סיכום בסיסי:

ערוצי input/output של קבצים בלינוקס:

הפקודה 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 ,

דוגמא: הפקודה ls מראה את כל הקבצים והתיקיות המצאות 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

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-IO buffering 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?