

מדבקה

האוניברסיטה העברית בירושלים
ביה"ס להנדסה ומדעי המחשב

מבחן במערכות הפעלה
קורס מס' 67808

תאריך: 5.7.10
זמן: 2.5 שעות

מועד א' תש"ע
המורה: פרופ' דרור פייטלסון

במבחן שני חלקים. בחלק הראשון יש לענות על 11 מתוך 12 שאלות. משקל כל שאלה 5 נקודות. בחלק השני יש לענות על 3 מתוך 4 שאלות, שמשקל כל אחת מהן 15 נקודות. ניתן להשתמש בצד האחורי של הדף לטיוטה. יש לכתוב את התשובות הסופיות בעט בטופס המבחן בצורה נקייה ומסודרת.

מהלכה!

חלק א' (55 נקודות)

ענה על 11 מתוך 12 השאלות הבאות. בכל שאלה יש להקיף בעיגול את התשובה הנכונה. נא לסמן באופן ברור את השאלה שאין לבדוק על ידי מתיחת קו אלכסוני על כל השאלה.

1. בגרף המצבים הבסיסי של תהליך, אין מעבר בין מצבים התלוי בגורם הבא:
 - (א) בקשה מפורשת של התהליך עצמו
 - (ב) שיקולי עדיפויות של מערכת ההפעלה
 - (ג) פעולה של רכיב חיצוני
 - (ד) מצב הריצה של התהליך (user או kernel)
2. הבדל אחד בין תהליך לבין kernel thread הוא
 - (א) עם ריבוי תהליכים ניתן לנצל ריבוי מעבדים, אבל עם kernel threads לא
 - (ב) עם kernel threads מונעים את הבעיה של חסימה כאשר אחד מבצע פעולת I/O, אבל הבעיה קיימת כשמשתמשים בריבוי תהליכים
 - (ג) תהליכים זקוקים לתיווך של מערכת ההפעלה כדי לתקשר, ו-kernel threads לא
 - (ד) kernel threads מריצים רק קוד של מערכת ההפעלה, ואילו תהליכים יכולים להריץ קוד משתמש
3. ניתן ליצור סגמנט של זכרון משותף בין שני תהליכים ע"י יצירת טבלת דפים משותפת, שיש מצביעים אליה מטבלאות הסגמנטים של שני התהליכים. במצב זה
 - (א) הסגמנט חייב להיות ממופה לאותה כתובת וירטואלית במרחבי הכתובות של שני התהליכים
 - (ב) אם אחד התהליכים פונה לכתובת משותפת, ויש page fault, אזי שניהם יחסמו
 - (ג) צריך אלגוריתם מיוחד לתאום החלטות אודות איזה דף לפנות כאשר אין מקום
 - (ד) אם יש במערכת רק מעבד אחד, העברת נתון מתהליך אחד לשני עדיין תיקח יותר זמן מאשר הזמן לביצוע פקודת trap

גם אם הזכרון משותף, כדי שתהליך אחד יקרא את מה שהשני כתב צריך להתבצע context switch, וזה כולל כמובן מעבר למערכת ההפעלה וחזרה ולכן לוקח יותר זמן מפקודת trap

4. בזכרון המחשב טעונה תכנית לביצוע, אבל חלקיק אלפא עובר דרך הזכרון וגורם להיפוך של ביט אחד. דבר זה גורם לתכנית לסבול מ-exception כשמריצים אותה. איזה מהתסריטים הבאים אינו מתאים לתאור זה?

- (א) שינוי הביט הפך פקודה רגילה לפקודה שמורה (privileged)
- (ב) שינוי הביט הפך פקודה שמורה לפקודה רגילה
- (ג) שינוי הביט גרם לקבוע בפעולת חילוק להפוך ל-0
- (ד) שינוי הביט גרם ליצירת צרוף ביטים שאינו מייצג אף פקודה במחשב

5. במנגנון הגישה לזיכרון הממופה בצורה של סגמנטים רציפים, הסיבה העיקרית להשוואת הכתובת היחסית עם אורך הסגמנט היא

- (א) הזיכרון מעבר לקצה הסגמנט עלול להיות לא מוקצה ולהכיל זבל
 - (ב) גישות לזיכרון שמעבר לקצה הסגמנט צריכות להעשות באמצעות סגמנט אחר
 - (ג) הזיכרון מעבר לקצה הסגמנט עלול להיות שייך לתהליך אחר
 - (ד) גישה מעבר לקצה הסגמנט תגרום לפסיקה ויש למנוע זאת
- הבדיקה נעשית בחומרה, ואם היא תיכשל זה מה שיגרום לפסיקה – ההיפך מלמנוע אותה

6. כדי לממש תזמון בשיטת SJF

- (א) צריך לממש מנגנון של החלפת תהליכים (context switching)
- (ב) צריך לדעת מראש את זמני הריצה של כל התהליכים
- (ג) צריך להשתמש במחשב עם מספר מעבדים
- (ד) זה אידיאל שאי אפשר לממש אותו כי אי אפשר להריץ מספר תהליכים בו זמנית על מעבד יחיד

7. פרוטוקול IP ישדר מנה (packet) שכבר שידר בעבר פעם נוספת אם

- (א) התקבל ack עבור מנה ששודרה מאוחר יותר אבל לא עבור המנה הזאת
- (ב) עבר מספיק זמן בלי שהתקבל עליה ack (כלומר היה timeout)
- (ג) מספר המנות שהלכו לאיבוד במהלך השידור גבוה מסף מסוים
- (ד) הוא לא יעשה זאת אף פעם

8. אילוץ חשוב על הבקשות שעוברות מהלקוחות לשרת במערכת הקבצים המבוזרת NFS הוא

- (א) שהן תהינה קצרות כדי להוריד את התקורה
 - (ב) שהן תתייחסנה לכמות מוגבלת של נתונים, ובפרט לא יותר ממה שניתן לשדר במנה (packet) אחת
 - (ג) שהן תהינה בלתי תלויות לחלוטין וכל אחת תכיל את כל המידע הנחוץ אודות הקובץ עליו רוצים לבצע פעולה
 - (ד) שניתן יהיה לבצע אותן כמה פעמים ולקבל אותה תוצאה lookup שהחזיר ידית (אמנם זמנית) המאפשרת גישה לקובץ
- הפקודות אינן כוללות את כל המידע – הן תלויות בביצוע

9. ב-UNIX המידע אודות הבלוקים המכילים את התוכן של קובץ שמור ב-inode בארגון דמוי עץ. הסיבה

- לכך שהעץ "מעוות" במקום להיות עץ מלא היא
- (א) הרצון להמנע ממצב בו השורש הופך לצוואר בקבוק
- (ב) הרצון למנוע את בזבוז הזמן והשטח הכרוך בשימוש בבלוקים בלתי ישירים עבור קבצים קטנים
- (ג) הרצון לנצל עד תום את השטח המוקצה ל-inode
- (ד) חוסר החשיבות של איזון העץ לאור כך שאין מדובר בחיפוש אחר איבר שרירותי

10. באיזה מהמצבים הבאים בהם קיים מעגל בגרף הקצאות המשאבים ניתן לומר בוודאות שיש deadlock?

- (א) כשיש תהליך במערכת שאינו חלק מהמעגל
- (ב) כשיש משאב במערכת שאינו חלק מהמעגל
- (ג) כשלכל אחד מהמשאבים במערכת יש רק מופע אחד
- (ד) כשלכל אחד מהמשאבים במערכת יש יותר ממופע אחד

11. כשיש page fault צריך לבחור איזה דף לפנות מהזכרון כדי שיהיה מקום למפות את הדף המבוקש. סיבה טובה לבחור מתוך כל הדפים במערכת היא
- (א) שכתוצאה מכך הקצאת הזכרון בין התהליכים תהיה הוגנת במובן שכולם יקבלו הקצאות זכרון שמובילות לבערך אותו קצב של page faults
- (ב) שכתוצאה מכך הקצאת הזכרון תהיה הוגנת במובן שכל התהליכים יקבלו אותו מספר דפים בתוחלת
- (ג) שאז מובטח שלא יהיה thrashing
- (ד) שאז אין סכנה שלתהליך מסוים לא ישארו דפים בכלל

12. המטרה של מנעול מסוג readers-writers היא
- (א) לתת עדיפות לתהליכים שקוראים את מבנה הנתונים
- (ב) לתת עדיפות לתהליכים שכותבים (משנים) את מבנה הנתונים
- (ג) לדאוג להגינות בין קוראים לבין כותבים
- (ד) להקטין את אילוצי הסנכרון ולאפשר יותר מקבילות במערכת
- מנעול כזה מאפשר למספר קוראים לעבוד בו זמנית, וחוסך את הצורך שיחכו זה לזה. יש גם גרסאות שנותנות עדיפות לכותבים או קוראים

חלק ב' (45 נקודות)
ענה על 3 מתוך 4 השאלות הבאות.

1. בעית החוצץ החסום (bounded buffer) עוסקת בשני סוגי תהליכים: האחד מיצר פריטי מידע (producer) והשני צורך אותם (consumer). החוצץ נועד לתאם ביניהם, ובפרט מאפשר ליצרן לאכסן כמה פריטים עד שהצרכן יקח אותם; בשאלה נניח שהחוצץ בגודל 10 פריטים. פתרון פשוט לבעיה המשתמש בסמפורים הוא

<pre> Producer: While (1) { t = new item; P(empty); P(mutex); buf[in++ % 10] = t; V(mutex); V(full); } </pre>	<pre> Consumer: While (1) { P(full); P(mutex); t = buf[out++ % 10]; V(mutex); V(empty); <use t> } </pre>
--	---

- 7 (נק') i. השימוש בסמפורים השונים הוא כדלהלן (מלא ערכים או הקף בעיגול את האפשרות הנכונה):
- (א) הסמפור empty מאותחל לערך 10
- (ב) הסמפור full מאותחל לערך 0
- (ג) הסמפור mutex מאותחל לערך 1
- (ד) הפקודה P(empty) נועדה לחסום/לשחרר את התהליך כשהחוצץ ריק/לא ריק/מלא/לא מלא
- (ה) הפקודה V(full) נועדה לחסום/לשחרר את התהליך כשהחוצץ ריק/לא ריק/מלא/לא מלא
- (ו) הפקודה P(full) נועדה לחסום/לשחרר את התהליך כשהחוצץ ריק/לא ריק/מלא/לא מלא
- (ז) הפקודה V(empty) נועדה לחסום/לשחרר את התהליך כשהחוצץ ריק/לא ריק/מלא/לא מלא

- 2) (נק' ii) בהנחה ש-in ו-out משתנים סטטיים משותפים המאותחלים ל-0, סמן את הטענה הנכונה לגבי המימוש הזה. בשאלה זו נתעלם מהסכנה של overflow אם מגדילים את in ואת out יותר מדי פעמים:
- (א) הפתרון נכון לכל מספר של יצרנים וצרכנים המשתמשים באותו חוצץ במשותף
 - (ב) הפתרון שגוי במקרה הכללי, אבל נכון במקרה שיש רק יצרן אחד ורק צרכן אחד
 - (ג) הפתרון שגוי אפילו במקרה שיש רק יצרן אחד ורק צרכן אחד

- 6) (נק' iii) במקרה שבו העתקת הפריטים לוקחת זמן רב, ניתן להשיג יותר מקביליות על ידי הפתרון הבא, שבו ההעתקה עצמה מתבצעת מחוץ לקטע הקריטי:

```

Producer:
While (1) {
    t = new item;
    P(empty);
    P(mutex);
    my_in = in++ % 10;
    V(mutex);
    V(full);
    buf[ my_in ] = t;
}

```

```

Consumer:
While (1) {
    P(full);
    P(mutex);
    my_out = out++ % 10;
    V(mutex);
    V(empty);
    t = buf[ my_out ];
    <use t>
}

```

- באותם תנאים כמו השאלה הקודמת, מה הטענה הנכונה עבור אופטימיזציה זו?
- (א) הפתרון נכון לכל מספר של יצרנים וצרכנים המשתמשים באותו חוצץ במשותף
 - (ב) הפתרון שגוי במקרה הכללי, אבל נכון במקרה שיש רק יצרן אחד ורק צרכן אחד
 - (ג) הפתרון שגוי אפילו במקרה שיש רק יצרן אחד ורק צרכן אחד
- הסכנה היא שהכותב יבחר אינדקס, אבל לפני שיבצע את ההשמה של הפריט לתא הזה יהיה context switch, והקורא יקרא מהתא זבל

2. במערכת Unix מסורתית העדיפות של תהליך מבוססת על זמן הריצה שלו. חישוב זמן הריצה מבוסס על דגימות בזמן הפסיקות של השעון: בכל פעם שיש פסיקה, מייחסים את כל הזמן מאז הפסיקה הקודמת לתהליך שרץ בזמן שהפסיקה קרתה. קצב הפסיקות הוא 100 פסיקות בשניה. תהליך חדש מתחיל בעדיפות מקסימאלית, והעדיפות יורדת ביחידה אחת עבור כל יחידת זמן של ריצה (כלומר בכל פסיקת שעון שקורית כשהוא רץ). כדי לאזן זאת, פעם ב-100 פסיקות המערכת מעלה את העדיפות של כל התהליכים בחצי הדרך בין העדיפות הנוכחית שלהם לעדיפות המקסימאלית האפשרית. המתזמן בוחר תמיד את התהליך בעל העדיפות הגבוהה ביותר באותו רגע.

- 5) (נק' i) התכונות של מערכת זו הן (סמן את כל הנכונות):
- (א) לתהליכים אינטראקטיביים (שמשלבים עיבוד קצר עם המתנה לתגובה של המשתמש) תהיה עדיפות יותר גבוהה מאשר לתהליכים שרצים באופן רצוף
 - (ב) אם יש במערכת כמה תהליכים שרצים באופן רצוף, הקצאת זמן הריצה ביניהם תהיה שוויונית
 - (ג) אם יש הרבה תהליכים במערכת חלק מהם עלולים לסבול מהרעבה ואף פעם לא לרוץ (אפילו אם לא מגיעים תהליכים חדשים כל הזמן)
 - (ד) אם יש רק תהליך אחד במערכת, העדיפות שלו תהיה בערך המקסימלי האפשרי כל הזמן
 - (ה) תהליכים שרצים לסירוגין יכולים להגיע לזמן ריצה מצטבר יותר גבוה מזה של תהליכים שרצים באופן רצוף
- יתכן מצב בו תהליכים רצים בלי שהמערכת תחייב אותם עבור זמן הריצה ותוריד להם את העדיפות (כמו בסעיף ii), ולכן באופן מצטבר הם ירוצו הרבה

5) (נק' 2) ii. תכנת media player מציגה סרט עם 50 מסגרות (תמונות) בשניה, כלומר מסגרת כל 20ms (20 אלפיות שניה). התכנה מנצלת את הפסיקות של השעון כדי להציג את המסגרות בזמן. הזמן הנחוץ להציג כל מסגרת הוא 7ms (7 אלפיות שניה), ואז התכנית הולכת לישון בהמתנה לפסיקה שתסמן שעבר הזמן הדרוש. עבור כמה זמן ריצה התכנה תחויב, בהנחה שבמערכת יש גם 2 או יותר

- תהליכים אחרים הרצים באופן רציף כבר הרבה זמן?
- (א) היא לא תחויב כלל
- (ב) היא תחויב עבור 7ms מתוך כל 20ms במוצע
- (ג) היא תחויב עבור 10ms מתוך כל 20ms במוצע
- (ד) אי אפשר לדעת – זה תלוי במספר התהליכים האחרים
- כיוון שה-media player תמיד מתחיל לרוץ מיד אחרי פסיקה, ורץ רק 7ms, הוא יגמור לרוץ לפני הפסיקה הבאה והמערכת לא תחייב אותו עבור זמן הריצה אלא את התהליך שרץ אחריו

5) (נק' 3) iii. בתנאים ובהנחות של השאלה הקודמת, מה יקרה את ננסה להריץ שני media player במקביל ולהציג שני סרטים?

- (א) אין בעיות – שניהם יוצגו בהצלחה מלאה בכל מקרה
- (ב) סרט אחד יוצג בהצלחה ואילו השני לא יצליח להציג את כל המסגרות של הסרט שלו
- (ג) תלוי במזל התחלתי: או ששניהם יוצגו בהצלחה, או שסרט אחד יוצג בהצלחה ואילו השני לא יצליח להציג את כל המסגרות של הסרט שלו
- (ד) שניהם לא יצליחו להציג את כל המסגרות
- אם הם התחילו בפסיקות שעון עוקבות, כל אחד בפני עצמו ירוץ כמו בסעיף ii, לא יחויב, ולכן יהיה תמיד בעדיפות עליונה ויוכל להציג את כל המסגרות. אבל אם התחילו ביחד, רק אחד יספיק להציג את המסגרת והשני יספוג פסיקת שעון וחויב, וכתוצאה יהיה בעדיפות נמוכה יותר ולא תמיד יזכה לרוץ כשצריך

3. מקבץ שאלות על מערכות קבצים:

7) (נק' 1) i. במערכת Unix, מערכת הקבצים מקיימת את התכונות הבאות (סמן את כל הנכונות):

- (א) שמות קבצים נשמרים בספרייה (directory) יחד עם המידע על המשתמש שלו שייך הקובץ והרשאות הגישה לקובץ
- (ב) לקובץ יכולים להיות מספר שמות בתנאי שהם באותה ספרייה (directory)
- (ג) תהליכים שונים שפתחו את אותו הקובץ יכולים לשתף את המצביע לתוך הקובץ (קרי המקום שבו תתבצע פעולת הקריאה/כתיבה הבאה) שיתוף אפשרי רק בין אב ובן, לא בין תהליכים שונים שפתחו (כלומר מימוש יעיל של טבלת ה-inodes הוא על ידי העתקה רציפה לזכרון של כל ה-inodes ביצעו פעולת open) שנמצאים על הדיסק, שכן אז הגישה לכל inode נעשית באופן ישיר לפי המספר שלו
- (ה) מובטח שבקשה לקרוא בית אחד של נתונים מקובץ פתוח לא תגרום לקריאה של יותר מבלוק אחד מהדיסק, אבל בקשה לשני בתים כבר יכולה לגרום לקריאה של שני בלוקים בלוק נתונים ובלוק indirect, אם מנסים לכתוב נתונים נוספים ואין מקום על הדיסק, התהליך יחסם עד שיתפנה מקום גם בקריאת בית בודד כאשר תהליך אחד כותב לקובץ ותהליך אחר קורא ממנו, יש סכנה שהקורא לא יקבל את המידע העדכני ביותר שנכתב אלא גרסה ישנה יותר ה-buffer cache משותף לכל המערכת, וכל בלוק יכול להופיע בו רק בעותק יחיד, אז הקורא יראה מה שהכותב כתב גם אם זה עוד לא הגיע לדיסק

5) (נק' 2) ii. בעת ביצוע גישות לקובץ, מערכת ההפעלה צריכה לעדכן ואולי לנעול מבני נתונים מסוימים. סמן את כל התאורים הנכונים של מקרים כאלה:

- (א) בפעולת seek (שינוי המקום בקובץ) אל סוף הקובץ, צריך לנעול את השדה המכיל את גודל הקובץ לא צריך לנעול כי רק קוראים אותו
- (ב) בפעולת קריאה אי אפשר לעדכן את המקום בקובץ כבר בתחילת הפעולה כי יתכן שהבקשה תחרוג מגודל הקובץ ניתן לזהות זאת מראש ולעדכן בהתאם
- (ג) עדיף לעדכן את המקום בקובץ בתחילת הפעולה כדי לאפשר לפעולות נוספות לקרות במקביל
- (ד) שימוש ב-readers-writers lock על השדה המכיל את המצביע לתוך הקובץ מאפשר להרבה פעולות קריאה מהקובץ להתרחש במקביל כולם צריכים לעדכן את השדה הזה, אז כולם writers וזה לא יעזור
- (ה) לא צריך לנעול את השדה המכיל timestamp של הגישה האחרונה כשמעדכנים אותו, כי מדובר בהשמה חדשה ולא בעדכון התלוי בערך הקודם (הנח timestamp של 32 ביט)

3 נק') iii. הבוס שלך בחברה למערכות הפעלה מציע חידוש: במקום לספק עם כל מחשב חדש DVD שמכיל את מערכת ההפעלה, ומאפשר לשחזר אותה במקרה הצורך, לשמור את העותק הנוסף במערכת הקבצים על הדיסק, כדי שתמיד הוא יהיה זמין. מה התגובה שלך?

- (א) רעיון גדול! איך לא חשבו על זה קודם?
- (ב) רעיון מבטיח, אבל זה לא יכול להיות במסגרת מערכת הקבצים אלא במקום נפרד בדיסק
- (ג) רעיון משונה, אני לא רואה איך זה יכול לעבוד אם צריך לשחזר את מערכת ההפעלה כנראה שהיא לא מתפקדת ובפרט לא מסוגלת לטפל בקבצים

4. תקשורת בין מחשבים נעשית באמצעות ממשק ה-sockets ופרוטוקולי TCP/IP.

7 נק') i. אילו פרטי מידע צפויים להיכלל ב-header של IP ואילו בשל TCP של כל מנה (packet)?

לא זה ולא זה	TCP	IP	
		✓	כתובת האינטרנט של השולח
	✓		מספר ה-port במחשב היעד
	✓		קוד בקרת שגיאות על כל המנה
	✓		הצהרה על מקום פנוי לקבלת מנות נוספות
	✓		אישור על קבלת מידע קודם
		✓	גודל (אורך) המנה
✓			ספירה של מספר המנות שאבדו עד כה

IP עוסק רק בניתוב בין מחשבים, לכן מכיל רק כתובות ולא ports אין כפילות: מה שמופיע כבר ב-header אחד לא משוכפל באחר

4 נק') ii. כדי לאפשר תקשורת, שרת מבצע את קריאות המערכת socket, bind, ו-listen. אילו מהמצבים הבאים יגרמו לכישלון לפחות אחת מהקריאות האלה? (סמן את כל הנכונים)

- (א) ה-port המבוקש כבר נתפס על ידי תהליך אחר
- (ב) המחשב שמריץ את הלקוח נפל
- (ג) ביצוע פקודת ה-bind לפני פקודת ה-listen
- (ד) הגענו לגבול של מספר ה-file descriptors המותרים לנו

4 נק') iii. מנגד, לקוח צריך לבצע את קריאות המערכת socket ו-connect. אילו מהמצבים הבאים יגרמו לכישלון לפחות אחת מהקריאות האלה? (סמן את כל הנכונים)

- (א) ה-port המבוקש כבר נתפס על ידי תהליך אחר
- (ב) המחשב שמריץ את השרת נפל
- (ג) אף תהליך לא נרשם עם ה-port המבוקש במחשב היעד
- (ד) הגענו לגבול של מספר ה-file descriptors המותרים לנו