שאלה 2

א. Trap to Instruction – זה פסיקה המתעוררת כתוצאה משגיאת המערכת או בקשה מפורשת של תוכנית משתמש (User mode) לבצע שירות של מערכת הפעלה (Kernel mode).ישנו ביט המתווסף לחומרת המחשב mode bit המציין מהו המצב הנוכחי ,כאשר 1 זה User mode ו 0 זה Kernel mode.כשמתבצע Trap to Kernel עוברים לכתובת מסוימת (בד"כ כתובת נמוכה שתמיד שייכת למערכת הפעלה) דרך וקטור הפסיקות, ולכן המשתמש לא יכול להעביר את התוכנית שלו ל- Kernel mode.מערכת הפעלה מקבלת שליטה בחזרה ו mode bit נכבה. מערכת הפעלה מעבירה חזרה ל User Mode כאשר היא מסיימת טיפול בפסיקה.בשל שימוש במשאבי מערכת משותפים של מערכת ההפעלה יש להבטיח שטעות בתוכנית אחת לא תגרור לטעויות בתוכניות אחרות . לצורך כך למערכת ההפעלה יש שני מצבי עבודה.

ב. הפונקציה ()write היא מעטפת קוד, המפעילה באמצעות פסיקת תוכנה את קריאת המערכת שמבצעת את השירות המבוקש, ומחזירה את תוצאת ביצוע השירות לקוד, במקרה של כישלון, מוחזר הערך 1-, וסוג השגיאה מוחזר במשתנה גלובלי הקרוי errno. כל קריאות מערכת ההפעלה ב-Linux מטופלות דרך פסיקת תוכנה אחת,מספר 128 (0x80). בהתחלה צריך להכין את הפרמטרים בתוך המחסנית כפי שפונקצית ביצוע השירות מצפה להם.סוג השירות המבוקש נקבע באמצעות מספר שירות המועבר ברגיסטר eax על-ידי פונקצית המעטפת.בתגובה לפסיקה המעבד עובר מ- User Mode ל- Kernel Mode או Trap to Kernel. System call בהיותה שגרת טיפול בפסיקה, פועלת תחת כללים legacy system call. העברת הפרמטרים לשגרה היא באמצעות רגיסטרים בלבד שכוללים: מספר השירות המבוקש, ברגיסטר eax, פרמטרים עבור השירות לפי הצורך, ברגיסטרים הבאים ebx, ecx, edx, esi, edi, ebp. בעבור פרמטרים הגדולים מ-32 ביט system call פועלת תחת כללים fast system call:מועבר מצביע לפרמטר ברגיסטר יחיד ואם נדרשים יותר מ-6 פרמטרים עבור השירות: מועבר רגיסטר יחיד המצביע לרשומה בזיכרון של התהליך המכילה את כל הפרמטרים.

ג.קריאת מערכת ()write דורשת Kernel mode לאומת ()printf שזה ספרייה סטנדרטית ועובדים איתה ב- User mode.קריאת מערכת ()write נועדה רק לכתוב רצף של בתים וזה נחשב בסיסי מדי לאומת ()printf שיכולה לכתוב נתונים בפורמטים שונים.אבל בסופו של דבר ()printf פונה ל-()write כדי לכתוב את הבתים שהיא הכינה.בעצם אפשר להגיד ש-()printf נועדה להקל אלינו ברמת המשתמש.

שאלה 3

פיתרון התור (strict alternation) איננו מהווה פתרון סביר מפני שאם קיים הבדל בזמן הביצוע של הקטעים הלא קריטיים של תהליכים,תיתכן הפרה של תנאי 3 של הפיתרון הסביר.התהליך שנמצא מחוץ לקטע קריטי יכול למנוע מתהליך אחר להיכנס לקטע קריטי שלו.הפיתרון אומר תורי-תורך עד שאני לא מסיים אתה לא תצליח להיכנס.

שאלה 4

תקן Pthereads אינו מתאר באופן פורמאלי את מודל הזיכרון והיה נדחה על ידי הרוב המכריע של מתכנתים.רוב העבודה התמקדה בזיכרון המסופק על ידי חומרה ולא תכנות היישומים דרך מהדר ובדיקות זמנים ריצה של המערכת.חוץ מזה נושאים של בטיחות והבטחה של מערכת יכולים להיות קריטי.במיוחד יש סכנה כי הרעיון הוא שיישומים יבטיחו גישה לכל מיקום זיכרון על ידי תהליכונים שמערכת הפעלה תעבוד על גלוי גם לתהליכונים אחרים.הסנכרון זיכרון שמדברים במאמר יעלה ביוקר מבחינת זמן המערכת. וכמובן חלק מהקומפיילרים והחומרה עלולים לשנות את סדר הפעולות של הזיכרון לצורך שיפור הביצועים עם הוספת ההליכונים.

אבל זה נראה סביר,אינטואיטיבי,כי בעצם מודל זיכרון יכול להיות מיושם עם ביצועים סבירים על ארכיטקטורה סטנדרטית.הכנסה תהליכונים עלולה לגרום ללוח זמנים טוב יותר להוראה אחרי שיפורים בביצועים. ע"י שימוש בפונקציות pthread_mutex_lock / pthread_mutex_unlock שהתקן מספק, פעולות זיכרון לא עוברות קטע קריטי וגם ייתכן שאין צורך בחסמי זיכרון מלאים .למתכנתים יהיה יעיל יותר וקל להשתמש בהצהרות פשוטות.אבל זה כמובן אינטואיטיבי ורוב חושבים שלא כדי להוציא מודל הזיכרון הזה בפועל.

שאלה 6

באלגוריתם Peterson מדובר על שני תהליכים/תהליכונים ראשון P0 ,שני P1. כאשר אחד התהליכים נמצא בקטע קריטי שלו,לא משנה איפה תתבצע context switch וכמה פעמים,התהליך השני לא יוכל להיכנס לקטע קריטי שלו,כלומר לגשת למשאב המשותף בזמן שהתהליך הראשון לא סיים להשתמש בו.יש לציין שהאלגוריתם עובד בשיטה של המתנה פעילה(busy waiting) כלומר,כאשר הקטע הקריטי לא סיים לרוץ,כל זמן שלא הגיע תור התהליך השני להריץ את הקטע הקריטי הוא יהיה תקוע בלולאה ה-while וידרוש זמן מעבד.בכדי ששני תהליכים יהיו בזמן המתנה יש צורך בערכים הפוכים של משתנה ממתין turn,דבר אינו אפשרי.ייתכן שתהליך ראשון יכנס כמה פעמים ברציפות לקטע קריטי ותהליך שני יחכה,באלגוריתם Peterson לא קיים דבר כזה,תהליך שרוצה להיכנס לקטע קריטי לא ימתין יותר ממה שלוקח מתהליך אחר להיכנס ולעזוב את קטע קריטי,כי ברגע שתהליך יוצא מקטע קריטי הוא מוגדר כממתין וימתין שם עד שתהליך השני לא יסיים ויצא מקטע קריטי.