מס' מחברת_____

מס' ת.ז._____

האוניברסיטה העברית

ביה"ס להנדסה ומדעי המחשב

מבחן במערכות הפעלה

קורס מס' 67808

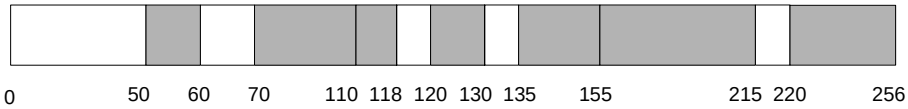
מועד א' תשס"ג
המורה: מ"ר גריגורי צ'וקלר

תאריך: 23.1.03
זמן: 2.5 שעות

במבחן שלושה חלקים. בחלק הראשון יש לענות על 10 מתוך 11 שאלות.
משקל כל שאלה 5 נקודות. בחלק השני יש לענות על 2 מתוך 3 שאלות, שמשקל כל אחת מהן 15 נקודות.
בחלק השלישי יש לענות על 2 מתוך 3 שאלות, שמשקל כל אחת מהן 10 נקודות. בכל חלק יש לסמן בבירור את השאלה שאינך רוצה שיבדקו ע"י מחיקתה בקו.
התשובות הסופיות לחלקים א' ו-ג' יש לכתוב על גבי טופס הבחינה בלבד. התשובות יש לכתוב בעט בלבד.

בהצלחה!

חלק א' (50%) יש לענות על 10 מתוך 11 שאלות בלבד. אם ענית על כל השאלות ייבדקו רק 10 השאלות הראשונות.



1. במערכת הפעלה נתונה משתמשים בשיטת הקצאת זיכרון דינאמית רציפה. האיור למעלה מראה את תמונת המצב של הזיכרון אחרי סדרה של פעולות הקצאה ושחרור. האזורים הכהים מסמנים קטעי זיכרון שהוקצו. המספרים הם ביחידות של 1K. הפעולה האחרונה שבוצעה הייתה פעולת הקצאה.
   א. איזה בלוק הוקצה אחרון אם מדיניות ההקצאה היא first-fit? רשום את כתובת ההתחלה של הקטע ואת כתובת הסיום שלו.
   ב. אם בקשת הקצאה הבאה היא לקטע בגודל 6K, היכן הוא יוקצה אם מדיניות ההקצאה היא next-fit? סמן את הקטע על האיור.

א': יש רק 2 אפשרויות לבחור מהם: 110-118, ו-155-215. מהם רק השניה, כלומר, 155-215 אינה מתאימה למקטעים פנויים משמאלה. התשובה היא 155-215.

ב': בהינתן ש 155-215, הייתה ההקצאה האחרונה, next-fit, יקצה הקטע שמתחילה בכתובת 0 כדי לספק בקשה ל-6K.

ניקוד: שווה לכל סעיף

2. במערכת מחשב נתונה, גודל המילה הוא 32 ביט. גודל הזיכרון הפיזי הוא 128 מגה בייט (MBytes). חומרת הדפדוף תומכת בגודל דף של 8K. כמה כניסות נדרשות לטבלת הדפים על מנת לתמוך בזיכרון מדומה במערכת זאת אם משתמשים ב-
   א. טבלת מיפוי ישירה (Forward Mapped Page Table)?
   ב. טבלת מיפוי הפוכה (Inverted Page Table)?
   ג. מה יהיו הגדלים של הטבלאות הנ"ל בהנחה שגודל המילה הוא 64 ביט.

תשובתכם יכולה להיות רשומה בתור ביטוי שמכיל חזקות של שתיים.

מספר הכניסות בטבלת מיפוי ישירה שווה לגודל מרחב הכתובות הוירטואלי חלקי גודל הדף. לעומת זאת, מספר הכניסות בטבלת מיפוי הפוכה הוא גודל זכרון פיזי חלקי גודל הדף. לכן התשובות הנכונות הן:

א: 2^32/2^13=2^19

ב: 2^17/2^3=2^14

ג: ישירה: 2^64/2^13=2^51, הפוכה: זהה לסעיף ב' לעיל.

ניקוד: שווה לכל סעיף

---

3. במערכת הפעלה נתונה מרחב כתובות מדומה של תהליך מורכב מ-512 דפים. זמן גישה לכניסה בטבלת הדפים אשר נמצאת בשלמותה בזיכרון הראשי הוא nsec 50. כדי ליעל את תהליך המיפוי של כתובות מדומות לכתובות פיזיות משתמשים ב-TLB בגודל של 32 כניסות כאשר זמן חיפוש ב-TLB הוא 5nsec. מהו יחס פגיעה (hit-rate) הדרוש כדי שזמן מיפוי ממוצע של כתובות מדומות לכתובות פיזיות יהיה 20 nsec. הראה את החישוב ונמק אותו בקצרה.

התקבלו שתי תשובות:

1. 50*(1-HITRATE)+5*HITRATE=20, בהנחה שחיפוש ב-TLB מתבצע במקביל למיפוי, או

---

2. 55*(1-HITRATE)+5*HITRATE=20, בהנחה שחיפוש ב-TLB ומיפוי הם סדרתיים

ניקוד: 5 לאחת התשובות לעיל (אם הסבר מתאים); 0-4, אחרת (בהתאם למהות הטעות ולהסברים)

---

4. אילו מהמדדים (metrics) הבאים ניתן לשפר באמצעות שימוש בהפקעה (pre-emption)? תניחו שתהליכים אינם מבצעים פעולות קלט/פלט (I/O).
   א. תפוקה (Throughput)
   ב. יחס תגובה ממוצע (Average Response Ratio)
   ג. זמן תגובה ממוצע (Average Response Time)
   ד. ניצולת (Utilization)
   ה. אף אחד מהמדדים הנ"ל לא ישתפר.

אם אין I/O, הפקעה יכולה לשפר אך ורק מדדים המבוססים על זמן. כלומר התשובות הנכונות הן ב' ו-ג'.

ניקוד: 5/4 על בחירת סעיף נכון או אי בחירת סעיף שאינו נכון בין הסעיפים א', ב', ג', ן-ד'. 2 על בחירת סעיף ה'.

---

5. במערכת קבצים נתונה משתמשים בהקצאה <u>רציפה סטטית</u> של שטח דיסק לקבצים. בחר/י במשפטים הנכונים מתוך המשפטים להלן:
   א. הקצאה רציפה גורמת לריסוק פנימי (internal fragmentation) <u>בלבד</u>
   ב. הקצאה רציפה גורמת לריסוק חיצוני (external fragmentation) <u>בלבד</u>
   ג. הקצאה רציפה גורמת גם לריסוק פנימי וגם לריסוק חיצוני
   ~~ד. הקצאה רציפה גורמת גם לריסוק פנימי וגם לריסוק חיצוני~~
   ה. הקצאה רציפה מיעלת גישות אקראיות לקבצים גדולים
   ו. הקצאה רציפה מיעלת גישות סדרתיות לקבצים גדולים

התשובות הנכונות הן: א', ה' (מיפוי יעיל של offset בקובץ ל-offset בדיסק), ו' (גישה סדרתית דורשת רק seek אחד לכל היותר).

ניקוד: 1 לבחירת סעיף נכון או לאי בחירת סעיף הלא נכון.

---

6. בזמן t_0 התור של תהליכים מוכנים לריצה (ready queue) מכיל שמונה תהליכים אשר התווספו לתור לפי סדר הגעתם. זמני הריצה של התהליכים הללו ידועים מראש. מצב התור הוא
4, 6, 9, 3, 25, 2, 18, 7
כאשר המספרים הנ"ל מסמנים את זמני ריצה והתהליך עם זמן ריצה 4 נמצא בראש התור. איזה מאלגוריתמי התזמון הבאים משיג זמן תגובה ממוצע מינימאלי? הניחו כי זמן החלפת הקשר הוא זניח.
   א. First-Come-First-Served (FCFS).
   ב. Shortest Job First (SJF).
   ג. Round-Robin עם קונטום של יחידת זמן אחת.
   ד. כל האלגוריתמים שקולים.

התשובה הנכונה היא ב'. לא היה שום צורך חישובים.

ניקוד: 5 אם נבחר אך ורק סעיף ב'; ניקוד חלקי אם נבחרו עוד סעיפים בנוסף ל-ב'; 0 אחרת.

---

7. באלגוריתם התזמון מבוסס עדיפויות (priority based scheduling), עדיפויות הן דינאמיות ונקבעות באופן הבא:
   1. כאשר תהליך מחכה למעבד (בתור של תהליכים המוכנים לריצה) עדיפותו משתנה לפי הנוסחה הבאה: $P(t) = P_0 + \alpha * (t - t_w)$, כאשר t_w הוא הזמן שבו התהליך הוסף לתור המוכנים לריצה.
   2. כאשר תהליך רץ עדיפותו משתנה לפי הנוסחה הבאה $P(t) = P_0 + \beta * (t - t_r)$, כאשר t_r הוא הזמן שבו התהליך התחיל לרוץ.
   3. ערכי עדיפות גדולים יותר משמעותם עדיפות גבוהה יותר.

   על ידי שינוי של α ו β נוכל לקבל אלגוריתמי תזמון שונים.
   א. איזה אלגוריתם תזמון מתקבל כאשר $\beta > \alpha > 0$ בהנחה שתהליך רץ כל עוד אין תהליך עם עדיפות גבוהה יותר?
   ב. איזה אלגוריתם מתקבל כאשר $\beta < 0, \alpha = |\beta|$ בהנחה שתהליך אינו מופקע (not pre-empted) כל עוד העדיפות שלו גדולה מ-0?

   ניתן להניח שהתהליכים אינם מבצעים פעולות קלט/פלט (I/O) והזמן הוא דיסקרטי (כלומר עדיפויות מחושבות בתום כל מספר קבוע של פעימות שעון).

א': FCFS מאחר ובכל רגע נתון, עדיפויות התהליכים יהיו בהתאם לזמני המתנה כאשר התהליך שמתבצע הוא בעל עדיפות הגבוהה מבין כולם ולכן לא מופקע אף פעם.

ב': Round Robin מאחר וכל תהליך רץ פרק זמן קבוע (בערך P_0/β) ואז מופקע, ובכל רגע נתון עדיפויות התהליכים הממתינים יהיו בהתאם לזמני ההמתנה.

ניקוד: 2.5 לתשובה נכונה לכל סעיף. 0-1 על תשובה לא נכונה (בהתאם להסברים).

---

8. סדרו את האלגוריתמים הבאים להחלפת דפים לפי קצב נפילת דפים (page faults) כאשר האלגוריתם עם קצב נפילת דפים המינימאלי מקבל דירוג 1 ואלגוריתם עם קצב נפילת דפים המקסימאלי מקבל דירוג 4. ניתן להניח שסדרות גישה לדפים הנתונות כקלט לאלגוריתמים הן סדרות טיפוסיות המקיימות את עקרון המקומיות (Locality Principle).
   א. LRU
   ב. FIFO
   ג. Optimal
   ד. Clock

Optimal, LRU, Clock, FIFO

ניקוד: 5 לתשובה נכונה; 0-2 לתשובות לא נכונות בהתאם למהות הטעות.

---

9. באילו מן הפעולות הבאות אין התערבות של מערכת ההפעלה:
   א. קריאת מערכת (system call).
   ב. גישה לכתובת וירטואלית הממופית לכתובת פיסית.
   ג. גישה לכתובת וירטואלית שאינה הממופית לכתובת פיסית.
   ד. פעולה אריתמטית שגורמת לחלוקה ב- 0.
   ה. פעולה אריתמטית שאינה גורמת לחלוקה ב- 0.
   ו. עדכון ה-Reference ביט בכניסה בטבלת הדפים.
   ז. עדכון ה-Valid ביט בכניסה בטבלת הדפים.
   ח. כל האפשריות נכונות.

   ב', ה', ו'

ניקוד: 5/7 על בחירת תשובה נכונה או על אי בחירת תשובה לא נכונה בין הסעיפים א'-ז'.
בגלל דו-משמעות מסוימת של סעיף ו' (מעודכן גם ע"י אלגוריתם Clock של מערכת הפעלה) אי בחירתו (במיוחד עם הסבר מתאים) לא השפיע על הניקוד הסופי.

---

10. במודל עבודה של Upload/Download (על קבצים מרוחקים) הקובץ ראשית כל מועתק אל מכונת הלקוח (client) שבה מתבצעות גישות אל תוכנו, וכאשר הגישות מסתיימות הקובץ מועתק חזרה אל השרת. מהו המשפט הנכון ביותר מתוך המשפטים הבאים:
   א. מודל עבודה זה מספק סמנטיקת שיתוף קבצים של UNIX (Unix File Sharing Semantics) אבל תמיד גורם לזמן השהיה (latency) גדול בפעולות קלט פלט עם הקובץ.
   ב. מודל עבודה זה אינו מספק את הסמנטיקה אבל יכול לשפר את זמן ההשהיה הממוצע של פעולות קלט פלט עם הקובץ.
   ג. מודל עבודה זה אינו מספק את הסמנטיקה ואף פעם אינו יכול לשפר את זמן ההשהיה הממוצע.
   ד. מודל עבודה זה מספק את הסמנטיקה ותמיד משפר את זמן ההשהיה הממוצע..
   ה. אף תשובה אינה נכונה.

   ב'

ניקוד: 5 לתשובה נכונה, אחרת 0-1 תלוי בהסברים וכו'

---

11. נתונה מערכת הפעלה התומכת בזיכרון וירטואלי. הביצועים של המערכת נמדדו בשלוש נקודות זמן שונות שבכל אחת מנקודות הזמן הללו רמת ריבוי תהליכים (multiprogramming level) במערכת הייתה זהה. תוצאות המדידות הן כדלקמן:
   1. ניצולת ה-CPU (CPU utilization) היא 13%; ניצולת של הדיסק המשמש להחזקת דפים מוחלפים (swap disk) היא 97%.
   2. ניצולת ה-CPU היא 87%; ניצולת של swap disk היא 3%.
   3. ניצולת ה-CPU היא 13%; ניצולת swap disk היא 3%.

   לכל אחד מהסעיפים לעיל ציין האם ניתן לעלות את רמת ריבוי התהליכים במצב זה. נמק בקצרה.

1. מערכת נמצאת במצב של thrashing, כלומר התהליכים האקטיביים אינם יכולים להתקדם מאחר וה-working set הנוכחי שלהם אינו נמצא כולו בזיכרון הראשי. כתוצאה מכך, התהליכים

האקטיביים מבלים רוב הזמן בהמתנה ל-I/O של הדפים החסרים כך שה-swap disk מנוצל כמעט עד תום וה-CPU ומשאירים את המעבד חסר עבודה. הוספת תהליכים במצב זה תפגע עוד יותר ב-working set של התהליכים הקיימים ולכן אין לעשות זאת.

---

2. במצב הזה המעבד מנוצל היטב ונפילות דפים כמעט ואין. אם נוסיף עוד מספר קטן של תהליכים (במיוחד אלה שהם I/O-bound) לא נפגע ב-working set של התהליכים הקיימים, ולכן ניתן לעשות זאת. שימו לב ששיקולים של M/M/1 אינם רלוונטיים פה מאחר והגעה של תהליכים חדשים

אינה תהליך רנדומי אלא נמצע תחת שליטה מלאה של המערכת (Long-Term Scheduler).

---

3. במצב הזה המערכת אינה מנוצלת כראוי מאחר וניצולת המעבד היא נמוכה מאוד ואין בעיות זיכרון. ברור שבמצב זה ניתן להוסיף עוד תהליכים (במיוחד אלה שהם CPU-bound).

ניקוד: היה מספיק לענות נכון ל-1 כדי לקבל לפחות 3 נקודות.

חלק ב' (30%) יש לענות על 2 מתוך 3 שאלות בלבד. אם ענית על כל השאלות ייבדקו רק 2 השאלות הראשונות.

1. נתונה פעולת חומרה אטומית swap(s,a) המוגדרת כדלקמן:

```
swap(s,a) {
        tmp = s;
        s=a;
        return tmp;
}
```

כאשר a הוא משתנה במרחב הכתובת המקומי של התהליך או קבוע; ו- s הוא משתנה משותף הנגיש אך ורק באמצעות פעולת swap (כלומר לא ניתן לבצע פעולות כתיבה/קריאה למשתנה s באופן ישיר). לדוגמה, הפעולות הבאות הן אינן חוקיות:

s=7 (פרט לזמן אתחול)

x=s+3, וכו'

השתמש בפעולת swap לפתרון בעיית קטע קריטי. עבור ניקוד מלא, פתרונך צריך לקיים את כל שלושת הדרישות של בעיית קטע קריטי (כלומר, את תנאי המניעה ההדדית (mutual exclusion), ההתקדמות (progress) וההוגנות (fairness)). ועבור ניקוד חלקי מספיק שפתרון יקיים אך ורק את הדרישות המניעה ההדדית וההתקדמות.

ראו את פתרון בית הספר לתרגיל עיוני מס' 2.

2. האלגוריתם הבא הוצע בתור פתרון לבעיית קטע קריטי (critical section) לשני תהליכים. להלן הקוד המבוצע ע"י תהליך i, $i\in\{0,1\}$:

```
Shared: Boolean flag[2];
        int turn;
Initially: turn=0;

flag[i] = true;
while (turn != i)  {
       while (flag[1-i]);
       turn = i;
}
    <CRITICAL SECTION>
flag[i] = false;
```

האם האלגוריתם הנ"ל מקיים את דרישת המניעה הדדית (Mutual Exclusion)? אם תשובתך חיובית – נמק בקצרה. אם תשובתך שלילית – פרט/י דוגמא נגדית.

האלגוריתם אינו מקיים את דרישת המניעה ההדדית. הדוגמה הנגדית הפשוטה ביותר היא לתת לתהליך 1 לרוץ ראשון ולעצור אותו לפני עדכון turn. אז להריץ את תהליך 0 ולתת לו להיכנס לקטע קריטי (שימו לב שזה חוקי מאחר ו-1 עדיין לא עדכן turn!). עכשיו ממשיכים את 1: הוא מעדכן את turn להיות שווה ל-1 וכתוצאה מכך גם נכנס לקטע קריטי.

ניקוד: פתרון הלא נכון קיבל בין 0 עד 5 נקודות בהתאם להסברים ולמהות האי הבנה.

3. נתונים N מופעים של מבנה נתונים "מחסנית" שעליו מוגדרות פעולות הבאות:
   - <item> pop(stack#)
   - push(stack#, <item>)

הפעולות הנ"ל הינן אטומיות. המופעים ממוספרים מ-1 עד N. פעולות push ו-pop אינן נחסמות (non-blocking): כלומר, אם המחסנית היא מלאה (ריקה), אזי פעולת ה- push() (pop()) חוזרת מייד עם אינדיקציה לכישלון.
ספק/י פיתרון עבור הפעולה pushpop(from, to) אשר באופן אטומי מעבירה איבר מהמחסנית שמספרה from למחסנית שמספרה to ואינה נתונה למצב קיפאון (deadlock).
הפתרון חייב לספק מידת בו-זמניות (concurrency) גדולה מ-1 (כלומר פעולות pushpop שאינן פועלות על אותן מחסניות תוכלנה להתקדם במקביל).
ראשית, נשים לב שאטומיות של pop ו-push לבד אינה מספיקה ל-pushpop אטומי. כדי לראות את זה, נתבונן בפתרון הבא:

```
pushpop(i,j) {
      item = pop(i);
      push(j, item);
}
```

כעת נניח ששתי פעולות pushpop(1,2) ו-pushpop(2,1) הופעלו במקביל. במצב הזה ייתכן שפעולות על המחסניות 1 ו-2 יופעלו בסדר הבא:

```
item1=pop(1);
item2=pop(2);
push(2,item1);
push(1,item2);
```

הסדר הזה אינו חוקי מאחר וכתוצאה מסדרת הפעולות הנ"ל האברים שהיו בראשי המחסניות יתחלפו בעוד שכל אחד מתוך שתי הפרמוטציות האפשריות של הפעולות pushpop(1,2) ו-pushpop(2,1) תשאיר את המחסניות ללא שינוי.
הפתרון הטריוויאלי למניעת ערבוב בין הפקודות הוא לסנכרן אותן באמצעות סמפור mutex גלובלי. יחד עם זאת הפתרון הזה אינו קביל מאחר והוא אינו מאפשר בכלל התקדמות של יותר מפקודה אחת במקביל.
כדי לאפשר סנכרון ביחידות עדינות יותר, נשתמש בסמפור mutex לכל מחסנית. אז כל פעולת pushpop(i,j) תצטרך לתפוס mutex[i] וגם mutex[j] לפני ביצוע פעולות pop ו-push. יחד עם זאת הפתרון הזה יהיה נתון למצב קיפאון אם לא נקפיד לתפוס את ה-mutexים לפי סדר מסוים. כדי לראות את זה נסתכל שוב בדוגמה לעיל ונניח שפעולה הראשונה מספיקה לתפוס את mutex[1] לפני השניה והפעולה השניה מספיקה לתפוס mutex[2] לפני הראשונה. במצב הזה שניהן ימתינו לנצח זו לזו. הפתרון הסופי הוא כדלקמן:

```
pushpop(i,j) {
      first=min(i,j);
      second=max(i,j);
      P(mutex[first]);
      P(mutex[second]);
      item=pop(i) ;
      push(j,item);
      V(mutex[second]);
      V(mutex[first]) ; // The release order is
insignificant
}
```

ניקוד: פתרונות דומות לפתרון לעיל קיבלו 15 נקודות. פתרונות שהציעו לתפוס mutex גלובלי לזמן המתנה על mutex[i] ו-mutex[j] קיבלו עד 12 נקודות מאחר וכל שאר הפעולות יהיו מנועות מלהתקדם במקביל כל עוד ה-mutex הגלובלי נשאר תפוס (שימו לב שהזמן הזה אינו חסום!).
פתרונות שמקיימים אטומיות אבל נתונים לקיפאון קיבלו עד 7 נקודות. פתרונות שלא מקיימים גם אטומיות וגם נתונים לקיפאון קיבלו עד 3 נקודות.

חלק ג' (20%) יש לענות על 2 מתוך 3 שאלות בלבד. אם ענית על כל השאלות ייבדקו רק 2 השאלות הראשונות.

1. מערכת קבצים דומה לזו של UNIX משתמשת בגודל בלוק של 4K וב-i-node בגודל של בלוק אחד. i-node מורכב משדות הבאים:
   1. 2 מצביעים ישירים (direct pointers);
   2. מצביע עקיף אחד (single indirect);
   3. מצביע עקיף כפול (double indirect);

   בכל בלוק ניתן להחזיק 256 מצביעים.

   להלן קטע קוד אשר כותב נתונים לקובץ. מהו מספר בלוקים שיוקצו לקובץ בסיומה של התכנית. ניתן להניח שאף תהליך אחר אינו ניגש לאותו קובץ.

```
fd = open("just_a_file", O_WRONLY |
O_CREAT);
lseek(fd, 40K, SEEK_SET);
write(fd, buff, 6.5K);
lseek(fd, 60K, SEEK_SET);
write(fd, buff, 1);
close(fd);
```

   א. 0
   ב. 2
   ג. 3
   ד. 5
   ה. 15
   ו. 20
   ז. אף תשובה לא נכונה. נמק.

ד'

---

2. בשאלה הזאת נבחן דרכים אפשריות להגנה על קטע קריטי באמצעות מערכת קבצים של UNIX.

   א. איזו מתוך ההצעות הבאות מקיימות את דרישת המניעה ההדדית (Mutual Exclusion)?
      1. תהליכים המשתמשים במנגנון מסכימים מראש על שם קובץ המנעול (למשל, "/tmp/lock"). תהליך המבקש להיכנס לקטע קריטי, יוצר קובץ ששמו ייחודי לתהליך (למשל, "foo.<pid>"). אח"כ התהליך מנסה ליצור לינק קשיח (hard link) מקובץ המנעול לקובץ הייחודי (כלומר קורא ל-link("foo.<pid>","/tmp/lock") בדוגמה שלנו). אם הפעולה הצליחה, התהליך רשאי להיכנס לקטע הקריטי, אחרת לא. כאשר תהליך יוצא מהקטע הקריטי הוא מוחק את הקובץ הייחודי ואת קובץ המנעול.
      2. קובץ המנעול קיים לפני שהתהליכים מתחילים לרוץ ומכיל את המספר השלם 0 בתחילת הקובץ. התהליך פותח את קובץ המנעול וקורא מתוכו מספר שלם (INTEGER). אם מספר שלם זה שווה ל-0 אז התהליך כותב את PID שלו לקובץ ואז הוא רשאי להיכנס

לקטע הקריטי. אם המספר גדול מ-0 אז התהליך לא רשאי להיכנס לקטע הקריטי. כאשר התהליך יוצא מהקטע הקריטי הוא כותב 0 לקובץ המנעול (במקום המתאים כמובן).

3. קובץ הנעילה קיים לפני שהתהליכים מתחילים לרוץ. כאשר התהליך מנסה להיכנס לקטע הקריטי הוא מנסה למחוק את קובץ המנעול. אם המחיקה הצליחה אז התהליך רשאי להיכנס לקטע הקריטי. אם פעולת המחיקה לא הצליחה אז התהליך לא רשאי להיכנס לקטע הקריטי. כאשר תהליך יוצא מהקטע הקריטי הוא יוצר את קובץ המנעול.

1, 3.

---

3. מהו תכן אפשרי של הקובץ בשם "/tmp/log_file" בסוף ריצת התוכנית הבאה?

```
int main() {

      int pid;
      int fd;

      fd = open("/tmp/log_file", O_WRONLY | O_CREAT, 0755);
      pid = fork();

      if(pid > 0) {
                lseek(fd, 0, SEEK_END);
                 write(fd, "aaaaa", 5);
      }
      else {
                lseek((fd, 0, SEEK_END);
                 write(fd, "bbbbb", 5);
     }
     return 0;
}
```

ניתן להניח ש-:

1. פעולת ה fork() מצליחה.
2. פעולת ה write() היא אטומית.
3. הקובץ "/tmp/log_file" לא קיים לפני תחילת הרצת התוכנית.

מאחר ושני התהליכים משתפים את המיקום (offset) בקובץ וכתיבות הן אטומיות, יש רק שתי אפשרויות לתכן הקובץ (תלוי בתזמון): bbbbbaaaaa, aaaaabbbbb.