

2. מה הם ה-4 הסיבות העיקריות להשתמש ב-fopen ולא ב-open:

a. Fopen כולל איתו את ה-buffering IO שהוא הרבה יותר מהיר מהשימוש ב-open.
b. Fopen יודע לתרגם את הקובץ כאשר הוא לא נפתח כקובץ בינארי, דבר זה מאוד מועיל במערכות שלא מבוססות UNIX.
c. Fopen אשר מחזיר את המצביע ל-File מאפשר להשתמש ב-fscanf ובפונקציות נוספות הקיימות בספריה stdio.
d. ישנם פלטפורמות שתומכות רק ב-ANSI C שרק fopen תומך בזה ולא open.

3. מהו עקרון copy on write (העתקה בזמן כתיבה)?

עקרון העתקה בזמן כתיבה אומר שכאשר יש מספר משאבים המשתמשים באותו מאותו מקום של זיכרון אז מיותר להעתיק את הזיכרון לכל משאב ולכן כל משאב יכיל מצביע לאותו מקום בזיכרון,

בלינוקס: כאשר מתבצעת הפעולה fork עותק של כל הדפים המותאמים לתהליך האב נוצרים ונטענים לאזור זיכרון נפרד על ידי מערכת ההפעלה לטובת תהליך הבן, פעולה זו אינה נדרשת במקרים מסוימים, למשל במקרה ונבצע דרך תהליך הבן את הקריאה execv אז אין צורך להעתיק את דפי האב, ו-execv מחליף את מרחב הכתובות של התהליך, במקרים כאלו עקרון COW בא לידי ביטוי, כאשר מתבצעת הפקודה fork דפי תהליך האב לא מועתקים לתהליך הבן, במקום זאת הדפים משותפים בין תהליך האב ותהליך הבן (לקוח מויקיפדיה).

כאשר רוצים לשתף קטע זיכרון בין תהליך אב לבן נדרש לסמן בטבלאות הדפים של האב והבן את הקטע בזיכרון כ-read only ולזכור שהם copy-on-write.

4. **מה זה systems calls (קריאות מערכת)?**

בקשה שמבצעת תוכנת מחשב מליבת מערכת ההפעלה (kernel) כדי לבצע פעולה שהיא לא יכולה לבצע בעצמה, קריאות מערכת אחראיות על החיבור בין מרחב המשתמש למרחב הליבה, נותנת למשתמש מספר פונקציות שגורמות לפעולות בליבת מערכת ההפעלה (למשל יכולת קבלת גישה לרכיבי חומרה כמו קריאה מכונן קשיח, יצירת תהליך חדש, העברת מידע בין תהליכים ועוד).

5. **מה ההבדל בין מרחב משתמש למרחב ליבה?**

kfir.goldfarb@msmail.ariel.ac.il