גירסה 1.00 - 2.12.2002

Virtual Memory

מסמך זה הורד מהאתר http://underwar.livedns.co.il.
אין להפיץ מסמך זה במדיה כלשהי, ללא אישור מפורש מאת המחבר.
מחבר המסמך איננו אחראי לכל נזק, ישיר או עקיף, שיגרם עקב השימוש במידע המופיע במסמך, וכן לנכונות התוכן של הנושאים המופיעים במסמך. עם זאת, המחבר עשה את מירב המאמצים כדי לספק את המידע המדויק והמלא ביותר.

Nir Adar
Email: underwar@hotmail.com
Home Page: http://underwar.livedns.co.il

אנא שלחו תיקונים והערות אל המחבר.

נדרש ידע מוקדם בנושא cache על מנת להבין מסמך זה.

מושגים

במרחב התוכנית יש מרחב כתובות אחיד. התוכנית לא מתעניינת באיזה מקום פיזית המידע יושב.
התוכנית רואה מרחב זיכרון רציף והיא לא מתחשבת בשאר התוכניות במערכת.
כתובת פיסית היא הכתובת שאיתה אנו ממש ניגשים אל הזיכרון הראשי.
כתובת וירטואלית היא הכתובת שהתוכנה רואה.
אנו משתמשים במנגנון שממפה כתובות וירטואליות לכתובות פיסיות.

יתרונות השימוש ב-Virtual Memory

1. ניתן להריץ מספר תוכניות במקביל. התוכניות לא יתנגשו בזיכרון אחת של השניה.
2. גודל הזיכרון המוקצה לתוכנית יכול להיות גדול מגודל ה-RAM. נוכל לתמוך בזיכרון גדול מגודל ה-RAM, על ידי כך שחלק מהמידע יאוכסן בדיסק.
3. הזמן הלוקח לתוכנית להתחיל לרוץ מהרגע שביקשנו להריץ אותה משתפר משמעותית (זמן טעינת התוכנית).

ניתן להתייחס אל הזיכרון הראשי כאל Level 3 Cache.
נחליף מושגים בהם אנו משתמשים ב-cache במושגים חדשים מתחום ה-VM:
ב-cache דיברנו על בלוקים. המקביל של הזיכרון הראשי הינו עמודים (pages).
המקביל ל-miss ב-cache הינו page-fault.

שאלות עקרוניות לגבי הזיכרון הראשי

1. כיצד נמקם page בזיכרון הראשי?
2. איך נמצא את הכתובת הפיסית ואת המידע, בהינתן כתובת וירטואלית?
3. מהי מדיניות ההחלפה של הזיכרון הראשי?
4. איך נבצע את הכתיבות לזיכרון הראשי?

תשובות:

1. הזיכרון הראשי מנוהל ב-fully assoc.. הניהול של הזיכרון נעשה על ידי תוכנה וזאת מכיוון שאנו פונים יחסית מעט אל הזיכרון הראשי, ולכן מימוש בתוכנה הוא הגיוני ופשוט יחסית.
2. קיימת טבלה שהקלט שלה הוא כתובת וירטואלית והפלט שלה היא כתובת פיסית.
   נשים לב כי ה-offset של הכתובת כלל לא משתתף בתהליך המיפוי.
3. מדיניות ההחלפה בה אנו משתמשים היא מתוחכמת, למשל LRU – המדיניות ממומשת על ידי תוכנה.
4. חלק מהכתיבות לזיכרון יהיו WT וחלקן יהיה WB.

cache

כאשר אנו מוסיפים למערכת שלנו תמיכה בזיכרון וירטואלי מתעוררות שאלות חדשות לגבי ה-cache של המערכת.
אחת מהן: אם נשמור ב-cache של המחשב כתובות וירטואליות או כתובות פיסיות?
אם נניח כי נרצה להשתמש בכתובות פיזיות, נצטרך לתרגם את הכתובות המגיעות מהמעבד לכתובות פיזיות שישמרו ב-cache. תרגום מלא של כתובת כולל גישה אל הזיכרון הראשי אל הטבלה המכילה את מיפוין של הכתובות הוירטואליות אל הכתובות הפיסיות. תרגום זה לוקח זמן רב. על מנת לשפר את הביצועים, נוכל להשתמש בשיטת TLB (חוצץ תרגום). נשמור בחוצץ את התרגומים האחרונים שביצענו. לפני הגישה אל מנגנון התרגום המסובך ניגש אל ה-TLB ונבדוק האם התרגום נמצא כבר שם.

שימוש בכתובות וירטואליות ב-cache

היתרון בשימוש בכתובות וירטואליות ב-cache הוא שאין צורך לבצע תרגום של הכתובות לכתובות פיסיות, ולכן ה-hit time של ה-cache הוא טוב.
עם זאת, אם נשתמש בכתובות וירטואליות ב-cache יתעוררו מספר בעיות:

1. נניח כי אנו מריצים את תוכנית a ואנו ממלאים את כל ה-cache במידע הקשור אליה. כעת נריץ את תוכנית b והיא תניח כי כל המידע הנמצא ב-cache הוא שלה.
   פתרון אפשרי אחד לבעיה זו, הוא לרוקן את ה-cache (flush) בכל Task Switch. הבעייתיות: כל תוכנית שתתחיל לרוץ תרוץ למעשה מההתחלה.
   פתרון אפשרי שני הוא להחזיק בצמוד ל-tag גם שדה – process id.
   הבעייתיות בפתרון זה: במערכות מתקדמות תהליכים רבים, ולכן כל שדה process id יתפוס מספר רב של ביטים. בנוסף הוספת שדה נוסף ל-tag תייקר את ה-cache.
2. נניח כי לשני תהליכים גישה לאותה כתובת מסוימת בזיכרון (למשל לצרכי תקשורת בין שני התהליכים). יכול להיווצר מצב שב-cache ישנן שתי כתובות וירטואליות הממופות אל אותה הכתובת הפיסית. במקרה כזה אין מנגנון המבטיח שאם אחד מהתהליכים עדכן כתובות זו, התהליך השני יהיה מודע לעדכון. בעיה זו נקראת **בעיית ה-aliasing**.
   פתרון אפשרי לבעיית ה-aliasing הוא לעבוד עם direct mapped cache משום שב-cache כזה אנו יודעים לאיזו קומה הכתובת שלנו תמופה. נחליט שכל הבתים המשותפים למספר תהליכים ימופו לקומות מסוימות ב-cache. התוכנית תיגש לקומות שבהם יתכן כי יושבת כתובת משותפת לה ולעוד תהליך, ואם תמצא כתובת כזו היא תזרוק אותה מה-cache, כך שכאשר התהליך השני יכול לפעול יהיה לו miss והוא ישים לב אל השינוי.
   פתרון נוסף: הזיכרון המשותף לשני תהליכים לא יהיה כלל ב-cache וישמר רק בזיכרון הראשי. כלומר: תהליכים יתקשרו ביניהם רק דרך הזיכרון הראשי.

3. בעיה נוספת היא שאם נרצה כל פעם שאנו מחליפים תהליכים להחליף את כל הנתונים ב-cache, אם נניח כי חצי מה-cache הוא במצב dirty, אז אנו משתקים את ה-bus לזמן ארוך על מנת לעדכן את כל הזיכרון הראשי.

שימוש בכתובות פיסיות ב-cache

אם נשמור ב-cache כתובות פיזיות, נפתרות כל הבעיות שהצגנו ב-cache המכיל כתובות וירטואליות.
עם זאת, יכולה להתעורר בעיה חדשה:
נניח כי יש בלוק ב-cache הממופה אל דל מסוים בזיכרון הראשי. נניח כעת שאנו רוצים מידע מדף שלא נמצא בזיכרון. נביא את הדף מהדיסק ובמקומו נאלץ למחוק את אחד הדפים. אם נמחק את הדף אליו ממופה הבלוק מה-cache, ייווצר מצב בו הבלוק יחשוב שהדף הקיים בזיכרון הראשי הוא הדף שלו, בעוד שבפועל המצב יהיה שונה.
כדי למנוע בעיות מסוג זה, כאשר מתרחש page fault מתרחשת גם פסיקה. הפעולה של התוכנית שגרמה לפסיקה מופסקת, ונקראת שיגרת הפסיקה.
הפסיקה תבדוק לאיזה כתובת התוכנית רוצה לגשת ואם היא בכלל מורשית לגשת לכתובת זאת.
עדיפות הריצה של הפסיקה חזקה בהרבה מעדיפות הריצה של תוכנית המשתמש משום שהפסיקה היא תוכנית של מערכת ההפעלה והיא יכולה לגשת לחומרה.
כאשר הפסיקה מוצאת בלוקים ששיכים לדף שעומד להיזרק מהזיכרון הראשי (ולכן גם מה-cache) היא זורקת אותם מה-cache או מאפסת להם את סיביות ה-valid.

מבנה כתובת

ניתן לחלק את הכתובת לשתי חלוקות שונות: חלוקה הקשורה לדפים וחלוקה הקשורה אל ה-cache.

	31 12	11 0	
חלוקה לדפים	Page Address	Page Offset	
חלוקה של cache	Address Tag	Index	Block offset

Page Offset נקבע לפי גודל הדף והוא איננו עובר תרגום.
נבחר את גודל הדף כך שהרוחב של ה-page offset יהיה שווה לרוחב של index+block offset.
בצורה כזו ה-index לא עובר תרגום בכתובת וירטואלית.
כעת נרצה למקבל את התהליכים המתרחשים:
נהמר כי היה לנו hit ב-cache. בעזרת ה-index ניתן לגשת אל הסט המתאים ולקרוא את המידע, ובנתיים נשלח את כתובת העמוד אל ה-TLB לתרגום.
נשים לב כי אם היה hit ב-TLB כל שאנו יודעים זה כי הדף נמצא בזיכרון הראשי. עדיין לא מובטח לנו כי יהיה hit ב-cache.

כמו כן, אם היה miss ב-TLB כל שאנו יודעים שאין בידינו את הכתובת הפיסית המתאימה לכתובת הוירטואלית אותה אנו מחפשים. עדיין ייתכן כי יתקיים hit ב-cache.
אחרי ה-hit ב-TLB מקבלים ב-cache את כתובת הדף הפיזי על מנת לבדוק האם יש גם hit ב-cache. נצטרך להשוות tags בין מה שקיבלנו מה-TLB, כלומר בין הדף הפיזי, לבין ה-tag השמור ב-cache.

אם נסכם:

$$\text{Page Offset} \geq \text{index} + \text{block offset}$$

$$\text{Cache Size} = \text{Number of sets} \cdot \text{Assoc.} \cdot \text{Block Size}$$

$$\text{Number of sets} = \frac{\text{Cache Size}}{\text{Assoc.} \cdot \text{Block Size}}$$

$$\text{Page Size} \geq \frac{\text{Cache Size}}{\text{Assoc.}} \Rightarrow \text{Cache Size} \leq \text{Page Size} \cdot \text{Assoc.}$$

מסקנה: קיים קשר בין גודל הדף לבין תצורת ה-cache!
זהו קריטריון שאנו חייבים למלא אם אנו רוצים לעבודה בצורה כזו (הכתובות הפיסיות נמצאות ב-cache).

גודל דף

ככל שהדפים יהיו יותר גדולים, גודל טבלת הדפים מצטמצם.
יעיל יותר להביא דפים גדולים לזיכרון הראשי, מכיוון שאם כבר היה page fault עדיף כבר להעביר כמה שיותר מידע (בכל מקרה אנו משלמים על ההעברה).
עם זאת, אם נקבע שאנו משתמשים בדפים גדולים מתעוררת בעיה של נצילות.
יתכן למשל שגודל הדף שלנו הוא 4K, בעוד גודל הבלוק בו אנו משתמשים הוא בגודל K אחד בלבד. במקרה זה יהיו 3K מבוזבזים.
בעיה נוספת – כאשר תהליך מתחיל לרוץ הוא צריך לטעון הרבה מידע שלא בהכרח נדרש על ידי האפליקציה (לא כל המידע בדף רלוונטי לתהליך).
הפתרון הינו צירוף של סגמנטים ודפים. לדף גודל משתנה וסגמנט הוא אוסף דפים והוא משתנה.