**ממן 11 – החלק העיוני**

שאלה 2:

1. TRAP היא פקודה שמעבירה את ה"שליטה" למערכת ההפעלה, היא נקראת בכל פעם שיש system call או בexceptions בuser mode (לדוגמא חלוקה ב0 או גישה לזיכרון בכתובת לא תקינה).

כלומר כל תוכנית צריכה לבקש שירותים מהמערכת הפעלה (לדוגמא עבודה עם זיכרון), בקשת השירות הזו מתבצעת בעזרת system calls ומעבר לביצוע פקודות של מערכת ההפעלה בkernel, הפקודה TRAP מבצעת את הswitch הזה בין הuser mode לkernel mode.

לאחר שבקשת השרות נגמרה ובוצעה על ידי מערכת ההפעלה, השליטה חוזרת לתוכנית המשתמש.

1. השלבים בקריאה לפונקציית write:

* הפרמטרים של הפונקציה (output\_file, buffer, buffer size), נשמרים ברגיסטרים מיועדים.
* האינדקס של הwrite syscall מועבר לרגיסטר שמוגדר שבו יהיה המספר של הsyscall
* מתבצעת פקודת TRAP – להעברת השליטה לkernel
* מערבת ההפעלה בkernel מקבלת את הפרמטרים ואת מספר הsyscall ומבצעת את קטע הקוד המתאים שמטפל בקריאת המערכת.
* המערכת מסיימת את הריצה ומעבירה את התשובה ברגיסטר יעודי, והשליטה חוזרת לתוכנית המשתמש.

ההבדל בא לידי ביטוי בשיטה שבה מועברים הפרמטרים למערכת.

Legacy system call : הפרמטרים נשמרים באוגרים (במקרה הזה ebx,ecx,edx) הערך של הsyscall מושם בeax (במקרה הזה ערך הsyscall של write הוא 4) ואז מתבצע int 0x80 שזה trap instruction.

Fast system call: הפרמטרים נדחפים למחסנית בסדר הבא: כתובת החזרה, הפרמטרים (נדחפים מהאוגרים ebx,ecx,edx) ומתבצע trap instruction, כך שבסיום התוכנית תדע לאחזר את הפרמטרים מהמחסנית ולאן לחזור.

1. Write היא פונקציית מעטפת לsyscall write שמקבלת מצביע וכותבת את הערך שאליו הוא מצביע בfd. בעוד printf היא פונקציה שמקבלת מחרוזת כformat עם פרמטרים ומדפיסה את הפלט לstdout.

Printf משתמשת גם היא בwrite.

שאלה 3:

הפתרון מבוסס על ההסבר בעמוד 102 במדריך הלמידה:

לכל סמפור S נבנה תיבת הודעות mailbox הנגישה לכל התהליכים שצריכים גישה לסמפור, מאותחלת בהודעה 0.

כמו כן תיבה ריקה L שמאותחלת בהודעה יחידה

הסבר: מעלים את המבנה – אם זה איבר ראשון שולחים הודעה לממתינים לאיבר הראשון.

Up(S)

{

Recv(L)

n = Recv (mailbox)

If (n=0)

{

Send (mailbox, n+1)

}

Send(L,1)

Send(mailbox, msg);

}

הסבר: מעלים את המבנה, אם n=0 מחכים להודעת האיבר הראשון.

Down(S)

{

Recv (L)

n = Recv(mailbox)

if (n=0)

{

Send(mailbox, n)

Send(L, 1)

Recv (e)

} else {

Send(e, 1)

}

}

שאלה 4:

התקן אינו מתאר באופן פורמאלי את מודל הזיכרון ואת הסמנטיקה של המקביליות בPthread משום שעלות הביצועים של הוספת ספריה לשפה שלא תמכה בתהליכונים מלכתחילה תהיה גבוהה, עקב כך הקומפיילר והמעבד עלולים לשנות את סדר ביצועם הפעולות כדי לשפר את הביצועים, ומכיוון שהם לא מודעים לthreads הם יגרמו לחוסר סנכרון בין הthreads, ודברים יתבצעו לא כמו שתוכנן.

לכן, מפתחי התקן מסבירים בכל זאת להשתמש בפונקציות pthread\_mutex\_lock, pthread\_mutex\_unlock שיבטיחו שמידע משותף בין threads לא ישתנה על ידי תהליך אחר בעוד תהליך אחר קורא/משנה אותו.

שאלה 6:

נניח שתהליך 0 מנסה להיכנס לקטע הקריטי שלו, ישנן 2 אפשרויות:

תהליך 1 מחוץ לקטע הקריטי – ואז תהליך 0 יכנס מיד.

תהליך 1 נמצא בקטע הקריטי – הראשון שיעדן את turn יכנס ראשון לקטע הקריטי.

נניח שהראשון שעדכן את turn הוא תהליך 1 והוא נכנס לקטע הקריטי ראשון, בינתיים תהליך 0 מחכה בלולאה ועוצר. ברגע שתהליך 1 מסיים את הקטע הקריטי הוא משנה את הinterested שלו לfalse ,עקב כך ברגע שתהליך 0 יקבל זמן ריצה הוא יצא מהלולאת waiting שלו ויכנה לקטע הקריטי כך שהוא חיכה רק פעם אחת שתהליך 1 היה בקטע הקריטי.

במידע ותהליך 0 לא מקבל זמן ריצה, תהליך 1 מנסה להיכנס שוב לקטע הקריטי, הוא משנה את הinterested שלו ל true ואת המשתנה turn ל1, ואז הוא יכנס ללולאת הwaiting שלו שמזהה שתהליך 0 ביקש זמן ריצה ולא קיבל, והוא מחכה. כעת מתי שהוא תהליך 0 יקבל זמן ריצה, לא יקיים את תנאי לולאת הwait שלו ויצא ממנה ויכנס לקטע הקריטי.

בכל אחד מהמקרים תהליך נכנס לכל היותר פעם אחת לקטע הקריטי בזמן שהתהליך השני מחכה, בגלל שברגע שהוא מנסה להיכנס שוב לקטע הקריטי הוא נעצר בלולאת הwait, עד שהתהליך השני יקבל זמן ריצה, יכנס לקטע הקריטי שלו, ויצא, וכך שוב..