**מעבדה במערכות הפעלה: מפגש 3**

**נושא: יצירת תהליכים ב-LINUX**

* + **מטרות:**
    - יצירת תהליכים בסביבת LINUX (פקודת FORK)
    - הרצת תוכנית בתהליך החדש
    - סנכרון תהליך אב עם סיום תהליך הבן ( קריאה: WAIT() )
    - **הצגת בעיית שיתוף משאבים**

**תוכנית מצורפות (מקור):**

* + - fork\_char\_by\_char.c
    - fork\_with\_exec.c
    - simple\_fork.c

**הערה עבור משתמשים של שרת המכללה :** תוך כדי קומפילציה מומלץ לתת לתוכית להרצה בשם ייחודי

בחלק הזה נעבוד עם תוכנית פשוטות המייצרות תהליכים חדשים.

הערה: אפשר לסיים תהליך עיי' פקודה **kill <PID>**

**משימה 1**

1. **תקראו/ותקמפלו את התוכנית simple\_fork.c**

תריצו את התוכנית עד קבלת הודעה **PRES any CHAR to continue….**

* + **כמה תהליכים של התוכנית יש במערכת ?**
  + הכניסו תו כלשהוא:
    - האם תהליך אב או בן קבל את התו ?
    - כמה תהליכים של התוכנית (**simple\_fork.c** ) יש במערכת? הסברו את התוצאה
  + הכניסו עוד תו. האם עדין יש תהליכי simple\_fork.c
  + שנו תוכנית כך הבן יבקש להכניס תו ורק אחרי שהבן יקבל (ויסיים את הריצה) תהליך האב יבקש להכניס תו לפי (תקראו לתוכנית החדשה בשם **simple\_fork\_ext.c**
    - **רמז:** השתמשו בקריאה **wait()** כדי לגרום לתהליך האב לחכות לסיום תהליך הבן

תגישו את התוכנית **simple\_fork\_ext.c**

1. **עד עכשיו תהליך הבן המשיך אותה תוכנית:**

נראה עכשיו איך אפשר להריץ תוכנית אחרת מתוך תוכנית עיי' פקודה **execvp**

**הערה:** יש מספר קריאות מערכת (SYSTEM CALLS) המאפשרות לעשות זאת (השוני רק בפורמט הפרמטרים) .

תקראו/תבינו/ותקמפלו את התוכנית **fork\_with\_exec.c**

**הערה חשובה:** התוכנית מפעילה תוכנית simple\_fork (אם נתתם לה שם אחר יש לשנות את השם בתוך fork\_with\_exec.c)

תריצו את התוכנית:

* + - כמה תהליכים נוצרים עיי' התוכנית ? עבור כל תהליך תציינו את שם התוכנית שהתהליך מריץ והאם הוא תהליך האב או הבן של התוכנית.
    - תסבירו במשפט אחד מה עושה כל אחד מהתהליכים הרלוואנטיים
      * + תסבירו למה ההדפסה הבא בתוכנית לא מופיעה

printf("FORK\_WITH\_EXEC:Hello world from child (pid=%d)\n", getpid());

* + - תלחצו על תו: איזה תהליכים רלוואנטיים אתם רואים עכשיו ?

מה הוא ה-PID של fork\_with\_exec ? למה עדיין רץ ? תסיימו אותו עם CNTL X

* + - * + איך היה אפשר לסיים אותו בצורה אחרת ?

1. **מה זה <defunct> ?**

בהרצה של סעיף הקודם כנראה נתקלתם בסימון <defunct> ליד חלק מתהליכים. תנסו לברר ב-GOOGLE מה זה אומר.

סימון הזה קורה עבור תהליכי בן, שהסתיימו מבלי שתהליכי אב מודעים לכך (ולכן מבחינת המערכת עדיין קיימים על אף שלעולם לא יהיו פעילים יותר).

המנגנון עיי' תהליך האב נעשה מודע לסיום של תהליך הבן הוא wait() . תתקנו את התוכנית fork\_with\_exec.c כך שתהליך האב נכנס ללולאת הדפסות רק אחרי שתהליך הבן שלו מסתיים. תחזרו על סעיף ב. עם התוכנית המתוקנת. וודאו שאתם לא רואים את <defunct>

להגיש את התוכנית **fork\_with\_exec.c** המתוקנת .

**משימה 2**

המטרה במשימה זאת היא להראות מה יכול לקרות כששני תהליכים משתמשים באותו משאב ללא תיאום.

1. תקראו/ותקמפלו את התוכנית **fork\_char\_by\_char.c**
   * 1. תסבירו בקצרה מה עושה תוכנית
2. תריצו את התוכנית. איזה בעיה אתם רואים ? מה לדעתכם הסיבה לבעיה ?
   * 1. הערה: הבעיה שאנו רוצים לראות לא מופיעה בכל הרצה – תריצו את התוכנית מספר פעמים עד שרואים דברים מוזרים בהדפסות
3. בהמשך הקורס נלמד איך אפשר לפתור את הבעיה ע''י תיאום בין התהליכים. תחשבו איך אולי אפשר לפתור את זה בדרך אחרת (ללא צורך בתיאום).
4. תשנו את התוכנית כך שתעשה את ההדפסות ללא לולאה על תווים (הדפסת מחרוזת בבת אחת) עיי': printf( "Hello world from child %d\n",i)

תריצו מספר פעמים את התוכנית ? האם אתם רואים את הבעיה ? תנסו להסביר

**משימה 3**

כתבו תוכנית מייצרת 10 תהליכים (עי'' קריאה ל-FORK).

* + כל תהליך בן ידפיס את המספר הסידורי של יצירת התהליך (מ-1 עד 10).
  + תהליך האב לאחר יצירת כל התהליכי הבן, מחכה עד סיום כל הבנים ורק לאחר מכן מדפיס הודעת סיום. דוגמא לפלט:

Text

Description automatically generated

מאוד חשוב שההדפסות יהיו לפי הסדר!!! בהמשך הקורס נלמד מנגנונים לסינכרון בין התהליכים. בינתיים: יכולים להשתמש בלולאות/SLEEP . ... כדי לקבל תוצאה נכונה.