פתרון מוצע לבחינת מה"ט במערכות הפעלה

מועד א' תשע"ז, יולי 2017
מחבר: ד"ר שייקה בילו, מכללת אורט רחובות

חלק א'

שאלה מס. 1

א. **יתרונות וחסרונות של ניהול מערך מחשוב ריכוזי מול מבוזר:**
ניהול מערך מחשוב ריכוזי

יתרונות:
1. חסכון בעלויות - מחשב אחד שעליו בסיס נתונים יחיד, יותר זול מלקשר את אותה כמות משתמשים לכמה שרתים.
2. אמינות גבוהה יותר - בסיס הנתונים במחשב מתעדכן בזמן אמת כל הזמן ויש פחות סיכוי לתהליכים שיתבצעו עם שגיאות על נתונים לא מעודכנים.
3. יש צורך במערכת אבטחה יחידה המותקנת במחשב הראשי.
4. מערכת תקשורת פשוטה המחברת בין המחשב המרכזי לאנשי החברה ולקוחותיו.
5. זמינות הנתונים והטיפול בהם יעיל וזול יחסית.
6. תחזוקה, פיתוח, שינויים וניהול, זולים נוחים וגמישים יותר כאשר הם מתופעלים במחשב אחד וממנו כל הלקוחות מקבלים את העדכונים והשדרוגים.
7. שיקום מהיר כי יש לשחזר רק מצב ממחשב אחד.
8. מערכת הגיבוי זולה כיוון שהיא מגבה מחשב בודד אחד.

חסרונות:
1. זמן תגובה איטי יותר – בגלל שמדובר בריבוי משתמשים, יש עומס על המחשב המרכזי כיוון שכל העבודה נעשית עליו.
2. אם המחשב הראשי מפסיק לעבוד – נפסקת העבודה לחלוטין וכל הרשת קורסת, אין תקשורת ואין נתונים גם ללקוחות וגם לאנשי ההנהלה.
3. פריצה למחשב המרכזי חושפת את כל הארגון ומאפשרת זליגת מידע מהמחשב הראשי ומכל הלקוחות והעובדים המחוברים אליו.
4. מערכת המבוססת על מחשב מרכזי אחד היא בעלת יכולת שרידות ואמינות נתונים נמוכה כי כל קריסה שלה משביתה את כל הארגון.

ניהול מערך מחשוב מבוזר

יתרונות:
1. זמן תגובה מהיר – על מספר גדול של מחשבי סניפים יש פחות עומס ולכן זמן התגובה המקומי שלהם גם יהיה מהיר יותר.
2. רמת השרידות של המערכת הממוחשבת, רמת אמינות ונכונות הנתונים, רמת האבטחה וזמינות הנתונים במערכות כולן תהיינה ברמה גבוהה יותר מאשר במערכת ריכוזית. הסיבה לכך היא שכל היכולות של מערכת מבוזרת הן כפולה של מספר המחשבים בסניפים.

3. איכות וזמינות גיבוי הנתונים הם ברמה גבוהה כיוון שניתן לבצע גיבויים בכל סניף ומכל סניף במחשב המרכזי של החברה.
4. אבטחה – במקרה פריצה או זליגה של נתונים בגלל הביזור ניתן למנוע זליגה של נתונים כי המערכת מאובטחת הן ברמת העובד, הן ברמת הלקוח, הן ברמת הסניף והן ברמת המחשב המרכזי בהנהלה. כל זה מעלה את רמת האבטחה של כל הארגון.
5. זמינות הנתונים מהירה ביותר, אם משהו חסר באחד ממחשבי הסניפים ניתן להשתמש במחשב של סניף אחר כי המחשב המרכזי בחברה מחובר לכולם ונותן להם גם יכולת גיבוי וגם ניהול. הליך זה יכול להיות שקוף לחלוטין הן ללקוח והן לעובדי החברה.
6. ביזוריות רבה המאפשרת חלוקת משאבים טובה יותר בין כל אמצעי המחשוב של החברה.
7. במקרה של קריסה מקומית - השיקום יהיה מהיר איטי יותר במערכת ממוחשבת כי יש לשקם רק מחשב סניף.

חסרונות:

1. עלות יקרה – כל המנגנונים של מערכת ההפעלה כולל אבטחה, זיהוי, ניהול, ביזור קבצים, תקשורת, תזמון ושיקום הכול מוכפל לכמות מחשבי הסניפים ועוד אחד למחשב הנהלת החברה. דבר זה מגדיל את העלויות הישירות של תפעול המערכת וכן את והתקורה.
2. הסתמכות על סנכרון בין המחשבים שבסניפים לבין המחשב המרכזי – מחשבי הסניפים צריכים לשמור על סנכרון קבוע ע"מ שהנתונים יהיו כל הזמן מעודכנים, כאשר מחשב בסניף אחד קורס מסיבה כלשהי זה פוגע בעבודה המקומית של אותו סניף. יתכן כי פעולות שבוצעו באותו סניף תצטרכנה שיקום מקומי ועד אז עלולות להתבצע פעולות על נתונים לא מעודכנים.
3. כל שינוי, שיפור או שדרוג במערכת הממוחשבת המבוזרת מחייב ביצוע בכל אחד ממחשבי הסניפים ובמחשב המרכזי כאחד.
4. מערכת תקשורת מורכבת המניעה מידע בין הלקוחות לבין מחשבי הסניפים לבין המחשב המרכזי.
5. במקרה של קריסה כללית - השיקום איטי יותר במערכת ממוחשבת כי יש לשקם את כל מחשבי הסניפים.

ב. **השיקולים לעבור למערכת ממוזגת של שתי החברות הן בשלושה מישורים:**

1. במישור הכלכלי - לאחר המיזוג ניתן להשתמש בשתי מערכות המחשוב במקביל, יש איגום משאבים, יש חיסכון בהשקעה ראשונית, כי אין צורך ברכישת מחשבים חדשים נוספים אלא להשקיע בתקשורת ובמערכת ניהול משוטפת שתמזג בין שתי המערכות. כל הניהול הכלכלי של החברה החדשה ינוהל ע"י המערכת החדשה גם במחשב אחד ראשי וגם במחשבי הסניפים. דבר שיביא לשליטה טובה יותר על הנתונים ויעזור להנהלה לנהל טוב יותר את החברה החדשה. ניתן יהיה לחלק את העומס בין המחשב הראשי למחשבי הסניפים, ניתן יהיה להשתמש במחשבי הסניפים כגיבוי למחשב הראשי כך שיחסכו עלויות גיבוי, שחזור ושיקום לאחר קריסה.
2. במישור המחשובי תקשובי - שתי המערכות תאוחדנה, נתוני שתי החברות תהייה נגישות גם במחשב המרכזי וגם במחשבים המקומיים. למערכת החדשה תהיה אמינות גבוהה יותר, יכולת שרידות גבוהה, יכולת התפתחות עתידית גדולה יותר, יכולת ביצוע גיבויים, שיקום, ושדרוג באופן יעיל ונוח יותר. מהירות התגובה תעלה כי בגדול יכולת

המחשוב של הארגון תעלה. השליטה על נתוני החברה המאוחדת תהיה בטוחה ואמינה יותר. כל נושא גיבוי הנתונים יהיה נוח יותר כי מצד אחד הם נמצאים במחשבי הסניפים וניתן לעשות גיבוי מקומי לכל סניף במחשב המרכזי ומצד שני ניתן לבצע גיבוי של המחשב המרכזי במחשבי הסניפים.

3. במישור ההתפתחותי עתידי - במקרה והחברה תמשיך להתפתח ותגדל ובעקבות זאת יעלו הדרישות לאמצעי מחשוב חדשים ונוספים תמיד ניתן להוסיף ולחבר עוד סניפים למחשב המרכזי בשיטה זו. במקרה והעומס יעלה ניתן להגדיל את המחשב המרכזי או לחילופין להוסיף עוד מחשבי סניפים שיחלקו עימו את העיבודים.

ג. **כדי להגיע לאופן התפקוד המיטבי של האיחוד ניתן להשתמש בשתי שיטות חיבור, הבחירה לבסוף תהיה בוודאי תוך כדי השוואה בין עלות החיבור לתועלת הנגזרת ממנו:**

1. חיבור המחשב המרכזי של החברה הראשונה לבין המחשב המרכזי הנמצא במשרדי הנהלת החברה השנייה.
במקרה זה התהליך פשוט יותר כי מבצעים חיבור בין שתי מערכות בלבד כאשר תהיה מערכת שליטה אחת משותפת לשתי החברות. במקרה זה מחשבי הסניפים לא יהיו מחוברים למחשב החברה השנייה, לא יגרמו לעליה בעומס העבודה שלו ולא יאפשרו גישה ישירה ממחשב החברה הראשונה לסניפי החברה השנייה. החיבור פה יהיה פשוט יותר, זול יותר וידרוש מעט משאבים יחסית. בקיצור חיבור זה ייתן מענה סביר וטוב לניהול החברה החדשה במחיר יחסית לא יקר.
2. חיבור בין שני המחשבים כמו בסעיף הקודם וביצוע חיבורים נוספים וכפולים בין כל מחשבי הסניפים של החברה השנייה לבין המחשב המרכזי של החברה הראשונה.
במקרה זה החיבור בין שתי מערכות המחשוב יהיה מורכב הרבה יותר אך מנגד הוא יוסיף למערכת החדשה יתרונות (אמינות, שרידות, אבטחה, שיקום) שאינם בחיבור הקודם. אפשרויות הגיבוי של כל מערך הסניפים של החברה השנייה גם במחשב של החברה הראשונה יתרמו רבות לאיכות ואמינות הנתונים. ברור כי המחיר של אפשרות חיבור זו יהיה יקר יותר, מערכת התקשורת תהיה מורכבת יותר, מערכות ההפעלה של שני המחשבים הראשיים ושל הסניפים יצטרכו להיות מסונכרנים. בקיצור חיבור זה ייתן מענה הכי טוב לניהול החברה החדשה אך במחיר יקר.

ד. **הבעיות הצפויות באיחוד שתי מערכות המחשוב הן חוסר התאימות שלהן:**

1. בעיקר נושא תאימות מערכת ההפעלה של שתי החברות, בעיות התקשורת שיש לפתור בעת חיבור המערכות, בעיות הניהוליות מי יהיה אחראי על מה בכל תת חברה. בנוסף לכך יצטרפו כל שלל הבעיות הצפויות מרגע הפעלת מערכת מבוזרת כגון עלות יקרה יותר מאשר מערכת אחודה, הפעלת כפילויות של מנגנונים בגין המצאות מחשבים גם בסניפים וגם במרכז. בעיות אבטחה כי כעת יש לאבטח בכמה רמות: ברמת הסניפים, ברמת הרשת ביניהם, ברמת הקישוריות למחשב המרכזי, ברמת הגישה למחשב הראשי. כל נושא זיהוי משתמשים, ניהול המערכת, חלוקת אחריות וסמכויות לביצוע פעולות פיתוח, אחזקה, שדרוג, רכש ועוד יהיו כעת מורכבות יותר. ניהול מערך הקבצים אשר יהיה הן מבוזר והן במחשב הראשי יהיה מורכב יותר. מערכות תזמון תהליכים חייבות להיות מסונכרנות כדי שמידע שעובר מהסניפים למרכזי וההיפך לא יתנגש או יפגע. מנגנון שיקום לאחר קריסה יצטרך להיות משוכפל אצל כל הסניפים בנוסף למרכזי מה שמגדיל את העלויות הישירות והתקורה.

2. כל נושא הסתמכות על סנכרון בין הסניפים והראשי יצטרך להיות מנגנון קבוע ע"מ שהנתונים יהיו כל הזמן מעודכנים, כאשר מחשב באחד הסניפים נופל זה פוגע בעבודה ועלולות להתבצע פעולות על נתונים לא מעודכנים. כל שינוי במחשבים המקומיים והראשי חייב להיות ידוע ומתואם. יש לשים לב כי מצד אחד אמינות ושרידות המערכת החדשה גבוהות יותר מהקודמות אולם במקביל העלויות של תפעול מערכת תקשורת מורכבת שתניע ותגבה מידע בין הסניפים לראשי תהיה יקרה יותר וכן תהליך השיקום יהיה איטי יותר כי יש לשקם אחרי קריסה יותר מחשבים במידה וקרסו כולם.

שאלה מס. 2

א. היתרונות של מערכת הפעלה המבוססת על שימוש בזיכרון ווירטואלי הן:

1. ניתן להריץ במערכת זו מספר תכניות במקביל.
2. התכניות הרצות במקביל לא יתנגשו אחת בשנייה בגלל שהן רצון בזיכרון ווירטואלי ולא בזיכרון הממשי ולכל אחת מהן הוקצה מרכב כתובות שונה.
3. גודל הזיכרון הווירטואלי אינו מוגבל בגודל זיכרון ה- RAM שיש במחשב כי הוא יכול לגלוש לדיסק החיצוני.
4. ניתן כיום בגלל גודל מילה ורוחב פס הכתובות לתמוך בזיכרון ווירטואלי גדול מאד ולמקם אותו בדיסק.
5. משך הזמן שלוקח לתוכנית להתחיל לרוץ במערכת מהרגע שהפעלנו אותה מתקצר משמעותית כי לא כולה עולה לזיכרון הממשי אלא היא משתמש בחלקה בזיכרון הממשי ובחלקה בזיכרון הווירטואלי.
6. זמן הטעות לתוכנית וזמני התגובה שלה מתקצרים.
7. במקום לנהל בזיכרון הממשי בלוקים של נתונים בזיכרון הווירטואלי אנו מנהלים דפים.
8. היתרון בשימוש בכתובות ווירטואליות הוא בזה שאין צרוך לבצע תרגום של הכתובות לכתובת פיזית ולכן הוא יותר מהיר הן בגישה, הן בהבאה והן בהוצאה משימוש של דפים.
9. זיכרון ווירטואלי הוא מרחב זיכרון מדומה העומד לרשות תהליך. מרחב הזיכרון הווירטואלי של תהליך יכול להיות גדול מהזיכרון הפיזי הראשי.
10. מרחב הזיכרון הווירטואלי ממופה בחלקו לזיכרון הפיזי הראשי ובחלקו לזיכרון הפיזי המשני.
11. עבור כל תהליך , הקוד המבוצע ע"י המעבד בנק' זמן כלשהי וכן הנתונים הדרושים, חייבים להיות בזיכרון <u>הפיזי הראשי</u> באותה נק' הזמן.

ב. דוגמאות לשימוש במערכת הפעלה עם זיכרון ווירטואלי:

- מערכות מחשבים אינטראקטיביים - **Time Sharing Systems**, במערכת זו זמן המעבד נחלק בין מספר משימות והמעבד מוקצה רק לצורך ביצוע משימה הנמצאת בזיכרון, כאשר אין מספיק זיכרון נעשה שימוש בזיכרון הווירטואלי.
- מערכות מחשבים מקביליים - **Parallel Computing Systems** , במערכת זו המעבדים במערכות אלה חולקים זיכרון ושעון התקשורת נעשית דרך פס תקשורת משותף. יתרונות המערכת המקבילית הן תפוקה מוגברת, נצילות משופרת, חיסכון במשאבי מחשב, אמינות גבוהה ויכולת גיבוי ושיקום גבוהות. מערכות מקביליות נחלקות לסימטריות בהן כל מעבד מריץ עותק זהה של מערכת ההפעלה ואסימטריות בהן לכל מעבד מקצים משימה ספציפית.

- מערכות מחשבים מבוזרות - **Distributed computer Systems**, במערכות אלה מתפעלים מחשב בהן מבוצע החישוב באופן מבוזר בין כמה מעבדים פיזיים במקביל. לכל מעבד במערכת כזו יש את הזיכרון המקומי שלו ואת הזיכרון הווירטואלי שלו במידה והפיזי אינו מספיק. יתרונות המערכת הזו הם שיתוף במשאבי מחשב, אמינות, יעילות זמינות ושרידות גבוהים ושיתוף במקרים בהם העומס גבוה כדי להגיע לחישוב מהיר יותר.

חלק ב'

שאלה מס. 3

א. מערכת מחשבים הכוללת שני מחשבים או יותר בעלי זיכרון ראשי משותף, כשהאחד הוא הראשי, שאליו מחוברות כל היחידות ההיקפיות, הוא האחראי לפעולות הקלט/פלט, והוא נקרא "אדון". האחרים מבצעים את העיבוד בלבד, והם נקראים "עבדים". (גם: יחסי אדון-עבד). מערכת הפעלה מסוג זה נחלקת לשני סוגים עיקריים:

1. **מערכת עיבוד סימטרי מקבילי סימטרי- Symmetric Multiprocessing** שבה על כל מעבד רצה אותה מערכת הפעלה. כל מערכות ההפעלה האלה מקושרות ביניהן ע"י פס תקשורת משותף הנקרא bus המבטיח כי כל קלט יגיע למעבד הנכון. במקרה זה מספר תהליכים יכולים לרוץ במקביל ובו זמנית בלי לפגוע בביצועים.

יתרונות:

- אמינות גבוהה ואפשרות המשך תפקוד חלק של המערכת במקרה של נפילת מעבד.
- חלוקה טובה של עומס העבודה בין המעבדים, כיוון שכל משימה יכולה להישלח למעבד אחר.
- בכל רגע נתון יש רק מעבד "אדון" אחד בלבד, תופעה שמונעת התנגשויות בין מעבדים.
- מעבדים יכולים לשתף פעולה ביניהם לצורך ביצוע של תהליך של משתמש.
- אחוזי הנצילות של כל אחד מהמעבדים גבוהה יותר מאשר במערכות אחרות.

חסרונות:

- הקוד של מערכת ההפעלה חייב להיות שיתופי ודרושה בלעדיות.
- בעיות של תחרות בין המעבדים יכולה לגרום להתנגשויות חומרה ותוכנה.
- נדרש מנגנון מיוחד נגד התנגשויות הן בחומרה והן בתוכנה.
- עלולות להיווצר חסימות במערכת בגלל תנועה רבה בין המעבדים.
- נדרשת תקורה נוספת עבור מערכת ההפעלה,
- יש תחרות רבה יותר ובמקביל על משאבי המערכת, חומרה ותוכנה.
- מתרחשים עיכובים בגלל מעבר מידע ניהולי רב בין ה"אדון" לבין המעבדים "העבדים".

2. **מערכת עיבוד אסימטרית – Asymmetric Multiprocessing** שבה ישנו מעבד אחד ראשי המריץ את מערכת ההפעלה ותפקידו להקצות משימות למספר מעבדים משניים באופן לא סימטרי. לכל מעבד יש את המשימות שהוקצו עבורו, אם המעבד הראשי נופל אחד ממעבדי "העבד" הופך ל"אדון".

יתרונות:

- מניעת חסימות הדדיות אינה קשה יותר מאשר במערכות מקבילות אחרות.
- אין צורך בקוד שיתופי כי כל מעבד מפעיל גרסת מערכת הפעלה משלו ואין הוא זקוק לקוד שיתופי כי רק הוא מבצע את התהליך בזמן נתון.
- מתאים למערכות בהן יש עיבוד רב מאוד וצורך רב ביכולות עיבוד עתיר תהליכי חישוב.
- מתאים למערכת בעלת כושר עבודה יציב וקבוע ללא שינויים תכופים.
- אופטימלי למערכת בעלת מתזמן פשוט ולא מורכב.

חסרונות:

- "העבדים" יכולים להריץ רק תהליכים עתירי חישוב.
- כל תהליך קלט פלט דורש לפחות שתי רמות של פסיקה.
- במערכות שהתזמון שלהן מורכב יותר יהיו תורים ארוכים אצל המעבד "האדון".
- נפילת מעבד "אדון" גורמת לקריסת המערכת כולה.
- מתזמן לא מוצלח יגרום לחוסר יעילות במערכת.

לסיכום ניתן לומר כי במערכות שיש בהן מספר רב של יחידות CPU נפוץ יותר השימוש בעיבוד האסימטרי כי אז ניתן לוותר על מעבד אחד בשביל ניהול האחרים "אדון". בנוסף, בעיבוד סימטרי יש overhead על גישות לזיכרון ויכולות להיות התנגשויות של מעבדים שונים, תופעה שעשויה להתרחש בשכיחות גבוהה ככל שיש יותר מעבדים.

ב. ניתן לומר כי במערכות בעלות ריבוי מעבדים ניתן לבצע שימוש יעיל בשתי שיטות, הראשונה שימוש בריבוי תהליכים אשר ירוצו במקביל על מעבדים שונים ויבצעו משימות שונות או ע"י שימוש בחוטים אשר יבצעו תקשורת בין תהליכים שיפוצלו בין כמה מעבדים אך יהיה ביניהם קשר באמצעות חוטים threads. כיוון שתהליך הוא בעצם המימוש של ריצת התוכנית על מערכת ההפעלה החוט הוא אמצעי המאפשר לתהליך בודד להתחלק לכמה תהליכים עצמאיים היכולים לרוץ במקביל. זהו מימוש קלאסי של מערכת ריבוי מעבדים ליעילות אופטימלית.

מימוש Thread –ים בא בשתי רמות:

- System Thread – התמיכה בריבוי תכניות נתמך ברמת מערכת ההפעלה.
- User End Threads – הניהול של תתי התכניות נעשה ביישום עצמו בדרך כלל בסיוע מערכת ההפעלה בכל הקשור למדידות זמן וכו'.

כיוון שאנו עובדים במערכות רב-מעבדים שהן גם מערכות הפעלה מוגנות, שבין השאר מגינות על תהליכים זה מזה. בגלל שתהליכים הם יקרים ומספרם במערכת מוגבל, לכן אנו זקוקים לחלופה יעילה והיא התהליכונים. מידור בין תהליכים הוא גבוה ותקשורת בין תהליכים היא דבר יקר ומורכב הדורש תקורה רבה ולכן מערכת Thread-ים נחשבת בעיני המערכת לתהליך אחד, ולכן אין מידור בין ה-Thread-ים: מרחב הכתובות של כולם משותף. יצירת Thread מחייבת הרבה פחות פעולות מיצירת תהליך ולכן יותר מהירה, יעילה וזולה. ברוב המערכות יש פחות הגבלות על יצירת Thread-ים לעומת יצירת תהליך שהוא מהלך התלוי במערכת ההפעלה. העברת מידע בין Thread –ים מתבצעת דרך שטחי זיכרון מה שיוותר מורכב זה הסנכרון ביניהם.

מהלך חייו של Thread:

- נולד במצב new
- כאשר עושים start עובר ל-runnable (רוצה לרוץ)
- אחרי שעובר ל-Thread runnable עשוי לעבור לאחד מהמצבים הבאים:
  1. למצב המתנה wait ביוזמתו לזמן לא מוגבל
  2. למצב המתנה ביוזמתו לזמן מוגבל timed wait

3. למצב חסימה אם ניסה לגשת למשאב נעול blocked
4. למצב סיום Terminated.

יצירת Thread ניתן לבצע ע"י:

1. הגדרת מחלקה חדשה כמחלקת בן של Thread.
2. הגדרת הקוד הביצועי של ה-Thread ע"י מימוש מתודה public void run()
3. בתוכנית הראשית להקצות מופע של המחלקה ולבצע על המופע ()start.

יישום מרובה Thread-ים הוא בעייתי מכמה סיבות שהחשוב בהם הוא הצורך בסנכרון מתמיד.
יש שתי אפשרויות ליצירת Thread –ים:
שיטה 1:

- המינימום הוא לממש מחלקה שמממשת את הממשק המובנה בשפה Runnable.
- חייבים לדרוס את המתודה run לספק תכנית ביצועית.

שיטה 2:

- להגדיר מחלקה שמממשת את Executer
- יעיל יחסית כאשר מקצים הרבה thread –ים לפעולות קטנות.
- בשיטה הזו נשמרים המשאבים של thread שסיים להקצאה חדשה בהמשך.
- מאפשר מימוש מה שנקרא מאגר thread–ים thread pool.
- מקצים מאגר של thread-ים, ע"י פקודה

ExecuteService *משתנה* = Exceutors.newCachedThreadPool();

- הקוד של ה-thread-ים עדיין כמו קודם, מיישמים של Runnable
- עדיין מקצים את המופעים ע"י new
- מפעילים את ה-thread-ים ע"י execute (במקום start) של מאגר ה-thread-ים.
- בתוכנית מרובת thread-ים, הקוד של ה-main נחשב בעצמו ל-thread.

העברת מידע בין Thread-ים היא עניין פשוט ע"י שימוש במשתנים סטטיים, הקצאה דינמית של שטח המשמש לתקשורת בין Thread-ים והעברת כתובת השטח לידי ה-Thread-ים המעורבים. לעיתים צריך לדאוג שתכניות עצמאיות שמשתפות פעולה יתבצעו בסדר מסוים. לפעמים חלק מה-Thread-ים לא יכולים להמשיך (אין להם מה לעשות) עד שאחד ה-Thread –ים יעשה פעולה מסוימת. דוגמא מיוחדת של הרעיון הזה הוא מניעה הדדית Mutual Exclusion. מניעה הדדית פירושו ש-Thread ניגש למשאב משותף רק כאשר הוא תקין קרי: לפני שינוי או לחלוטין לאחר השלמת שינוי. הבעיה נובעת מכך שבמסגרת מכסות זמן, Thread יכול להיעצר בכל נקודה בכל רגע. מקור הבעיה היא בעצירת thread תוך כדי שינוי משאב גלובלי.

פתרון אפשרי הוא מנגנון נעילות:

thread נועל משאב משותף לפני השינוי, וכל thread המנסה לגשת למשאב הנעול – <u>נעצר</u>.
כאשר ה-thread הנועל משחרר את המשאב, הראשון מבין הממתינים משתחרר וחוזר חלילה.
מנגנון נעילות הוא מנגנון רצוני שאינו מונע עצירה של thread באמצע.
אפשר לנעול כל מופע של מחלקה כלשהיא משום שמנגנון נעילות ממומש במחלקה Object.
מתודה synchronized – אם thread ראשון קורא למתודה על מופע מסוים – הוא נועל את המתודה.
המתודה משתחררת רק כאשר הוא יוצא מהמתודה.
אם ה-thread נעצר באמצע – המתודה על המופע נשארת נעולה.
אם בזמן שהמתודה על המופע נשארת נעולה היא נקראת ע"י thread אחר ה-thread האחר ייעצר.

ג. מערכות הפעלה מודרניות מאפשרות לריצה של תהליך (Process) לתפעל מספר תהליכונים במקביל בתוך מרחב כתובות אחד. כל תהליך חדש מתחיל את ביצועו באמצעות 'תהליכון ראשי' שמיצר תהליכונים נוספים. מנגנון הריצה באמצעות תהליכונים נותן

למשתמש מהירות תגובה ורציפות פעולה כאשר התהליך (יישום) מבצע כמה משימות במקביל. מערכות הפעלה שיש בהם ריבוי תהליכונים (Multithreading) בתהליך אחד, מאפשרות גישה חופשית של התהליכונים בתהליך לכל המידע במרחב הכתובות של אותו תהליך. מצב זה מאפשר למתכנת מערכות תוכנה שיתוף מידע פשוט ומהיר בין התהליכונים, המחיר של זה הוא כמובן סיבוכיות בכתיבת התוכנית, והגדלת הסיכוי לתקלות. דוגמה למצבים מורכבים אלה היא תחרות בין התהליכונים העשויה לגרום לקריסה ודריסה וצורך בשיקום של מידע שנפגע. הפתרון לכך הוא הכלים שמספקות מערכות ההפעלה ככלי סנכרון כדוגמת נעילה ,סמפור וקטע קריטי. כלים אלו מאפשרים לתאם את הגישה למידע על ידי התהליכונים.

השימוש הנכון בחוטים הוא בדרך כלל במצבים בהם מפעילים תכניות מרובות תהליכים אשר הן מסובבות מודולריות, כוללת שימוש יעיל במשאבי מחשוב, ומבוססות חישוב מבוזר.

המשימות המאפיינות מערכות בעלות ריבוי מעבדים וריבוי חוטים הן:

1. Non-Blocking I/O
2. משימות בלתי תלויות.
3. טיפול במימוש אלוגריתמים מקביליים.
4. טיפול במשימות הרצות ברקע כמו Collection Garbage
5. רצון לנצל מספר מעבדים פיזיים במקביל במחשב.
6. רצון לנצל את המעבד בזמן שהתוכנית ממתינה למשאב או להתקן איטי כמו קריאה מדיסק.
7. מתן אשליה למשתמש שמספר פעולות, כמו טיפול בקלט נעשות במקביל.
8. מימוש שרתים שבהם כל לקוח מקבל חוט נפרד.
9. רצון להפריד מודולים שיכולים לרוץ בסדרי ביצוע שונים.

דוגמא לתוכנית עם שימוש בחוטים לטיפול בבעיית יצרן צרכן:

```
semaphore freeSpace,
        initially n
Semaphore availItems,
        intiailly 0
Producer:
  repeat
    wait( freeSpace);
    buff[pp] = new item;
    pp = (pp+1) mod n;
    signal( availItems);
  until false;
Consumer:
  repeat
    wait( availItems);
    consume buff[cp];
    cp = (cp+1) mod n;
    signal( freeSpace);
  until false;
```

שאלה מס. 4

א. לעבדות Batch יש שתי שיטות תזמון :

1. להערכתי שיטת ה- FCFS היא אחת משתי השיטות המתאימות לשימוש במחשב זה כיוון שהיא מתאימה יחסית די טוב לאלגוריתם של הפעלת עבודות בשיטת האצווה Batch אולם היא לא האופטימלית ביותר. בשיטה זו אין חשיבות לגודל העבודה, לזמן בו היא הגיעה, למשך זמן הריצה שלה במעבד למשאבי המחשב אליהם היא זקוקה, לעדיפות שלה. בפועל במערכת זו כל תהליך עבודה מגיע סדרתית ונכנס לעיבוד לפי סדר הגעתו עד לסיום ריצתו. השימוש בזיכרון ווירטואלי במערכת זו יכול רק לעזור במקרים בהם העבודה דורשת שימוש בזיכרון רב אשר אינו תמיד בנמצא בזיכרון הפיזי.

יתרונות: שיטה פשוטה ללא סיבוכיות יתר, זולה ומתאימה לעבודות קצרות, לא אופטימלית, לא מאפשרת שימוש מקבילי במעבדים.

חסרונות: יכולה לגרום לאפקט השיירה בו תהליכים רבים יגיעו אך יצטרכו להמתין עד שקודמיהם יסיימו ריצתם.

נחשבת שיטה לא מתקדמת ומתאימה לעבודה מול מעבד אחד בנצילות לא גבוהה במיוחד.

2. שיטה נוספת שגם היא מתאימה ביותר להפעלת תהליכים בשיטת האצווה היא ה- SJF היא נחשבת ליעילה יותר מ- FCFS כיוון שהיא מתחשבת בזמן הריצה הנדרש עבור כל עבודה ,היא מקדמת ונותנת עדיפות לעבודות קצרות על פני עבודות ארוכות. בשיטה זו העבודות הקצרות יסיימו במהירות ויינתנו זמן ריצה ומשאבי מחשב גדולים יותר לעבודות גדולות אחרי שהקטנות יסיימו ריצתן. יש שני סוגים של SJF אחת Pre-emptive והשנייה Non-Pre-emptive. העבודה הקצרה הבאה היא יתרון בגלל הפשטות שלה, כי זה ממזער את כמות הזמן הממוצע שכל תהליך צריך לחכות עד שביצועו הושלם. עם זאת, היא אינה חפה מחסרונות. העבודה הקצרה הבאה משמשת בסביבות מיוחדות שבהן אומדנים מדויקים של זמן ריצה זמינים. העבודה הקצרה הבאה יכולה להיות יעילה עם תהליכים אינטראקטיביים אשר בדרך כלל יעבדו בדפוס עבודה הנע לסירוגין בין המתנה לפקודה לבין ביצועה. אם ההתפרצות הביצועית של תהליך נחשבת ל"משימה "נפרדת, התנהגות העבר יכולה להצביע על התהליך הבא, בהתבסס על אומדן זמן הריצה שלו.

יתרונות: עבודות קצרות תרוצנה בעדיפות גבוהה. נותנת את זמן ההמתנה הממוצע הנמוך ביותר.

חסרונות: יש צורך לדעת מראש לפני ריצת העבודה מהו זמן הריצה שלה כדי לארגן את התור בצורה הכי אופטימלית. בעוד שאי אפשר תמיד לחזות את זמן הביצוע באופן מושלם, ניתן להשתמש במספר שיטות לאמוד אותו, כגון ממוצע משוקלל של זמני ביצוע קודמים.

יש מצב שיתרחש תהליך של הרעבה למשאבים אצל תהליכים הממתינים לריצה. **דבר שעשוי לגרום לבזבוז זמן.**

ב. שיטת ניהול זיכרון חיונית כאשר מתפעלים מערכת שאין בה צורך וחשיבות למתן עדיפויות לעבודות בעלות זמן ריצה קצר. הכוונה למערכות שמבצעות באופן סדרתי אותן עבודות באותו, באותו משך זמן ריצה כאשר הכל במערכת יציב וללא שינויים. הביצוע של כל העבודות הוא בדרך כלל בעל אותו עומס צפוי כך שאין כל צורך בחישובי אומדנים מדויקים של זמני ריצה זמינים. זה מתאים למערכות שאין בהן תהליכים אינטראקטיביים ואין צורך לצמצם את המרחק בין המתנה של עבודה אחת לחברתה כיון שכולן פחות או יותר די

דומות. במערכות כאלה גם אין צורך להתבסס על ריצות של עבודות מהעבר כי אין לכך כל חשיבות לגבי עתיד ריצת העבודות הבאות בסדר אחת אחרי השנייה.

ג. זיכרון ווירטואלי מוגדר כמרחב זיכרון מדומה העומד לרשות תהליך, והוא גדול מהזיכרון הפיזי המשמש תהליך זה. מרחב זה נמצא על אמצעי לאחסון נתונים, כגון דיסק קשיח וחלקים ממנו, כאלה הדרושים למעבד ברגע נתון מובאים לזיכרון הממשי על ידי מערכת ההפעלה. כאלה שאינם דרושים מפונים.

כיון שכך מרכיבי של זיכרון ווירטואלי הקובעים את מידת בתפקוד שלו הם:

1. **גודל הזיכרון הווירטואלי:** יש לזכור שבניהול זיכרון ווירטואלי ישנה העברת אחריות על ניהול חלק מהזיכרון מהתוכנית אל מערכת ההפעלה. מגבלת הזיכרון הזמין לתהליך איננה תלויה בגודל הזיכרון הראשי, אלא בפרמטרים אחרים: גודל הדיסק הקשיח (שיכול להגיע לאלפי ג'יגה בייט), גודל המילה במחשב ושיטת הייצוג והתרגום של כתובות בזיכרון. אם תהליך דורש זיכרון רב ממה שמוקצה עבורו בזיכרון הפיזי, יוקצה לו זיכרון נוסף, ווירטואלי על הדיסק.
2. **שמירה על רצף כתובות:** במקרה בו תהליך מבקש זיכרון רציף, עבור טיפול במבני נתונים כגון מערכים ,הוא יכול להתייחס לזיכרון ווירטואלי רציף, ואז לא נדרש לאתר בזיכרון הראשי רצף של כתובות שאינן בשימוש.
3. **בידוד והפרדה בין תהליכים:** במקרים בהם קיימים לפחות שני תהליכים שאינם משתפים זיכרון אך מנסים יחד לגשת לאותה כתובת, הם ניגשים בפועל לכתובות שונות, אלא אם מערכת ההפעלה המבוססת על הזיכרון הווירטואלי מגיעה למסקנה שאין סכנת התנגשות ושאין בעיה בשיתוף של כתובות פיזיות - אם המידע הדרוש זהה, והגישה היא לקריאה בלבד, או שמדובר במידע שמשותף במפורש בין תהליכים.
4. **ניהול יעיל יותר של החומרה:** מימוש של ניהול הזיכרון ע"י הקטנת השברור הפנימי והחיצוני כמו במקרים של ניהול דיסקים קשיחים, ע"י איחוד גישות לסקטורים סמוכים, וניהול יעיל של זמן העיבוד, ע"י שימוש במתזמן תהליכים בהתאמה למידת הנגישות המידית של המידע הדרוש להם.

שאלה מס. 5

א. סביבת העבודה בה פועל המחשב היא **אדון-עבד**. הסיבה לכך נובעת מזה שהמחשב שולטת במכונה המבצעת מספר תהליכים שונים תוך כדי ייצור חלקי אלומיניום. למכונה אין אפשרות להשפיע על המחשב המתפעל אותה והיא בעצם נחשבת ל"העבד" שלו. המושג מערכת מבוססת **אדון-עבד** בנויה על העיקרון שבו מחשב אדון מתפעל מערכת אחת או יותר שהן נחשבות עבד. המודל הזה נפוץ בתעשיית הטכנולוגיה, לא רק בתחום האלקטרוניקה אלא גם בתחומי המכניקה:

- **בטכנולוגיה אלקטרונית -** הוא משמש לעתים קרובות כדי לפשט את התקשורת כמו, במקום שיש ממשק נפרד לתקשר עם כל כונן דיסק, אנחנו יכולים לחבר את רובם באמצעות ממשק אחד וכבל ואת המחשב רק צריך לתקשר עם כונן אחד המשמש מאסטר, אז כל פקודה מלאה מופצת לעבדים מהמאסטר.
- **בטכנולוגיה מכנית -** המונח יכול להתייחס לתצורה של מנועים כגון שני מנועים המחוברים לכוננים שונים הפועלים על אותו עומס; כונן אחד מוגדר כאדון, עושה את המהירות ואת השליטה של העומס, ואילו העבד הוא שם כדי לעזור להגדיל את המומנט. במערכות פנאומטי הידראולי, יש לנו גם צילינדרים הורים, אשר יכולים להעביר את הלחץ ולשלוט על כמה צילינדרים המוגדרים בסטטוס העבד.

ב. המרכיבים החיוניים של מחשב זה הם:
1. מנגנון לניהול ובקרת פעולות קלט/פלט
2. מנגנון לטיפול והגנה בפני מקטעים קריטיים
3. מנגנון לטיפול בהתנגשויות
4. מנגנון לזימון תהליכים
5. מנגנון לטיפול בפסיקות
6. מנגנון לטיפול בתהליכונים
7. מנגנון לניהול התקשרות וממשק גראפי למשתמש
8. מנגנון לניהול הזיכרון והזיכרון הווירטואלי
9. מנגנון לניהול הליבה, המעבד ויחידת העיבוד המרכזית – CPU
10. מנגנון לניהול מערכת הקבצים
11. מנגנון לניהול מצבי מערכת הפעלה ושיקום מקריסה
12. מנגנון לניהול רשת תקשורת מחשבים וניהול תקשורת בין תהליכים

ג. אמצעי הגיבוי המומלצים במקרה זה נחלקים לפי דעתי לשניים:
1. תהליך גיבוי מקומי שיבצע גיבוי של פעולות המחשב מידי כמה דקות. הגיבוי הזה צריך לשמור הן במחשב והן במקום אוסף את כל הפקודות אותן נתן המחשב למכונה ומה היו תוצאות אותן פקודות. בנוסף יש לגבות מידי סוף יום את שנעשה במהלך אותו יום במסגרת פקודות הביצוע של המחשב למכונה.
2. תהליך גיבוי מקומי המבצע גיבוי של פעולות המכונה מידי כמה דקות לצורכי ניטור, מעקב, בדיקה ובקרת איכות מתמשכת. תהליך גיבוי זה צריך להישמר לפחות בשני מקומות, הראשון במחשב המפעיל והשני במקום מרוחק נוסף. בגיבוי זה יש צורך גם להוסיף צילום או תיעוד אופטי אחר שייתן תמונה לכל הפעילויות אותן עשתה המכונה בהתאם לפקודות המחשב.

הצורך בגיבויים אלה נדרש מכמה סיבות והן: לאפשר שחזור של ייצור מוצלח של רכיב מסוים הן בהיבט התכנותי והן בהיבט המכני. לאפשר בקרת איכות למקרים בהם ייצרו ע"י המכונה מוצרים לא שמישים למרות שהמחשב נתן את אותה סדרת פקודות למכונה. לצורך למידת לקחים בנושא חומרים מתכלים שהוזרמו למכונה ואולי לא היו באיכות טובה, לצורך בדיקות על הרס של רכיבי האלומיניום שנותרו והתאמתם לתקנים מסוימים. כל הצורך בגיבוי מקבל חיזוק מכך שהרכיבים המיוצרים הם חלק ממערכות חיוניות ולכן חשוב לתעד, לגבות, לצלם, להקליט וליצור כל גיבוי אפשרי שייתן מענה לצרכים עתידיים והכל כדי שייצור החלקים למערכות החיוניות לא יפגע או שלא יעמוד בסטנדרטים הנדרשים. אם יהיה מבדק בקרת איכות נוכל להראות לבודק את התיעוד של התהליך כל לוודא שהמוצר עומד בתקן.

שאלה מס. 6

א. ככל שמסד הנתונים גדל ונפחו עולה הניהול שלו מתחיל להיות בעייתי. יש צורך לנייד יותר נתונים, יש צורך לאחות חלקי קבצים שהופרדו ונמצאים בכתובות מרוחקות. יש פגיעה בזמני התגובה של המערכת, יש האטה של יכולת העיבוד של הנתונים כי להביא אותם מהמסד נתונים היושב על הדיסק אל הזיכרון הפיזי ולכתוב אותם בחזרה לוקח זמן רב. ניצול משאבי המחשב הופך ללא יעיל. ככל שמטפלים ביותר נתונים יש שחיקה רבה ויש סכנה לקריסה של הדיסק בגלל תנועה רבה של ראש קורא כותב.
ככל שמסד הנתונים גדול יותר העבודה על המחשב נעשית מסורבלת, איטית, לא יציבה, בעלת שרידות נמוכה ויכולת שיקום נמוכה. כל התוכנות יעלו לאט יותר, מעבר בין תכניות לוקח זמן רב, טעינת מנגנונים והפעלת תהליכים של מערכת ההפעלה נעשית איטית ולא

תמיד מתבצעת בהצלחה. פתיחה ,עדכון ,שמירה והעתקת קבצים יבוצעו בקצב איטי. תופעות כמו קריסה של מערכת ההפעלה צפויות בכל רגע נתון.

ב. בכל מערכת יש אפשרות לבדוק בכל רגע נתון את כמות הנתונים הנמצאים במערכת, הן באחסון, הן בריצה, הן בקבצים זמניים ועוד. כאשר מגלים כי יש האטה בקצב עבודת המערכת, כאשר פוגשים בכל הבעיות שהוזכרו בסעיף א' לשאלה זו יש צורך ראשוני לבצע ניקוי של בסיס הנתונים.

יש למחוק את כל הנתונים שאין בהם צורך, יש להעתיק נתונים שאין בהם צורך עכשווי אבל יש צורך לשמור אותם כהיסטוריה לצורכי תיעוד לדיסק חיצוני מהמערכת, יש למחוק אותם אחר כך מבסיס הנתונים, יש לבצע טיוב נתונים ע"י בדיקת הנתונים מציאת מי מהם נחוץ והכרחי כי הוא יישאר במסד הנתונים ומי אינו חיוני ולמחוק אותו, יש לבצע איחוי של מסד הנתונים כדי להביא את הקבצים למצב בו יהיו רשומים באופן רציף ככל שניתן במסד הנתונים, יש לבדוק אפשרות הגדלת מסד הנתונים או החלפת הרכיבים הטכניים שלו כמו מעבר מדיסק רגיל ל- SSD.

בנוסף יש לעשות את הצעדים הבאים כדי לטפל בבעיה וכדי לשפר את הביצועים:

- איחוי דיסק – שהקבצים יהיו רציפים אחד אחרי השני, צמצום פרגמנטציה ואיחוי הדיסק בזמנים קצובים מראש.
- בדיקה האם יש תוכנות מיותרות שמעמיסות על מע"ה.
- ביצוע גיבוי הנתונים לכונן חיצוני ופרמוט מחדש של הדיסק, החזרת הנתונים מהגיבוי חזרה לדיסק לאחר הפרמוט והתקנת מערכת ההפעלה מחדש כולל כל ההגדרות.
- ביצוע פרמוט הדיסק של מסד הנתונים לאחר פרקי זמן מסוימים וגיבוי קבוע של הנתונים בכונן חיצוני.
- הפעלת היישום לניקוי דיסק בפרקי זמן קצובים מראש.
- הפעלת היישום לסריקת הדיסק לבדיקת שגיאות טכניות.
- חיפוש אחרי עדכונים של גרסת מערכת ההפעלה הקיימת במחשב.
- מחיקת קבצים ישנים, תמונות, סרטים, שירים, משחקים ותוכנות ומחיקת תכניות ישנות ויישומים לא רלוונטיים.
- ניקוי המחשב מקבצים זמניים ומתכניות והתקנות ישנות שאינן נחוצות לתפעול היום יומי.
- שחזור מערכת הפעלה למועד קודם.

ג. המנגנון במערכת ההפעלה אשר במסגרתו נבצע ניהול של העברת הנתונים מהזיכרון אל יחידות אחסון נוספות כדי לפזר את בסיס הנתונים כך שכל פעם שנוצר צורך לבצע שינוי במסדי הנתונים השינוי יועבר לשתי יחידות האחסון במקביל או יותר(RAID). הוא מומלץ ביותר כי הוא ישפר את שרידות ואמינות הנתונים. במידה וגם נוסף מנגנון נוסף שתפקידו יהיה לטפל בחומרה במקביל לטיפול בתוכנה כלומר יטפל ביחידות האחסון הוא נקרא RAID תפקידו יהיה להפעיל את הכונן (יחידת אחסון הנתונים) החילופית במידה ויחידת האחסון הראשית קרסה מצב המערכת ישתפר לאין ערוך.

היתרונות של RAID:

ביצועים, גמישות ועלות הם בין היתרונות העיקריים של RAID. על ידי הצבת מספר כוננים קשיחים יחד, RAID יכול לשפר את העבודה של כונן קשיח יחיד, בהתאם לאופן שבו הוא מוגדר, יכול להגדיל את מהירות המחשב ואמינות לאחר התרסקות. עם RAID 0, הקבצים מתחלקים ומופצים על פני כוננים שפועלים יחד באותו קובץ. ככזה, קורא וכותב ניתן לבצע

מהר יותר מאשר עם כונן אחד. פונקציה זו מאפשרת ל- RAID לספק זמינות מוגברת. עם שיקוף, מערכי נתונים יכולים להיות בשני כוננים ולהכיל את אותם נתונים, אבטחת המשכיות העבודה של האחד במידה והשני נכשל. RAID עדיין יכול לגרום לעלויות נמוכות יותר באמצעות דיסקים במחיר נמוך יותר במספרים גדולים.

שאלה מס. 7

א. לא, למרות ששתי העבודות הן בגודל דומה, בשיטת RR מוקצה לכל אחת מהן פרק זמן עיבוד זהה במעבד. כלומר זמן העיבוד שלהן במעבד יהיה דומה אבל בגלל שעבודה אחת יש לה יותר גישות לקבצים, היא "תבזבז" זמן עבור הגישות לקבצים.

ב. לפי דעתי ניתן היה במקרה זה להפעיל מנגנון תזמון אחר, כמו SJF, אשר נותן עדיפות לעבודת קטנה יותר כך שהעבודה הקטנה תקבל עדיפות גבוהה יותר ותסתיים לפני השנייה. כך העבודה השנייה שיש לה ריבוי גישות תוכל לבצע את משימותיה אחרי סיום העבודה ה"קצרה" יותר.

ג. לשינוי פלח הזמן יש מספר משמעויות:

- ב- RR קיימים מספר תורים לפי סדר עדיפות כאשר תור גבוה מוקצה לתהליכים בעלי עדיפות גבוהה יותר והתורים הנמוכים מקבלים אחוז קטן יותר של זמן מעבד לפני שמפסיקים אותו. כל תהליך מתחיל בתור הגבוה ביותר, ובסוף ה- quantum הוא יורד לתור נמוך יותר. הדבר יוצר אפליה מתקנת של תהליכים עתירי קלט/פלט לעומת תהליכים עתירי חישוב (אם תהליך משחרר את המעבד לפני תום ה- quantum הוא עולה לתור גבוה יותר). הדבר יותר עדיפות דינמית לפי השימוש במעבד, כך שככל שממתינים יותר ,העדיפות גדלה וככל שזמן הריצה במעבד גדול יותר – העדיפות קטנה.
- אם נניח שזמן הריצה הממוצע שכל חוט מקבל בשנייה הוא c וזמן המעבד הממוצע שכל תהליך מקבל בשנייה הוא p , נסיק כי אין טעם לייצר מ c/p-חוטים. כמות כזו מבטיחה ניצול מלא של זמן המעבד המוקצה לתהליך, תוספת חוטים רק תגדיל את התקורה של החלפת ההקשר ואת הזמן המבוזבז על פעולת סנכרון, ואם נגיע למצב של עודף חוטים, נגרום לכך ששטח המחסנית של כל חוט יקטן. ב- pooling threadאנו מייצרים רק כמות סופית של חוטים. החוט הראשי הינו המנהל ובמקום ליצור חוט עבודה עבור כל בקשה, הוא יאכסן את הבקשות המגיעות בתור. ב- FIFOכל חוט אחר הינו חוט עובד והוא מוציא מהתור משימה ומבצע אותה .גודל ה- pool הינו דינמי, כאשר אם הגודל לא מספיק, כלומר אורך תור המשימות גדל והתהליך לא מספיק לנצל את כל ה-quantum שלו, אז מגדילים ב- k נוספים עד לסף עליון קבוע. באותו אופן, אם במשך הרבה זמן יש חוטים מובטלים, מבטלים k מהם.

שאלה מס. 8

FAT32 החליפה את קודמתה ה- FAT16 והיא נחשבת למערכת הקבצים הוותיקה יותר מבין השלוש. את ה- FAT32 ניתן היה לראות בפעם הראשונה בשימוש במערכת ההפעלה ווינדוס 95. ל- FAT32 יש מספר מגבלות, כאשר העיקרית שבהן היא שלא ניתן להעתיק אליה קבצים שהם מעל לגודל של GB4. ה- FAT32 נמצאת בדרך כלל בשימוש בכרטיסי זיכרון או דיסק און

קי, ולא תוכלו להשתמש בה כמערכת הקבצים למערכות הפעלה מסוג Windows 8 או Windows 10.

NTFS הופיעה לראשונה ב- Windows XP והיא נחשבת כיום למערכת הקבצים החדשה והטובה ביותר. על NTFS מומלץ להתקין מערכות הפעלה חדשות, כמו: Windows 7, Windows 8 או Windows 10. אם מותקן לכם במחשב עוד כונן קשיח אשר עליו אתם מעוניינים לאחסן קבצים, מומלץ מאוד לפרמט אותו כ- NTFS. NTFS תומכת בשלל מאפיינים אשר חשובים מאוד לכונן הקשיח ולמערכת ההפעלה, כמו: הרשאות קבצים, תיקון מהיר של שגיאות במקרים של תקלה במחשב, הצפנת קבצים ועוד. היתרון החשוב ביותר שיש ל- NTFS על פני קודמותיה הוא היכולת של המשתמש להעתיק אליה קבצים שהם מעל לגודל של 4GB.

<u>השוואה בין FAT32 ו- NTFS</u>

ניתן לראות שמערכת הקבצים FAT32 מותאמת יותר לכוננים שהנפח שלהם לא עולה על GB32, אך החיסרון הבולט שלה הוא שלא ניתן להעתיק אליה קבצים שהם מעל לגודל של GB4. לעומת זאת, ה- NTFS מותאמת במיוחד למערכות הפעלה חדשות בזכות שלל המאפיינים המתקדמים שיש לה, וגם מומלץ להשתמש בה בכוננים קשיחים שהם מעל ל- GB32. דרך אגב, ה- NTFS אינה מומלצת לזיכרונות ניידים שהם מעל GB32.

א. ניתן להתקין באותו מחשב שני דיסקים המפורמטים ומנוהלים כל אחד בשיטה אחרת. מערכות ההפעלה כיום יודעות לטפל בכל דיסק המותקן בכל תצורה של פרמוט ובכל סוג דיסק, רגיל או ssd. שיטת הפרמוט אינה מפריעה למערכת ההפעלה לנהל את הקבצים הנמצאים בכל דיסק בנפרד והיא יודעת כיצד לתפעל את שתי השיטות במקביל.

ב. לפי דעתי הגישה לקבצים בדיסק החדש תהיה מהירה יותר מכמה סיבות הנובעות מהיות הדיסק חדש, מהיר, לא שחוק, לא עמוס, כל טבלאות הניהול של הקבצים עליו מעודכנות וכמובן עצם השימוש בשיטת ניהול NTFS תוסיף לכך כי בשיטה זו מחזיקים בכל רגע נתון 3 אינדקסים המציינים איפה נמצא הקובץ. הבלוק הראשון מחולק לקבוצות של 4 בתים, כאשר כל 4 בתים כאלה הם הפנייה לבלוק מסוים בדיסק. כלומר, הבלוק הראשון מצביע לכל יתר הבלוקים וזה מאד שימושי ומהיר.
יתרונות: תמיכה בגישה ישירה, כאשר הדבר לא היה ניתן בהקצאה משורשרת כאשר אין FAT.
חסרונות: בזבזני עבור קבצים קטנים, כי עדיין יש צורך בבלוק לטבלה.
ניגשים לכל קובץ לפחות פעמיים – קודם לטבלה ואח"כ לנתונים.
מה קורה אם הקובץ גדול והטבלה לא מספיקה? אזי משתמשים באינדקסים בשתי רמות המכילים בלוקים של 4K, כל בלוק מצביע ל- K כניסות וכל כניסה מצביעה גם היא לבלוק של 4K נוסף. ניתן גם למצוא פתרון נוסף כאשר הכתובת האחרונה בבלוק הראשון תצביע לבלוק אינדקסים נוסף, וכעת נקבל קובץ של 8MB פתרון נוסף – הבלוק הראשון מפנה ל- K בלוקים שכל אחד מהם הוא בלוק אינדקסים , כלומר מפנה ל- K בלוקים אחרים. גודל מקסימאלי 4G.

ג. נפח האחסון בשני הדיסקים יהיה שונה כי הארגון הפנימי של הנתונים וטבלאות הכתובות של כל דיסק הוא שונה בגלל שיטת בפרמוט. לכן בניהול של הדיסק עם FAT32 יהיה נפח אחסון קטן יותר פנוי בגלל צורת הניהול ואילו בשיטת NTFS השטח הפנוי יהיה גדול יותר בגלל ניהול יותר יעיל של המקומות הפנויים בדיסק.

ד. כן ואף רצוי מאד לעשות זאת כך מערכת ההפעלה תוכל לנהל באופן יעיל יותר כל אחד מהדיסקים וכמובן תוכל להתייחס אליהם כאילו הם יחידת אחסון אחת וכך לייצר במידה הצורך זיכרון ווירטואלי גדול יותר בעל כמות כתובות המשכיות גדולה יותר.

בטבלה הבאה ניתן לראות את כל היתרונות והחסרונות של כל אחת ממערכות הקבצים הפופולאריות.

	FAT32	NTFS
קריאה וכתיבה של קבצים בגודל של מעל GB4	x	✓
יצירת כוננים (מחיצות) גדולים מ- GB32	x	✓
ניהול חכם של מקום פנוי (אין יותר מידי צורך בפעולת איחוי דיסק)	x	✓
דחיסת קבצים וחיסכון בשטח דיסק	x	✓
הרשאות משתמשים לקבצים ותיקיות	x	✓
הצפנת קבצים	x	✓
מותאמת לשימוש עם רוב מערכות ההפעלה	✓	x
תופסת פחות מקום בזיכרונות ניידים (דיסק און קי)	✓	x

<u>שאלה מס. 9</u>

א. בהחלט יתכן כי תתקיים בעיית ריסוק וחוסר רציפות של קבצים. הדבר מתרחש כיוון ישנן שתי יחידות הקצאת קבצים המוגדרות ב- UFS שהן: גודל בלוק הידוע גם בשם גודל בלוק לוגי ופרגמנט. בארכיטקטורה Sun4u נקבע כי גודל בלוק UFS הוא 8KB וגודל השברור כברירת המחדל הוא 1KB. ניתן להגדיר את השבר כחלק הקטן יותר שניתן לטפל בו ב- UFS. כאשר נוצר קובץ גדול מ - 8KB ה- UFS מקצה 8KB בלוקים וכל החלק שנותר הבלוק מוקצה כשבר אחד או יותר בתוך הבלוק האחרון. הדבר נכון לגבי קובץ שגודלו קטן מ - 96KB, קבצים ללא שברים יגיעו בעת יצירתם לגודל של לא יותר מ - 96KB כלומר, רק בלוקים מלאים באים לידי שימוש. מערכת ה- UFS תקצה קטע אחד או יותר מאותו קובץ לבלוק מערכת לוגי אחד עבור קבצים בגודל של פחות מ -8KB. עם זאת תמיד קיימת הגבלה כי UFS לא מאפשרת שחלקים של הקובץ יהיו מפוזרים על פני בלוק אחד או יותר של מערכת ההפעלה.
בכל זאת, אותו בלוק חוסם של מערכת ההפעלה עשוי להכיל שברי קבצים שונים, למשל 5 מתוך file1, 2 מתוך קובץ 3 וכו' בהנחה שגודלם אינו עולה על 1KB לכל מקטע. כאשר נעשה במערכת הקבצים שימוש רב, בלוקים רבים יכולים להיות מלאים חלקית עם שברים שהוקצו. לכן, נוצר מצב בו מוגבר הפיצול לשברי קבצים ואז אנחנו מקבלים מה שאנו מכנים מערכת קבצים מקוטעת הדורשת איחוי.
במקרים קיצוניים מאד ידוע כי מערכות מבוססות UFS עשויות להיות מקוטעות כל כך, עד כי הן לא יכולות להקצות בלוק שלם אחד לקובץ חדש למרות שיש בפועל הרבה שטח מקוטע.

ב. זמן הגישה לכל קובץ במערכת כזו תלוי לא רק בגודלו של הקובץ אלא גם במיקומו הפיזי שלו בדיסק באיזו מחיצה וכמה היא רחוקה ממחיצת השורש. גישה לקובץ גדול תדרוש במערכת כזו המבוססת על inode תיקח 5 גישות, 3 מתוכן לבלוקי המצביעים, בלוק נתונים, ולקובץ קטן רק 2 .

ה inode-נשמר תמיד בזכרון עבור כל הקבצים פתוחים. לרוב inodes נשמרים בנפרד מנתונים באזור מסוים בדיסק. כך בהבאת inode מובאים מצביעים נוספים ומתקיימת האצת פתיחת קבצים וסריקת מדריכים. מצד שני יתכן בזבוז מקום – כי לעיתים אין מספיק מקום בזיכרון עבור ה- inode-אך יש מספיק מקום לנתונים, או אין מקום לנתונים פרט באזור זה. מערכות מתקדמות מקצות מקום דינאמי ל- inode לפי צורך.

ג. בנושא שמות קבצים, לכל קובץ במערכות של יוניקס ובכלל במערכות של מחשבים מתקדמים שם הקובץ אינו בודד אלא כולל הכוונה לנתונים רבים הנותנים מידע הקובץ.

שמות קבצים: שם קובץ הוא שם שהוקצה לקובץ כדי להבטיח מקום אחסון בזיכרון המחשב והוא מאפשר גישה חוזרת למקום זה. בין אם מערכת הקבצים משתמשת בהתקן אחסון ובין אם לאו, בדרך כלל יש במערכות קבצים ספריות ששמות הקבצים מקשרים ביניהם לבין הקבצים, לרוב על ידי חיבור שם הקובץ לאינדקס בטבלת הקצאת קובץ כלשהו, כגון FAT בתוך מערכת הקבצים של DOS, או inode בתוך מערכת קבצים של Unix . מבנה הספריות עשוי להיות שטוח, או לאפשר היררכיות שבהן ספריות עשויות להכיל תת-ספריות. בכמה מערכות קבצים, שמות הקבצים הם בעלי תחביר מיוחד עבור סיומות ומספרי גרסה. במקרים אחרים, שמות הקבצים הן מחרוזות פשוטות, וכל המטא מידע של הקובץ מאוחסן במקום אחר. נתונים נוספים: המידע שנשמר עבור כל קובץ תלוי במערכת הקבצים הספציפית (למשל מערכות הפעלה רבות בתחילת ימי המחשוב לא עקבו אחר הזמנים הקשורים לקובץ) אולם לרוב מכיל את הפריטים הבאים:

- שם הקובץ (כולל גם את הנתיב)
- גודל הנתונים השמורים בקובץ - עשוי להיות מאוחסן כמספר בלוקים שהוקצו עבור הקובץ או כגודל הקובץ המדויק
- מזהה הקובץ
- מיקום פיזי שבו נמצא הגוש הראשון של המידע בקובץ (למשל בדיסק קשיח מדובר על מספר הדיסק הפיזי ועל מספר הסקטור)
- זמן השינוי האחרון של הקובץ, לרוב כחותמת הזמן של הקובץ
- זמן יצירת הקובץ
- זמן הגישה האחרונה לקובץ
- הזמן בו מטא המידע של הקובץ שונה
- סוג הקובץ (לדוגמה, התקן בלוקים, התקן תווי ,socket, ספריית משנה, וכו')
- זיהוי של בעל הקובץ והקבוצה אליה הוא משתייך
- הגדרות הגישה (למשל, אם הקובץ הוא לקריאה בלבד, ניתן להפעלה, וכו')
- הרשאות
- במחשבים גדולים מקובל לשמור גם את הפרטים הבאים:
- טיפוס הקובץ - סדרתי, אינדקס סדרתי, גישה ישירה
- סוג השדות וגודלם
- השדות המשמשים לאינדקס

במערכות קבצים מתקדמות (כגון XFS, ext2-4, כמה גרסאות של UFS, וב־ (+HFSניתן להוסיף גם פריטי מידע שרירותיים, באמצעות שינוי של מאפייני קובץ כאלו. תכונה זו מיושמת בליבות של Linux, FreeBSD ו־ Mac OS X, והיא מאפשרת למידע על המידע (באנגלית : (Metadata)להיות משויך לקובץ ברמת מערכת הקבצים. מאפיינים אלו יכולים, למשל, להכיל את שם המחבר של מסמך, את קידוד התווים של מסמך טקסט רגיל, או סיכום ביקורת .מאפייני המסמך בחלונות לעומת זאת מאוחסנים בתור-stream ים נפרדים, כלומר כחלק מהקובץ עצמו.

ד. נפילת מתח במערכת UFS אינה יכולה לגרום נזק רב כי בשיטה זו של ניהול הקבצים מה שעשוי להיפגע זה רק מצביע המסמן את מי הבאנו ועל מי עובדים. באופן עקרוני כיוון

שמערכת הפעלה מבוססת UFS מורכבת מהרכיבים הבאים הגורמים לכך שבמידה ויש קריסה הנזק לקבצים הוא מזערי.
בכל UFS יש אוסף של כמה בלוקים בתחילת המחיצה אשר הם שמורים עבור בלוקי האתחול (אשר חייבים להיות מאותחלים בנפרד ממערכת הקבצים) בהמשך יש בלוק-על, המכיל מספר המזהה אותו כמערכת קבצים של UFS, וכמה מספרים חיוניים אחרים המתארים את הגיאומטריה והסטטיסטיקה של מערכת הקבצים ופרמטרים של כוונון התנהגותי ובסיום נמצא אוסף של קבוצות הצילינדר.
לכל קבוצת צילינדר יש את הרכיבים הבאים:

- עותק גיבוי של המחסום
- ראש קבוצת צילינדר, עם נתונים סטטיסטיים, רשימות חינם וכו ', על קבוצת צילינדר זו, הדומה לאלו שבבלוק.
- מספר inodes, שכל אחד מהם מכיל את התכונות הנגררות של קובץ.
- מספר הבלוקים הנתונים במערכת.
- Inodes ממוספרים ברצף, החל מ- 0. Inode 0 ששמור עבור ערכי ספריות שלא הוקצו, inode 1 היה inode של קובץ הבלוק הפגום בגרסאות UNIX היסטוריות. אחריו inode עבור ספריית השורש, כאשר תמיד inode 2 ו- inode מוגדרים ושמורים עבורם. בהמשך תאוחסן הספרייה בה מאוחסנים הקבצים שהלכו לאיבוד אך נמצאו ב- inode 3.
- קבצי Directory מכילים רק את רשימת שמות הקבצים בספריה ואת inode המשויך לכל קובץ.
- כל המטא נתונים של הקבצים נשמרים ב- inode.

<u>שאלה מס. 10</u>

א. תוצאת ריצת ה-script בשם game 7 – תודפס ההודעה:
Sorry, incorrect.

ב. תוצאת ריצת ה-script בשם game 5 – תודפס ההודעה:
you got the answer correct!

ג. תוצאת ריצת ה-script בשם game 0 : תודפס ההודעה:
The number must be between 1 and 10 !

ד. ה-script מבצע קלט אל מספר ואז בודק האם הוא בתחום של גדול מ-1 או קטן מ-10. במידה וכן בודק האם המספר שנקלט שווה לערכו של המשתנה num שהוא 5, אם כן תודפס ההודעה: you got the answer correct אחרת תודפס ההודעה:
Sorry, incorrect.