ממן 11 – מערכות הפעלה

שם: אוראל בריאנצב

ת"ז:318264132

2.א) פעולת TRAP מעבירה את השליטה מuser mode לkernel mode.
כאשר לתוכנית יש צורך בשירותים ממערכת ההפעלה,נשלחת בקשה בצורת system call ופעולת הTRAP מבצעת את מעבר השליטה לkernel לצורך ביצוע פעולות של מערכת ההפעלה (לדוגמה,פעולות עם זיכרון).
כאשר מבוצעת הTRAP,המעבד עובר לכתובת של פונקציית ה-handler של מערכת ההפעלה ומשנה את ההרשאות למעבד לkernel .
לאחר שמערכת ההפעלה מסיימת את הפקודות הרלוונטיות,השליטה חוזרת לuser.

ב) בעת קריאה לפונקציית ()write מתבצעים השלבים הבאים:

- מתבצעת העברה של שלושה פרמטרים אשר ערכם נשמרים ברגיסטרים מיועדים.
  שלושת הפרמטרים הם:
  - fd - יעד הפלט
  - buff - מצביע להודעה הנפלטת
  - nbytes - מספר התווים בהודעה

- מספר הsyscall של פונקציית write (4) נשמר ברגיסטר ייעודי (מסופק על ידי מערכת ההפעלה).
- מתבצעת הוראת TRAP המתוארת בסעיף הקודם.
- מערכת ההפעלה מבצעת בדיקה על תקינות הפרמטרים שהתקבלו. במידה והפרמטרים תקינים, מתבצעת הדפסה.
- השליטה חוזרת לתוכנית המשתמש.

ג.פונקציית write היא בעצם פונקציית מעטפה לwrite syscall.הפונקציה מקבלת את פרמטר buff המצביע על הערך שייכתב בfd (שהוא קובץ היעד). לעומת זאת ,printf מקבלת מצביע למחרוזת המופיעה כפורמט.לאחר מכן,פונקציית printf משתמשת בפונקציית write.

3.ההוספה מסומנת בצהוב

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

int run_in_monitor (int * m, int (* f)(int count, va_list list), int count, ...){
int result = 0;
va_list list;
va_start (count,list);
down(&m.mutex); // lock monitor
result = f(count, list);
up(&m.mutex);//unlock monitor
va_end(list);
return result;
  {
```

4)תקן Pthreads אינו מתאר באופן פורמלי את מודל הזיכרון ואת הסמנטיקה של המקביליות משום שהוספת ספריה לשפה שלא תמכה בתהליכונים תוביל לעלויות ביצועים גבוהות משום שהמעבד והקומפיילר עלולים לשנות את סדר ביצוע הפעולת לטובת שיפור בביצועים. היות וההמעבר והקומפיילר אינם מודעים לthreads הם עלולים לגרום לבעיות סנכרון בין הthreads.

מפתחי התקן מסבירים שכל תוכנה צריכה לוודא בעצמה שמידע המשותף בין תהליכים אינו משתנה על ידי תהליך\תהליכון בזמן שתהליך\תהליכון אחר קורא אותו. התקן מספק את פונקציות pthread_mutex_unlock / pthread_mutex_lock על מנת לבצע זאת.

5)נוכיח כי בפתרון של Peterson ל2 תהליכים,תהליכים אינם ממתינים זמן אינסופי על מנת להיכנס לקטע הקריטי:
לצורך ההוכחה נסמן את שני התהליכים – תהליך 0 ותהליך 1. נניח (ללא הגבלת הכלליות) כי תהליך 0 מנסה להיכנס לקטע הקריטי-במצב כזה יש 2 אפשרויות:

(1).תהליך 1 נמצא בקטע הקריטי – במקרה כזה,התהליך הראשון מבין השניים שיעדכן את ערכו של turn יכנס ראשון לקטע הקריטי.

(2).תהליך 1 מחוץ לקטע הקריטי-במקרה כזה,תהליך 0 יכנס לקטע הקריטי.

נניח כי תהליך 1 הוא הראשון שעדכן את ערכו של turn ל-1.אז תהליך 1 נמצא בקטע הקריטי ותהליך 0 נמצא בלולאה שמונעת ממנו להתקדם.
כאשר תהליך 1 מסיים את הקטע הקריטי, הוא משנה את הinterested שלו false.כאשר תהליך 0 יקבל זמן ריצה הוא יצא מהלולאה ויכנס לקטע הקריטי.

במידה ותהליך 0 לא יקבל זמן ריצה,תהליך 1 יינסה להיכנס שוב לקטע הקריטי,הוא ישנה את ערך הinterested שלו ל true ואת ערכו של turn ל-1 ואז יכנס ללולאת ה while (תהליך 0 לא קיבל זמן ריצה) ותהליך 1 יחכה.
כאשר תהליך 0 יקבל זמן ריצה,הוא לא יעמוד בתנאי הלולאה (כי ערכו של turn שונה ל-1) וייכנס לקטע הקריטי.
בשני המקרים שציינתי,כל תהליך 1 נכנס לכל היותר פעם אחת לקטע הקריטי,בזמן שהתהליך 0 ממתין בלולאה.בכל אחד מהמקרים ,זמן ההמתנה של תהליך 0 לא ארוך יותר מזמן הריצה של תהליך 1 בקטע הקריטי.