מרצים: פרופ' דן צפריר, דר' לאוניד רסקין

מתרגלים: ארתור קייאנובסקי, אריה טל, אסף רוזנבאום, גיל קופפר, מתיאס בונה, סאהר עודה

מערכות הפעלה (234123)

סמסטר אביב תשע"ו

מועד א'

הנחיות

1. ענו על כל השאלות; מלאו מספר סטודנט בעמוד זה.
2. אסור שימוש בכל חומר עזר
3. משך המבחן: שלוש שעות.
4. במבחן זה 23 דפים, כולל דף זה. וודאו שכל הדפים נמצאים.
5. יש לענות על כל השאלות בטופס הבחינה (מחברת הבחינה משמשת לטיוטה בלבד).
6. **תשובות לא מנומקות לא תתקבלנה**

ת.ז.: _____

שאלה 1	24
שאלה 2	25
שאלה 3	24
שאלה 4	27
סה"כ	100

בהצלחה!

שאלה 1 - פסיקות וסיגנלים (24 נק')

השאלות מתייחסות לחומר שנלמד בהרצאה. יש לנמק את כל התשובות.

א. (4 נק') פסיקה פנימית (internal interrupt) הינה:

1. פסיקה שנגרמת ע"י התקנים פנימיים במחשב (דיסק, שעון, וכו') ולא חיצוניים (עכבר, דיסק USB וכו').
2. פסיקה שמקורה בשליחת נתונים שמגיעים מתוך המחשב אל הרשת (חבילה יוצאת) ולא מהרשת אל המחשב (חבילה נכנסת).
3. פסיקה סינכרונית.
4. פסיקה שמתרחשת למשל כשיש page fault.
5. Inter Processor Interrupt (פסיקת IPI).
6. יש יותר מתשובה אחת נכונה (אם כן, רשום איזה תשובות בנימוק).
7. אין תשובה נכונה.

ב. (4 נק') פסיקה משתמעת (implicit interrupt) הינה:

1. פסיקה שניתן לדעת עליה רק בדיעבד כתוצאה מאירוע אחר.
2. פסיקה שמתרחשת למשל כתוצאה מחלוקה באפס.
3. פסיקה שמתרחשת למשל כתוצאה מקריאה ל- system call.
4. פסיקה שמתרחשת למשל כשמדבגים תהליך.
5. פסיקה שבגינה התהליך מת.
6. יש יותר מתשובה אחת נכונה (אם כן, רשום איזה תשובות בנימוק).
7. אין תשובה נכונה.

ג. (4 נק') שיגרת הפסיקה (interrupt handler) שמוצבעת ע"י ה- IDT:

1. נקראת מייד כשמתחוללת הפסיקה, אלא אם תוכנית המשתמש נמצאת בקטע קריטי (critical section).
2. נקראת מייד כשמתחוללת הפסיקה, אבל חלקה יתבצע בהמשך.
3. נקראת כאשר המתזמן (scheduler) מחליט, אלא אם כן מדובר בפסיקה דחופה.
4. רצה תמיד על אותה ליבה בעבור מספר פסיקה (interrupt vector) נתון, כך שכל מספר פסיקה משיוך לליבה אחרת.
5. רצה בכל פעם על ליבה אחרת לפי סדר Round Robin.
6. יש יותר מתשובה אחת נכונה (אם כן, רשום איזה תשובות בנימוק).
7. אין תשובה נכונה.

ד. (4 נק') איזה מבין ההיגדים הבאים נכון:

1. לתוכנית מותר לחסום את כל הסיגנלים שלה.
2. לתוכנית אסור לחסום את כל הסיגנלים שלה אך היא יכולה להתעלם מכולם.
3. לתוכנית מותר לחסום חלק מהסיגנלים שלה כל עוד היא תאפשר אותם בהמשך.
4. יש 2 סיגנלים שלא ניתנים לחסימה.
5. יש 3 סיגנלים שלא ניתנים להתעלמות.
6. יש יותר מתשובה אחת נכונה (אם כן, רשום איזה תשובות בנימוק).
7. אין תשובה נכונה.

ה. (4 נק') איזה מבין ההיגדים הבאים נכון:

1. סיגנלים יכולים לאפשר עבודה אסינכרונית עם קלט/פלט.
2. סיגנלים יכולים לאפשר למתשמשים לממש מתזמן תהליכים ב- user level.
3. סיגנלים יכולים לאפשר לתוכניות לדווח למשתמשים מה הן עושות בזמן ריצה.
4. סיגנלים יכולים לאפשר לתוכניות להגיב ל- segmentation fault.
5. סיגנלים יכולים לאפשר לתהליך הורה לדעת מתי תהליכי הילד שלו נעצרו.
6. יש יותר מתשובה אחת נכונה (אם כן, רשום איזה תשובות בנימוק).
7. אין תשובה נכונה.

ו. (4 נק') איזה מבין ההיגדים הבאים נכון:

1. יש לעבוד עם סיגנלים בכדי להשתמש ב- DMA.
2. יש להשתמש בסיגנל בכדי לדעת מתי DMA מסתיים.
3. יש לעבוד עם פסיקות בכדי להשתמש ב- DMA.
4. יש להשתמש בפסיקה בכדי לדעת מתי DMA מסתיים.
5. משתמשים חייבים לאפשר DMA בכדי להשתמש בהתקנים כגון כרטיס רשת או דיסק.
6. יש יותר מתשובה אחת נכונה (אם כן, רשום איזה תשובות בנימוק).
7. אין תשובה נכונה.

שאלה 2 - זימון תהליכים בלינוקס (25 נק')

נתונה מערכת דמוית לינוקס המשתמשת באלגוריתם הזימון SCHED_OTHER הבא:

- עדיפות התהליכים נקבעת על פי השדה static_prio אשר טווח ערכיו בין 1 ל - 5 וערכו נקבע על ידי המשתמש (1 הוא העדיף ביותר ו- 5 הכי פחות עדיף).
- בכל רגע נתון מוגדר זמן ריצה מינימלי למשימות בעלות עדיפות i אשר נסמנו q_i, על פי הנוסחה:

q_i = max(target_latency / N_i , min_granularity)

כאשר target_latency ו- min_granularity קבועים המוגדרים במערכת ו- N_i הוא מספר המשימות עם עדיפות i.

- לכל משימה יש שדה vruntime המאותחל ל- 0 בעת יצירתה. בכל פסיקת שעון מתעדכן שדה באופן הבא:

current->vruntime += current->static_prio

- בכל פעם שמשימה נבחרת לרוץ נשמר ערכו של השדה vruntime כך:

current->start_vruntime = current->vruntime

- בכל פסיקת שעון נבדקים שני התנאים הבאים:
  1. קיימת משימה בעלת vruntime קטן יותר משל המשימה הנוכחית
  2. q_{current->static_prio} <= current->vruntime - current->start_vruntime

  אם שני התנאים הללו מתקיימים, מתבצעת החלפת הקשר (בעזרת הדגל need_resched).
- בהחלפת הקשר המשימה הבאה שנבחרת לרוץ היא זאת בעלת ה- vruntime המינימלי.

א. (4 נק') בהנחה שהאלוגריתם הנ"ל עושה שימוש בעץ חיפוש מאוזן הממוין לפי vruntime.

i. מהי סיבוכיות הזמן של בחירת המשימה הבאה?

ii. מהי סיבוכיות הזמן של הוספת משימה חדשה?

iii. מהי סיבוכיות הזמן של הסרת משימה?

ב. (4 נק') נניח שבמערכת יש שני תהליכים CPU-bound בעלי עדיפות זהה ושהשניים הללו הינם התהליכים היחידים במערכת. איזה אחוז מהזמן ירוץ כל אחד מהתהליכים?

ג. (4 נק') נניח שבמערכת שני תהליכים CPU-bound, אחד בעל עדיפות 1, והשני בעל עדיפות 3. נניח שהשניים הללו הינם התהליכים היחידים במערכת. איזה אחוז מהזמן ירוץ כל אחד מהתהליכים?

ד. (5 נק') באלגוריתמי הזימון SCHED_OTHER שלמדנו (בתרגול ובהרצאה), המערכת מחשבת לכל תהליך עדיפות דינמית כדי להבדיל בין תהליכים שהם IO-bound לתהליכים שהם CPU-bound. באלגוריתם הנ"ל, אין הפרדה כזו. כיצד בכל זאת מתעדפת המערכת תהליכי IO-bound כראוי?

ה. (5 נק') באיזו בעיה היינו עלולים להיתקל אם לא היה נעשה שימוש בקבוע min_granularity?

סעיף זה מתייחס למערכת כפי שנלמדה בתרגולים. בפרט, אלגוריתם זימון המשימות הוא זה שנלמד בכיתה ולא האלגוריתם שהוזכר לעיל.

ו. (3 נק') נניח כי תהליך מסויים מודע לזמן בו מתרחשת פסיקת שעון והוא מסוגל לבחור להריץ קוד כרצונו בדיוק לפני/אחרי שמתקבלת פסיקת שעון. כיצד יכול התהליך לנצל זאת כדי "לרמות" את אלגוריתם הזימון?

שאלה 3 - זיכרון וירטואלי ואחסון (24 נק')

א. (4 נק') הסבירו איך עובד מנגנון הסופר-דפים (superpages) בארכיטקטורת x86 32bit.

ב. (4 נק') מה הגודל של סופר-דף בארכיטקטורה זו? מדוע?

ג. (4 נק') הסבירו מה היתרונות של השימוש במנגנון זה.

ד. (4 נק') הסבירו את הקשיים/חסרונות של השימוש במנגנון זה.

ה. (4 נק') מנגנון ה- extent דומה במידה מסוימת למנגנון הסופר-דפים, אך מטרתו שונה. הסבירו מדוע.

ו. (4 נק') בהינתן קובץ בגודל 100MB במערכת קבצים ext4, תארו שתי צורות שונות באמצעותן ניתן למפות את הבלוקים של הקובץ בדיסק מתוך ה- inode.

שאלה 4 - סינכרוניזציה בגרעין (27 נקודות)

שאלה זו עוסקת במערכת עם מספר מעבדים, שפועלת כמו המערכת עליה למדנו בתרגולים.

הגדרות לצורך שאלה זו:

לצורך פתרון השאלה אין צורך בידע מוקדם על hardirq או softirq, כל הדרוש לפתרון השאלה מופיע בהגדרות להלן.

- בהרצאות למדתם שטיפול בפסיקה (interrupt) מתחלק ל2 חלקים top half ו bottom half. בשאלה זו top half הוא hardirq ו bottom half הוא softirq.
- קוד שמטפל ב-system call יכול להיקטע בכל שלב לצורך טיפול ב-hardirq- או ב-softirq, אלא אם טיפול כזה נחסם (למשל על-ידי spin_lock_bh או spin_lock_irq כפי שתראו בהמשך).
- קוד שמטפל ב-softirq יכול להיקטע בכל שלב לצורך טיפול ב-hardirq, אלא אם טיפול כזה נחסם (למשל על-ידי spin_lock_irq), אבל לא יכול להיקטע לצורך טיפול ב-softirq נוסף.
- קוד שמטפל ב-hardirq לא יכול להיקטע על-ידי קוד אחר.
- אם L הוא spin-lock אזי:
  - הפונקציה spin_lock(&L) נועלת את L.
  - הפונקציה spin_lock_bh(&L) חוסמת טיפול ב-softirq על המעבד הנוכחי ונועלת את L.
  - הפונקציה spin_lock_irq(&L) חוסמת טיפול ב-hardirq על המעבד הנוכחי ונועלת את L.
- אם S הוא סמפור אזי:
  - הפונקציה down_interruptible(&S) נועלת את S.
  - הפונקציה up(&S) משחררת את S.
- תזכורת - tasklet הוא סוג של softirq והוא מוגן מפני הפעלה במקביל ממעבדים שונים ולכן איננו צריך להיות re-entrant.

א. (4 נק') קוד מסויים בקרנל צריך להשתמש בשני מבני נתונים בו-זמנית. אחד מהם מוגן על-ידי spin-lock שנקרא A_lock, והשני מוגן על-ידי סמפור שנקראת B_sem. איזו מהדרכים הבאות לנעול את שני מבני הנתונים פועלת באופן תקין?

2	1

`down_interruptible(&B_sem);`
`spin_lock(&A_lock);`
 	
`/* … */`
 	
`spin_unlock(&A_lock);`
`up(&B_sem);`
	`spin_lock(&A_lock);`
	`down_interruptible(&B_sem);`
	
	`/* … */`
	
	`up(&B_sem);`
	`spin_unlock(&A_lock);`

1 / 2 / שתי הדרכים תקינות (יש להקיף בעיגול ולהסביר את תשובתכם).

בסעיפים ב-ו יש לבחור את הפתרון **היעיל ביותר** (זה שמספק ביצועים אופטימליים) מבין האפשרויות הנתונות. אין להשתמש במנגנון אחר מאלה הרשומים בכל סעיף.

ב. (4 נק') מבנה נתונים C נמצא בשימוש רק על-ידי קוד שרץ בזמן טיפול בtop half של timer interrupt. נניח ש-C מוגן על-ידי spin-lock שנקרא C_lock. באיזה מהמנגנונים הבאים כדאי להשתמש כדי להגן על הגישה ל-C? הסבירו.

1. spin_lock(&C_lock)
2. spin_lock_bh(&C_lock)
3. spin_lock_irq(&C_lock)
4. לא נדרשת נעילה כלשהי.

ג. (4 נק') מבנה נתונים D נמצא בשימוש על-ידי tasklet וגם על-ידי system call מסויים. D מוגן על-ידי spin-lock שנקרא D_lock. באיזה מהמנגנונים הבאים כדאי להשתמש בכל אחד מהמצבים כדי להגן על הגישה ל-D הסבירו?

מה-tasklet:

1. spin_lock(&D_lock)
2. spin_lock_bh(&D_lock)
3. spin_lock_irq(&D_lock)
4. לא נדרשת נעילה כלשהי.

מה-system call:

1. spin_lock(&D_lock)
2. spin_lock_bh(&D_lock)
3. spin_lock_irq(&D_lock)
4. לא נדרשת נעילה כלשהי.

ד. (5 נק') קיים בקרנל מנגנון דומה ל-spin-lock שמבדיל בין נעילה לקריאה לבין נעילה לכתיבה, בדומה ל-reader/writer lock שראינו בתרגולים. המנגנון נקרא rwlock והוא תומך בפעולות הבאות:

- read_lock(&L) נועל את L לקריאה.
- write_lock(&L) נועל את L לכתיבה.
- read_lock_bh(&L) חוסם טיפול ב-softirq על המעבד הנוכחי ונועל את L לקריאה.
- write_lock_bh(&L) חוסם טיפול ב-softirq על המעבד הנוכחי ונועל את L לכתיבה.
- read_lock_irq(&L) חוסם טיפול ב-hardirq על המעבד הנוכחי ונועל את L לקריאה.
- write_lock_irq(&L) חוסם טיפול ב-hardirq על המעבד הנוכחי ונועל את L לכתיבה.

נניח שמבנה נתונים E מוגן על-ידי rwlock כזה, שנקרא E_lock. הגישה ל-E_lock מתבצעת בשלושה הקשרים שונים:

- מתוך קוד שרץ בזמן טיפול ב-hardirq וקורא נתונים מ-E.
- מתוך tasklet שקורא נתונים מ-E.
- מתוך tasklet שמעדכן נתונים ב-E.

באיזה מהמנגנונים הבאים כדאי להשתמש בכל אחד מהמצבים כדי להגן על הגישה ל-E? הסבירו.

מהקוד שמטפל ב-hardirq:	מה-tasklet הקורא:	מה-tasklet הכותב:
1. read_lock(&E_lock) 2. read_lock_bh(&E_lock) 3. read_lock_irq(&E_lock) 4. לא נדרשת נעילה כלשהי.	1. read_lock(&E_lock) 2. read_lock_bh(&E_lock) 3. read_lock_irq(&E_lock) 4. לא נדרשת נעילה כלשהי.	1. write_lock(&E_lock) 2. write_lock_bh(&E_lock) 3. write_lock_irq(&E_lock) 4. לא נדרשת נעילה כלשהי.

ה. (5 נק') מערך F מכיל את מספר ה-interrupts מכל סוג שהתקבלו במערכת (F[i] מכיל את מספר הפעמים שהתקבל interrupt מספר i). בכל פעם שמתקבל interrupt, הקוד שמטפל בו (בtop half) מגדיל באחד את האיבר המתאים ב-F. בנוסף לזה קיים system call שמקבל כפרמטר מספר i ומחזיר לתהליך הקורא את F[i]. המערך F מוגן על-ידי rwlock שנקרא F_lock (הגישה ל-F[i] ללא נעילה מתאימה אינה אטומית). באיזה מהמנגנונים הבאים כדאי להשתמש בכל אחד מהמצבים כדי להגן על הגישה ל-F? הסבירו.

מקוד שמטפל ב-interrupt (בtop half) ומעדכן את F[i]:

1. write_lock(&F_lock)
2. write_lock_bh(&F_lock)
3. write_lock_irq(&F_lock)
4. לא נדרשת נעילה כלשהי.

מה-system call שמחזיר את F[i]:

1. read_lock(&F_lock)
2. read_lock_bh(&F_lock)
3. read_lock_irq(&F_lock)
4. לא נדרשת נעילה כלשהי.

ו. (5 נק') מכיוון שעדכון F מתבצע בתדירות גבוהה מאוד, כדי לשפר את ביצועי המערכת הוחלט שלכל מעבד יהיה מערך F משלו, שיכיל את מספר ה-interrupts מכל סוג שהתקבלו על-ידו בלבד. כעת:

- בכל פעם שמעבד מסויים מטפל ב- interrupt (בtop half), הוא מעדכן רק את האיבר המתאים במערך F שלו.
- ה-system call שתואר בסעיף הקודם עובר על המערכים של כל המעבדים במערכת, ומחזיר את הסכום של האיברים המתאימים בהם.

למרות שלכל מעבד יש מערך משלו, כל המערכים מוגנים עדיין על-ידי rwlock יחיד, F_lock. באיזה מהמנגנונים הבאים כדאי להשתמש בכל אחד מהמצבים כדי להגן על הגישה ל-F? הסבירו.

מקוד שמטפל ב-interrupt (בtop half) ומעדכן את F[i] של המעבד הנוכחי:

1. read_lock(&F_lock)
2. read_lock_bh(&F_lock)
3. read_lock_irq(&F_lock)
4. write_lock(&F_lock)
5. write_lock_bh(&F_lock)
6. write_lock_irq(&F_lock)
7. לא נדרשת נעילה כלשהי.

מה-system call שמחזיר את סכום האיברים F[i] במערכים של כל המעבדים:

1. read_lock(&F_lock)
2. read_lock_bh(&F_lock)
3. read_lock_irq(&F_lock)
4. write_lock(&F_lock)
5. write_lock_bh(&F_lock)
6. write_lock_irq(&F_lock)
7. לא נדרשת נעילה כלשהי.

<u>פתרונות:</u>

שאלה 1 - פסיקות וסיגנלים

א. 3 ו- 4, שכן פסיקה פנימית מוגדרת כפסיקה סינכרונית שהמעבד עשוי לחולל בעבור עצמו כאשר הוא מריץ תוכנה כלשהי. סימון אחת מההתשובות מזכה בחצי מהנקודות. סימון תשובה לא נכונה מוריד 2 נקודות (בכל השאלה, לא רק בסעיף זה).

ב. 2, שכן פסיקה משתמעת מוגדרת כפסיקה סינכרונית שקורת בגין פקודת מחשב שהמעבד לא יכול להשלים כתוצאה משגיאה (זמנית או קבועה). בפרט, זימון של system call או דיבוג של תהליך מתבצע באמצעות פסיקות מפורשות (explicit interrupts) -- שהתוכנה יזמה במפורש.

ג. 7 - אין תשובות נכונות. 1 לא נכון מפני שהחומרה איננה מודעת לקטעים קריטיים של תוכניות משתמש. 2 לא נכון מפני שהשיגרה המוצבעת ע"י IDT מתבצעת כולה בבת אחת (היא ה- top half; היא איננה ניתנת לחלוקה). 3 לא נכון כיוון ששיגרת פסיקה קורת מייד כשמתחוללת הפסיקה (אלא אם כן הפסיקה חסומה). 4. לא נכון מפני שפסיקות אסינכרוניות יכולות להתבצע על כל ליבה. 5 לא נכון למשל מפני שפסיקות סינכרוניות מתבצעות תמיד על הליבות שחוללו אותן.

ד. 5, שכן אי אפשר להתעלם מ- (או לחסום) <u>שלושה</u> סיגנלים: SIGKILL, SIGSTOP, ו- SIGCONT. תשובה 4 קיבלה את מחצית הנקודות. השילוב של 4+5 קיבל את מלוא הנקודות.

ה. 6, שכן כל חמשת התשובות נכונות: 1 באמצעות SIGIO ו- 2 באמצעות SIGSTOP/SIGCONT ו- 3 באמצעות SIGUSR (למשל, ראו דוגמא בהרצאה) ו- 4 מעצם זה שניתן לתפוס את הסיגנל segmentation fault ו- 5 באמצעות SIGCHLD. סימון של 3 או 4 מתוך חמשת התשובות זיכה ב 3 נקודות, סימון של שתי תשובות זיכה ב 2 נקודות. סימון של תשובה אחת בלבד זיכה בנקודה אחת בלבד.

ו. 7, אפשר לבצע DMA עם polling; לא חייבים פסיקות כלל. מי שהתעלם מאופציית ה- polling וסימן את תשובות 3, 4, 5 (כל קומבינציה שלהם) קיבל מחצית הנקודות.

שאלה 2 - זימון תהליכים

אלגוריתם הזימון שתואר בשאלה הוא למעשה אלגוריתם CFS אשר נמצא בשימוש בגרסאות רבות של לינוקס.

<u>להרחבה</u>.

הערה כללית - סטודנטים רבים התייחסו ל - q_i כ - timeslice וזה לא נכון. באלגוריתם הנ"ל אין timeslice, הערך q_i קובע את תדירות החלפות ההקשר.

א.

i. O(1) - נחזיק מצביע לבן השמאלי ביותר בעץ .מי שכתב O(log n) קיבל נקודה אחת מתוך שתיים.
ii. O(log n), כאשר n מספר המשימות במערכת
iii. כמו ii

חלק מהסטודנטים כתבו O(logNi) אבל זה לא נכון כי בעץ יושבים כל התהליכים לפי vruntime. בשאלה לא הוגדר שיש עץ נפרד לכל עדיפות i.

ב. 50% כל אחד מהתהליכים ירוץ q_i בכל פעם שיבחר.

חלק מהסטודנטים איבדו נקודות על כך שכתבו שתהיה החלפת הקשר כל פסיקת שעון או כל שתי פסיקות שעון - זה לא נכון. כמות פסיקות השעון שיהיו בין החלפות הקשר תלויה בqi - ומכיוון שהקבועים target_latency, ו min_granularity אינם ידועים הרי שאין לדעת תוך כמה פסיקות שעון תנאי 2 יתקיים ולכן אין לדעת מה בדיוק כמות פסיקות השעון קורות בין החלפות ההקשר.

ג. התהליך בעל עדיפות 1 ירוץ 75% והתהליך בעל עדיפות 3 25%. q1=q3 אבל vruntime גדל לאט יותר. עבור התהליך עם עדיפות 1, הוא ירוץ q1 בכל פעם שיבחר ואילו התהליך בעל עדיפות 3 ירוץ q_1/3 בכל פעם שיבחר. גם כאן בדומה לסעיף הקודם ירדו נקודות למי שטען שתהליך 1 ירוץ 3 פסיקות שעון לפני החלפת הקשר ואילו תהליך 2 ירוץ פסיקת שעון אחת. זה לא נכון מאותה סיבה כמו בסעיף ב. זה נכון לדוגמא עבור qi=1,2,3 אבל לא נכון עבור qi=4 שם יחס הפסיקות יהיה 6:2. וכדומה.

בנוסף היו סטודנטים שטענו שהתשובה נכונה החל מqi מסויים (לדוגמא qi>3) ואילו יש התנהגות אחרת עבור qi קטנים יותר. גם טענה זו לא נכונה. אם תריצו את האלגוריתם עבור כל qi תראו שהמערכת תתאזן על 25%/75% די מהר.

ד. אם תהליך הוא IO bound יהיה לו vruntime נמוך ולכן כאשר הוא רוצה לרוץ הוא יקבל עדיפות גבוהה. מכיוון שהוא לא רץ הרבה זמן הוא (בדרך כלל) לא יגיע לרוץ q_i ולכן לא נעצור אותו באמצע.

טעות נפוצה הייתה לטעון שתהליכים אינטראקטיביים תהיה עדיפות טובה, דבר זה אינו בשליטת המערכת. ככל הנראה הבלבול נובע מכך שתהליכים אינטראקטיביים במערכת שלמדנו בכיתה מקבלים עדיפות דינמית טובה. כאן שאלנו כיצד <u>המערכת</u> מתעדפת תהליכים אינטראטיביים ולכן תשובות מסוג זה לא התקבלו.

ה. כאשר יש הרבה תהליכים יהיו החלפות הקשר רבות וזה לא כדאי, במקרים קיצוניים כל מה שנעשה זה החלפות הקשר.

יש לשים לב שגם אם q_i הוא אפס עדיין תהליך יזכה לרוץ, פשוט לזמן קצר. בכל מקרה לא תהיה הרעבה, אלה overhead גבוה של החלפות הקשר.

ו. הוא יכול לבצע yield או לצאת להמתנה <u>קצרה</u> בדיוק לפני פסיקת שעון וכך גם לא יגמר לו ה - timeslice וגם יהיה לו בונוס גבוה. כדאי לשים לב שתהליך שביצע yield לא נבחר שוב לרוץ כאשר יש משימות נוספות בתור (כמו שראיתם בשיעורי הבית).

למעשה מה שתהליך עושה הוא לרוץ רק בין פסיקות שעון. מכיוון שהמערכת אוספת את הנתונים על הזמן שתהליך מסוים רץ רק בזמן פסיקת שעון, היא תחשוב שהתהליך לא רץ בכלל. מי שלא התייחס לעובדה זו לא קיבל ניקוד.

סטודנטים רבים כתבו שהתהליך יכול לשנות את ה - timeslice שלו או את sleep_avg לפני/אחרי פסיקת השעון. תהליך לא יכול לגשת לנתונים של הגרעין ואם היה יכול אין שום חשיבות לעובדה שהוא מודע לפסיקות השעון. הוא יכול פשוט להגדיל לעצמו את ה - timeslice כל כמה זמן וזהו. תשובות מסוג זה לא קיבלו ניקוד.

שאלה 3 - זכרון וירטואלי ואחסון

א.סופר דף רציף בזכרון הפיזי. בטבלת התרגום "העליונה" (page directory) שמים כתובת פיזית של רצף (דף) בגודל 4MB.

ב. הגודל של סופר דף בארכיטקטורה זו חייב להיות 4MB על מנת לא ליצור "חורים" בזיכרון וירטואלי. הוא צריך להיות שווה בגודלו לכמות הזיכרון שדף PT מצביע עליו.

ג. מגדיל את ה- TLB coverage.

שימו לב שתשובה שזה מקטין מספר פסיקות דף לא נכונה ולא קיבלה ניקוד.

ד .

i. ניהול של דפים בעלי גדלים שונים מורכב יותר: למשל עלינו לנהל "חורים" עם גדלים שונים ולנייד דפים קטנים בכדי לפנות מקום לגדולים.

ii. בלוקים גדולים גורמים לשיברור פנימי

iii. שיממוש בדפים בגדולים שוניפ גורם לשיברור חיצוני

ה. היתרון בשימוש של Extent הוא שאנחנו יכולים לבצע גישה רציפה לדיסק ומניעת גישה רנדומית.

ו.

i. extent בודד של 100MB.

ii. שני extents בגודל 50MB כל אחד.

שאלה 4 - סינכרוניזציה

א. התשובה הנכונה היא 2. כדי לקבל את כל הנקודות מספיק היה לכתוב שאסור לנעול סמפור (ובאופן כללי יותר --- אסור לבצע פעולה שעלולה לעשות sleep) בזמן שמחזיקים spin-lock. ההסבר המלא הוא שפעולה כזו עלולה לגרום ל-deadlock, באופן הבא: נניח שיש לנו N מעבדים וכולם מבצעים את הקטע הנתון בו-זמנית, כאשר B_sem נעול על-ידי תהליך כלשהו. אחד מהמעבדים, נקרא לו cpu0, יצליח לנעול את ה-spin-lock, והשאר ימתינו בסבלנות עד שהוא ישתחרר. כעת cpu0 מחזיק ב-A_lock ומנסה לנעול את B_sem. מכיוון ש-B_sem נעול, התהליך הנוכחי נכנס ל-sleep ונותן לתהליך אחר לרוץ על cpu0 במקומו. נניח שעכשיו התהליך האחר נכנס לאותו קטע בדיוק --- הוא ינסה לנעול את A_lock, אבל לא יצליח כי הוא נעול על-ידי התהליך שעזב את המעבד. הגענו למצב שבו כל המעבדים ממתינים לשחרור A_lock, כאשר הוא מוחזק על-ידי תהליך שמחכה לשחרור B_sem, שהוא בתורו מוחזק על-ידי תהליך אחר, שלא יוכל לחזור לרוץ אף פעם כי כל המעבדים עסוקים בניסיון לנעול את A_lock.

בסעיף זה תשובה נכונה בלי הסבר, או עם הסבר חלקי או שגוי, קיבלה ניקוד חלקי (2 מתוך 4).

תשובה לא נכונה לא קיבלה ניקוד.

בסעיפים ב-ו מפתח הניקוד היה כדלקמן:

- פתרון אופטימלי עם הסבר סביר קיבל את כל הנקודות.
- פתרון אופטימלי בלי הסבר, או עם הסבר חלקי או שגוי, קיבל ניקוד חלקי (2 מתוך 4, או 3 מתוך 5).
- פתרון תקין אבל לא אופטימלי עם הסבר סביר קיבל ניקוד חלקי (2 מתוך 4, או 3 מתוך 5).
- פתרון תקין אבל לא אופטימלי בלי הסבר, או עם הסבר חלקי או שגוי, קיבל נקודה אחת.
- פתרון לא תקין לא קיבל ניקוד.

ב. מכיוון שלפי הנתון hardirq לא יכול להיקטע על-ידי hardirq, המקרה הבעייתי היחיד שצריך להגן מפניו הוא ששני מעבדים שונים ינסו לגשת ל-C בו-זמנית. כדי למנוע מצב כזה מספיק לנעול את C_lock, ולכן הפתרון האופטימלי הוא 1.

תשובה 4 לא מונעת את המצב הזה ולכן היא לא נכונה.

תשובות 2 ו-3 נכונות אבל לא אופטימליות, ולכן קיבלו ניקוד חלקי (2 מתוך 4).

ג. כאן יש כמה מצבים בעייתיים:

1. שני מעבדים שונים מריצים את ה-system call בו-זמנית ומנסים לגשת ל-D.
2. מעבד אחד מריץ את ה-tasklet בזמן שמעבד אחר מריץ את ה-system call, ושניהם מנסים לגשת ל-D בו-זמנית.
3. מעבד מריץ את ה-system call, ניגש ל-D, ונקטע על-ידי ה-tasklet שניגש גם הוא ל-D.

כדי למנוע את שני המקרים הראשונים יש לנעול את D_lock מה-system call וגם מה-tasklet, לכן תשובה 4 בכל אחד מהם לא נכונה. כדי למנוע את המקרה השלישי יש לחסום softirq בזמן הטיפול ב-system call, ולכן יש להשתמש ב-spin_lock_bh. לכן הפתרון האופטימלי הוא:

מה-tasklet: תשובה 1.

מה-system call: תשובה 2.

תשובות 2 ו-3 עבור ה-tasklet נכונות אבל לא אופטימליות, ולכן קיבלו ניקוד חלקי (2 מתוך 4).

כל תשובה אחרת עלולה לגרום לאחד מהמקרים הבעייתיים שתוארו לעיל ולכן לא קיבלה ניקוד.

ד. כמו שלמדנו בתרגולים, במצבים שבהם יש קוראים וכותבים מותר לאפשר למספר בלתי-מוגבל של קוראים לרוץ במקביל, אבל אסור שיהיה יותר מכותב אחד, ואסור שירוצו קוראים בו-זמנית עם הכותב. לכן המקרים הבעייתיים בסעיף זה הם:

1. מעבד אחד מריץ את ה-tasklet הכותב בזמן שמעבד אחר מריץ את ה-tasklet הקורא, ושניהם ניגשים ל-E בו-זמנית.
2. מעבד אחד מריץ את ה-tasklet הכותב בזמן שמעבד אחר מטפל ב-hardirq (הקורא), ושניהם ניגשים ל-E בו-זמנית.
3. מעבד מריץ את ה-tasklet הכותב, מתחיל לכתוב ל-E, ונקטע על-ידי ה-hardirq שקורא מ-E.

כדי למנוע את שני המקרים הראשונים מספיק לנעול את E_lock בכל אחד משלושת ההקשרים. אם אחד מהם לא נועל כלל (תשובה 4), הוא עלול לרוץ בזמן שמעבד אחר מריץ את אחד הדברים האחרים ולכן לגרום למקרה (1) או (2).

כדי למנוע את המקרה השלישי ה-tasklet הכותב חייב לחסום hardirq על המעבד, לכן יש להשתמש ב-spin_lock_irq (תשובה 3).

לכן הפתרון האופטימלי הוא:

מהקוד שמטפל ב-hardirq: תשובה 1.

מה-tasklet הקורא: תשובה 1.

מה-tasklet הכותב: תשובה 3.

תשובות 2 ו-3 עבור ה-tasklet הקורא ו/או הטיפול ב-hardirq נכונות אבל לא אופטימליות, ולכן קיבלו ניקוד חלקי (3 מתוך 5).

כל קומבינציה אחרת של תשובות עלולה לגרום לאחד מהמקרים הבעייתיים שתוארו לעיל ולכן לא קיבלה ניקוד.

ה. כמו בסעיף הקודם, עלינו לדאוג שלא יהיה יותר מכותב אחד ושלא יהיו כותבים וקוראים בו-זמנית. לכן המקרים הבעייתיים הם:

1. שני מעבדים מטפלים ב-hardirq במקביל וכותבים ל-F בו-זמנית.
2. מעבד אחד מטפל ב-hardirq וכותב ל-F, בזמן שמעבד אחר מטפל ב-system call שקורא מ-F.
3. מעבד מטפל ב-system call, מתחיל לקרוא מ-F, ונקטע על-ידי hardirq שכותב ל-F.

כדי למנוע את שני המקרים הראשונים יש לנעול את F_lock בשני המקומות (לקריאה או לכתיבה בהתאם). כדי למנוע את המקרה השלישי יש לחסום hardirq בזמן הטיפול ב-system call, ולכן יש להשתמש ב-read_lock_irq. לכן הפתרון האופטימלי הוא:

מהקוד שמטפל ב-hardirq: תשובה 1.

מה-system call: תשובה 3.

תשובות 2 ו-3 עבור הטיפול ב-hardirq נכונות אבל לא אופטימליות, ולכן קיבלו ניקוד חלקי.

כל תשובה אחרת עלולה לגרום לאחד מהמקרים הבעייתיים שתוארו לעיל ולכן לא קיבלה ניקוד.

ו. סעיף זה זהה לסעיף הקודם, פרט לכך שאין לנו כאן בעייה ששני מעבדים יעדכנו את F במקביל (מקרה (1) מהסעיף הקודם). מכיוון שה-system call ניגש למערכי F של כל המעבדים במערכת, עדיין יש לנו בעיות עם שני המקרים האחרים:

1. מעבד אחד מטפל ב-hardirq וכותב למערך F הלוקאלי שלו, בזמן שמעבד אחר מטפל ב-system call שקורא מאותו מערך F.
2. מעבד מטפל ב-system call, מתחיל לקרוא מהמערך F הלוקאלי שלו, ונקטע על-ידי hardirq שכותב לאותו מערך F.

כדי למנוע את המצב הבעייתי הראשון יש לנעול בשני המקומות, ולפחות באחד מהם הנעילה צריכה להיות לכתיבה. מכיוון שטיפול ב-interrupts מתבצע בתדירות גבוהה הרבה יותר מטיפול ב-system calls, עדיף לנעול לכתיבה ב-system call ולנעול לקריאה בטיפול ב-interrupt, כך שמעבדים שונים יוכלו לטפל ב-interrupts במקביל. כדי למנוע את המצב הבעייתי השני יש לחסום hardirq בזמן הטיפול ב-system call, ולכן יש להשתמש ב-write_lock_irq. לכן הפתרון האופטימלי הוא:

מהקוד שמטפל ב-hardirq: תשובה 1.

מה-system call: תשובה 6.

תשובות 2-6 עבור הטיפול ב-hardirq נכונות אבל לא אופטימליות, ולכן קיבלו ניקוד חלקי. כמו-כן תשובה 3 עבור ה-system call יחד עם אחת מהתשובות 4-6 עבור הטיפול ב-hardirq (כך שה-system call נועל לקריאה והקוד שמטפל ב-hardirq נועל לכתיבה) נכונה אבל לא אופטימלית, ולכן גם תשובה כזו קיבלה ניקוד חלקי.

כל קומבינציה אחרת עלולה לגרום לאחד מהמצבים הבעייתיים שתוארו לעיל ולכן לא קיבלה ניקוד.