

פרק 6

מערכות קבצים

שני מנגנונים

❖ מיפוי ממרחב שמות מדרגי קריא לבני אדם (מחרוזות) לעצמים של מערכת ההפעלה:

▪ `/usr/bin/ls`

▪ `/dev/lp0`

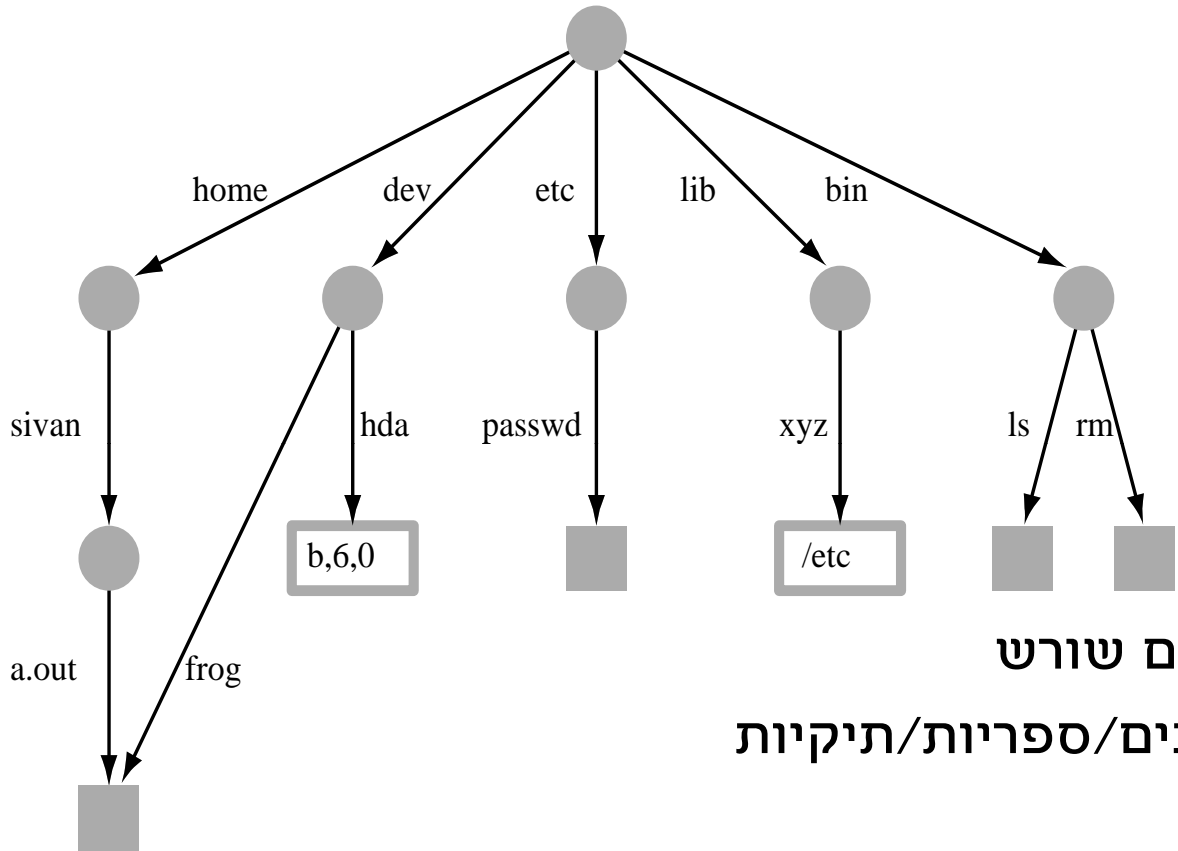
▪ `C:\winnt\system32\kernel32.dll`

❖ מנגנון לשמירת ואחזור קבצים על התקני זיכרון חיצוניים (בעיקר דיסקים, כרטיסי זיכרון, ותקליטורים)

▪ במערכות הפעלה מסוימות (מקינטוש, חלונות) קובץ יכול להיות מורכב ממספר רצפים; אנו נתעלם מאפשרות זו, שמייצגת בעיקרון סוג שונה מעט של מדריך.

מרחב השמות

מרחב השמות



- ❖ עץ או גרף חסר מעגלים עם שורש
- ❖ צמתים פנימיים הם מדריכים/ספריות/תיקיות (directories, folders)
- ❖ עלים הם קבצים או עצמים אחרים (התקנים, צינורות, וכו')
- ❖ מצביע סימבולי הוא עלה מסוג מיוחד; נדון בו בהמשך
- ❖ ביוניקס/לינוקס לקשתות יש שמות
- ❖ התייחסות לעצמים על ידי שרשור שמות הקשתות לאורך המסלול מהשורש

מרחב השמות בחלונות

- ❖ לעצמים עצמם יש שמות, לא לקשתות
- ❖ לעצם יכולים להיות מספר שמות נרדפים (למשל למימוש שמות של 8.3 תווים לקבצים)
- ❖ מרחב השמות כולל גם עצמים ששמורים במערכת הקבצים (כלומר בדיסק) וגם עצמים נדיפים כמו מנעולים ואירועים
- ❖ שמות קבצים במרחב השמות האחד הזה מתפענחים כאילו התחילו ב- "??"\" ועצמים נדיפים כאילו התחילו ב-
"BaseNamedObjects"

פעולות על מרחב השמות

- ❖ יצירת עלה מסוג מסוים ומצביע אליו ממדריך כלשהו `creat, open socket, mknod`
- ❖ יצירת מדריך ומצביע אליו `mkdir`
- ❖ יצירת מצביע נוסף לעצם קיים `link`
- ❖ יצירת מצביע סימבולי `symlink`
- ❖ מחיקת מצביע; העצם נמחק כאשר אין מסלול מהשורש אליו ואינו בשימוש על ידי תהליך `unlink`
- ❖ שינוי הרשאות גישה לעצם `chmod, chown`
- ❖ שתילת ועקירת מערכת קבצים שלמה בצומת מסוים `[u]mount`
- ❖ פענוח שם למזהה פנימי והודעה למערכת ההפעלה שהעצם בשימוש והודעה על סיום שימוש בעצם `open, close`
- ❖ קריאת תוכן מדריך או חיפוש במדריך `opendir, readdir`

פענוח שמות (open)

- ❖ פירוק שם למרכיבים שמופרדים ב- \ או ב- / ,
- ❖ למשל, `/usr/bin/ls`
- ❖ מעקב אחרי המסלול במרחב השמות שמוביל לעצם
- ❖ בכל שלב בפענוח מערכת ההפעלה מחפשת את המרכיב הנוכחי של השם במדרוך
- ❖ לכל תהליך נשמר מדרוך נוכחי (current working directory) שהוא התחלת המסלול אם השם אינו מתחיל ב- \ או ב- /
- ❖ מצביעים מיוחדים: `..` , `.` (המדרוך הזה וההורה שלו)

מצביעים סימבוליים

❖ עלים במרחב השמות שמכילים שם של עצם אחר

❖ כאשר מנגנון הפענוח מגיע למצביע סימבולי, הערך של המצביע

משורשר לסיומת השם המתפענת, והפענוח מתחיל מחדש:

▪ למשל, `/lib/xyz` הוא מצביע סימבולי ל-`/etc`

▪ כאשר מגיעים למצביע בפענוח של `/lib/xyz/passwd`, משרשרים

את תוכן המצביע לסיומת `passwd`

▪ הפענוח מתחיל מחדש עם השם המשורשר `/etc/passwd`

❖ ניתן ליצור מצביע סימבולי גם אם העצם המוצבע לא קיים, וניתן

למחוק את העצם המוצבע

נקודות הצבה

- ❖ מצביע מצומת במרחב השמות לשורש של מרחב שמות אחר
- ❖ מאפשר "לשתול" מערכת קבצים מדיסק, מחיצה, או מחשב מרוחק במרחב השמות
- ❖ כאשר תהליך הפענוח מגיע לנקודת הצבה, התהליך ממשיך עם הסיומת של השם מהשורש של המערכת המוצבת.
- ❖ ביוניקס/לינוקס: טבלה נדיפה במערכת ההפעלה
- ❖ בחלונות: עצם ששמור בדיסק (נקודת פענוח, reparse point)
- ❖ נקודת פענוח יכולה גם לגרום להפעלת שגרה שממשיכה את תהליך הפענוח

מחיקת קבצים ומעגלים

- ❖ אם מערכת ההפעלה הייתה מתירה ליצור מעגלים בגרף, קשה היה לאפיין ביעילות עצמים שלא ניתן להגיע אליהם מהשורש
- ❖ ניתן לסמן את כל העצמים שניתן להגיע אליהם ולמחוק את השאר, אז זהו תהליך יקר
- ❖ מערכות הפעלה מונעות יצירת מעגלים ומוחקות עצמים שמספר ההצבעות אליהם יורד ל-0; מכיון שאין מעגלים ואין עצמים שאין אליהם הצבעות, ניתן להגיע לכל עצם שיש אליו הצבעות
- ❖ דרך פשוטה למנוע היוצרות מעגלים: איסור על יותר ממצביע אחד למדריכים
- ❖ תוכניות לטיפול בקבצים צריכות להיזהר ממעגלים שכוללים מצביעים סימבוליים (למשל תוכניות גיבוי או תוכניות ליצירת אינדקסים)

שמירת ואחזור קבצים

סמנטיקה של גישה לקבצים

❖ קונסיסטנטיות בגישה ממספר תהליכים:

- כתיבות וקריאות הן אטומיות; לא רואים מידע חלקי
- כתיבה לקובץ נראית "מייד" לתהליכים אחרים; סדר את כל הכתיבות והקריאות בסדר שלם וקריאה תמיד רואה את הקובץ לאחר הכתיבה האחרונה; פעולות לא מתבצעות בצורה הדרגתית (sequential consistency)

❖ עמידות (durability, persistence) למרות נפילות:

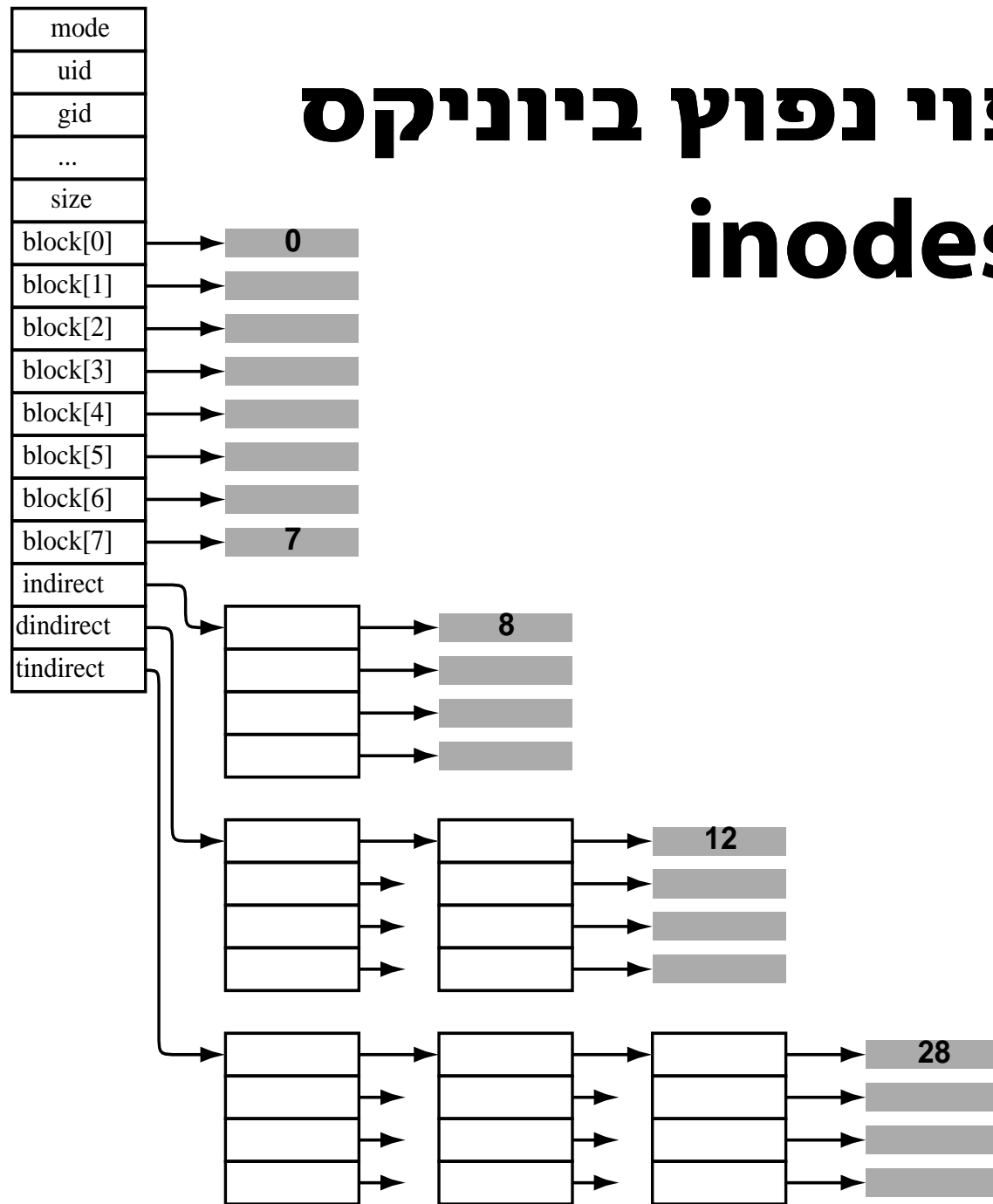
- כתיבה רגילה אינה מבטיחה עמידות אם המחשב ייפול
- סגירה של קובץ מבטיחה עמידות לכתיבות של התהליך אליו
- ניתן להבטיח עמידות על ידי פתיחת הקובץ בצורה מיוחדת או על ידי קריאות מערכת מיוחדות
- הבטחת עמידות פוגעת בביצועים (יותר כתיבות ולפי סדר שרירותי)

מיפוי קבצים

- ❖ מערכת ההפעלה מחלקת קבצים לבלוקים בגודל קבוע ושומרת אותם בדיסק, לא בהכרח ברצף
- ❖ מנגנון המיפוי מוצא את המיקום הפיזי בדיסק של בלוק מסוים ברצף הנתונים של הקובץ (היכן הבלוק ה-17 של הקובץ?)
- ❖ התעקשות על הקצאה ברצף בלוקים פיזי מפשטת את המיפוי
 - אבל גורמת לפיצול חיצוני (הרבה חורים, קטנים מדי לקובץ גדול),
 - לא מאפשרת הגדלה של קובץ ללא הזזה אם אין אחריו חור
- ❖ מיפוי של בלוקים גדולים מקטין את התקורה של המיפוי ואת גודל מבנה הנתונים, ומשפר את ביצועי הדיסק
 - אבל בלוקים גדולים מבזבזים יותר מקום בבלוק האחרון של כל קובץ (פיצול פנימי); לפעמים אפשר לדחוס כמה זנבות לקובץ אחד

מנגנון מיפוי נפוץ ביוניקס

ולינוקס : inodes



מיפוי קבצים ב-NTFS בחלונות

❖ קובץ מיוצג על ידי רשומה בקובץ master file table (MFT)

❖ רשומה מכילה את

- תחילת רצף הנתונים (קובץ קטן ישמר בתוך הרשומה)
- נתונים אודות הקובץ: שמו או שמותיו, הרשאות גישה, זמן יצירה, ...
- מערך של מצביעים לבלוקים של הקובץ בדיסק

❖ אם אין מקום במערך המצביעים לכל הבלוקים משתמשים

ברשומת המשך

מיפוי קבצים ב-FAT

- ❖ בדיסק (מחיצה) שמורה טבלת הקצאה אחת של מצביעים שגודלה כמספר הבלוקים בדיסק
- ex.doc → 54
- 0 :
- ❖ קובץ מיוצג על ידי מספר הבלוק הראשון שלו
- 1 :
- ❖ בכל איבר בטבלת ההקצאה שמור מצביע לבלוק הבא של הקובץ
- ... 23 : 47
- ❖ הבלוקים הפנויים משורשרים לרשימה גם הם מיפוי בקובץ ארוך דורש זמן פרופורציוני למספר הבלוקים בקובץ (מעבר על רשימה מקושרת)
- ... 47 : -1
- ❖ הקובץ בדוגמה שמור בבלוקים 47, 23, 54
- ... 54 : 23

מדיניות הקצאת בלוקים לקבצים

- ❖ איזה בלוקים כדאי להקצות לקובץ?
- ❖ רצוי למפות רצפים ארוכים בקובץ לרצפים פיזיים ארוכים משום שיישומים קוראים לעיתים קרובות קבצים שלמים ברצף או חלקים גדולים שלהם, וקריאת רצף בלוקים פיזי יעילה יותר
- ❖ רצוי להקצות קבצים שמשתמשים בהם ביחד קרוב בדיסק על מנת למזער את הצורך בהזזת הזרוע, למשל קבצים מאותה ספרייה
- ❖ מערכות קבצים כמו FAT ששומרות את הבלוקים הפנויים ברשימה מקצות מתחילת הרשימה, ובמשך הזמן קבצים מקבלים בלוקים אקראיים פחות או יותר ← ביצועים גרועים

הקצאה חכמה ב-FFS

❖ BSD Unix-מ Fast File System

❖ הקצאה חכמה טיפוסית: רצפים פיזיים ארוכים וקבצים מאותה

ספריה קרובים בדיסק

❖ הדיסק מחולק לקבוצות גלילים עוקבים; בכל קבוצה שומרים

▪ את הפרמטרים של מבנה הדיסק ומבנה מערכת הקבצים, לשרידות

▪ מאגר של inodes כדי שה-inodes ישמרו קרוב לקבצים שלהם

▪ מערך סיביות לייצוג הבלוקים הפנויים בקבוצת הגלילים

▪ בלוקים של נתונים

❖ מדיניות הקצאה דו-שלבית:

▪ בחירת קבוצת גלילים שבה יוקצה הבלוק הדרוש לקובץ

▪ בחירת הבלוק הספיציפי בקבוצה

FFS: פרטי ההקצאה

❖ בחירת קבוצת גלילים:

- אם הקובץ קיים ובקבוצה שבו הוא מוקצה יש מקום, והקובץ תופס פחות מרבע ממנה, נקצה באותה קבוצה, אחרת נמצא קבוצה אחרת עם תפוסה נמוכה
- אם הקובץ חדש, נקצה בקבוצה שמכילה את המדריך שבו הקובץ

❖ בחירת בלוק:

- הבלוק הקרוב ביותר מבחינה סיבובית של הדיסק לבלוק האחרון בקובץ, אם הוא פנוי, אחרת משתמשים במדיניות משנית

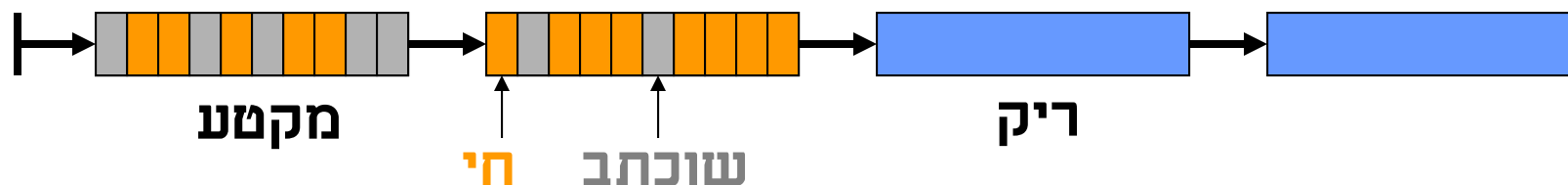
- ❖ מערכת הקבצים מנסה למפות אשכולות של 64 KB של נתונים לבלוקים רצופים פיזית על מנת למרב את קצב ההעברה

סיכום FFS

- ❖ ביצועים מצוינים בסביבות שבהן יש הרבה העברת נתונים מכיון שהזרוע זזה מעט (הקצאה בקבוצות גלילים) ומכיון שלא ממתינים הרבה לסיבוב הדיסק (הקצאה מיטבית של בלוקים והעברה של אשכולות שלמים)
- ❖ ביצועים פחות טובים בסביבות שבהן יוצרים ומוחקים קבצים בתכיפות: להקצאת בלוקים אין השפעה אבל תכונות אחרות של FFS מגבילות את הביצועים

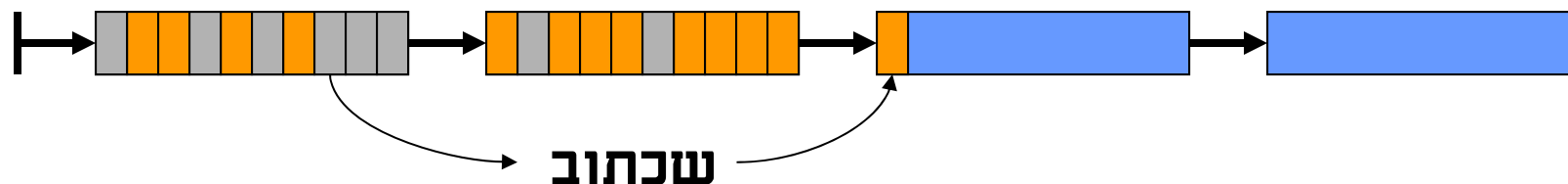
גישה אחרת:

Log-Structured File System



- ❖ הדיסק מחולק למקטעים בגודל קבוע (כמיליון בתים) שמשורשרים לרשימה
- ❖ נתונים נכתבים על מקטעים כאילו הרשימה הייתה מגילה אינסופית; נתונים נכתבים ומשוכתבים רק בסוף הרשימה
- ❖ תחילת המגילה מכילה בלוקים "חיים" (חלק מקבצים) ובלוקים של קבצים שנמחקו ובלוקים ששוכתבו
- ❖ סוף הרשימה ריק
- ❖ תהליך מיוחד דואג שתמיד יהיה מקטעים ריקים בסוף הרשימה

שכתוב בלוק ב-LFS



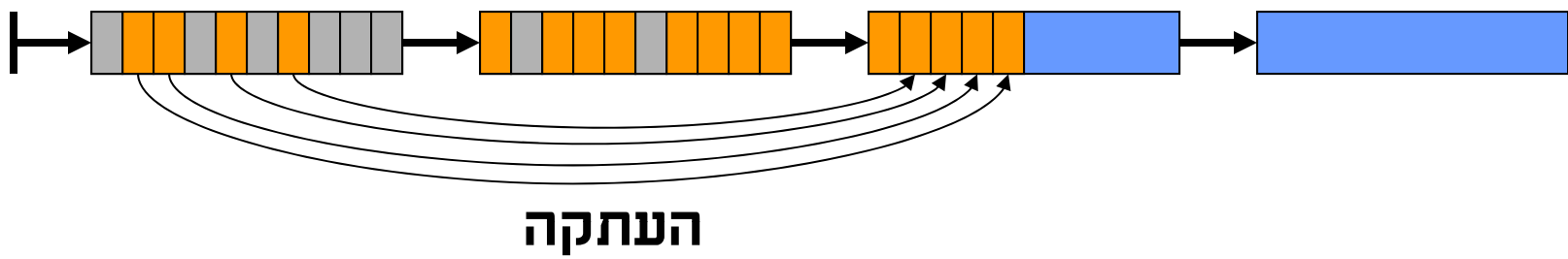
❖ בלוקים משוכתבים לסוף המגילה

❖ כאשר משכתבים בלוק, צריך לשכתב גם את ה-inode או בלוק המצביעים שמצביע עליו, משום שמקומו השתנה

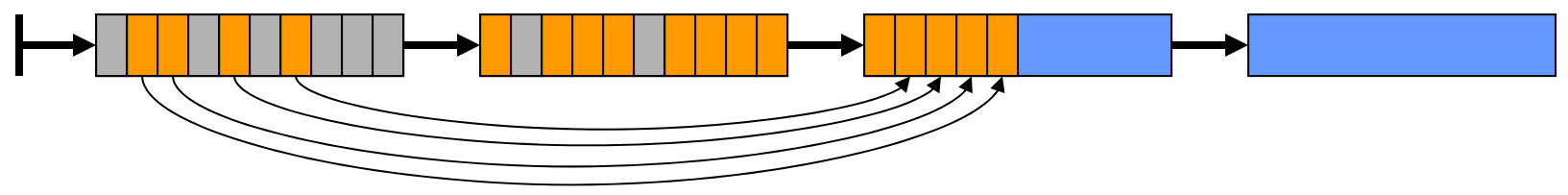
❖ שרשרת השכתובים נעצרת ב-inode משום שמדריכים מצביעים ל-inodes לוגיים שטבלה ממפה למיקום בדיסק

❖ בלוקים אינם נכתבים אחד אחד אלא בקבוצות, כאשר מצטבר מקטע שלם או כאשר חייבים לכתוב בלוק לקובץ

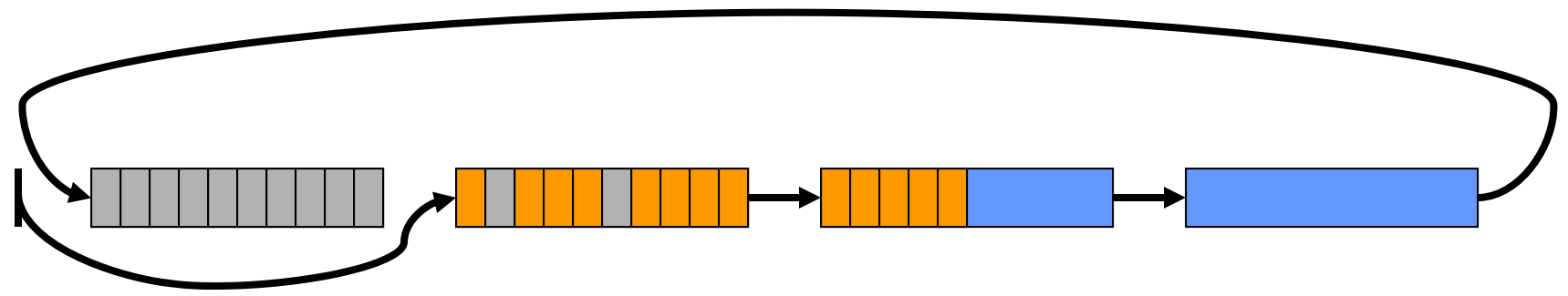
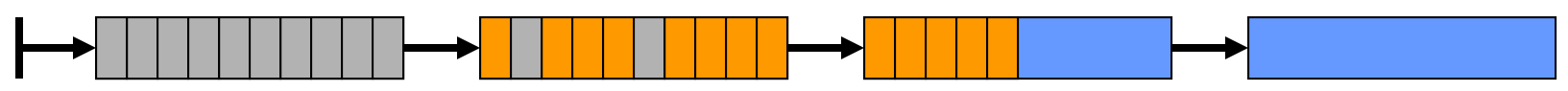
ניקוי מקטעים ב-LFS



ניקוי מקטעים ב-LFS



העתקה



❖ זיהוי בלוקים חיים לעומת משוכתבים: כל מקטע מתאר את הזהות (קובץ+מספר בלוק) של כל בלוק בו; ממפים את הבלוק ומשווים

סיכום LFS

- ❖ כתיבות יעילות מאוד:
 - תמיד לבלוקים רצופים בדיסק
 - בדרך כלל מקטעים שלמים או חלקים גדולים שלהם
- ❖ קריאות מבלוקים שרירותיים בדיסק, אבל
- ❖ אם כתיבה ביחד מנבאת קריאה ביחד, אז קריאות מאזורים קרובים בדיסק
- ❖ ביצועים מצוינים כאשר ביצועי כתיבה חשובים יותר מביצועי קריאה או כאשר יש יותר כתיבות מקריאות (קבצי לוג, למשל)
- ❖ ביצועים גרועים במערכות מרובות משתמשים: כתיבות בו-זמניות לא מנבאות דבר לגבי קריאות
- ❖ רעיון שקם לתחיה במערכות אחסון מבוססות זיכרון הבזק כי זול בהן לכתוב ברצף פיזי ויקר לשכתב in place

נפילות והתאוששות

- ❖ השגרות של מערכת הקבצים מניחות הנחות מסוימות אודות מבנה הנתונים על הדיסק; נכונות השגרות תלויה בקיום ההנחות
- ❖ נפילה פתאומית עלולה להשאיר את המערכת במצב לא תקין
- ❖ הנחות שחייבות להתקיים:
 - בלוק שהוא חלק מקובץ אינו מסומן כפנוי
 - בלוק אינו ממופה כחלק משני קבצים או יותר
 - מצביע במדריך ל-inode שמסומן כפנוי
 - ...
- ❖ הנחות פחות חשובות שניתן לתקן בזמן שהמערכת פועלת:
 - בלוק שאינו ממופה כחלק מקובץ וגם אינו מסומן כפנוי
 - inode שאין אליו מצביע ואינו מסומן כפנוי

גישות להתאוששות

❖ כתיבה זהירה

❖ עדכונים רכים

❖ כתיבה עצלה

❖ מערכות מתאוששות

❖ שימוש בזיכרון לא נדיף

❖ זיהוי הצורך בהתאוששות: הדלקת סיבית מיוחדת במערכת

הקבצים מכריזה שהמערכת ירדה באופן מסודר והיא

קונסיסטנטית. הכתיבה הראשונה למערכת קבצים היא כיבוי

הסיבית. אם היא כבויה, דרושה התאוששות

כתיבה זהירה

- ❖ פעולות שמשנות את מבנה הנתונים בדיסק מתבצעות מיידית (לפני שקריאת המערכת חוזרת) והבלוקים נכתבים לדיסק בסדר שמבטיח שלא ייווצר חוסר קונסיסטנטיות חמור
- ❖ למשל, בזמן מחיקת קובץ קודם ימחק המצביע ל-inode מהמדריך, אחר כך ימחקו ההצבעות לבלוקים מה-inode, ואז יסומנו ה-inode והבלוקים כפנויים
- ❖ הגישה משמשת את FAT, FFS, LFS, ועוד
- ❖ גישה זו משפיעה לרעה על הביצועים כאשר קבצים נוצרים ונמחקים באופן תכוף; פחות ב-LFS כי הכתיבות רצופות
- ❖ אחרי נפילה, תהליך התאוששות רץ ברקע ומתקן את הבעיות הלא חמורות (סימון בלוקים כפנויים וכדומה)

עדכונים רכים

- ❖ מבנה נתונים בזיכרון שומר את כל השינויים שצריך לבצע בדיסק; לא משנים ישירות את העותרים של הבלוקים בזיכרון
- ❖ לגבי כל שינוי, המערכת זוכרת את השינויים שחייבים לבצע לפניו כדי לא ליצור מצב לא קונסיסטנטי חמור
- ❖ זהו למעשה גרף שבו השינויים הם צמתים
- ❖ כאשר צריך לכתוב לדיסק, מערכת הקבצים מזהה את כל השינויים שצריך לכתוב ואת כל מי שצריך לכתוב לפניהם, מסדרת, משנה את העותקים בזיכרון וכותבת לדיסק
- ❖ יתכנו בלוקים שצריך לכתוב יותר מפעם אחת בגלל שהם מכילים כמה שינויים שביניהם צריך לכתוב שינויים אחרים
- ❖ למרות זאת, ניסויים הראו שהביצועים טובים הרבה יותר מכתובה זהירה, מכיון שניתן לעכב בזיכרון עדכונים

כתיבה עצלה

- ❖ מבצעים שינויים על עותקים של בלוקים בזיכרון וכותבים לפי סדר שנוח מבחינת תזמון הדיסק; מתעלמים מבעיית הקונסיסטנטיות
- ❖ כאשר המערכת עולה אחרי נפילה, יש צורך להריץ תהליך התאוששות שמתקן את מבנה הנתונים
- ❖ לפני שהתהליך מסיים לא ניתן להשתמש במערכת הקבצים כיון שהנחות בסיסיות על מבנה הנתונים עלולות שלא להתקיים
- ❖ בשימוש בלינוקס, למשל (מערכת הקבצים ext2), גישה לא נפוצה במערכות הפעלה מסחריות
- ❖ ביצועים מצוינים, התאוששות איטית (עד שעות במערכות קבצים גדולות מאוד)

מערכות מתאוששות

- ❖ שימוש בטכנולוגיה של מסדי נתונים על מנת לוודא שפעולות שמשנות את מבנה הנתונים ודורשות שינוי במספר בלוקים מתבצעות במלואן או לא בכלל, גם אם מתרחשת נפילה
- ❖ כל פעולה כזו מוגדרת כ**תנועה** וכל תנועה מקבלת מזהה
- ❖ כל שינוי בבלוק של מבנה הנתונים (לא בבלוק נתונים של קובץ!) נרשם בקובץ יומן מיוחד עם מזהה התנועה
- ❖ בסיום כל פעולה נרשמת רשומה מיוחדת ביומן עם מזהה התנועה
- ❖ מערכת הקבצים לא מבצעת את השינויים בבלוקים מייד, וגם לא כותבת לדיסק את רשומות היומן מייד, אבל היא מוודאת שבלוק נכתב לדיסק רק אחרי כל רשומות היומן שרלוונטיות אליו

אחרי נפילה של מערכת מתאוששת,

- ❖ מערכת הקבצים לא קונסיסטנטית ואי אפשר להשתמש בה לפני הרצת תהליך התאוששות (כמו בעצלות, לא כמו בזהירות)
- ❖ יתכן שיש בדיסק שינויים ששייכים לתנועות שאולי לא בוצעו במלואן ואולי אפילו לא נכתבו ליומן בדיסק במלואן
- ❖ יתכן שחסרים בדיסק שינויים ששייכים לתנועות שמתוארות ביומן בדיסק במלואן

התאוששות מערכת מתאוששת

- ❖ קריאת היומן מהדיסק
- ❖ זיהוי כל התנועות שנכתבו בחלקן אבל לא במלואן ליומן
 - בעזרת רשומות היומן מחזירים את הבלוקים הרלוונטיים למצבם הקודם
- ❖ זיהוי כל התנועות שנכתבו ליומן במלואן אבל אולי לא בוצעו מול הדיסק במלואן (חשוב לנסות למזער את מספרן)
 - בעזרת רשומות היומן מבצעים שוב את השינויים בבלוקים הרלוונטיים
 - יתכן שהשינויים כבר בוצעו; אי אפשר לדעת
 - לכן, רשומות היומן חייבות לתאר עדכונים אידמפוטנטיים
- ❖ כלומר, כל רשומת יומן מאפשרת לבצע עדכון בדיסק וגם להחזיר את המצב לקדמותו

סיכום מערכות מתאוששות

- ❖ התאוששות מהירה בגלל שצריך בדרך כלל לבצע שוב או לבטל רק תנועות מהשניות האחרונות לפני הנפילה
- ❖ ביצועים מצוינים: מעט אילוצים על סדר הכתיבה (יומן לפני בלוק)
- ❖ מערכות מורכבות מאוד בגלל הצורך לאפשר זיהוי מהיר ומדויק של תנועות שצריך לבטל או לבצע שוב (זיהוי מקורב יאט את ההתאוששות), בגלל הצורך לכפות אילוצים, ובגלל הצורך להתמודד עם מצבים מורכבים, כמו נפילה בזמן התאוששות
- ❖ מערכות נפוצות מאוד במערכות הפעלה מודרניות כולל חלונות, רוב הגרסאות המסחריות של יוניקס, ורוב הגרסאות של לינוקס

שימוש בזיכרון לא נדיף

- ❖ אחסון בלוקים של מבנה הנתונים (ואולי אף בלוקים של נתונים מקבצים) בזיכרון מיוחד לא נדיף במקום בזיכרון הראשי הנדיף
- ❖ לאחר נפילה פשוט כותבים את הבלוקים הללו לדיסק
- ❖ הזיכרון הזה צריך לשרוד לא רק תקלות חומרה (הפסקות חשמל למשל) אלא גם תקלות תוכנה שעלולות לכתוב לתוכו: עדיף להגן עליו מכתיבה רוב הזמן ולשחרר את ההגנה רק בשגרות של מערכת הקבצים
- ❖ ניתן למימוש בעזרת זיכרון מגובה בסוללה או אל-פסק או בעזרת זיכרון מגנטי מהיר
- ❖ בשימוש בשרתי קבצים ייעודיים; לא נפוץ במערכות הפעלה רגילות

מערכות קבצים וזיכרונות הבזק

- ❖ בדרך כלל זיכרונות הבזק (flash) מנוהלים על ידי בקר ייעודי שגורם להם להיראות כמו דיסק מגנטי (אפשר לקרוא לכתוב בלוקים קטנים) אבל ממפה כתובות לוגיות לפיזיות
- ❖ בהתקנים יקרים המיפוי דומה ל-log structured file system
- ❖ בהתקנים זולים יש טבלת מיפוי בגרעיניות גסה; כתיבות אקראיות הן איטיות (גורמות להעתקת הרבה נתונים)
- ❖ שימוש במערכות קבצים רגילות, לפעמים עם התאמות קלות (העדפת קריאה על כתיבה, פעולת trim לסימון בלוק שלא יקרא)
- ❖ כאשר זיכרון ההבזק חשוף למעבד: מערכות קבצים מיוחדות, בדרך כלל מבוססות על רעיונות של LFS