חלק עיוני – ממ"ן 11

1. א)

פקודת ה-TRAP היא קריאה שמעבירה את השליטה ממצב המשתמש למצב הKernel.

מדובר בפסיקה שמתבצעת על ידי תוכנית המשתמש או על ידי שגיאה (כאשר מערכת ההפעלה נדרשת על ידי התוכנית לבצע קריאות מערכות או פעולה). הפקודה דומה לקריאת מערכת בכך שהפקודה הבאה נלקחת ממיקום רחוק ושכתובת החזרה נשמרת במחסנית לשימוש בהמשך, אך שונה ממנה בכך שהיא עוברת למצב Kernel ובכך שבמקום לתת כתובת של מיקום הפרוצדורה, הפקודה לא יכולה לקפוץ לכתובת שרירותית – או שהיא קופצת למקום קבוע או שיש שדה בן 8 ביטים בפקודה אשר נותן את האינדקס לטבלה המכילה את הכתובות לקפוץ אליהן.

ב) הקריאה לפונקציית write כותבת מידע מbuffer שהוכרז על ידי המשתמש על ידי מכשיר נתון או קובץ. היעד מזוהה על ידי קוד מספרי. הנתונים מוגדרים על ידי מצביע לbuffer

בו הם מאוחסנים ועל ידי מספר המציין את מספר הביטים אשר נכתבים מהbuffer.

הפרמטרים הם קוד הקובץ, המצביע לbuffer ומספר הביטים (fd, buf, nbytes).

הקריאה היא פונקציה ברמה נמוכה, לכן פונקציות I/O ברמה גבוהה יותר קוראות

לפונקציות אחרות כדי לאסוף נתונים ואז קוראות לwrite כדי לכתוב את הנתונים.

ההבדל בין הקריאה write לקריאה printf הוא שWRITE מתוכנן לכתוב סדרת ביטים בלבד, אך מכיוון שקריאה זו בסיסית בלבד, יש את הקריאה printf אשר מאפשרת לנו לכתוב נתונים בפורמטים שונים(כגון float). Printf למעשה ממירה את המידע לסדרה של ביטים ואז קוראת לwrite לכתוב את אותם ביטים לפלט. קריאת מערכת כזאת עולה בביצועי המערכת ולכן בפונקציות הדפסה רבות יש buffering מובנה אשר דואג שwrite לא תמיד תקרא רק במקרה הצורך.

1. פתרון התור אינו מהווה פתרון סביר כיוון שהוא מפר את תנאי 3 ( אף תהליך שרץ מחוזר לאיזור הקריטי לא יכול לחסום תהליכים אחרים). במצב שבו יש תהליך שאיטי משמעותי יותר מתהליך אחר, יצירת תור בין התהליכים עלול לגרום לבעיה – במצב שבו תהליך אחד מסיים את פעולתו באיזור הלא קריטי ומתחיל לולאה, הוא לא ראשי להכנס לאיזור הקריטי כיוון שמשתנה הturn מוצב כ1. באותו זמן, התהליך השני עדיין עסוק באיזור הלא קריטי, מה שמוביל לכך שהתהליך הראשון ממתין בלולאה עד שהתהליך השני יסיים ויציב 0 ב-turn. בכך למעשה, תהליך אחד נחסם על ידי תהליך אחר שלא נמצא באיזור הקריטי שלו – הפרה מובהקת של תנאי 3.
2. תקן Pthreads אינו מתאר באופן פורמלי את הסמנטיקה ומודל הזיכרון הסמנטיקה של המקביליות הממומשו כיוון שהגדרות רשמיות של מודל הזכרון נחשבות בלתי קריאות על ידי כמות רבה של מתכנתים. בנוסף, רוב המידע בנושא התרכז בזיכרון המסופק על ידי החומרה ולא על ידי המתכנת דרך הקומפיילר וזמן הריצה. התקן קובע כי תוכנות יוודאו שגישה לכל מיקום זיכרון על ידי יותר מתהליכון אחד של שליטה, תוגבל כך שאף תהליכון שליטה לא יכול לקרוא או לשנות מיקום זיכרון בזמן שתהליכון שליטה אחר משנה אותו. גישה שכזאת מוגבלת באמצעות פונקציות שמסנכרנות את ביצעו התהליכונים ומסנכרנות את הזיכרון תוך כדי התייחסות לתהליכונים אחרים.

בשפות כמו C וC++ נהוג לבצע כך על מנת לקיים את מודל הזיכרון : 1) פונקציות כגון pthread mutex lock() שדואגות לסנכרן את הזיכרון וכוללות הוראות חומרה על מנת למנוע סידר מחדש של זיכרון בזמן הפעולה המתבצעת 2) כדי למנוע סידור מחדש של הזיכרון בזמן קריאות לפונקציות כמו הנ"ל, הקומפיילר מניח כי הקיראה עלולה לקרוא או לכתוב בכל משתנה גלובלי. בכך, התייחסות זיכרן לא יכולה לזוז ולהשתנות בזמן הקירה. גישה זו מבטיחה שקריאות טרנזיביטות מנוהלות באותה צורה ולא זזות בזמן הקריאה. גישה זו לא בהחלט נכונה ולא תמיד עובדת וגם תוכניות נכונות עלולות להכשל מדי פעם כתוצאה מסתירות בלוחות זמנים ש תהליכונים. בנוסף, שיטה זו עלולה לגרום לסתירה עם פתרונות אלגוריתמייים יעילים.

1. בפיתרון פיטרסון, החלק הקריטי הוא סופי והתהליך הנכנס יצא לפני שהוא יתן לתהליך אחר להכנס – כאשר תהליך יכנס, הוא יבדוק אם יש עוד תהליך אחר שרוצה להכנס. התהליך לא ימתין זמן רב מדי להכנס לאיזור הקריטי, כיוון שהתהליך שכבר באיזור הקריטי יבצע את לולאת הגיון שלו בזמן ששאר התהליכים הם בbusy waiting. ברגע שהתהליך הנמצא באיזור הקריטי יבקש לעזוב את האיזור הקריטי, תהליך ההמתנה יכנס מיד לאיזור הקריטי. אחת הלולאות while תהפוך לfalse וכך התהליך יכנס לאיזור הקריטי. תהליך זה לא יקח זמן אינסופי ולא יגרום לstarvation של ההמתנה במהלך לולאת הwhile, כי אנו מניחים שהתהליך באיזור הקריטי יסיים את כל הפקודות שלו בזמן קצוב.