מערכות הפעלה 20594

ממן: 11
מגיש: טל גלנצמן
ת.ז: 30280035
תאריך: 19/11/2021

שאלה 2

סעיף א

כאשר בקר הפסיקות מקבל פסיקה מספר X כלשהי, הבקר מצד אחד מרים דגל למעבד שמצהיר על עלייה של פסיקה ומצד שני מכניס למעבד את מספר הפסיקה.

כאשר המעבד מוכן לעבד את הפסיקה, מתבצעת הפקודה TRAP אשר בה

- המעבד עובר למצב מיוחס
- רגיסטרים הדרושים לחזרה נדחפים למחסנית (pc, psw...)
- הרגיסטר pc מקבל את הערך שנמצא ברשומה ה-X בוקטור הפסיקות (ערך זה למעשה מבטא את כתובת הפונקציה לביצוע בעת הפסיקה הנתונה)
- המעבד ממשיך בביצוע הפקודות (נזכור כי כעת ה-pc מצביע על הפונקציה)
- בסיום, נשלפים הנתונים מהמחסנית חזרה אל האוגרים הרלוונטים (בפרט pc)
- המעבד מועבר חזרה למצב משתמש
- המעבד חוזר לעבודה שגרתית מהיכן שעצר קודם לכן במצב משתמש בביצוע התוכנית

סעיף ב

במקרה ה-legacy, שהרי הפרוטוקול בלינוקס לקריאת מערכת מציין כי בעת פסיקה תוכנתית מספר 0x80, ייקרא האוגר הכללי הראשון להיות מספר פקודת המערכת לביצוע. שאר האוגרים הכלליים, עפ"י סדרם יהוו את הפרמרטים האחראיים על אופן ביצוע הקריאה - בפועל זה תלוי במימוש וייתכן שהעברת הפרמטרים היא דרך המחסנית. כמובן, מסעיף קודם, ביצוע הקריאה יהיה במצב מיוחס.

מהאמור, פונקציית הספרייה write, תכתוב את הערך 4 לאוגר הראשון, את הפרמטרים file descriptor, מחרוזת לכתיבה ואורך המחרוזת תדחוף למחסנית, ותבצע פסיקה תכנתית עם המספר 0x80. לבסוף, היא תקרא ערך ההחזר מהאוגר הראשון.

במקרה ה-fast, פונקציית הספרייה תמצא את הכתובת בה יושבת הפונקציה "__kernel_vsycall", תכתוב את הערך 4 לאוגר הראשון ואת הפרמטרים כמו במקרה ה-legacy, ותקרא לפונקציה "__kernel_vsyscall". בנוסף, למען החזרה, תירשם כתובת החזרה לאוגר edx, שכן הקוד המיוחס יבצע sysexit שמסתמך על כך שכתובת החזרה רשומה שם.

סעיף ג

פונקציית הספרייה printf מכילה לוגיקה נוספת מלבד כתיבה שכן היא מעבדת פורמט נתון ופרמטרים למחרוזת אחת. רק לאחר מכן אותה מחרוזת נכתבת לקובץ. בפועל, יצירת המחרוזת המפורמטט אינה דורשת הרשאות מיוחסות או קריאות מערכת, ולכן חלק זה יתבצע במצב משתמש. רק לאחר בניית המחרוזת, תתבצע קריאת מערכת עם אותה מחרוזת. המוטיבציה היא לבצע כל שניתן מחוץ למצב מיוחס.

שאלה 3

```
/*
 * insert a node to a queue
 * @param q the queue to insert a node to
 * @param n the node to insert to the queue
 * @return the current total number of elements in the queue (?)
 */
int insert(Queue *q, Node *n);

typedef struct Monitor {
  /* assuming the semaphore is an integer */
  int semaphore;

  /* a reference to the monitored queue */
  Queue *queue;
} Monitor;

/*
 * creates a new Monitor instance
 * @param queue the queue to provide mutual excluded access to
 */
Monitor create_monitor(Queue *queue) {
  return {
    .semaphore = 1,
    .queue = queue
  };
}

/*
 * runs some function (with the specified prototype) in a monitored environment and in a mutual exclusion way.
 * @param m a reference to a Monitor
 * @param fptr the function to invoke on the monitor's queue
 * @param node the node to provide to fptr alongside the monitor's queue
 * @return the returned value of fptr
 */
int run_in_monitor(Monitor *m, int (*fptr)(Queue *q, Node *n), Node *node) {
  down(&m->semaphore);
  int ans = fptr(m->queue, node);
  up(&m->sempahore);
  return ans;
}
```

שאלה 4

הספסיפיקציה בחרה לא להגדיר את הסמנטיקה שכן

1. הגדרות פורמליות של מודל הזכרון לא היו מובנות דיים לרוב המפתחים
2. ההגדרות הרווחות מתעסקות במודל זכרון כפי שנובע מהחומרה ולאו דווקא מהאפליקציה. לדוגמא,

- הקומפיילר יכול לבחור להזיז מבני זיכרון מסויימים כפי שנראה לו לנכון בתנאים מסויימים וכאשר אלו נמצאים באותו תהליך
- החומרה בעצמה יכולה לבצע פקודות זיכרון בסדר שונה (שמירה ביחס לטעינה מהזיכרון)

על מנת למנוע את חוסר הבהירות, הספסיפיקציה בוחרת לא להגדיר מודל כזה בבהירות, אלא לתת למפתח יכולות חזקות לסנכרן תהליכים באמצעות פונקציות הנתונות ע"י ממשי הספריות. לאותן פונקציות יש משמעות והבטחות (gaurantees) ברורות שאותן יש לממש ולאכוף מטעם הממשים. כעת, במקום להבין מראש מודל זכרון ומקביליות כלשהם, על המפתח להשתמש באותם מימושים כרצונו ועפ"י דרישות המערכת שאותה הוא מפתח.

שאלה 5

נניח שתהליך 1 מחכה להיכנס לקטע קריטי שתהליך 2 מבצע.

כאשר תהליך 2 יוצא מהקטע הקריטי, הוא משנה את הדגל "interested" לשלילי.

מכאן ואילך יכולים לקרות שני מקרים

מקרה א' וה"קל" מבחינת המקביליות"די מהר" תהליך 1 קיבל זמן חישוב, בדק את "interested" של תהליך 1 שהינו שלילי ולכן נכנס לקטע הקריטי.

מקרה ב' ובמקרה "הכי גרוע", תהליך 1 לא קיבל זמן חישוב כלל ותהליך 2 ממשיך לקבל את מלוא תשומת הלב של המעבד ומגיע שוב עד ל"זמן בדיקת" כניסה לקטע הקריטי. נשים לב כי בהגיעו הוא שינה את האינדיקציה "turn" כי הוא זה שמתנדב להיות הבא שמחכה וכרגע הוא בעצמו נכנס ל-"busy waiting" ותהליך 2 יהיה הבא ל-"לא לחכות, ולהיכנס". לא משנה כמה זמן ייקח למתזמן עד שייתן זמן חישוב לתהליך 1, כרגע תהליך 2 (שהתנדב לכך) מחכה, ולבסוף תהליך 1 ייכנס לקטע הקריטי, ורק בצאתו יהיה רשאי תהליך 2 להיכנס.