אוניברסיטת בן-גוריון בנגב, המחלקה למדעי המחשב

מועד א' במערכות הפעלה (21 ביולי 2009)

מרצים: דני הנדלר ואמנון מייזלס; מתרגלים: דניאל גורדון, אלון גרובשטיין, תומר הבר ועדי סוויסה

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

1. ניהול זיכרון (25 נקודות)

נתונה תוכנית המשתמשת ב- n+1 דפים, דף אחד לקוד ו-n דפי נתונים. התוכנית רצה במערכת זיכרון וירטואלי שעובדת ב- Demand Paging. התוכנית עוברת על דפי הנתונים אחד אחרי השני n פעמים על פי הסדר.
כלומר: D1,D2,..,Dn D1,D2,..,Dn D1,D2,.., Dn.
מחשב מריץ את התוכנית הנ"ל בשני תהליכים שונים בו-זמנית, כאשר <u>דף הקוד משותף בין שני התהליכים ואילו n דפי הנתונים נפרדים</u>. ה- time sharing בין התהליכים הוא כזה שכאשר תהליך מסיים לעבור על דף נתונים מסוים ולפני שהוא עובר לדף הנתונים הבא ה- scheduler מקצה זמן לתהליך השני, ששוב רץ עד שהוא מסיים לעבור על דף נתונים מסוים, ושוב לפני שהוא עובר לדף הנתונים הבא ה-scheduler חוזר להקצות זמן לתהליך הראשון, וחוזר חלילה. בזיכרון הפיסי של המחשב נמצאים x Frames (לאחר שהוקצו ה-Frames הדרושים למערכת ההפעלה). לתהליך 1 הוקצה $\frac{x}{4}$ מה- frames (הניחו כי x מתחלק ב-4 ללא שארית), לתהליך 2 הוקצה $\frac{3x}{4}$ מה- frames. בזיכרון של תהליך 2 יושב דף הקוד המשותף.

א. חשבו את מספר ה-Page faults עד לגמר ריצת שני התהליכים, כאשר שיטת החלפת הדפים היא LRU (עם demand paging), החלפת הדפים היא <u>לוקאלית</u> לכל תהליך ו – $x = 2(n+1)$. הסבירו חישוביכם. (**10 נק'**)

ב. חשבו את מספר ה-Page faults עד לגמר ריצת 2 התהליכים, כאשר שיטת החלפת הדפים היא LRU (עם demand paging) , החלפת הדפים היא <u>גלובאלית</u> לכל התהליכים ו – $x = 2(n+1)$. הסבירו חישוביכם (**10 נק'**).

הסעיף הבא איננו קשור לשני הסעיפים הקודמים.

ג. הסבירו את ההבדלים בין TLB (translation lookaside buffer) המותאם למערכת multi-programming לבין TLB אשר יש לאתחלו (לבצע flush) כל אימת שהמערכת מבצעת process context switch.(**5 נק'**)

<u>פתרון שאלה 1:</u>

הגישה לנתונים נעשית באופן הבא: $D_{1,1}, D_{2,1}, D_{1,2}, D_{2,2}, D_{1,3}, D_{2,3}, ..., D_{1,n}, D_{2,n}$ (n פעמים), כאשר $D_{i,j}$ הינו גישה של תהליך i לדף j. נשים לב כי הגישה לדף הקוד מתבצעת כל הזמן (כל פעם שצריך לקרוא שורת קוד של התוכנית).

א. (התקבלו מספר תשובות)

לפי נתוני השאלה – בשיטת החלפת דפים לוקאלית, לתהליך 1 מוקצים $\frac{x}{4}$ מה- frames, שהם $\frac{n+1}{2}$ frames;

ולתהליך 2 מוקצים $\frac{3x}{4}$ מה- frames, שהם $\frac{3n+1}{2}+1$ frames.

תהליך 1 מתחיל לרוץ ומבצע page-fault על מנת להביא את דף הקוד המשותף לזיכרון. לפי נתוני השאלה, דף זה יוקצה בזיכרון של תהליך 2, ולכן ישארו לתהליך 2 $\frac{3n+1}{2}$ frames למידע.

כל אחד מהתהליכים ניגש לכל אחד מ- n דפי הנתונים. כיוון שדפים אלה אינם משותפים אזי בגישה הראשונה של תהליך לדף נתונים יתרחש page-fault.

- תהליך 2 יכול לאחסן $\frac{3n+1}{2} > n$ דפי נתונים, ולכן רק בגישות הראשונות לדפים אלו יתרחשו page-faults
  $\Leftarrow$ סה"כ n page-faults עבור תהליך 2.
- תהליך 1 יכול לאחסן עד $\frac{n+1}{2} < n$ דפי נתונים, ובגלל ששיטת החלפת הדפים היא LRU, אז כשינסה לגשת לדף $\frac{n+1}{2}+1$ יתרחש page-fault והדף הראשון של התהליך יוחלף. כאשר ינסה לגשת לדף $\frac{n+1}{2}+2$ יתרחש page-fault והדף שני של התהליך יוחלף, וכך הלאה. כיוון שניגשים n פעמים ל- n דפי נתונים, וכל פעם יש page-fault $\Leftarrow$ סה"כ n^2 page-faults עבור תהליך 1.

סה"כ page-faults: n^2+n+1.

ב. (התקבלו מספר תשובות)

בשיטת החלפת דפים גלובאלית אין משמעות להקצאה של מספר דפים לתהליך, כיוון שהאלגוריתם מחליף את הדף הכי ישן שאחד התהליכים רצה בו.

נשים לב שגודל הזיכרון מאפשר לכך שיהיו $2(n+1)=2n+2$ דפים בזכרון הפיסי.

יש צורך בדף אחד עבור הקוד, ולכן נשארו $2n+1$ frames עבור דפי הנתונים. כל תהליך ניגש ל- n דפים שונים, ולכן סה"כ נצטרך עוד $2n$ frames עבור דפי הנתונים. לכן רק בגישה הראשונה של כל תהליך לדף המידע יתרחש page-fault $\Leftarrow$ סה"כ n page-faults עבור כל אחד מהתהליכים בגישה לדפי הנתונים.

סה"כ page-faults: $2n+1$ (1 עבור גישה לדף הקוד).

ג.

TLB המותאם למערכת multi-programming נכיל בכל שורת המרה גם את מספר התהליך (בנוסף לכתובת הוירטואלית), ולכן בעת ביצוע process context-switch אין צורך לבצע flush (invalidation) של כל הטבלה. לעומת זאת ב- TLB **<u>שאיננו</u>** תומך ב- multi-programming יש צורך לבצע flush לטבלה על מנת שהתהליך החדש לא ינסה לגשת לדף פיסי של תהליך אחר.

2. קבצים – NFS (25 נקודות)

נתונה מערכת קבצים של LINUX אשר תומכת בשיתוף קבצים (NFS).

א. במערכת קיימים שלושה סוגי מבני נתונים: r-nodes, i-nodes ו-v-nodes. הסבירו את תפקידו של כל אחד ממבני הנתונים הללו ואת הקשרים ביניהם (**5 נק'**).

ב. נתון קטע הקוד הבא.

```
int fd=open("data.txt",O_RDONLY,0666);
lseek(fd,50,SEEK_SET); // Sets the offset to 50
write(fd,buf,150); // Write 150 bytes from buf
lseek(fd,50,SEEK_SET); // Sets the offset to 50
read(fd,buf,150); // Read 150 bytes into buf
```

ידוע שהקובץ data.txt יושב על גבי שרת חיצוני (ולא במערכת הקבצים המקומית).

הסבירו אילו מן הפקודות שבקוד נשלחות לשרת ואילו לא. נמקו תשובתכם בקצרה (**6 נק'**).

ג. נתון קטע הקוד הבא.

```
int fd=open("data.txt",O_RDONLY,0666);
lseek(fd,0,SEEK_SET); // Sets the offset to 0
read(fd,buf,500);
read(fd,buf,500);
read(fd,buf,1000);
```

נתון גם כי לקליינט יש שכבת NFS caching והשרת שולח לקליינט בלוקים בגודל קבוע של 1KB.

הסבירו אילו מן הפקודות בקוד שלמעלה נשלחות לשרת ואילו לא. נמקו תשובתכם בקצרה (**6 נק'**).

ד. נתונים שני תהליכים. התהליך הראשון פותח את הקובץ data.txt לכתיבה ולקריאה. התהליך השני מנסה למחוק את הקובץ בעודו פתוח ע"י התהליך הראשון.

i. בהנחה ששני התהליכים רצים על **מחשבים שונים** (שני קליינטים שונים של השרת החיצוני). האם המחיקה תצליח? נמקו בקצרה (**4 נק'**).

ii. בהנחה ששני התהליכים רצים **על אותו מחשב** (קליינט של השרת החיצוני). האם המחיקה תצליח? נמקו בקצרה (**4 נק'**).

<u>פתרון שאלה 2:</u>

<u>הערה</u>: שרת NFS הוא STATELESS (על פי מה שנלמד בכיתה).

א. INODE: מבנה נתונים המתאר קובץ במערכת הקבצים המקומית.
RNODE: מבנה נתונים **אשר נמצא אצל הקליינט** ומתאר קובץ במערכת קבצים מרוחקת.
VNODE: נמצא בשכבת ה-VFS ומצביע או ל-INODE או ל-RNODE (משמש לאבסטרקציה).

ה-SYSTEM CALL LAYER "עובדת" רק עם VNODES. כלומר ה"דיבור" עם קבצים ב- SYSTEM CALL LAYER יתבצע באותה "שפה" בלי קשר אם הקובץ מקומי או מרוחק. כאשר RNODE (המוצבע ע"י VNODE) יתרגם את הפקודות להודעות לאיזשהו שרת מרוחק שעליו יושב הקובץ. ואילו INODE (המוצבע ע"י VNODE) יתרגם את הפקודות להודעות למערכת הקבצים המקומית של מערכת ההפעלה.

ב. שורה 1: לא תשלח פקודת OPEN לשרת, אלה פקודות LOOKUP אשר תחזיר לקליינט FILEHANDLER. לאחר קבלת ה-FILEHANDLER, הקליינט ייצר VNODE ו-RNODE כאשר ה- VNODE יצביע ל-RNODE, ואילו ה-RNODE יחזיק את ה-FILEHANDLER.
לשורה 1 התקבלו גם התשובות שבהן הנבחן רשם שה-OPEN נשלח והשרת מחזיר FD, וגם התשובות שבהן הנבחן רשם שה-OPEN לא נשלח בגלל שהשרת הוא STATELESS.
שורה 2: השרת הוא STATELESS ולכן מידע כגון ה-OFFSET של הקובץ נמצא אצל הקליינט. כלומר פקודת LSEEK תמיד תתבצע בצד של הקליינט בלבד.
שורה 3: השרת הוא STATELESS ולכן מידע כגון הרשאות כתיבה/קריאה של הקובץ נמצאות אצל הקליינט. מכיוון שהקובץ נפתח כ-RD_ONLY, הקליינט יזהה שאסור לבצע פעולת כתיבה לקובץ. ולכן פקודת ה-WRITE לא תשלח לשרת.
שורה 4: (ראה הסבר לשורה 2).
שורה 5: מכיוון שהתוכן של הקובץ יושב אצל השרת (ולא אצל הקליינט) פקודת ה- READ תשלח לשרת ותבקש את 150 הביטים מ-OFFSET 100.

ג. **הערה:** בשאלה זו פקודת ה-READ יכולה להחזיר יותר מידע ממה שהקליינט ביקש. כאשר המוטבציה מאוחרי קבלת מידע "נוסף", היא שסביר להניח שבאיזשהו שלב הקליינט ירצה לקרוא את המידע הנוסף הזה. שכבת ה-CACHE אשר בה יאוחסן המידע יכולה לחסוך גישות לשרת.
שורה 1: (ראה הסבר לשורה 1 בסעיף ב')
שורה 2: (ראה הסבר לשורה 2 בסעיף ב')
שורה 3: הקליינט יראה שה-CACHE ריק ולכן עליו לקבל את תוכן הקובץ מהשרת. פקודת ה- READ תשלח לשרת ותבקש את 500 הביטים מ-OFFSET 0. מכיוון שנתון שהשרת מחזיר תשובות בבלוקים של 1kb. הקליינט יקבל חזרה את תוכן הקובץ מ-0 עד 1024 בייט (או 1000 בייט) ויאכסן את המידע ב-CACHE.
שורה 4: כעת הקליינט רוצה את תוכן הקובץ מ-500 עד 1000 בייט. הפעם הקליינט יגלה שהמידע שהוא צריך יושב ב-CACHE ולכן פקודת ה-READ **לא** תישלח לשרת, אלה תיקח המידע ישירות מה-CACHE.
שורה 5: כעת הקליינט רוצה את תוכן הקובץ מ-1000 עד 2000 בייט. אולם ל-CACHE יש רק את המידע עד הבייט ה-1024 (או 1000), ולכן הפעם פקודת ה-READ תישלח לשרת.

ד.
i. השרת הוא STATELESS ולכן אין לו שום מידע לגבי איזה תהליכים פתחו את הקובץ data.txt. בהנחה שיש לקליינט הרשאות מתאימות, המחיקה תצליח.
ii. מכיוון ששני התהליכים רצים על אותה מערכת הפעלה באותו מחשב, מערכת ההפעלה של הקליינט יכולה לזהות שהקובץ פתוח (הקליינט הוא STATEFUL). האם המחיקה תצליח? תלוי במערכת ההפעלה.
נבחן שהבין שהפעם המחיקה תלויה בקליינט (ולא בשרת) ונתן הסבר מספק, קיבל את מלוא הנקודות.

3. סינכרוניזציה - סמפורים (25 נקודות)

נתונה בעייה של יצרן אחד ושלושה צרכנים שבה ישנם שלושה מוצרים שונים המסופקים על ידי היצרן וכל אחד מן הצרכנים נזקק לשניים מהם. היצרן מקבל כל כמה זמן אספקה של 2 מהמוצרים (ראנדומאלית) ומציע אותם לצרכנים. הצרכן, שאלו שני המוצרים הדרושים לו, לוקח את שני המוצרים. רק לאחר שהמוצרים נלקחו, חוזר היצרן על התהליך- ומציע שוב שני מוצרים. כך חוזר התהליך על עצמו שוב ושוב. נדרש פתרון לבעיה המשתמש אך ורק בסמפורים כאמצעי סנכרון.
לבעייה הוצע הפיתרון הבא, המשתמש במערך S[0..2] של סמפורים ובסמפור נוסף בשם producer (אין זה משנה אם הסמפורים בינאריים או כלליים) המאותחלים לערך 0. בפתרון זה, כל אחד מן הסמפורים S[i] מיצג אחד מן המוצרים.

Code for Consumer i∈{0,1,2}

```
Down(S[i])
Down(S[(i+1) mod 3)
Up(producer)
```

Producer code

```
Let i, j be two random products from {0,1,2}
Up(S[i])
Up(S[j])
Down(producer)
```

א. תנו תסריט המוכיח כי בפתרון זה יתכן deadlock. (**10 נק'**)

ב. הציעו קוד חלופי הפותר בעייה זו ומשתמש בדיוק באותם משתנים (אך משנה את משמעותם). ניתן להניח כי היצרן יודע מי הם הצרכנים ואילו מוצרים כל אחד מהם צריך (**15 נק'**).

פתרון שאלה 3

א. התרחיש הבא יוביל ל- deadlock:

Producer:	*i=0, j=1*	
	Up(S[0])	
	Up(S[1])	
	Down(producer)	*==> blocked*
C_1*:*	*Down(S[1])*	
	Down(S[2])	*==> blocked*
C_0*:*	*Down(S[0])*	
	Down(S[1])	*==> blocked*
C_2*:*	*Down(S[2])*	*==> blocked*

⇨ **Deadlock!**

ב. נחליף את תפקיד הסמפורים שבמערך. כעת, במקום לציין מוצרים ייצג סמפור תנאי עבור צרכן מסוים. לשם כך נגדיר (אך לא נממש) פונקציה consumer(int i,int j) אשר בהינתן זוג מספרים המייצגים מוצרים תחזיר את מספר הצרכן שיוכל להשתמש במוצרים אלו.

Code for Consumer i∈{0,1,2}
Down(S[i])
Up(producer)

Producer code
Let i, j be two random products from {0,1,2}
Up(S[consumer(i,j)])
Down(producer)

4. סינכרוניזציה – בעיית המניעה ההדדית (mutual exclusion) (25 נקודות)

להלן אלגוריתם המאֶפיה (Bakery) של Lamport, כפי שנלמד בכיתה.

int number[1..N] initially 0
boolean choosing[1..N] initially 0

Code forProcess i

1: choosing[i] := true
2: number[i] := 1+ max {number[j] | (1 ≤ j ≤ n)}
3: choosing[i] := false
4: for j := 1 to n do
5: await choosing[j] = false
6: await (number[j] = 0) ∨ (number[j],j) ≥ (number[i],i)
7: od
8: critical section
9: number[i] := 0

כפי שהסברנו בכיתה, אלגוריתם המאפיה מקיים את תכונות המניעה ההדדית (mutual exclusion), חופש מהרעבה (starvation freedom) והוגנות (first-in first-out).

א. נניח כי מחליפים את שורה מספר 6 של האלגוריתם בשורה הבאה.

await (number[j] = 0) ∨ (number[j] ≥ number[i])

ציינו איזו מתכונות האלגוריתם מפסיקה להתקיים. תארו תסריט מדוייק המוכיח זאת. (**7 נק'**)

ב. נניח כי מסירים את שורה מספר 5 מן האלגוריתם (שורה 6 המקורית נותרת). ציינו איזו מתכונות האלגוריתם מפסיקה להתקיים. תארו תסריט מדוייק המוכיח זאת. (**8 נק'**)

ג. ד"ר אינטלקטואלק מציע את הוריאציה הבאה של אלגוריתם המאפיה (הדגשנו בקו תחתון את קטע הקוד שהשתנה).

Int number[1..N] initially 0
Boolean choosing[1..N] initially 0

Code forProcess i

1: choosing[i] := true
2: if i is even
3: number[i] := 1 + max {number[j] | (1 ≤ j ≤ n) for all even j}
4: else
5: number[i] := 1 + max {number[j] | (1 ≤ j ≤ n) for all odd j}
3: choosing[i] := false
4: for j := 1 to n do
5: await choosing[j] = false
6: await (number[j] = 0) ∨ (number[j],j) ≥ (number[i],i)
7: od
8: critical section
9: number[i] := 0

עבור כל תכונה מתכונות האלגוריתם המקורי (מניעה הדדית, חופש מהרעבה והוגנות), ציינו האם היא נשמרת או שהיא מופרת על ידי האלגוריתם החדש. עבור כל תכונה הנשמרת לדעתכם, הסבירו בקצרה אך במדויק מדוע היא נשמרת. עבור כל תכונה המופרת לדעתכם, תנו תסריט מדויק המוכיח זאת. (**10 נק'**)

פתרון שאלה 4:

א. התכונה המופרת היא *mutual exclusion*. נסתכל בשני תהליכים – תהליך 0 ותהליך 1 - ששניהם מבצעים לקיחת מספר לתור באותו זמן. זה אפשרי כיוון שאין אף פעולה אטומית בשורות הקוד. לכן, שני התהליכים יכולים לקבל את אותו מספר, נאמר שמספרם הוא 1. שניהם בודקים מערך של אפסים ומוסיפים לו 1. כעת שני התהליכים מגיעים לשורה 6 בקוד ושניהם עוברים אותה כיוון שלכל אחד מהם מתקיים שהוא ≥ מהשני. לכן שניהם נכנסים בו זמנית לקטע הקוד הקריטי.
שימו לב כי הקוד בסעיף א. שונה מהדוגמא בכיתה כיוון שכאן יש אי-שוויון חלש. בדוגמא בכיתה היה אי-שיוויון חזק והתהליכים יכלו להיקלע למצב של *deadlock*.

ב. גם כאן התכונה המפסיקה להתקיים היא *mutual exclusion*. ניקח שני תהליכים – תהליך 0 ותהליך 1 - הנמצאים בתהליך בחירת מספר בו זמנית ושניהם יקבלו בסופו של דבר את אותו המספר. אם התהליך 1 יקבל את מספרו ראשון ותהליך 0 עדיין לא בחר מספר, אזי עקב העדר שורה 5 תהליך 1 לא ימתין וייכנס לקטע הקוד הקריטי. כל מתחריו, כולל תהליך 0, הם עדיין בעלי מספר 0. כעת תהליך 0 מסיים לקבל את המספר 1 וגם הוא ייכנס לקטע הקוד הקריטי כיוון שלקסיקוגרפית הוא קטן יותר.

ג. תכונה ברורה שנפגעת, מסיבה שתיכף נבהיר, היא ההוגנות ונוצרת אפשרות הרעבה. אפשר גם להראות שנשברת תכונת *mutual exclusion*. חלוקת מספרי התור נעשית לחוד לתהליכים זוגיים ואי-זוגיים ואילו התור לכניסה לקטע הקוד הקריטי מתנהל באופן יחיד. לכן, תהליך אי-זוגי יחיד שמספרו 1 יקבל את המספר 1 ואילו שני תהליכים זוגיים, למשל 0 ו- 2 יקבלו את המספרים 1 ו- 2 בהתאמה. זהו כמובן תסריט אפשרי בקוד הנתון. כעת מצב של הרעבה הוא ברור, כי תהליך 1 ייכנס לקוד מספר פעמים בלתי מוגבל כל זמן שהוא האי-זוגי היחיד ואילו תהליך מס' 2 ימתין כל

הזמן כי מספרו יהיה גדול יותר. התסריט הבא מוכיח כי גם תכונת mutual exclusion מופרת. תהליך 2 רץ לבדו, לוקח מספר 1 ונכנס לקטע הקריטי. כעת, רץ תהליך מספר 1. גם הוא לוקח מספר 1, מצליח לעבור את ההשוואה הלקסיקוגראפית מול תהליך 2 ונכנס אף הוא לקטע הקריטי.

בהצלחה !