**234123 - מערכות הפעלה - סמסטר אביב תשפ"ד**

תרגיל יבש 1

מגישים: אייל אמדור, בארי זיטלני

ת.ז: 318849270, 209351626

אימייל: beerizitelny@campus.technion.ac.il, eyal.amdur@campus.technion.ac.il

**שאלה 1**

* סעיף 1

ריצת הקוד הנ”ל תתקיים כך:

* + בתחילת הריצה התכנית תכנס ל-main ותדפיס את ה-pid של התהליך הנוכחי (נסמן p1). לאחר מכן יתבצע fork() ונקבל שני תהליכים (p1, p2) כאשר שניהם יקפצו בלולאת הFORK ויגיעו לתחילתה עם i=1.
  + כעת שני התהליכים (p1, p2) יבצעו fork() (כלומר סה"כ יש 4 תהליכים עם i=1), ארבעתם יעלו את i ב-1 (כלומר i=2 אצל כולם) ושוב כל ארבעת התהליכים יקפצו בלולאה כיוון שבכולם i<3.
  + כעת ארבעת התהליכים (p1, p2, p3, p4) יבצעו fork() (כלומר כעת יש 8 תהליכים עם i=2), כל השמונה יעלו את i ב-1 (כלומר i=3 אצל כולם) והפעם כל שמונת התהליכים לא יקפצו בלולאה כיוון שבכולם i=3.
  + לאחר מכן, בשורת while (wait(NULL) != -1) כל תהליך ימתין לסיום \*כל\* הבנים שלו ולאחר מכן ימשיך להדפסה הסופית. זאת מכיוון שפעולת wait() תכשל (תחזיר 1-) רק כאשר לתהליך לא נותרו בנים.
  + נשים לב: ההדפסה הראשונית תמיד תהיה של תהליך האב הראשון ורק הוא ידפיס אותה.  
    ההדפסה השנייה תהיה של אחד מארבעת התהליכם שנוצרו אחרונים (p5, p6, p7, p8). לאחר רק תהליך ללא בנים חיים יכול להדפיס ולכן הסדר הוא אקראי.
  + כלומר מצד אחד פלט התכנית אינו דטרמינסטי לחלוטין כיון שיש דרגת חופש לפלט התכנית בין כל תהליכים שרצים במקביל בלי תלות אחד בשני. מצד שני הוא אינו אקראי לחלוטין כיוון שתמיד תהליך p1 ימתין שכל בניו יסיימו את ריצתם ולכן יודפס אחרון בכל ריצה של התכנית.
  + לכן התשובה הנכונה היא תשובה א' - בפלט התכנית יופיע הביטוי “3 “fork:8 פעמים בדיוק שכן סה"כ ישנם 8 תהליכים וכולם יגיעו להדפסה עם i=3.
* סעיף 2
  + כפי שהוסבר למעלה התהליך שמודפס ראשון הוא תהליך האב המקורי (p1) וכיוון שהוא מחכה לסיום ריצת כל בניו (ראו הסבר סעיף 1) הוא גם יודפס אחרון. מהנתון הpid שלו הוא 1000 ולכן השורה האחרונה תהיה בעלת pid 1000. כמו כן שורת ההדפסה הנ"ל תודפס רק כאשר i=3 כלומר התשובה הנכונה היא d: (PID: 1000, fork:3) .
* סעיף 3
  + כעת כל תהליך שמגיע לשורה : while (waitpid(p, NULL, 0) != -1);ממתין רק לתהליך הבן האחרון שלו (כיוון שהערך p מחזיק את הpid של הבן האחרון שלו) בניגוד לסעיפים הקודמים בהם כל תהליך חיכה לסיום ריצת כל בניו.
  + לכן, מובטח שכל אב ידפיס מתישהו לאחר הבן האחרון שלו אך מעבר לכך הסדר הוא אקראי.
  + בפרט, האב המקורי (p1) לא חייב להדפיס אחרון ולכן התשובה לסעיף 2 במקרה זה תהיה e.
* סעיף 4
  + כעת אין התניה על הקפיצה לFORK ולכן בהגעת כל תהליך לשורה 14 הוא יקפוץ חזרה לשורה 11, יעלה את i ב1 ויבצע fork().
  + ללא תלות בהצלחת/כשלון הfork() התהליך יגיע לשורה 14 ויחזור על הפעולות הנ"ל ללא תנאי עצירה.
  + בעצם ישנה לולאה אין סופית שתיצור תהליכים חדשים עד למספר תהליכים מקסימלי וגם כאשר תגיע למספר זה וה-fork()-ים יכשלו הלולאה תמשיך לרוץ.
  + נשים לב כי אף תהליך לא יגיע לשורה 16 כלומר לא יגיע לפקודת wait() ולא לפקודת print. לכן לא יודפס כלום למסך.

**שאלה 2**

* סעיף 1
  + הקוד הנ"ל מאתחל מערך בגודל 4 עם הערכים 0,1,2,3 .
  + אחר כך בלולאה ללא תנאי עצירה מודפסים הערכים מסוף המערך להתחלתו   
    ((x[3] -> x[2] -> x[1] ->x[0] -> x[-1] -> … בגלל שהלולאה אין סופית הקוד ימשיך להדפיס את תאי הזכרון שנמצאים לפני המערך בקפיצות של 4 בתים (כגודל של int) עד אשר יגיע לכתובת לא חוקית.
  + במקרה זה ישלח signal SIGSEGV אשר מעיד על segmentation violation (פנייה לכתובת לא חוקית בזכרון) ולכן יריץ את הפונקציה seg\_fault\_catcher שמדפיסה את ערך הsignal (שהוא 11) ויוצאת מהתכנית (ולכן לא נגיע לשורה 23 – שורת ההדפסה של Hi).
  + ההדפסה תהיה: 3->2->1->0->…->11 (כאשר ... הוא ערך לא יודע של תמונת הזכרון).
  + סה"כ ההדפה האחרונה תהיה "11" והתשובה הנכונה היא d.
* סעיף 2
  + הפונקציה catcher\_fault\_seg תיקרא פעם אחת בלבד . התשובה: **לא.**

נימוק:  
כיוון שבטיפול שליחת SIGSEGV הפונקציה מגדירה את הטיפול בSIGFPE להיות הפונקציה עצמה, מתקיים כי בקבלת SIGSEGV (שערכו 11) הרצת שורת היציאה תבצע חלוקה ב-0 ולכן תזרק שגיאת SIGFPE כלומר הפונקציה תקרא לעצמה פעם נוספת עם ערך SIGFPE (שערכו 8) ולכן בפעם השנייה הפונקציה תצליח לצאת עם ערך exit(1) (3/11-8) .

לכן סה"כ הפונקציה תקרא יותר מפעם אחת (פעמיים).

* + התכנית תסתיים בצורה תקינה (ע"י קריאה מוצלחת לexit) התשובה: **כן.**

נימוק:  
כפי שהוסבר למעלה, קבלת סיגנל SIGSEGV בפעם הראשונה תטופל ע"י הפונקציה ובמקביל תגדיר את טיפול בזריקת סיגנל SIGFPE.

הפעם ההדפסה האחרונה תהיה של ערך הSIGFPE כלומר 8.

כך הטיפול בשגיאת החלוקה ב-0 יטופל גם הוא ע"י הפונקציה ויסיים את ריצת התכנית בקריאה מוצלחת לexit() עם הערך 1.

* סעיף 3

לאחר שינוי שורה 13 כפי שהוצג מתקיים כעת כי כאשר יישלח signal SIGSEGV התהליך יתעלם ממנו ולא יטפל בו, דבר זה יגרום להתנהגות לא צפויה של התכנית ולכן מערכת ההפעלה תטפל בסיגנל הנ"ל. בפרט התכנית תקרוס ולא תסתיים כהלכה.

לפרט יותר.

**שאלה 3**

* סעיף 1  
  האם יתכן שהאירועים הבאים יגרמו לסיום הריצה המיידי של תהליך A?

|  |  |  |
| --- | --- | --- |
| אירוע | כן / לא | דוגמה / נימוק |
| תהליך A כותב לpipe | לא |  |
| תהליך A קורא ל()kill | כן | שליחת kill(my pid, SIGINT) תשלח kill סיגנל לעצמי ותסיים את התהליך. |

* סעיף 2  
  האם האירועים הבאים יגרמו בהכרח לסיום הריצה המיידי של תהליך A?

|  |  |  |
| --- | --- | --- |
| אירוע | כן / לא | דוגמה / נימוק |
| תהליך A כותב לpipe | לא | כתיבה תקינה לpipe לא מסיימת את התהליך (שכן אחרת אין משמעות לצינור תקשורת שהורג את עצמו) |
| תהליך A קורא ל()kill | לא | שליחת kill(other pid, SIGINT) תשלח kill סיגנל לתהליך אחר ותסיים אותו, התהליך הנוכחי ימשיך בפעולתו. |

* סעיף 3  
  האם האירועים הבאים יגרמו בהכרח למעבר מיידי ממצב משתמש ב-A למצב גרעין?

|  |  |  |
| --- | --- | --- |
| אירוע | כן / לא | דוגמה / נימוק |
| תהליך A כותב ל(pipe) מחובר לתהליך B | כן | ביצוע כתיבה לpipe מנוהלת על ידי הגרעין – בפרט הכתיבה משתמשת בקריאת מערכת sys\_write שיכולה להיות מופעלת רק במצב גרעין. לכן כתיבה לpipe בהכרח מלווה במעבר ממצב משתמש למצב גרעין. |
| תהליך A קורא ל()dup | כן | Dup היא קריאת מערכת ולכן המשתמש חייב לעבור למצב גרעין כדי שתהיה לו גישה אליה. |
| תהליך B קורא מה (pipe) הוא רץ על ליבה שונה מזו של A | כן | כמו הכתיבה, ביצוע קריאה לpipe מנוהלת על ידי הגרעין – בפרט הקריאה משתמשת בקריאת מערכת sys\_read שיכולה להיות מופעלת רק במצב גרעין. לכן קריאה לpipe בהכרח מלווה במעבר ממצב משתמש למצב גרעין. |

* סעיף 4  
  האם execv יוצרת מופע חדש בעבור האובייקטים הבאים?

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| אירוע | Stack | FDT | Heap | PCB |
| כן / לא | כן | לא | כן | לא |