**ממ"ן 13 – מערכות הפעלה**

**מגיש: רועי אבידב**

**ת.ז: 209729185**

חלק עיוני

2)

א.

היתרון של מדיניות LIFO הוא שהבקשה האחרונה מטופלת ראשונה, ותהליכים קרובים, כאלו שהשתמשו בהם לאחרונה ויש יותר סיכוי שהם רלוונטיים, דברים שהמשתמש יותר צריך ברגע זה, יטופלו בצורה מהירה יותר. הדבר אומר שתהליכים אלה ככל הנראה יכתבו או יקבלו את המידע שהם צריכים בצורה מהירה יותר.

בנוסף, תהליכים סדרתיים – כאלה ששולחים בקשות רבות אחת אחרי השנייה, כאשר הן מטופלות, בעזרת שימוש במדיניות LIFO, הסיכוי לקרבה פיזית של הבלוק הבא לקריאה גבוה מאוד, מה שבאופן ממוצע ישפר את ניצול הדיסק מבחינת זמנים.

מש"ל 2א.

ב.

החיסרון של מדיניות LIFO הוא שבקשות "רחוקות" יכולות לרעוב, כלומר אם ישנם תהליכים רבים המשתמשים בדיסק בצורה מתמשכת, תהליכים ישנים יותר עלולים לא לקבל או לכתוב את המידע שהם צריכים למשך זמן רב מאוד, מכיוון שהמחסנית תמשיך להתמלא, והתחתית שלה לא תתבצע לעולם.

מש"ל 2ב.

3)

הקצאה רציפה

ראשית, עבור הבחירה במדיניות של הקצאה רציפה, נשים לב כי גודל הקובץ הנתון בכל רגע יכול להשתנות בהפרשים משמעותיים . הדבר עלול לגרום לריסוק חיצוני בכמות גבוהה – ולבעיה באחסון קבצים חדשים.

נעיר כי אם הקבצים היו בגודל קבוע, ולא משתנה, היינו בוחרים בשיטה זו, מכיוון שהיא מוודאת כי אין ריסוק פנימי.

רשימה מקושרת

שנית, עבור הבחירה במדיניות של רשימה מקושרת, מכיוון שגודל קובץ נע בין 4K ל4Mb, נבחר כי גודל הבלוק המקסימלי שלנו יהיה 4K, על מנת לצמצם את מספר הקריאות ברשימה המקושרת. במידה והרשימה המקושרת מנוהלת על ידי טבלת FAT, והיא שמורה בזיכרון, הדבר יתפוס כמות גדולה מאוד של זיכרון. במידה ולא, והרשימה שמורה על הדיסק עצמו, ובמקרה הגרוע ביותר, בו גודל הבלוק הוא מינימלי, נצטרך לעבור 1024 בלוקים על מנת להגיע אל האיבר האחרון ברשימה המקושרת – שעבור כל אחת מהן נצטרך לקרוא לבלוק המידע שלה. מדובר ב-1024 קריאות נוספות, סך הכל 2048 קריאות.

i-node

בשיטה האחרונה, שיטת ה-i-node, בהנחה שגודלו של מצביע הוא 4 בתים – נקבל כי I-node יחיד יכול להחזיק קובץ שלם בדיוק בתוך בלוק אחד – מכיוון שנעדיף לבחור את גודל ה-i-node שלנו כגודל בלוק.

נראה זאת:

נבחר את גודל הקובץ המינימלי להיות 4Kb, כלומר ישנם מצביעים בקובץ. נוכל להכניס את כל אלה ברמה הראשונה של i-node, מכיוון שבקובץ בגודל מקסימלי של 4Mb נכנסים בדיוק 1024 קבצים בגודל 4Kb, כפי שחישבנו למעלה.

מצב זה הוא אופטימלי, מכיוון שאנחנו נצטרך רק קריאה יחידה מהדיסק (מעבר לקריאות של מידע).

ולכן, במקרה זה, הייתי בוחר במדיניות ה-i-node

4)

א.

מודל ה-Sandboxing המתואר דומה לקונטיינרים הנידונים בחלק המעשי של מטלות הקורס בכמה דברים. ראשית, שניהם סוגים של וירטואליזציות. בנוסף, שניהם יוצרים מודל של חלוקה, כלומר "אריזה" של משאבים בסביבות הזרות אחת לשנייה, משיקולי אבטחה, יעילות, וכו'.

עם זאת, בין שני המודלים ישנם גם הבדלים רבים. ראשית, Sandbox הוא עבור אפליקציה מסויימת, ולעומת זאת Container של לינוקס הוא עבור המערכת כולה, ומופרד אך ורק עבור יוזרים שונים. בנוסף, Sandbox דורש מאפליקציה כלשהי לבקש באופן ספציפי את כל הקשרים שלה עם אפליקציות אחרות, ונשמר תיעוד קפדני של גרף יחסים זה, ואילו עבור קונטיינרים הדבר לא נחוץ, מכיוון שאין גישה בין אפליקציות שונות בכלל.

מש"ל 4א.

ב.

מדובר בסוג בידוד פר אפליקציה, כלומר החיבור היחיד בין כל שתי אפליקציות הוא חיבור שהמשתמש אישר באופן ספציפי.

מש"ל 4ב.

ג.

היתרון בשיטת ניהול זיכרון Copy on Write שמערכת אנדרואיד משתמשת בה בשביל מימוש מודל התהליכים שלה נגזר מהעובדה שמשאב כלשהו יכול להיות משוכפל ולאחר מכן לא יצטרך שום שינוי נוסף בכדי להגיע למצב הרצוי שלו. הדבר יכול לחסוך שימוש רב במשאבים, עבור הוספת תקורה קטנה בלבד לפעולות המשנות תהליכים.

מש"ל 4ג.

5)

שיטת הוירטואליזציה Trap And Simulate עובדת כך:

באופן רגיל, כל מערכת הפעלה שהיא, מתוכננת כך שתהיה לה שליטה מלאה ומוחלטת על המערכת, והיא לא מצפה שיהיו מערכות הפעלה נוספות שפועלות במקביל אליה. הדבר גורם לבעיה, כאשר יש מצב שבו שתי מערכות ההפעלה מתנגשות בפעולתן, כלומר ששתי מערכות הפעלה שונות מבקשות בו זמנית את אותו המשאב, או רוצות לבצע את אותה הפעולה. במקרה זה, נכנסת לתוקף שיטת ה-Trap And Simulate, שאומרת כי במצב כזה, ה-Virtual Machine Manager "יתפוס" (Trap) את הפעולה הכפולה באחת ממערכות ההפעלה, ו"יסמלץ" (Simulate) אותה. לדוגמא, אם גם מערכת הפעלה של Windows וגם מערכת הפעלה של Unix מבקשות לצייר על המסך באותו הזמן ובאותו המקום, שמבחינתן זה בסדר, כי כל מערכת הפעלה לא יודעת שמערכת ההפעלה השנייה רצה, תיכנס לתוקף שיטת הוירטואליזציה הנ"ל, וה-VMM יקח את שתי הבקשות הנפרדות, ויהפוך אותן לבקשה יחידה, שתצייר את שני הדברים הרצויים על המסך אחד ליד השני, כך שלא יהיה ניתן להרגיש חוסר סנכרון בין מערכות ההפעלה.

שיטה זו עדיפה על binary translation, מכיוון ש-binary translation בא לפתור בעיה ספציפית, של פעולות שאותן המערכת לא יכולה "לתפוס", כי קיימות פעולות כאלה, ובמקרה זה, אם הוא מוצא בעיה, הוא כותב לה פתרון ידני בצורה דינמית ו"על המקום". בעבר, זה היה הפתרון היחיד לבעיה, אבל כיום, בעזרת trap and simulate, המעבד יכול לדעת שהוא משתמש במערכת הפעלה "אורחת", ושהוא יכול להגביל את פעולותיה, כדי להימנע ממצבים שאי אפשר לעשות להם trap.

מש"ל 5.