החלק העיוני

שאלה 2

א. פעולת ה TRAP מעבירה את מצב מערכת ההפעלה ממצב משתמש(user mode) למצב ראשוני (kernel mode). היא מתבצעת בעקבות קריאה מתוכננת של המשתמש ל system call או כאשר קורה exception ב user mode (כמו חילוק באפס או גישה לא תקינה לזיכרון). זאת הדרך המקובלת למשתמש ב user process כלשהו לקרוא ל system call על מנת לבצע פעולות בkernel mode שלא ניתן לבצע בuser mode. לאחר סיום בקשת השירות מצב מערכת ההפעלה מועבר חזרה ממצב ראשוני למצב המשתמש ותוכנית המשתמש ממשיכה לרוץ). למעשה ניתן לראות את פעולת ה TRAP במעין סוויץ' למצב ראשוני(kernel mode) שבו מערכת ההפעלה מבצעת פעולות לפני שהיא חוזרת לביצוע הקוד של תוכנית המשתמש.

ב. שלבים בעת הקריאה לפונקציה write של C library

1. הפרמטרים(במקרה שלנו הפרמטרים הם file descriptor, buffer, bytes to write) של הפונקציה write נשמרים באוגרים המיועדים לכך(במקרה שלנו ebx, ecx, edx).
2. שמים באוגר eax את מספר ה system call של write(במקרה שלנו את המספר 4).
3. מתבצעת פקודת ה trap משמע מתבצע switch במצב של מערכת הפעלה(מעבירים את השליטה ממצב המשתמש למצב הראשוני(kernel mode)).
4. מערכת ההפעלה במצב הראשוני(kernel mode) מבצעת את מה שעליה לעשות על פי הsystem call הנקרא והפרמטרים שהתקבלו.
5. ערך ההחזרה נשמר באוגר המיועד.
6. נגמרו הדברים לביצוע במצב הראשוני(kernel mode) ומצב מערכת ההפעלה מועבר חזרה למצב המשתמש ותוכנית המשתמש ממשיכה.

ניתן להבדיל בין המקרים של fast ל legacy באופן בו הפרמטרים מועברים:

- כאשר מדובר ב legacy system call ה Linux kernel מצפה מהמשתמש לשים באוגר eax את מספר ה system call במקרה שלנו מדובר במספר של write. ובשאר האוגרים הכלליים לשים את הארגומנטים בשביל ה syscall. לאחר מכן מתבצע trap מסוג int 0x80 על מנת לעבור למצב הראשוני.
- כאשר מדובר ב fast system call יש לשים בערכי האוגרים בזהה לlegacy **ובנוסף כתובת החזרה והפרמטרים באוגרים נדחפים למחסנית** ולאחר מכן מאוחזרים בשינוי מצב המערכת.

ג. write היא **מעטפת** ל **write system call** אשר ניתן להשתמש בה באמצעות הכללת קובץ ההדר unistd.h אשר מטרתה היא לכתוב מספר מסוים של בתים (אשר ערכם מתקבל בתור פרמטר וגם כמות הבתים לכתוב) לתוך קובץ מסוים אשר ה file descriptor(**אין הגבלה** לקובץ אחד מסוים) שמתקבל בתור פרמטר.

בעוד ש printf היא פונקציה אשר ניתן להשתמש בה באמצעות הכללת קובץ ההדר stdio.h אשר מטרה היא לכתוב סטרינג(אשר מתקבל בתור פרמטר) בפורמט מסוים לקובץ הפלט הסטנדרטי stdout **בלבד**. למעשה printf אפילו **משתמשת ב write** על מנת לבצע את מטרתה.

שאלה 3

פסאדו קוד נדרש:

```
typedef struct Monitor {
        semaphore mutex = 1;
}Monitor;

int run_in_monitor (Monitor *m, int (*fptr)(Queue *q,Node *n), Queue *q, Node *n) {
        int retVal;

        down(&m->mutex); /* enter a critical region */
        retVal = fptr(q, n); /* calling to insert node and saving it's return value*/
        up(&m->mutex); /* leave critical region*/

        return retVal; /* returning insert's return value */
}
```

שאלה 4

לפי פרק 3 של המאמר שדן בנושא הוספת תהליכונים כספריה לשפה שלא תמכה בהם מלכתחילה, התקן של Pthreads אינו מתאר באופן פורמאלי את מודל הזיכרון ואת הסמנטיקה של המקביליות כיוון שלמרות הקוד הכתוב הקומפיילר והמעבד לעיתים יחליפו את סדר הפקודות הכתוב בקוד(בשביל אופטימיזציה בדרך כלל) אשר המתכנת ציפה שיקרה משמע הthreads יפעלו בצורה בה המתכנת לא תכנן שיפעילו.

בכל זאת מפתחי התקן Pthread מסבירים פתרון ל-מהו מודל הזיכרון באמצעות כך שעל מנת למנוע חוסר סכרון אשר גורם להשמות לא צפויות/רצויות וכדומה נשתמש בפונקציות כמו pthread_mutex_lock(), pthread_mutex_unlock() על מנת שתהיה הבטחה לכך שthreads בעלי משאבים מסוימים משותפים לא יוכלו לבצע שינוי במשאב המשותף בזמן שthread מסוים משתמש במשאב.

שאלה 6

נניח כי תהליך 0 מעוניין להיכנס לקטע קוד קריטי; ישנם שתי אפשרויות:

1. תהליך 1 אינו מעוניין להיכנס לקטע קוד קריטי וכתוצאה תהליך 0 נכנס ישר לקטע הקוד הקריטי ללא המתנה.
2. תהליך 1 מעוניין להיכנס לקטע קוד קריטי. במצב זה ישנם שתי אפשרויות.
   2.1. תהליך 0 עדכן ראשון את ערך המשתנה turn ולכן לפי עמוד 63 במדריך הלמידה הוא הראשון אשר נכנס לקטע הקוד הקריטי.
   2.2. תהליך 1 עדכן ראשון את ערך המשתנה turn, משמע הוא נכנס ראשון לקטע הקוד הקריטי ותהליך 0 מחכה בלולאה ועוצר. לאחר שתהליך 1 מסיים את קטע הקוד הקריטי הוא משנה את ערך המשתנה interested[1] ל false וכתוצאה מכך תהליך 0 יוצא מן הלולאה ונכנס לקטע הקוד הקריטי.

ובכך הראנו כי לא משנה מה, אם תהליך מעוניין להיכנס לקטע קוד קריטי, בסופו של דבר הוא יכנס.

בפרט, תהליך שרוצה להיכנס לקטע קוד קריטי אינו ממתין יותר ממה שלוקח לתהליך אחר להיכנס ולעזוב את הקטע הקריטי, כיוון שאם תהליך רוצה להיכנס לקטע קוד קריטי בזמן שתהליך אחר נמצא בו או בדיוק נכנס אליו, ברגע שהתהליך אשר נמצא בקטע הקוד הקריטי יצא ערך המקום המתאים במערך interested ישתנה, וכתוצאה מכך התהליך שחיכה מפסיק לחכות כיוון שתנאי הלולאה אינו מתקיים יותר וישר נכנס לקטע הקוד הקריטי.