אוניברסיטת בן-גוריון בנגב, המחלקה למדעי המחשב

מועד א' במערכות הפעלה (202-1-3031) (8 ביולי 2012)

מרצים: דני הנדלר ורועי זיוון; מתרגלים: אלון גרובשטיין, איליה מירסקי, אמיר מנצ'ל ודולב פומרנץ

משך המבחן: שלוש שעות
חומר עזר: אסור
פתרי את כל השאלות: סה"כ 100 נקודות.

1. מערכת הקבצים (25 נק')

א. (15 נק') שלומציונה משתמשת במערכת קבצים יוניקס מבוססת i-nodes אשר בה מנגנון ה-buffer cache מנוטרל. הניחו כי במערכת זו גודלו של כל בלוק הוא 1KB. שלומציונה כותבת תוכנית המגרילה בכל פעם באופן רנדומלי ויוניפורמי מיקום (file position, offset) בקובץ נתון וקוראת 100 בתים של אינפורמציה ממיקום זה בקובץ.

<u>הניחו כי לכל קובץ ממנו קוראת התוכנית של שלומציונה מוקצה מקום רציף בדיסק.</u>

a. **(8 נק')** לאחר הרצת התכנית מספר רב של פעמים על קבצים שונים שלומציונה מבחינה בתופעה הבאה: משך הזמן הנדרש להרצת התכנית שונה עבור קבצים בגדלים שונים. עבור קבצים קטנים (לכל היותר 100KB) משך ההרצה קצר בצורה משמעותית מקבצים גדולים (יותר מ- 1GB). כיצד ניתן להסביר תופעה זו בהסתמך על מאפייני המערכת? הסבירו בקצרה אך במדויק.

תשובה: קל לראות כי עבור הקבצים הקטנים יותר נזדקק ברוב המקרים לשלוש גישות (ארבע במקרה והמיקום דורש הבאת שני בלוקים): inode,single,data. מנגד, עבור קבצים גדולים כפי שהוגדרו בשאלה, ברוב המקרים נאלץ לגשת לדיסק לפחות חמש פעמים (לשם המחשה, בקובץ בגודל 1GB ורוחב כתובת של 32 ביטים, נאלץ להעזר ברמת triple ב-75% מהדגימות). מכיוון שהגישה לדיסק אורכת זמן רב יחסית ההבדל בין קבצים גדולים וקטנים ניכר.

b. **(7 נק')** זרובבל טוען כי אם שלומציונה תריץ את התוכנית לעיל במערכת קבצים FAT ההבדל בזמן ההרצה בין קבצים גדולים לקטנים יקטן משמעותית וכי במערכת קבצים NTFS כלל לא יהיה הבדל. האם זרובבל צודק? הסבירו בקצרה אך במדויק.

תשובה: זרובבל צודק!

במקרה של מערכת הקבצים FAT: תחילה יש לאתר את הבלוק הרצוי ולאחריו להביאו (לכל היותר שתי גישות דיסק במידה ונידרש להביא שני בלוקים). חשוב לציין שאמנם החיפוש במערכת קבצים מבוססת FAT דורש מעבר על רשימה מקושרת אך רשימה זו נמצאת כולה בזכרון (ה-FAT נטענת לזכרון) ולכן משך החיפוש זניח ביחס לזמן הנדרש בגישה לדיסק ועל כן כמעט ולא ניתן יהיה להבחין בהבדל.

במקרה של מערכת הקבצים NTFS: מכיוון שמהנתון ברור כי יש run יחיד לכל רשומת MFT נידרש לפעולת דיסק יחידה (שתי גישות דיסק במידה ונידרש להביא שני בלוקים) בשני המקרים. שווה לציין כי למרות שבכיתה דיברנו על כך שניתן לשמר מידע של קבצים קטנים בתוך הרשומה עצמה, הקבצים ה"קטנים" כפי שהוגדרו בשאלה אינם קטנים דיים.

ב. (10 נק')

a. **(5 נק')** במערכת הקבצים NTFS ישנו וקטור של ביטים (bitmap) הממפה את הבלוקים הפנויים בדיסק. אם מבנה נתונים זה נמחק ואין עותק נוסף שלו, האם יש דרך לשחזר אותו (ואם כן, כיצד)? ענו בקצרה אך במדוייק.
תשובה: כן, ניתן לשחזר וקטור זה ע"י סריקה של מערכת הקבצים. מאחר וקיימת ברשומות קובץ ה MFT האינפורמציה אילו בלוקים תופס כל קובץ, ניתן למצוא אילו בלוקים בדיסק אינם בשימוש.

b. **(5 נק')** מאיזה מבנה נתונים במערכת הקבצים FAT ניתן להסיק אילו בלוקים פנויים בדיסק? אם מבנה נתונים זה נמחק ואין עותק נוסף שלו, האם יש דרך לשחזר את מספרי הבלוקים הפנויים בדיסק? ענו בקצרה אך במדוייק.
תשובה: ניתן להסיק זאת מטבלת ה- FAT. אם טבלה זו נמחקת ולא נשמר עותק נוסף שלה אין דרך לשחזר את האינפורמציה השמורה בה ובפרט אין דרך לשחזר אילו בלוקים בדיסק פנויים.

2. **ניהול זיכרון (30 נק')**

שימו לב: בשאלה זאת סעיפים א' ו-ב' בלבד מתייחסים לעבודה 3 ולמערכת ההפעלה xv6.

א. (5 נ') . ראינו כי מערכת ההפעלה xv6 תומכת בריבוי תהליכים (multiprogramming), ומכאן שבזמן נתון יכולים לרוץ מספר תהליכים שונים. בנוסף, אנו יודעים כי לכל תהליך טבלת דפים יחודית משלו (2-level page table). כאשר המעבד מריץ את פקודות התהליך, הוא עושה שימוש בחומרה על מנת לתרגם כתובות וירטואליות לפיזיות, וזאת בעזרת טבלת הדפים.

כיצד מודיעה מערכת ההפעלה למעבד מהי טבלת הדפים הנוכחית איתה עליו לעבוד? תארו בקצרה אך באופן מדוייק וברור.

תשובה: מערכת ההפעלה מעדכנת את האוגר CR3 ומכניסה לתוכו את כתובת טבלת הדפים של התהליך הנוכחי. פעולה זאת מתבצעת בכל החלפה של תהליך.

ב. (5 נ') . כאשר המעבד מריץ את פקודות התהליך ומתבצע שימוש בחומרה על מנת לתרגם כתובות וירטואליות לפיזיות, עלול להיווצר מצב בו מתבצעת גישה לכתובת לא חוקית בזיכרון הוירטואלי. במקרה כזה, המעבד מעביר שליטה למערכת ההפעלה xv6 על מנת שתטפל במצב. כיצד מודיע המעבד למערכת ההפעלה מהי הכתובת הוירטואלית שהגישה אליה אינה חוקית? תארו בקצרה אך באופן מדוייק וברור.

תשובה: המעבד שומר באוגר CR2 את הכתובת הוירטואלית אשר בגישתה נוצרה הבעיה.

ג. (5 נ') . מהי תופעת ה-thrashing? כיצד מתמודדים מנגנוני ניהול הזיכרון של מערכות הפעלה עם תופעה זו? הסבירו בקצרה אך במדויק.
תשובה: Thrashing היא תופעה בה ביצועי מערכת ההפעלה - ובפרט ניצולת המעבדים - יורדים כתוצאה מקצב גבוה של page faults הנובע מכך שלחלק מן התהליכים לא מוקצה מספיק זכרון פיסי. מערכת ההפעלה מתמודדת עם התופעה ע"י נסיון להגדיל את כמות הזכרון המוקצת לתהליכים הסובלים מן הבעיה ואם הדבר אינו עוזר, ע"י suspension של תהליכים לדיסק.

ד. (15 נ') . נתונה מערכת הפעלה העובדת עם paging ואלגוריתם החלפת דפים LRU, אשר מקבלת את סדרת הגישות (ה- reference string) הבאה: 6, 3, 1, 7, 1, 9, 6, 3, 5, 1, 3, 8, 7, 1

סדר בקשת הדפים הוא משמאל לימין.

i. **(7 נ')** חשבו את ה-distance string המתאים ל-reference string שלעיל ולאלגוריתם LRU.

Ref. string	**6**	**3**	**1**	**7**	**1**	**9**	**6**	**3**	**5**	**1**	**3**	**8**	**7**	**1**
	6	3	1	7	1	9	6	3	5	1	3	8	7	1
		6	3	1	7	1	9	6	3	5	1	3	8	7
			6	3	3	7	1	9	6	3	5	1	3	8
				6	6	3	7	1	9	6	6	5	1	3
						6	3	7	1	9	9	6	5	5
									7	7	7	9	6	6
												7	9	9

Distance	∞	∞	∞	∞	2	∞	5	5	∞	5	3	∞	7	4

ii. **(8 נ')** נסמן ב-F את מספר דפי הזכרון הפיסיים (page frames) במערכת. נסמן ב- P(F) את מספר ה-page faults לו תגרום סדרת הגישות שלעיל באלגוריתם LRU. חשבו את הערך של F שיביא למינימום את הביטוי הבא: F+P(F). צריך להיות ברור מתשובתכם כיצד הגעתם לערך זה.

תשובה: במקרה כזה הערך האופטימלי הינו 5.

R	1	2	3	4	**5**	6	7
P(R)	14	13	12	11	**8**	8	7
P(R) + R	15	15	15	15	**13**	14	14

3. סינכרוניזציה – סמפורים ובעיית המניעה ההדדית (30 נק')

שני הסעיפים הראשונים מתייחסים לבעייה הבאה.
במערכת ישנם שלושה חוטים – A, B ו-C. יש לכתוב אלגוריתם המממש קטע קריטי בעל התכונות הבאות:

- אם A נמצא בקטע הקריטי, אזי הוא נמצא בו לבדו.
- חוטים B ו-C יכולים להימצא בקטע הקריטי בו זמנית. בפרט, אם B (C) נמצא בקטע הקריטי אזי C (B) יכול להיכנס לקטע הקריטי גם כן, אפילו אם A ממתין.
- על האלגוריתם לקיים חופש מקיפאון (deadlock freedom).

א. **(10 נק')** ממשו את הפרוצדורות A-enter, A-exit, B-enter, B-exit, C-enter ו- C-exit המממשות את קטעי הכניסה והיציאה של שלושת ההליכים.

בסעיף זה מותר להשתמש במימוש אך ורק במשתנים הבאים:
סמפורים (סמפורים מונים או בינאריים – כרצונכם) m1 ו-m2 המאותחלים שניהם לערך 1 ומשתנה c התומך בקריאות וכתיבות בלבד ומאותחל לערך אפס. אין להשתמש ב-busy waiting. כלומר, תהליכים יכולים להמתין רק ע"י שינה על סמפור.

A-enter
m1.down( )

A-exit
m1.up( )

B/C-enter
m2.down( )
c++
if c=1
  m1.down()
m2.up()

B/C-exit
m2.down( )
c--
if c=0
  m1.up()
m2.up()

ב. **(10 נק')** גם בסעיף זה עליכם לממש את הפרוצדורות A-enter, A-exit, B-enter, B-exit, C-enter ו- C-exit. בסעיף זה מותר להשתמש אך ורק במשתנים הבאים:
משתנים m1 ו-m2 התומכים בפעולות קריאה, כתיבה ו- test-and-set בלבד ומאותחלים לאפס
ומשתנה c התומך בקריאות וכתיבות בלבד ומאותחל לערך אפס גם הוא. תהליכים הנאלצים להמתין יעשו זאת באמצעות busy-waiting (כלומר, פקודת await).
להזכירכם, אם משתנה v תומך בפעולת test-and-set אזי פעולת v.test-and-set( ) מתבצעת באופן אטומי, כותבת ערך 1 למשתנה ומחזירה את ערכו הקודם.

A-enter
await(m1.test-and-set() = 0)

A-exit
m1=0

B/C-enter
await(m2.test-and-set() = 0)
c++
if c=1
  await(m1.test-and-set() = 0)
m2=0

B/C-exit
await(m2.test-and-set() = 0)
c--
if c=0
  m1=0
m2=0

ג. (10 נק')

במערכת ישנם שני תהליכים, P1, P2 העשויים לבצע מדי פעם את הקוד להלן.
עליכם לכתוב מימוש לפרוצדורות Before_A( ) ו- After_B( ) המבטיח את התכונות הבאות:

P1	P2
Before_A( ) Print("A")	Print("B") After_B( )

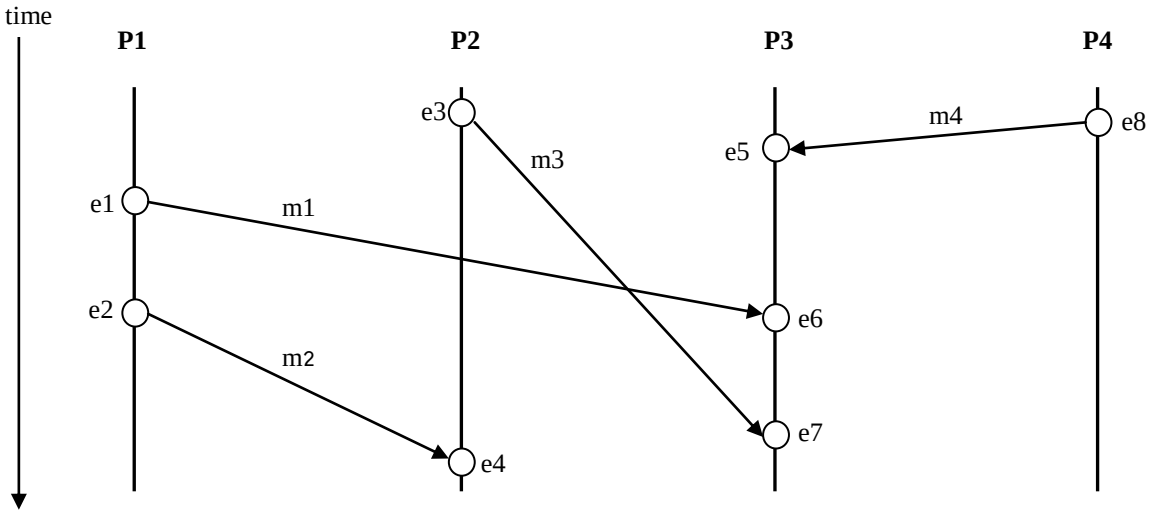
1. לעולם אין ארבע הדפסות רצופות של "A". במילים אחרות, בין כל ארבע הדפסות של "A" יש לפחות הדפסה אחת של "B".
2. אם תהליך P1 רץ לבדו (כלומר, P2 אינו מבצע את הקוד שלו באותו זמן) וההדפסה הבאה שלו אינה הרביעית ברציפות, אזי הוא יכול לבצעה.
3. תהליך P2 יכול תמיד לבצע עוד ועוד הדפסות.
4. לעולם לא קורה מצב בו P1 ו-P2 ממתינים זה לזה (deadlock).

המימוש שלכם יכול להשתמש אך ורק בסמפורים (מונים או בינאריים – כרצונכם) ובמשתנים התומכים בקריאות וכתיבות בלבד. אין להשתמש ב-busy waiting. כלומר, תהליכים יכולים להמתין רק ע"י שינה על סמפור.

סעיף זה שגוי ואין פתרון הממלא את כל התנאים הנדרשים בו. הבעייה היא זו:
נניח כי P1 עומד לבצע את ההדפסה השלישית שלו ונעצר ממש לפניה (כלומר,מייד לאחר ביצוע Before_A). כעת, P2 מבצע את After_B. P2 חייב לאפשר לP1 שלוש הדפסות נוספות שכן עד כמה שP2 יכול לדעת יתכן שP1 כבר ביצע את ההדפסה השלישית. לפיכך P1 יוכל לבצע ארבע הדפסות ברציפות.

עם הסטודנטים הסליחה. מלוא 10 הנקודות ניתנו בסעיף זה לכל הסטודנטים.

4. סינכרוניזציה מבוזרת ותיאום שעונים **(15 נק')**
נתונה הדיאגרמה הבאה של צירי הזמן של 4 מעבדים (Processors) $P_1...P_4$ ומשלוח ההודעות ביניהם.
נשלחות ארבע הודעות הגורמות לשמונה מאורעות (events) אצל המעבדים. את התשובות לשאלות שלהלן יש לכתוב במחברת ולא על גבי הדיאגרמה.



א. (5 נק') עבור כל אחד משמונה המאורעות (events), כתבו את חותמת הזמן של Lamport (Lamport timestamp).

e1: 1.1
e2: 2.1
e3: 1.2
e4: 3.2
e5: 2.3
e6: 3.3
e7: 4.3
e8: 1.4

ב. (5 נק') עבור כל אחד משמונה המאורעות, כתבו את חותמת הזמן הווקטורית (Vector timestamp).

e1: (1,0,0,0)
e2: (2,0,0,0)
e3: (0,1,0,0)
e4: (2,2,0,0)
e5: (0,0,1,1)
e6: (1,0,2,1)
e7: (1,1,3,1)
e8: (0,0,0,1)

ד. (5 נק') האם קיים מקרה של causality violation בתסריט המתואר בדיאגרמה לעיל? אם כן, בין אילו הודעות? נמקו תשובתכם בקצרה אך במדויק.

לא. אין שתי הודעות אשר נשלחות בזמנים בעלי סדר קוזלי מסוים ומגיעות בסדר זמנים קוזלי הפוך.

בהצלחה!