20594 מערכות הפעלה, ממן 13

טל גלנצמן 302800354

16 January 2021

תשובה 1

יתרון

פשוט למימוש. אין לוגיקה מיוחדת, כל שדרוש למימוש זו מחסנית.

חסרון

אלגוריתם גורם, באופן עקבי, להרעבה של בקשות. שכן כאשר קצב מימוש הבקשות נמוך מקצב הגעת בקשות, האלגוריתם לעולם לא יגיע לממש בקשות שהגיעו קודם.

תשובה 2

$RAID3$ מוסיף ביט זוגיות עבור מילה בניגוד ל־ $RAID2$ ששומר את קוד האמינג של המילה איתה.

לכן, ב־ $RAID3$ ניתן לזהות כאשר יש עיוות במידע אך לא ניתן לתקן אותו ללא אינפורמציה נוספת. אכן, כאשר אחד מהדיסקים תקול, נדע איזה ביט מעוות ולכן נוכל לבצע תיקון.

$RAID2$ למעשה נותן לנו את היכולת לבצע תיקון גם במקרים בהם אנו לא יודעים איזה ביט מעוות. למשל, כאשר לא זה המצב בו דיסק מסוים תקול, אלא ישנן שגיאות רנדומליות בקריאות.

תשובה 3

גודל הקובץ דינמי ונע בין טווח יחסית גדול של גדלים אפשריים. לכן לא כדאי לבחור בשיטת הקצאה רציפה שכן הפוטנציאל לריסוק חיצוני גדול.

נניח כעת כי אנו מחלקים דיסק בגודל D לבלוקים בגודל B.

נניח כי גודל מצביע לחוליה בשיטת FAT הוא a בתים. גודל טבלת ה־ FAT הוא כעת נניח כי אנו מחלקים את הדיסק לבלוקים בגודל x כעת נניח כי אנו מחלקים את הדיסק לבלוקים בגודל x

$$F = a \cdot \frac{D}{B}$$

ככל ש־ F קטן יותר, כך נעדיף את שיטת ה־ FAT ביחס ל־ $I-Node$ שכן מציאת בלוק רלוונטי הוא חיפוש בזיכרון ולא בדיסק.

שיקול נוסף הוא מהי רמת ההצבעה הנדרשת על מנת לייצג קובץ מקסימלי באמצעות שיטת ה־ $I-Node$. זה

תלוי בגודל מצביע, גודל בלוק ובמבנה ה $i-node$ עצמו. אם ניתן להסתפק בפחות רמות נטה את הבחירה יותר לטובת ה־ $I-Node$.

לסיכום, נאלץ לבחור בין FAT ל־ $I-Node$. הבחירה עדיין יחסית לגודל הדיסק והבלוקים כאשר גודל דיסק גדול ויחס בלוק לדיסק גבוה ייטה לבחירת $I-Node$. ככל שיותר ריאלי לשמור את טבלת ה־FAT בזכרון, כך ניטה לבחירת שיטה זו.

תשובה 4

$binary\ translation$ היא שיטה בה ה־ $hypervisor$ נוקט על מנת לבצע הוראות רגישות - כאשר הוראות רגישות אלו הוראות שלהן יש משמעות שונה בין אם במצב משתמש או מצב מיוחס.

השיטה היא למעשה, מנקודת מבט של ה־ $hypervisor$, להחליף בזמן ריצה את ההוראות הרגישות שהתקבלו ממערכת ההפעלה ברצף הוראות שונה, לא רגיש אך שקול מבחינת התוצאה.

השיטה קשה לביצוע בפרט שכן לא תמיד ברור מה הרצף להחליף בו פקודות.

בפרט, השיטה לא כ"כ יעילה שכן

- פעולת החלפת ההוראות בעצמה לוקחת משאבים
- הוראה אחת מתחלפת לרוב ברץ הוראות שונה, כאשר במקור אותה הוראה הייתה מינימלית בעצמה.

ישנן כמה חלופות.

ניתן לבצע אמולציה מלאה, כלומר לבצע את כל הפעולות בתוכנה - לא יעיל כלל, אף פחות מהשיטה הנידונה.

$Trap\ And\ Emulate$ היא שיטה **הנתמכת ע"י החומרה**, ובה כאשר יש הוראה רגישה המגיעה ממערכת ההפעלה החומרה מבצעת $trap$ ומעבירה את השליטה ל־ $hypervisor$ לבצע אמולציה. תמיכת החומרה מתבטאת גם בכך שהיא נותנת לשכבת וירטואליזציה לקנפג עבור אילו הוראות להעביר אליה את השליטה.

$Paravirtualization$ היא שיטה בה מערכת ההפעלה מודעת לכך שהיא עובדת מעל שכבת ורטואליזציה ומודיעה על פקודות רגישות ומעבירה שליטה ל־ $hypervisor$. זוהי שיטה יעילה שכן אין הוראות עודפות ומערכת ההפעלה היא בלאו הכי זו שיודעת מתי יש פקודות רגישות לביצוע. יתרון נוסף הוא שאין צורך בתמיכה חומרתית לכך. עם זאת החסרון ברור, מערכת הפעלה אידיאלית צריכה להכתב בצורה אגנוסטית לחומרה תחתיה, ממשית או וירטואלית.