מערכות הפעלה

סיון טולדו

שקפים לליווי הספר "מערכות הפעלה" בהוצאת אקדמון.

התלמידים אינם צריכים עותק של השקפים ואינם צריכים להעתיק אותם למחברת משום שכל מה שכתוב בהם כתוב בספר.

אסור להדפיס; אסור לשכפל; אסור להעביר את הקובץ לאדם אחר ואסור להפיץ את הקובץ באינטרנט או בכל דרך אחרת.

נא להשאיר הודעה זו על השקף.

פרק 1
מבוא

מערכת ההפעלה

- מגינה על חומרה על מנת לתחם תקלות ועל מנת להגן על מידע
- מנהלת את משאבי החומרה (זיכרון, זמן מעבד, דיסקים, רשת תקשורת) ביעילות והגינות
- מספקת לתוכניות ממשקים אחידים ונוחים לשימוש בהתקני חומרה

חשיבות ההגנה על חומרה

- תוכנית אחת לא יכולה לקרוא או לכתוב מהזיכרון של אחרת
- תקלה בתוכנית אחת לא משפיעה על אחרות
- בפרט, גם אם תוכנית נתקעת בלולאה אין–סופית תוכניות אחרות ממשיכות לרוץ

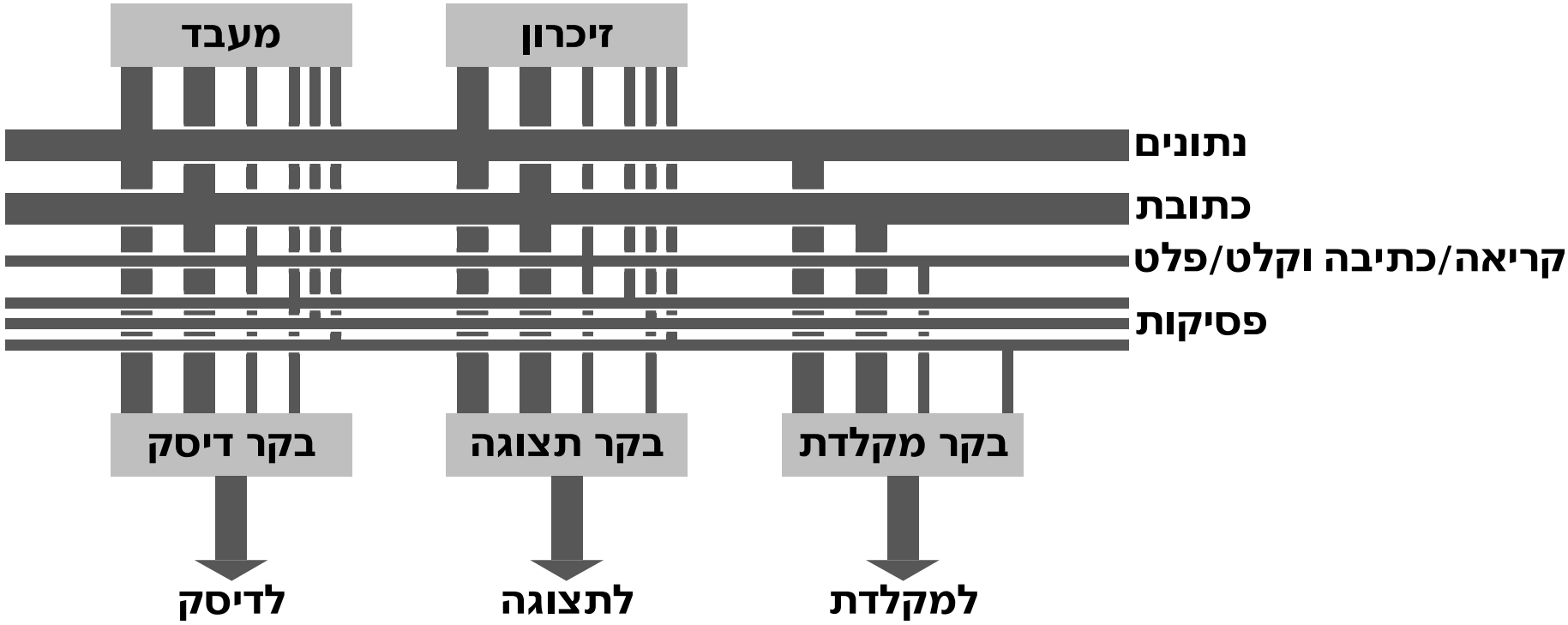
הגנה על חומרה: המעבד

- שני מצבי ריצה: מצב משתמש ומצב מיוחס
- במצב מיוחס הכל מותר – המעבד מציית לכל פקודה
- במצב משתמש פקודות מסוימות אסורות ולא ניתן לגשת לאוגרים מסוימים
- אחת לפרק זמן קבוע פסיקת שעון מעבירה את המעבד לביצוע שגרה של מערכת ההפעלה; המעבר לשגרת הפסיקה מעביר את המעבד למצב מיוחס, חזרה מחזירה אותו למצב הקודם
- הכתובת של השגרה הזו שמורה באוגר שלא ניתן לשנות במצב משתמש
- מערכת ההפעלה אינה "רצה ברקע"

הגנה על זיכרון

- שיתוף פעולה בין מערכת ההפעלה והמעבד
- כל גישה לזיכרון נבדקת על ידי המעבד מול טבלאות שמערכת ההפעלה מתחזקת
- פירוט בפרק 2

שליטה בהתקנים חיצוניים



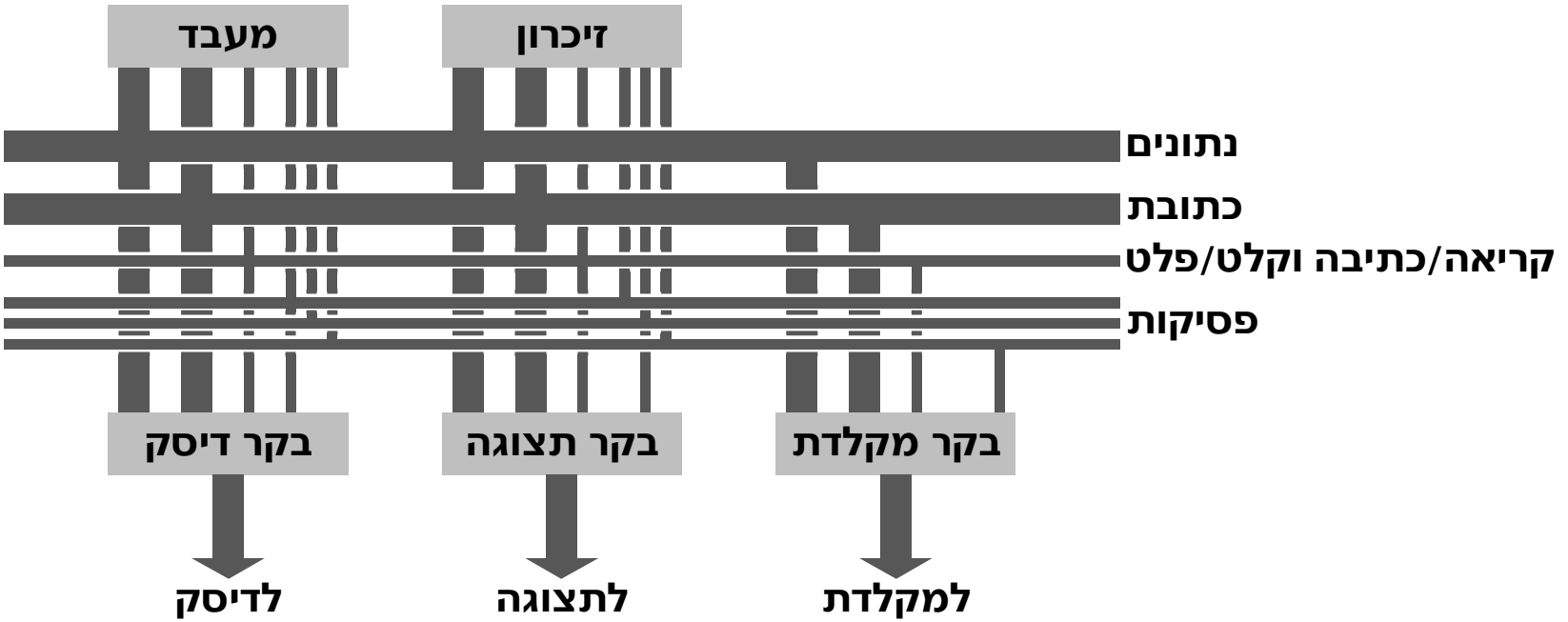
- התקנים חיצוניים מחוברים לבקרים שמחוברים לפס התקשורת המרכזי של המחשב (לעיתים יש למחשב מספר פסים)

תקשורת עם בקרים

- מידע מועבר בין המעבד ובקרים בעזרת פקודות קריאה/כתיבה מיוחדות שמדליקות את סיבית הקלט/פלט, או בעזרת פקודות קריאה/כתיבה מזיכרון
- כל בקר מגיב לתחום כתובות מסוים

```
load     r5 = *(r4)
store    *(r4) = r6
add      r3 = r3 + r8
bc       if (r3) goto loop
ioread   r5 = *(r4)
iowrite  *(r4) = r6
```

תקשורת על הפס




```
load     r5 = *(r4)
store    *(r4) = r6
add      r3 = r3 + r8
bc       if (r3) goto loop
ioread   r5 = *(r4)
iowrite  *(r4) = r6
```

הגנה על התקנים חיצוניים

- פקודות קלט/פלט שמדליקות את סיבית הקלט/פלט בפס מותרות רק במצב מיוחס
- בקרים שמגיבים לפקודות קריאה/כתיבה מזיכרון מוגנים בעזרת מנגנון ההגנה על זיכרון
- על מנת לגשת להתקן חיצוני, תוכניות קוראות לשגרה של מערכת ההפעלה
- הקריאה מתבצעת בנוהל מיוחד שמעביר את המעבד למצב מיוחס, על מנת שמערכת ההפעלה תוכל לתקשר עם הבקר

קריאות מערכת

- נוהל קריאה לשגרות של מערכת ההפעלה
- משתמש במנגנון הפסיקות או במנגנון דומה
- מעביר את המעבד למצב מיוחס; חזרה מקריאת מערכת מחזירה את המעבד למצב משתמש
- ציון השגרה שרוצים לקרוא לה על ידי מזהה מספרי ולא על ידי כתובת (למשל, **`read==1`**, **`write==1`**, וכדומה)
- אין שימוש בכתובת על מנת למנוע מעבר לשגרה שאינה של מערכת ההפעלה במצב מיוחס

ממשקים אחידים לגישה לחומרה

- מגוון אדיר של התקני חומרה דומים אך לא זהים
- תוכניות זקוקות לממשק אחיד לגישה לחומרה על מנת שיוכלו לרוץ ללא שינוי על מחשבים עם התקנים שונים
- פתרון: ספריית שגרות בעלות ממשק אחיד שכל אחת מהן ניגשת לסוג אחר של התקן חומרה
- הספרייה הזו היא חלק ממערכת ההפעלה, בעיקר משיקולי ביצועים
- הממשק לתוכניות בעיקר על ידי קריאות מערכת

ממשקים משופרים לגישה לחומרה

- הממשק שהשגרות הללו מספקות יכול לספק שירותים שהחומרה אינה מסוגלת לספק בעצמה
- למשל, אחסון או העברת נתונים אמינה בעזרת חומרה לא אמינה
- יכולות כאלה כלולות במערכות הפעלה משיקולי ביצועים או שווק

ישויות עיקריות במערכת ההפעלה

- תהליך: מחשב וירטואלי לתוכנית יחידה ומשתמש יחיד
- חוט: מעבד של מחשב וירטואלי מרובה מעבדים
- קובץ: התקן אחסון וירטואלי
- קשר: ערוץ תקשורת וירטואלי בין תהליכים, אולי במחשבים שונים
- חלון: מקלדת, עכבר, ותצוגה וירטואליים. המשתמש יכול לסמן שהוא מעוניין באינטראקציה עם חלון זה או אחר
- תור הדפסה: מדפסת וירטואלית
- משתמשים וקבוצות משתמשים לבקרת גישה למשאבים ומידע

ניהול יעיל והוגן של חומרה

- יעילות: שאיפה למרב את הנצילות של משאבי חומרה לטובת תוכניות רגילות ומשתמשים; מזעור כמות המשאבים שמערכת ההפעלה עצמה צורכת או שמתבזבזים סתם בגלל ניהול כושל
- הגינות: חלוקה הוגנת של המשאבים בין משתמשים; חלוקה הוגנת של משאבים בין תוכניות שרצות בו–זמנית

מערכות Unix

- מגוון גדול של מערכות הפעלה תואמות זו לזו ברמת קוד המקור (לא ברמת קוד המכונה), שמוצא כולן ממערכות הפעלה בשם unix שפותחה במעבדות המחקר של חברת Bell
- AIX של IBM, Solaris של Sun, HP-UX של HP, Tru64 של Compaq, IRIX של SGI
- מערכת ההפעלה של המקינטוש (החל מגרסה 10) היא מערכת יוניקס עם ממשק ייחודי
- ממחשבים אישיים ותחנות עבודה עד שרתי 7/24 עם מאות מעבדים

מערכות Linux

- מערכת הפעלה תואמת-יוניקס שפותחה החל מ-1991 על ידי סטודנט פיני בשם Linus Torvalds
- מחולקת חינם עם קוד המקור שלה
- חבילות הפצה (distributions) כוללות את מערכת ההפעלה, תוכנת התקנה, ותוכנות שימושיות
- ממחשבי כף יד ועד שרתים בינוניים (איזור ה-4 מעבדים), כולל כמעט כל מחשב שיכול להריץ חלונות

מערכות Windows

- מפותחות ומשווקות על ידי חברת מיקרוסופט.
- גרסאות למחשבי כף יד, למחשבים ביתיים (95/98), לתחנות עבודה בארגונים גדולים (2000/NT) ולשרתים קטנים ובינוניים (גרסאות מיוחדות של NT ו–2000)
- שתי משפחות של מערכות: 95/98 ו–2000/NT; גרסת XP אמורה לאחד את שתי המשפחות
- תאימות גבוהה ברמת קוד המכונה בין הגרסאות השונות

מערכות הפעלה אחרות

- מערכות הפעלה למחשבים מרכזיים (mainframes); בעיקר לצורך שימור מערכות מחשוב ארגוניות ישנות
- מערכות הפעלה למערכות זמן אמת; שליטה בכור גרעיני, בטייס אוטומטי, או בטיל דורשת יכולות תזמון מדויקות שאין למערכות הפעלה רגילות; אם אפשר לסבול כשלים נדירים ניתן להשתמש בגרסאות מתאימות של חלונות או לינוקס
- מערכות הפעלה למחשבים מעוטי יכולת כגון מחשבי כף יד; גם כאן גרסאות מתאימות של חלונות או לינוקס באות בחשבון

סיכום

- מערכות ההפעלה מגינה על החומרה על מנת לתחם תקלות ולהגן על מידע
- מערכת ההפעלה מנהלת את החומרה ביעילות והגינות
- מערכת ההפעלה מספקת ממשקים אחידים ונוחים לגישה לחומרה
- רוב מערכות ההפעלה כיום שייכות לשתי משפחות עיקריות (לינוקס/יוניקס וחלונות), לכולן יכולות בסיסיות דומות, וכולן משתמשות באותם מנגנונים
- ההבדלים העיקריים כיום בין מערכות שונות מתבטאים בסגנון הממשקים וביכולת לנהל ביעילות שרתי ענק

פרק 2
קלט/פלט

העברת נתונים אל ומאת בקרים

- דגימה: מערכת ההפעלה דוגמת את הבקר מדי פעם על מנת לבדוק האם קרא אירוע שמצריך תקשורת עם הבקר (הגיעו נתונים, הבקר מוכן לקבל פקודות נוספות, וכדומה)
- פסיקות: הבקר מודיע למעבד בעזרת קו תקשורת מיוחד בפס שקרא אירוע שמצריך תקשורת עמו. המעבד מגיב בהפעלת שגרת פסיקה של מערכת ההפעלה שמטפלת באירוע
- גישה ישירה לזיכרון: מערכת ההפעלה מורה לבקר להעביר בלוק גדול של נתונים בין התקן חיצוני (דיסק, רשת) ובין הזיכרון. הבקר מעביר את הנתונים ללא התערבות נוספת של המעבד

יתרונות וחסרונות

- דגימה כמעט שאינה בשימוש. דגימה איטית מדי עלולה להוביל לאובדן נתונים ודגימה כשלא קורה כלום מבזבזת משאבי מחשב
- פסיקות פותרות את הבעיות שדגימה מציבה והן בשימוש נרחב
- גישה ישירה לזיכרון מייקרת את הבקר אך מאפשרת העברת נתונים מהתקנים מהירים ללא עומס על המעבד. בשימוש נרחב בגישה לדיסקים ורשתות תקשורת מהירות. ניתן להוזיל את הבקרים על ידי שימוש בבקר גישה ישירה משותף לכמה בקרים אחרים

אבחנה בין פסיקות שונות

- כאשר מספר בקרים משתפים קו פסיקה, מערכת ההפעלה צריכה לברר איזה בקר הפעיל את הפסיקה
- ניתן לחסוך את הבדיקה על ידי שימוש במספר קווי פסיקה בפס
- במעבדים עם כניסת פסיקה אחת (למשל מעבדי IA32 כולל פנטיומים למיניהם) משתמשים בבקר פסיקות שיש לו כמה כניסות פסיקה ויציאה אחת. בקר הפסיקות מודיע למעבד שאחד הבקרים הפעיל פסיקה ומערכת ההפעלה יכולה לברר בעזרתו איזה בקר הפעיל את הפסיקה

מניעת הפעלה רקורסיבית של שגרות פסיקה

- פסיקה משעה את השגרה שרצה ומפעילה את שגרת הפסיקה
- רצוי לא להשעות את שגרת הפסיקה עצמה ולהפעילה שוב אם מתרחשת פסיקה נוספת לפני שהיא חוזרת
- הפעלה רקורסיבית עלולה לגרום להשחתת מבני נתונים (עוד על כך בפרק 5)
- מערכות הפעלה פשוטות משעות את הטיפול בפסיקות כאשר שגרת פסיקה רצה
- מערכות הפעלה מתקדמות יותר מסווגות פסיקות על פי עדיפויות ומשעות רק פסיקות בעדיפות נמוכה יותר מזו המטופלת

הפעלה דחויה של שגרות

- אם הטיפול בפסיקה ארוך, עדיף לבצע כמה שיותר מהטיפול מחוץ לשגרת הפסיקה, על מנת שלא לדחות טיפול בפסיקות אחרות
- מערכת ההפעלה מבצעת בשגרת הפסיקה את הטיפול שיש לבצע מייד ואורזת ייצוג של שאר הטיפול במבנה נתונים שמייצג הפעלה דחויה של שגרה
- מערכת ההפעלה מבצעת את הקריאות הדחויות לאחר הטיפול בפסיקות אך לפני החזרה לתוכנית המשתמש
- זהו למעשה מנגנון תזמון זעיר שמבטיח טיפול מהיר באירועים דחופים (פסיקות)

מנהלי התקן

- מנהל התקן הוא מודול תוכנה עם ממשק סטנדרטי ששולט על התקן חומרה מסוים, כמו בקר דיסקים מסוג מסוים או בקר רשת מסוג מסוים
- השגרה שמופעלת על ידי קריאת מערכת קוראת למנהל ההתקן המתאים
- לעיתים מערכת ההפעלה קוראת למנהל התקן באופן לא ישיר: תוכנית קוראת מקובץ ומערכת ההפעלה קוראת למנהל ההתקן של הדיסק על מנת לחפש ולקרוא את הנתונים
- לא כל מנהל התקן שולט על חומרה; לפעמים מנהל התקן שולט על התקן וירטואלי, למשל מערך דיסקים

ממשק של מנהל התקן

- שגרת פסיקה שמופעלת כאשר ההתקן מפעיל פסיקה
- שגרה לאתחול ההתקן
- שגרות להתחברות והתנתקות (open, close)
- שגרות לקריאת וכתיבת נתונים (read, write)
- שגרה להזזת מצביע הקלט/פלט (seek)
- העברת פקודות מיוחדות להתקן או למנהל (ioctl)
- עוד שגרות פחות חשובות
- ממשק אחר למנהלי התקן של בקרי רשת תקשורת ושל בקרי תצוגה גרפית

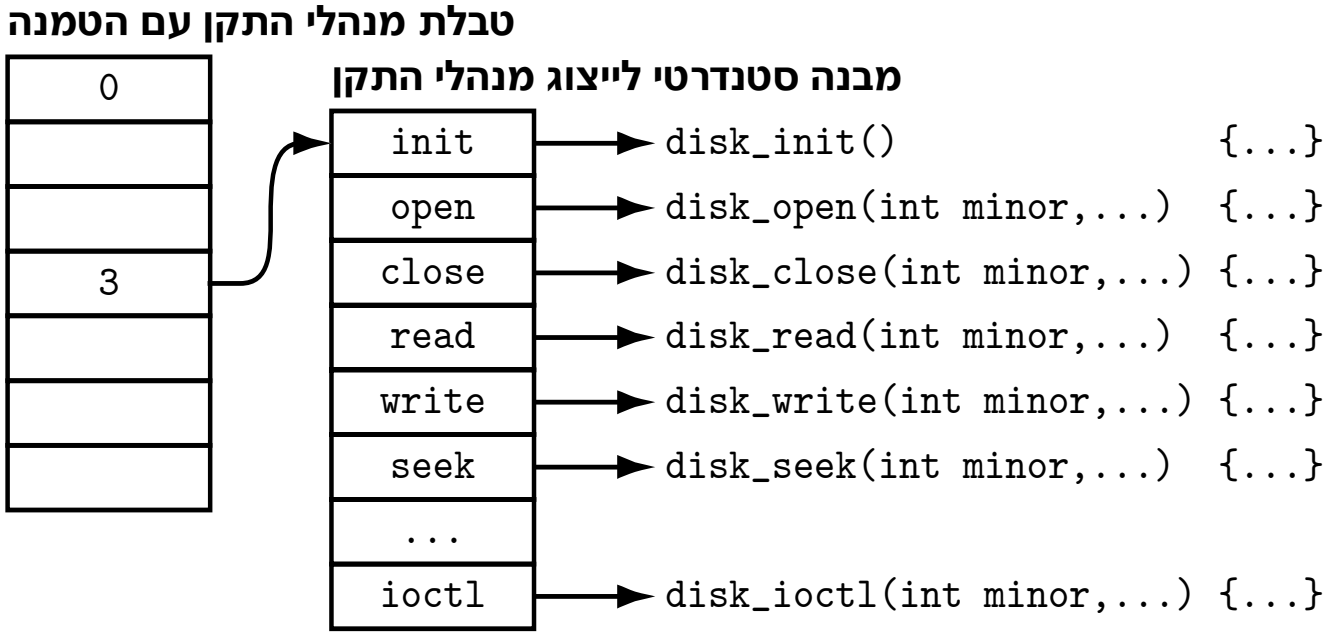
הטמנה וחציצה

- **הטמנה:** מערכת ההפעלה שומרת בזיכרון הראשי בלוקים של נתונים שנקראו לאחרונה מהתקני זיכרון חיצוניים (דיסקים)
- גישה חוזרת לבלוק מחזירה את תוכנו מהמטמון ללא גישה להתקן החיצוני
- הטמנה חוסכת גישות להתקן חיצוני איטי
- **חציצה:** מערכת ההפעלה שומרת בזיכרון הראשי בלוקים שמיועדים להיכתב להתקני זיכרון חיצוניים ומעכבת את הכתיבתם להתקן
- כתיבה חוזרת לאותו בלוק משנה את תוכן הבלוק בחוצץ; רק הנתונים המאוחרים יכתבו להתקן החיצוני
- חציצה (עיכוב כתיבה) חוסכת גישות שמשכתבות נתונים ומאפשרת לכתוב להתקן כאשר הוא פנוי

מנהלי התקן עם וללא הטמנה

- מערכת ההפעלה מנהלת מאגר חוצצים מרכזי (מטמון) עבור מנהלי התקן של התקני זיכרון חיצוניים
- התקנים שאינם התקני זיכרון (מקלדת, עכבר, מדפסת, רשת תקשורת) אינם זקוקים להטמנה, אבל חלקם זקוקים לחציצה
- חציצה מאפשרת לקרוא נתונים שמגיעים מההתקן (מקלדת למשל) גם אם תוכנית אינה מבצעת קריאה באותו רגע ממש, ולכתוב נתונים בקצב שמתאים להתקן (מדפסת, מודם)
- בלינוקס/יוניקס יש שתי טבלאות של מנהלי התקן: עם הטמנה (**block**) וללא הטמנה (**character**), חוץ ממנהלי התקן לבקרי רשת ולבקרי תצוגה גרפית

התייחסות להתקנים ביוניקס



- התקן מיוצג על ידי שני מספרים: **ראשוני** שמייצג את מנהל ההתקן שמטפל בהתקן **ומשני** שמייצג את ההתקן עבור המנהל
- תוכניות מתייחסות להתקן על ידי גישה לקובץ מיוחד שמכיל מזהה טבלת מנהלים (עם/ללא הטמנה) ואת שני המספרים
- ניתן להתייחס להתקן כאל קובץ, להתחבר אליו ולקרוא/לכתוב נתונים

מנהלי התקן בלינוקס/יוניקס

- מנהל ההתקן קיים וההתקן עצמו קיים:

```
# ls -l /dev/lp0
crw-rw---- 1 root daemon 6,0 may 5, 1988 /dev/lp0
# cat < /dev/lp0
skfjkl...
```

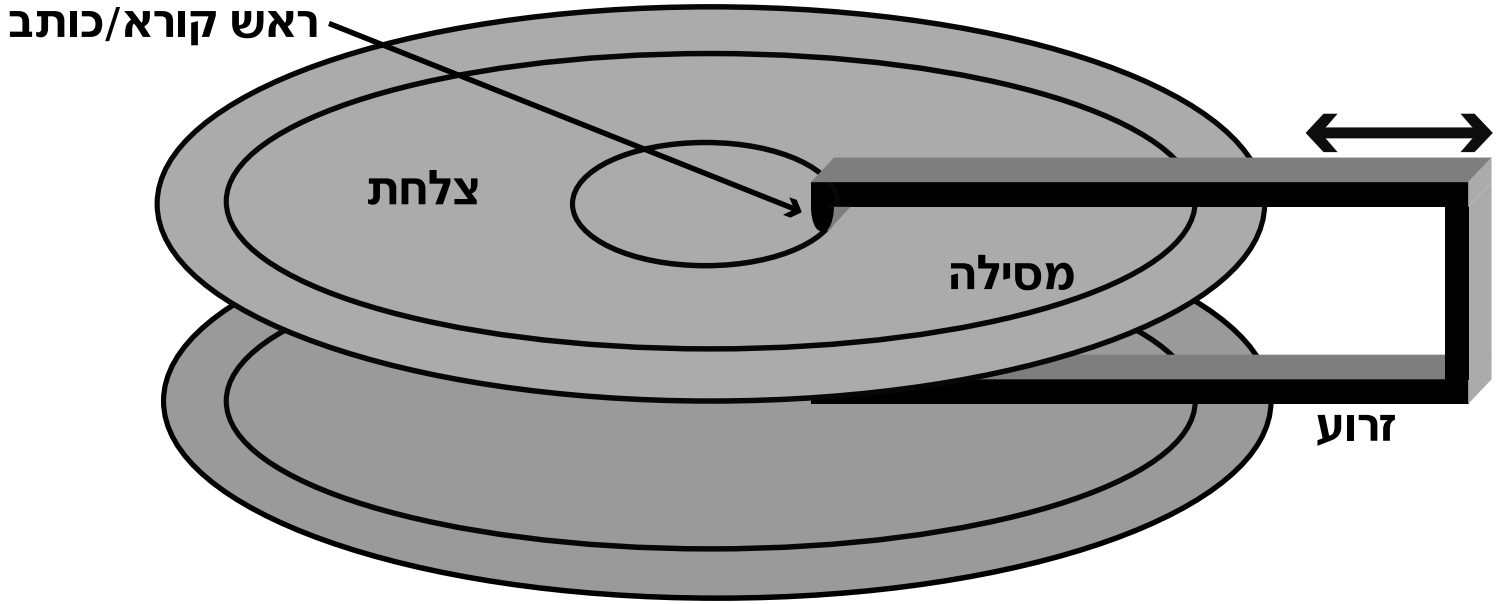
- מנהל ההתקן קיים, אבל אין התקן פיזי:

```
# cat < /dev/lp0
cat: -: Input/output error
```

- הקובץ המיוחד קיים, אבל אין מנהל התקן בטבלה:

```
# cat < /dev/lp0
bash: /dev/lp0: No such device
```

דיסקים קבועים



- משמשים לאחסון קבצים וכהרחבה של הזיכרון הפיזי ולכן ביצועיהם משפיעים ישירות על ביצועי מערכת המחשב כולה
- מסתובבים במהירות קבועה של 3600–15000 סיבובים בדקה
- קיבולת של עד 160 GB, קצבי העברה אפקטיביים של 10–20 MB/s במחשבים אישיים
- המספרים נכונים לשנת 2001

תזמון דיסקים קבועים

- בקשות גישה מצטברות בתור
- ביצוע לפי סדר קבלה (FCFS): הוגן לחלוטין אבל לא יעיל; הזרוע עלולה לנוע שוב ושוב מצד לצד על מנת לקרוא קטע (סקטור) בודד בכל פעם
- מזעור זמן השיוט (SSTF): הבקשה הבאה שתשורת היא הבקשה שניתן להגיע אליה תוך פרק הזמן המזערי; לא הוגן עד כדי הרעבה
- מדיניות מעלית: הזרוע נעה מהמסילה הפנימית לחיצונית וחזרה ומשרתת את כל בקשות הגישה לפי סדר תנועת הזרוע, בדומה לאוטובוס עם מסלול קבוע; הוגן ויעיל

וריאציות על מדיניות מעלית

- **scan**: הזרוע נעה פנימה והחוצה בין שתי המסילות הקיצוניות ומשרתת בקשות שהיא פוגשת בדרך
- **c-scan**: כנ"ל, אבל משרתים בקשות רק בדרך פנימה, את הדרך החוצה עושים במהירות; יותר הוגן ואולי מעט פחות יעיל (מעט כי הזרוע מאיצה)
- **look**: כמו **scan** אבל הזרוע לא נעה מעבר למסילה הקיצונית שיש עבורה בקשה. יותר יעיל ופחות הוגן
- **c-look**: מובן מאליו
- אפשר להפעיל את מדיניות דומה גם לגבי קטעים בתוך מסילה
- לא כדאי לשרת בקשות למסילה שנמצאים בה שנכנסו לתור לאחר שהזרוע הגיע למסילה. מדוע?

מחיצות

- חלוקה של דיסק למספר דיסקים לוגיים על פי טבלה ששמורה בתחילת הדיסק
- כל מערכות ההפעלה יודעות לפענח את הטבלה
- משמשת להתקנת מספר מערכות הפעלה או מספר מערכות קבצים על אותו דיסק פיזי
- בשימוש בעיקר במחשבים אישיים

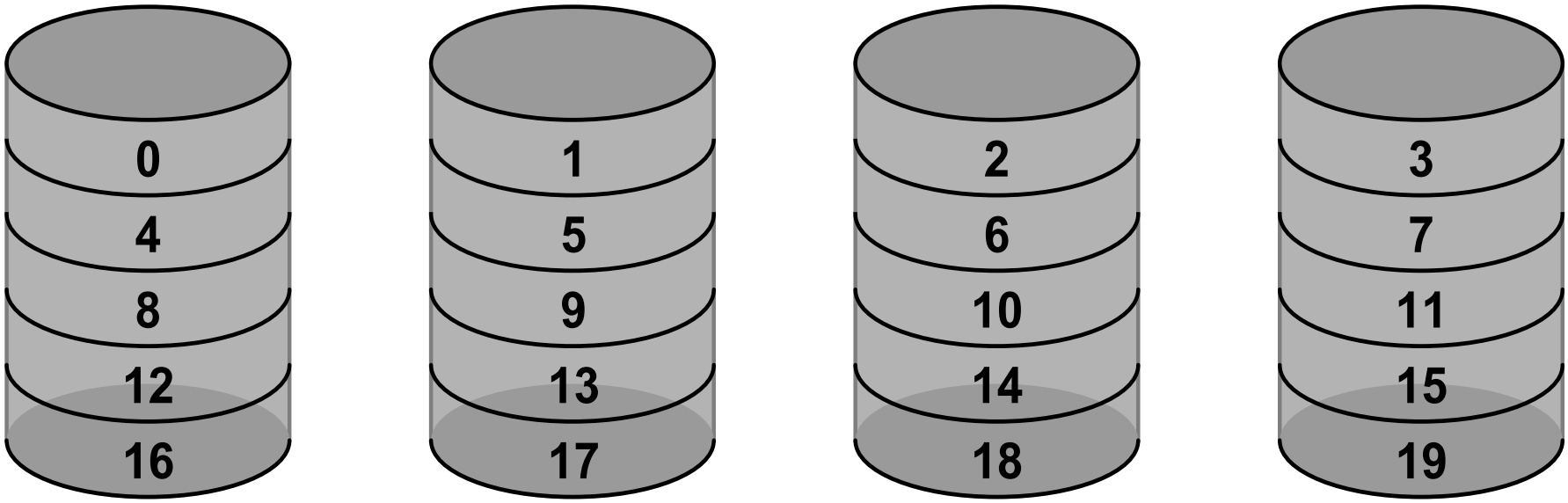
דיסקים לוגיים (Logical Volumes)

- דיסק אחד או יותר מחולקים למקטעים (extents) בגודל קבוע (מספר MB)
- ניתן להרכיב דיסק לוגי מכל קבוצה של מקטעים
- מאפשר להרכיב דיסקים לוגיים מהירים יחסית (למשל על ידי שימוש במסילות חיצוניות של מספר דיסקים)
- מאפשר להגדיל ולהקטין דיסק לוגי על ידי הוספת או גריעת מקטעים (מערכת הקבצים, מבנה הנתונים שעל הדיסק, צריכה לתמוך בהגדלת והקטנת הדיסק)
- בשימוש בעיקר ביוניקס ולינוקס

מערכי דיסקים (RAID)

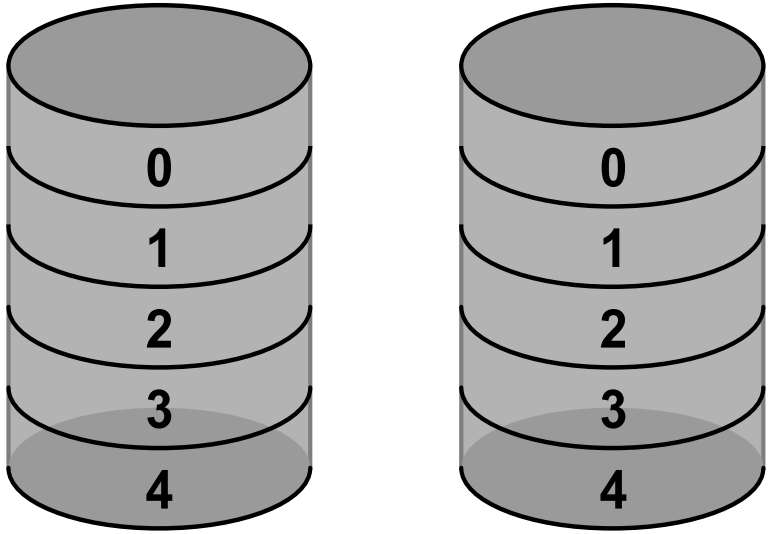
- מספר דיסקים משמשים כהתקן אחסון אחוד
- ניתן למימוש ברמת מנהל ההתקן, כמו דיסקים לוגיים ומחיצות, או ברמת בקר הדיסקים שדרושה רמת ביצועים גבוהה
- מבוסס על מיפוי בלוקים של נתונים בהתקן הוירטואלי (מערך הדיסקים) לדיסק+היסט, כלומר למיקום של בלוק פיזי של נתונים
- מספר ואריאציות שנקראות רמות שמספקות קומבינציות שונות של שירותים; רמה גבוהה אינה בהכרח טובה יותר לכל שימוש

RAID 0



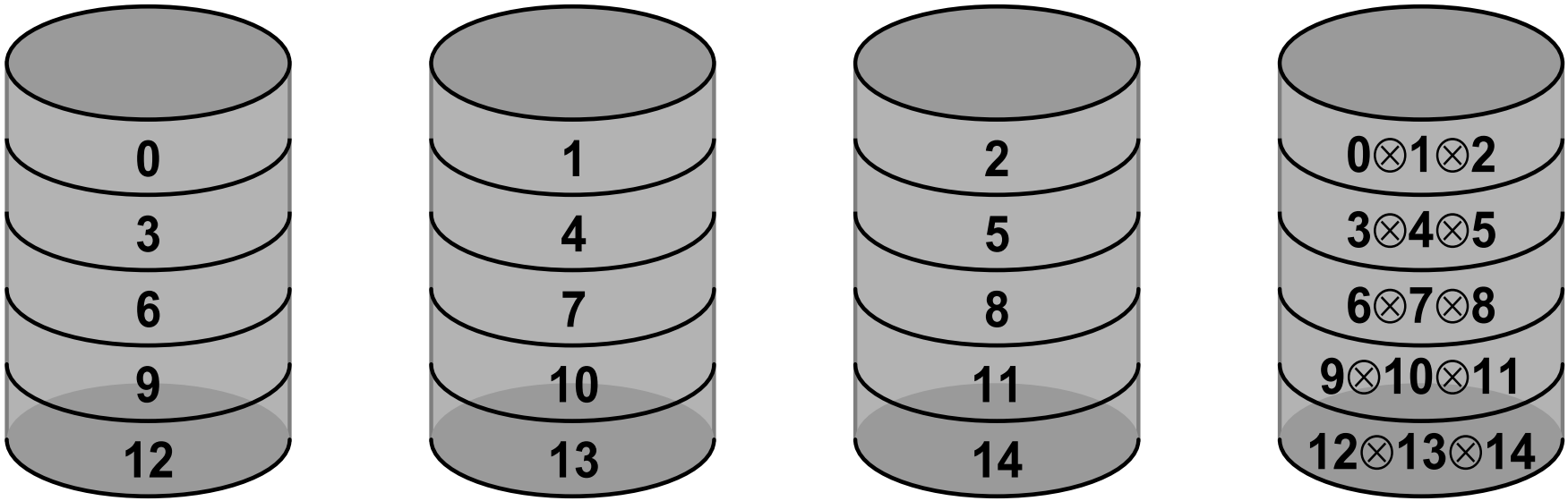
- בלוקים מפוזרים על הדיסקים באופן מחזורי
- קבוצת כל הבלוקים ששמורים באותו היסט על כל הדיסקים נקראת **רצועה**
- ניתן להעביר נתונים אל/מכל הדיסקים במקביל אם יש בקשות גישה לכולם בתור, למשל אם ניגשים תמיד לרצועות שלמות
- אמינות נמוכה; גידול במספר הדיסקים מקטין את האמינות

RAID 1



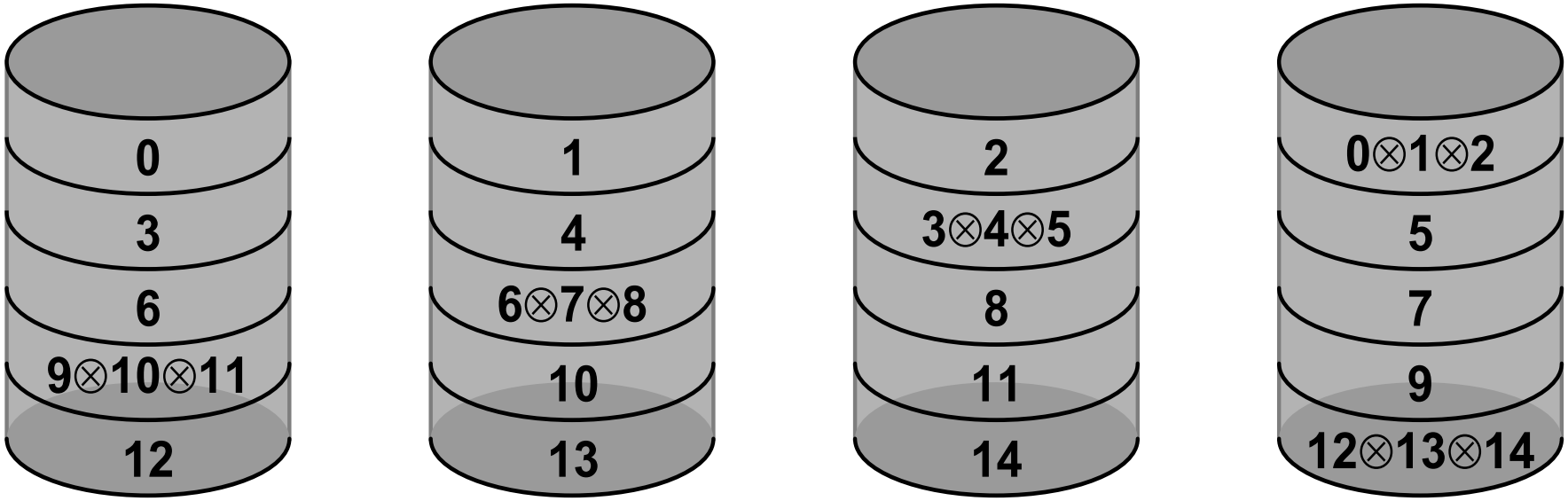
- שיקוף (תמונת ראי) נתונים על שני דיסקים
- אין אובדן נתונים כאשר דיסק מתקלקל
- המערכת ממשיכה לעבוד עם הדיסק התקין אם דיסק מתקלקל
- אחרי שמחליפים את הדיסק המקולקל המערכת מעתיקה אליו את הנתונים מהדיסק התקין
- בשרתים, דיסקים באריזה נשלפת (hot-swap) מאפשרים החלפת דיסק מקולקל בזמן עבודה רצופה של המערכת

RAID 3



- מייחדים דיסק אחד לשמירת זוגיות
- הזוגיות מהווה יתירות מספיקה להמשך עבודה רצופה עם דיסק מקולקל
- כל הגישות הן לרצועה שלמה בבת אחת, מכיון שכל כתיבה לבלוק מצריכה שינוי בלוק הזוגיות ברצועה וכתיבה לדיסק הזוגיות
- אמינות וקצב העברה גבוה ליישומים שניגשים לרצועות שלמות

RAID 5



- בכל רצועה בלוק הזוגיות נשמר בדיסק אחר, באופן מחזורי
- כתיבת בלוק מיושמת על ידי קריאת הבלוק ובלוק הזוגיות ברצועה, עדכונם (החלפת סיבית זוגיות עבור כל סיבית נתונים שערכה מוחלף), וכתיבתם חזרה לשני הדיסקים
- עדיף על 3 אם יש מעט כתיבות או כתיבות לבלוקים בודדים, 3 עדיף אם רוב הכתיבות הן לרצועות שלמות

דיסקים אופטיים

- תקליטורים מאחסנים 650–700 MB, תקליטי DVD עד 17 GB
- מידע מיוצג על ידי חורים במשטח
- תהליך דפוס מהיר להפצה של מידע ספרתי, אפשרות צריבה (איטית יחסית) בכוננים מתאימים
- אורך חיים גבוה מתאים לשמירת מידע ארכיונית
- צורבים פשוטים מתאימים לגיבוי של כמויות קטנות של נתונים
- לגיבוי כמויות נתונים גדולות דרושה ספריית תקליטים רובוטית

סרטים מגנטיים

- עד עשרות GB בכל קלטת
- מחיר נמוך ל–GB
- כוננים יקרים (טכנולוגיה שונה מזו של קלטות וידאו למניעת שחיקה של הראש והסרט)
- שימוש נרחב לגיבוי ושמירה ארכיונית של כמויות נתונים גדולות, תוך שימוש בספריות קלטות רובוטיות

תצוגות גרפיות

- הבקר מכיל יחידת זיכרון שכל מילה בה ממופה לפיקסל (לפעמים נעשה שימוש בזיכרון הראשי של המחשב)
- בבקרים פשוטים המעבד צובע פיקסלים על ידי כתיבה לזיכרון הזה
- בבקרים מתוחכמים המעבד יכול להורות לבקר לצבוע משולש שלם, לחשב הסתרות של עצמים במרחב, וכולי
- המבנה נגזר מהצורך להעביר לתצוגה נתונים בקצב גבוה, בעיקר עבור אנימציה ווידאו
- למשל: 30 f/s X 1 Mpixel/frame X 3 B/pixel = 90 MB/s

פרק 3
ניהול זיכרון

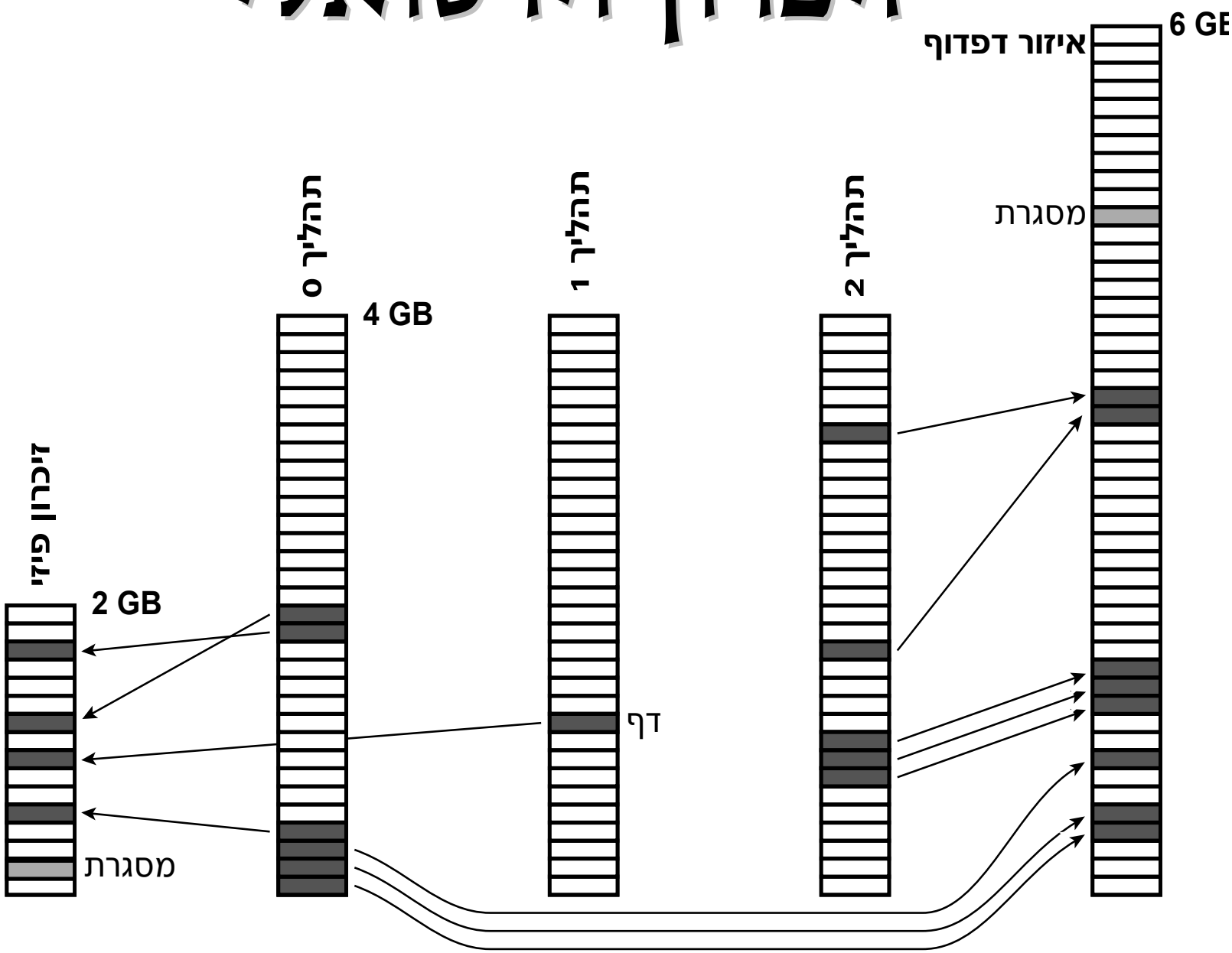
מטרות ניהול הזיכרון

- יכולת להריץ מספר תוכניות בו–זמנית תוך כדי הגנה על הזיכרון של כל תוכנית מפני האחרות
- יכולת להשתמש בדיסקים כהרחבה זולה אך איטית של הזיכרון; המעבד אינו יכול להתייחס ישירות למידע בדיסק
- יכולת להזיז את מבני הנתונים של תוכנית במהלך ריצתה בלי שהתוכנית מודעת להזזות הללו; הזזות כאלו מאפשרות לנצל "חורים" קטנים בזיכרון, או לאחד חורים לזיכרון פנוי רצוף, או להעביר מבנה נתונים מהזיכרון לדיסק ומשם למקום אחר בזיכרון

הפרדה בין כתובות בתוכנית וכתובות על הפס

- תוכנית מציגה למעבד כתובות של תאי זיכרון שיש לקרוא או לכתוב מהם נתונים
- הכתובות מאוחסנות ברגיסטרים או בזיכרון
- מצביע התוכנית מציג למעבד כתובות שיש לקרוא מהן פקודות
- המעבד **אינו** מציג את הכתובות הללו על ערוץ הכתובות בפס, אלא מתרגם אותם לכתובות אחרות תוך שימוש במבנה נתונים שמערכת ההפעלה מקימה ומתחזקת
- **כתובות** וירטואליות בתוכנית, **כתובות** פיזיות על הפס

זיכרון וירטואלי



תרגום כתובות וירטואליות לפיזיות

- מערכת ההפעלה מקימה **טבלת דפים**, מבנה נתונים שממפה את הדפים של תהליך למסגרות פיזיות
- טבלת הדפים שמורה בזיכרון ומערכת ההפעלה מודיעה למעבד היכן היא בעזרת אוגר מיוחד
- המעבד מסוגל לחפש מיפוי של דף למסגרת בטבלה
- על מנת לחסוך את הצורך בחיפוש בכל גישה לזיכרון, המעבד שומר בהתקן חומרה מיוחד (חלק מהמעבד) בשם TLB מיפויים שנעשה בהם שימוש לאחרונה
- במעבדים מסוימים מערכת ההפעלה אחראית לביצוע חיפוש בטבלה והכנסת מיפויים ל–TLB; לא נפוץ

תרגום כתובות (המשך)

- בהינתן כתובת פיזית, המעבד מפרק אותה למספר דף (סיביות בכירות) ולהיסט בתוך הדף (סיביות זוטרות)
- לדוגמה, אם גודל דף 8192 בתים, אזי 13 סיביות מציינות את ההיסט והשאר את הדף
- המעבד ממפה את הדף למסגרת פיזית ומשרשר את ההיסט למספר המסגרת על מנת ליצור כתובת פיזית שניתן להציג על הפס, למשל

1001 0000 0000 0110 0000 0000 0000 0000

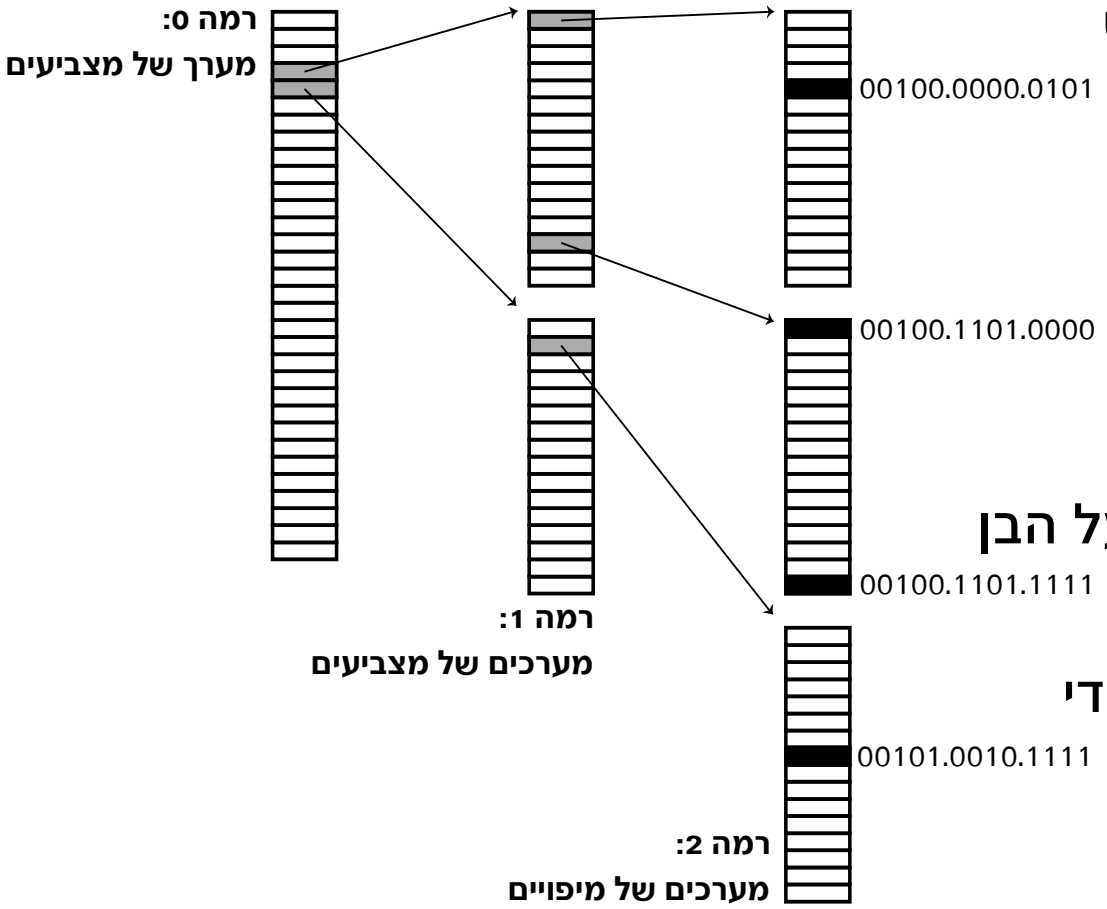
- הכתובת הוירטואלית מתייחסת לבית ה–9 בדף מספר 11; אם הדף הזה ממופה למסגרת מספר 255, הכתובת שתוצג על הפס היא

1001 0000 0000 1110 1111 0000 0000 0000

טבלאות דפים שטוחות

- מערך בגודל מספר הדפים במרחב זיכרון וירטואלי
- בכל איבר במערך מצוין מספר המסגרת ועוד מספר סיביות לתיאור מצבים שונים, כמו דף שאין מוקצה כלל לתהליך ([in]valid) או שהמסגרת שמכילה אותו אינה בזיכרון ([not]present)
- חיפוש פשוט, טבלה גדולה
- למשל, מרחב של 4 GB (מצביעים של 32 סיביות) עם דפים בגודל 8 KB ו–4 דורש טבלה עם 512 K איברים. אם גודל כל איבר 4 בתים הטבלה של כל תהליך צורכת 2 MB

טבלאות דפים היררכיות דלילות



- המיפויים שמורים בעץ חיפוש
- תתי עצים שאין בהם מיפויים תקפים אינם מיוצגים כלל (המצביע אליהם הוא **null**)
- מספר הדף מפורק לקבוצות סיביות
- הסיביות הבכירות מצביעות על הבן של השורש וכך הלאה
- המיפויים בעלים מיוצגים על ידי מערכים של עלים

טבלאות דפים היררכיות

- חיפוש מורכב יותר מאשר בטבלה שטוחה (ממומש בחומרה!)
- יותר גישות לזיכרון בזמן חיפוש (אחת לכל רמה בעץ)
- חיסכון עצום בזיכרון עבור מרחבי זיכרון שרק חלק קטן מהם מוקצה

טבלאות דפים הפוכות

- טבלה אחת לכל התהליכים
- שומרת רק מיפויים של דפים בזיכרון; מיפויים לאזור הדפדוף שמורים במבנה נתונים אחר
- ממומשת כטבלת גיבוב (hash) שמפתח החיפוש שלה הוא מספר הדף (ואולי מספר התהליך)
- כניסה צריכה לציין גם מספר מסגרת וגם מספר תהליך
- אלגוריתם חיפוש מורכב יחסית
- גודל הטבלה תלוי רק בגודל הזיכרון הפיזי ואינו תלוי בגודל מרחבי הזיכרון הוירטואליים או במספר התהליכים

תחזוקת טבלאות הדפים וה-TLB

- מערכת ההפעלה מחליטה היכן לשכן דפים
- אי לכך, מערכת ההפעלה היא שמתחזקת את טבלאות הדפים
- בדרך כלל המעבד מחפש בטבלה ומכניס מיפויים ל–TLB
- שינוי כניסה בטבלת דפים דורש מחיקת המיפוי מה–TLB אם הוא נמצא שם, בעזרת פקודת מכונה מיוחדת
- מערכת ההפעלה צריכה להודיע למעבד מה הכתובת של טבלת הדפים של התהליך שרץ (באוגר מיוחד)

הגנה על זיכרון

- כל הפקודות שעוסקות ב– TLB ובכתובת של טבלת הדפים מותרות רק במצב מיוחס
- אם טבלת הדפים הפוכה, מערכת ההפעלה מודיעה למעבד מה המזהה של התהליך שרץ כרגע על מנת להשתמש רק במיפויים שלו
- אי לכך, המעבד משתמש רק במיפויים שמערכת ההפעלה יצרה עבור התהליך שרץ כרגע; תהליך לא יכול להתייחס לכתובות פיזיות כלל, ולא יכול להתייחס למסגרות שאינן ממופות לדפים שלו

הגנה על זיכרון (המשך)

- כל כניסה בטבלת הדפים כוללת מספר סיביות שמתארות הרשאות גישה
  - האם מותרת גישה לדף במצב משתמש או רק במצב מיוחס?
  - האם מותרות קריאה, כתיבה, ושימוש בתוכן כפקודות מכונה?
- היכולת לאסור גישה במצב משתמש מאפשרת למערכת ההפעלה למפות את הקוד ומבני הנתונים שלה לכל התהליכים בלי לאפשר לתוכנית שהתהליך מריץ לגשת לזיכרון הזה
- מערכת ההפעלה ממופה בדרך כלל לכל התהליכים באותו מקום
- היכולת לאסור סוגי גישה מסוימים, יחד עם העובדה שהדפים הראשונים לעולם אינם ממופים, עוזרת לגלות שגיאות בתוכניות (התייחסות דרך מצביע **null**, שכתוב קוד, ...)

חריגי דף (page faults)

- המעבד מזהה גישות לדפים לא ממופים, לדפים שממופים למסגרת בדיסק, ולדפים שהרשאותיהם אינן מתירות את סוג הגישה או אינן מתירות גישה במצב משתמש
- המעבד מייצר חריג (exception) בשם חריג דף שגורם להפעלת שגרה של מערכת ההפעלה, בדומה לטיפול בפסיקות
- השגרה של מערכת ההפעלה מזהה את הסיבה לחריג ומגיבה
- אם צריך לקרוא מסגרת מהדיסק, מערכת ההפעלה משעה את התהליך, מביאה את המסגרת, מעדכנת את טבלת הדפים, וחוזרת לתהליך; הפקודה שגרמה לחריג תרוץ שוב
- רוב המקרים האחרים גורמים להעפת התהליך או הפעלת שגרה מיוחדת של התהליך (שגרה לטיפול באיתות, signal)

סיבות לחריגי דף והטיפול בהם

- דף שכלל אינו מוקצה לתהליך
- דף מוקצה בתהליך אך אינו ממופה למסגרת פיזית
- הרשאות לא מספיקות לסוג הגישה הנדרש
- במצב מיוחס: דף של התהליך שאינו מוקצה, מוקצה אך אינו ממופה, או מוקצה וממופה אך ללא הרשאות לסוג הגישה (בדרך כלל החריג הוא תוצאה של גישה של מערכת ההפעלה לארגומנט של קריאת מערכת שכתובתו הועברה לקריאה)
- במצב מיוחס בגלל דף של מערכת ההפעלה: שגיאה בתהליך (ארגומנט לא נכון לקריאת מערכת) או תקלה במערכת ההפעלה

התייחסות לכתובות פיזיות

- מערכת ההפעלה צריכה להתייחס לכתובות פיזיות ולהקצות מערכים בזיכרון פיזי רצוף בשביל טבלאות דפים וזיכרון שגם בקרים ניגשים אליו, כמו חוצצים שבקר DMA מעביר מהם/אליהם נתונים
- דרך פשוטה להתייחס לכתובות פיזיות היא למפות את כל הזיכרון הפיזי או חלקו לזיכרון הוירטואלי של כל התהליכים
- דוגמה: מיפוי 2 GB של זיכרון פיזי לחצי העליון של מרחב כתובות של 32 סיביות: כתובת פיזית X ממופה לכתובת וירטואלית $X+2^{31}$
- רשומות מיפוי לדפים גדולים (למשל 4 MB) מייעלות את זה

פינוי מסגרות

- מערכת ההפעלה צריכה מאגר של מסגרות פנויות שניתן להקצות במהירות עבור תהליכים או עבור מערכת ההפעלה עצמה.
- מסגרות נקיות (שיש עותק עדכני שלהן בדיסק) ניתן לפנות במהירות
- מסגרות מלוכלכות, שאין עותק עדכני שלהן בזיכרון משום ששונו מאז שנקראו או שמעולם לא נכתבו לדיסק, יש צורך לשמור בדיסק לפני פינויין
- כניסות בטבלת הדפים וה–TLB כוללות סיבית ניקיון/לכלוך

מנגנונים לפינוי מסגרות

- המנגנון משתמש בשלושה מאגרים (רשימות) של מסגרות
  - מסגרות מלוכלכות מועמדות לניקוי
  - מסגרות נקיות מועמדות לפינוי
  - מסגרות פנויות וריקות
- תהליך מיוחד מתעורר מדי פעם, מוצא מסגרות ראויות לפינוי, מוחק אותן מטבלת הדפים המתאימה, ומעביר לרשימה המתאימה
- תהליך אחר שגם הוא מתעורר מדי פעם כותב לדיסק מסגרות מלוכלכות מועמדות לפינוי ומעביר אותן לרשימת הנקיות
- תהליך שלישי (פחות חיוני) מאפס מדי פעם מסגרות נקיות ומעביר אותן למאגר הפנויות

הצלת מסגרות מועמדות לפינוי

- בזמן טיפול בחריג דף מערכת ההפעלה בודקת האם הדף (שאינו ממופה) שמור במסגרת באחד משני ממאגרי המועמדות לפינוי
- במצב כזה מערכת ההפעלה מוציאה את המסגרת ממאגר המועמדות לפינוי ומשחזרת את המיפוי בטבלת הדפים, ללא גישה לדיסק
- זו הסיבה שכדאי להחזיק מאגר גדול של מועמדות נקיות אבל לא לפנות אותן

קביעת קצב הפינוי

- התהליכים המיוחדים ומפנים כמות מסגרות שנקבעת על פי מצב המערכת
- כאשר יש מעט מסגרות נקיות/פנויות, התהליכים יפנו מספר גדול כל אימת שהם מתעוררים
- כאשר יש הרבה מסגרות נקיות, התהליכים יפנו מעט או שלא יתעוררו כלל
- הפרמטרים שקובעים את קצב הפינוי תלויים בגודל הזיכרון הפיזי והם ניתנים לכיוונון

מדיניות לבחירת קורבנות לפינוי

- מיטבי: מספר חריגי הדף ממוזער אם בכל חריג דף מפנים את המסגרת שהשימוש הבא בה רחוק ביותר
- לא מעשי כי מערכת ההפעלה אינה יודעת לאיזה מסגרות ייגשו בעתיד וגם משום שרצוי להחזיק מאגר של פנויות
- מיטבי במודל מתמטי שאינו מתיר הצצה לעתיד: **least recently used** (LRU), מפנים את המסגרת שהשימוש האחרון בה היה רחוק ביותר בעבר
- מדיניות LRU פועלת היטב בפועל משום שההיסטוריה חוזרת
- קשה למימוש כי דרושות חותמות זמן שימוש ב–TLB ובטבלת הדפים וצריך למצוא את הישנה ביותר בכל טבלאות הדפים

קירובים ל-LRU

- מעבדים אינם תומכים בחותמות זמן שימוש מדויקות למסגרות
- מעבדים כן מממשים חותמת שימוש בת סיבית אחת שמודלקת אוטומטית בכל שימוש; צריך לאפס אותה מפורשות
- כיבוי של הסיבית הזו מדי פעם בכל מיפוי ובחירה של מועמדות שהסיבית שלהן עדיין כבויה מבטיחה שלא נעשה בהן שימוש לאחרונה
- על ידי שמירת הסיבית הזו באוגר הזזה מדי פעם ניתן לקבל קירוב מדויק יותר ל–LRU
- ניתן לסמלץ סיבית כזו במעבד שאינו תומך בה בעזרת מנגנון חריגי הדף

מדיניות פינוי דו-שלבית

- במערכות עם טבלת דפים לכל תהליך (היררכית למשל), קשה לסרוק את כל המסגרות על מנת לחפש כאלה שלא היו בשימוש לאחרונה
- מערכות הפעלה בוחרות מסגרות לפינוי בשני שלבים: תהליך שמסגרות שלו יפונו, ואז מסגרות ספציפיות
- מדיניות דו –שלבית כזו גם מאפשרת לממש עקרונות של הגינות בין תהליכים, למשל לתגמל תהליכים שמשתמשים במעט מסגרות

בחירת תהליך בפינוי מסגרות

- בלינוקס: מדיניות פשוטה, התהליך שגרם למספר חריגי הדף הקטן ביותר בזמן האחרון
- גרסאות ישנות יותר של לינוקס: התהליך עם מספר המסגרות הגדול ביותר
- חלונות: מדיניות working sets:
  - מערכת ההפעלה משערכת את מספר המסגרות שכל תהליך זקוק להן, כתלות בקצב חריגי הדף שלו ותוך כיבוד חסמים עליון ותחתון
  - התהליך שייבחר לפינוי מסגרות הוא התהליך שחורג ממספר המסגרות שהוא "זכאי" להן במידה הגדולה ביותר
  - שערוך חוזר מדי פעם

אזורי דפדוף

- מסגרות שאין להן מקום בזיכרון הראשי מועברות לאזור דפדוף בדיסק
- אזור הדפדוף יכול להיות קובץ רגיל (טיפוסי בחלונות), מחיצה (טיפוסי בלינוקס), או אפילו מספר קבצים, מחיצות ודיסקים שלמים (אפשרי בלינוקס)
- בלינוקס ניתן להגדיר עדיפות לכל אזור דפדוף, ולהוסיף ולהסיר אזורים בזמן שהמערכת פועלת
- בכל מערכות ההפעלה ניתן להקצות בלוק של זיכרון וירטואלי שממופה לקובץ מסוים

מיפוי קבצים לזיכרון

- מאפשר לקרוא קובץ פשוט על ידי מיפויו וגישה לזיכרון הממופה
- במיפוי **משותף** כתיבה לזיכרון הממופה משנה את תוכן הקובץ (מאפשר להעביר מידע בין תהליכים)
- במיפוי **פרטי** כתיבה לזיכרון יוצרת עותק פרטי של המסגרת, ואם יהיה צריך לפנות אותה היא תפונה לאזור דפדוף, לא חזרה לקובץ הממופה
- מיפוי קבצים משמש לטעינה של קוד מכונה של תוכניות וספריות שגרות לזיכרון
- מיפוי משותף של תוכניות וספריות חוסך מסגרות פיזיות

פרק 4
תהליכים ותזמון מעבדים

תהליכים

- תהליך הוא מחשב וירטואלי שמריץ תוכנית אחת עבור משתמש אחד
- תהליך יכול לרוץ על מעבד פיזי או להיות מושעה
- מערכת ההפעלה משעה תהליך כאשר הוא מחכה למשאב כלשהו (למשל לבלוק נתונים מקובץ או לטיפול בחריג דף) וכאשר צריך להריץ תהליך אחר במקומו

מבנה נתונים לייצוג תהליך

- מצב המעבד: ערכים של האוגרים, כולל מצביע התוכנית, כדי שאפשר יהיה להחזיר תהליך מושעה לריצה
- מפת זיכרון וירטואלי (טבלת דפים)
- טבלת שגרות איתות (signals; רק ביוניקס/לינוקס) שיופעלו כאשר מאותתים לתהליך או בזמן אירוע חריג. בחלונות יש מנגנון שונה לטיפול בחריגים
- זהות המשתמש המריץ והרשאות
- טבלת משאבים זמינים: קבצים פתוחים, קשרים פתוחים לתהליכים אחרים, וכדומה
- מצב תזמון וסטטיסטיקות

טבלת משאבים זמינים

- תהליך שמבקש משאב (גישה לקובץ, ערוץ תקשורת למחשב אחר, גישה להתקן חומרה) מקבל ממערכת ההפעלה מזהה משאב (handle בחלונות, file descriptor בלינוקס/יוניקס)
- המזהה הוא למעשה אינדקס בטבלת המשאבים של התהליך ששמורה בזיכרון שנגיש רק ממצב מיוחס
- כל איבר בטבלה של תהליך מכיל הרשאות (קריאה/כתיבה וכולי) ומצביע למבנה נתונים שמייצג את המשאב עצמו בזיכרון של מערכת ההפעלה
- ההצבעה הכפולה מונעת מתהליכים לזייף מזהה משאב
- חלק ממנגנון ההגנה של מערכת ההפעלה

מיתוג תהליכים

- כך מערכת ההפעלה מעבירה מעבד פיזי מתהליך אחד לאחר:
  - המעבד עובר מקוד של התוכנית לשגרה של מערכת ההפעלה כתוצאה מקריאת מערכת או פסיקה (כולל פסיקת שעון)
  - מערכת ההפעלה מטפלת באירוע, כלומר מבצעת את השירות שקריאת המערכת ביקשה או מטפלת בפסיקה
  - מערכת ההפעלה מחליטה שצריך להעביר את המעבד לתהליך אחר
  - מערכת ההפעלה שומרת את מצב המעבד במבנה הנתונים של התהליך כדי שניתן יהיה להמשיך את הרצתו
  - מערכת ההפעלה משחזרת את מצב המעבד של התהליך שצריך לרוץ
- זהו למעשה מנגנון מיפוי של מעבדים לתהליכים; אנלוגי למנגנון המיפוי של דפים למסגרות

תהליכים לעומת חוטים (תהליכונים)

- חוט הוא מעבד וירטואלי
- תהליך מרובה חוטים מדמה מחשב וירטואלי מרובה מעבדים
- כל החוטים של תהליך משתפים את כל השדות במבנה הנתונים של התהליך פרט למצב המעבד וחלק ממצב התזמון
- בפרט, כל החוטים משתמשים באותה מפת זיכרון
- לכל חוט יש מחסנית משלו
- חוטים של תהליך עשויים לרוץ במקביל במחשב מרובה מעבדים או בזה אחר זה
- במיתוג בין חוטים של תהליך אין צורך להחליף את טבלת הדפים

תהליכי (חוטי) גרעין

- תהליכים של מערכת ההפעלה עצמה שניגשים רק למבני הנתונים שלה ולא לזיכרון של תהליך כלשהו
- מבני הנתונים של מערכת ההפעלה ממופים בצורה זהה לכל התהליכים ולכן תהליך גרעין יכול לרוץ עם כל טבלת דפים
- שדה טבלת הדפים במבנה הנתונים שלהם ריק
- בזמן מיתוג אליהם מערכת ההפעלה לא מחליפה טבלת דפים
- מיתוג כזה חוסך החלפה אחת או שתיים של טבלאות דפים
- תהליכי גרעין מועילים כאשר מעונינים שמטלות של מערכת ההפעלה יתחרו על משאבי מעבד עם תהליכים רגילים
- בלינוקס משמשים לפינוי מסגרות, למשל

יצירת תהליכים בלינוקס/יוניקס

- קריאת המערכת **fork** יוצרת תהליך חדש שהוא תאום זהה לתהליך הקורא פרט למספר התהליך; הקריאה חוזרת פעמיים
- לכל אחד מהתהליכים יש מרחב זיכרון זהה אבל פרטי
- מנגנון **copy-on-write** מאפשר לשכפל מרחב זיכרון בלי להעתיק את כל המסגרות:
  - הרשאת הכתיבה מוסרת מכל הדפים בשני התהליכים
  - ניסיון כתיבה גורם לחריג דף, מערכת ההפעלה מאבחנת את הסיבה, מעתיקה את המסגרת, ומשחזרת את הרשאת הכתיבה המקורית
- בחלונות תהליך חדש תמיד מריץ תוכנית מתחילתה

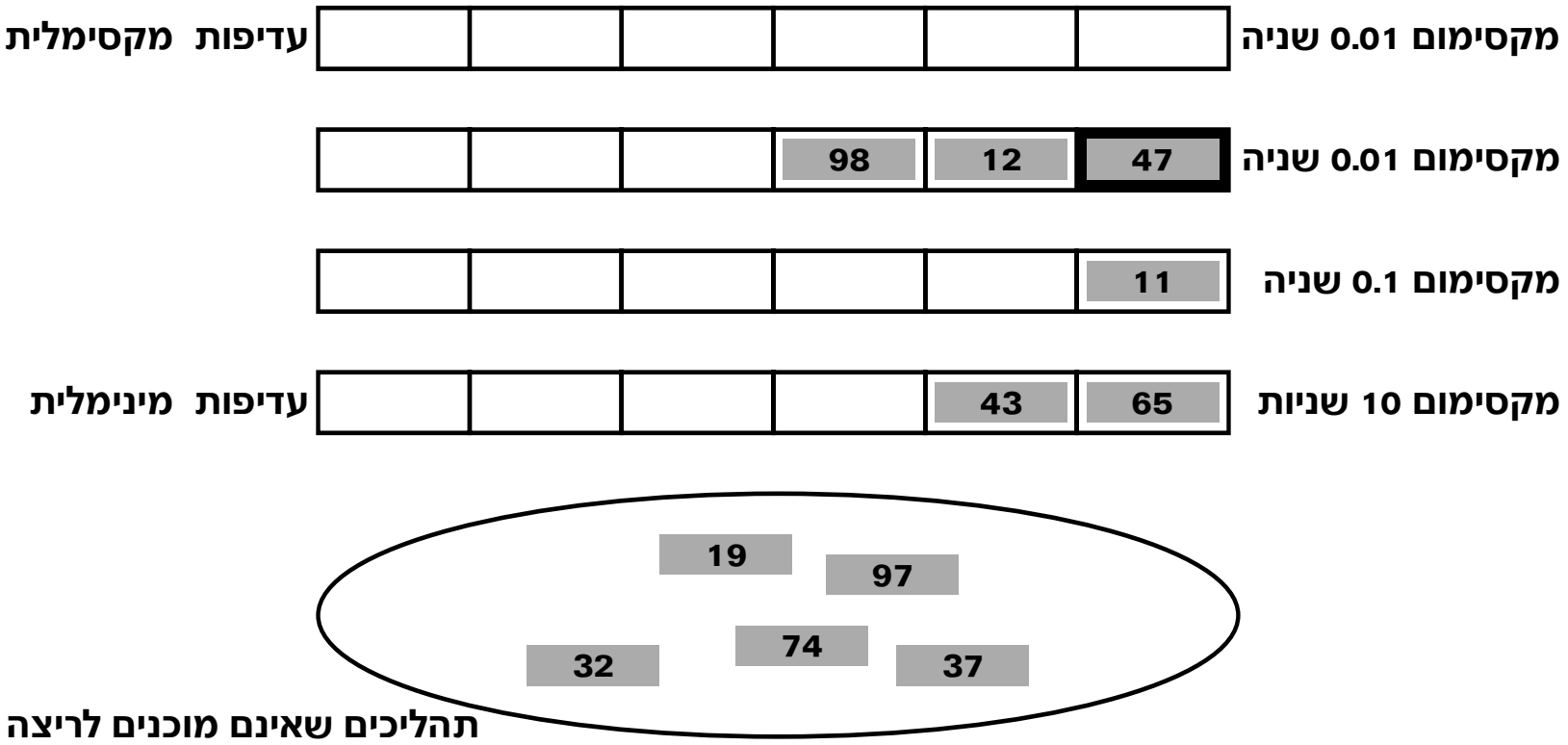
מדיניות לתזמון מעבדים: מטרות

- שימוש יעיל במעבדים פיזיים: כדאי להמעיט במיתוג תהליכים
- זמן תגובה ראוי:
  - לתוכנית אינטראקטיבית, זמן תגובה שייצור תחושה של תגובה מיידית בכפוף למגבלות החושיות של בני אדם
  - לתוכניות ששולטות על מערכות פיזיות (מטוס, מפעל), זמן תגובה שמאפשר שליטה (לפני שהמטוס מתרסק)
  - לסימולציית מזג אויר, סיום לפני מהדורת החדשות
  - ...
- תוכניות שגרתיות (לא זמן אמת) אינן מכריזות על זמן תגובה נדרש לכל פעולה ולכן מערכת ההפעלה מסתפקת בתזמון הגון תוך התייחסות כללית לעדיפויות

מטרות תזמון משניות

- עמידה בלוחות זמנים: במערכות זמן אמת בלבד; מערכות הפעלה כלליות הן מורכבות מכדי להבטיח עמידה בלוחות זמנים (למשל בגלל חריגי דף)
- תזמון דביק: במחשב מרובה מעבדים, רצוי להריץ תהליך על המעבד שהריץ אותו לאחרונה, על מנת למרב את התועלת מה–TLB וזיכרונות המטמון; במחשב עם זיכרון מבוזר, כדאי להריץ תהליך על מעבד שקרוב למסגרות של התהליך
- תזמון כנופיות: כשמחשב מרובה מעבדים מריץ תוכניות מרובות חוטים, כדאי לתזמן הרצת כל החוטים של תהליך בבת אחת

מבני נתונים לתזמון מעבד: multilevel feedback queues



- התהליך בראש התור הגבוהה ביותר רץ
- מבנה הנתונים כולל כללים למעבר בין תורים

מעבר בין תורים

- מעבר בין תורים:
  - תהליך שצבר זמן ריצה רב עובר לתור נמוך יותר
  - תהליך שלא רץ זמן רב עובר לתור גבוה יותר, למניעת הרעבה
  - תהליך שחיכה למשאב וקיבל אותו נכנס לתור גבוה מאוד כדי שישחרר את המשאב מהר ככל האפשר
- העדיפות שאנו מעניקים לתהליכים משפיעה על הפרמטרים של מעבר בין תורים ולא קובעת תור הספציפי לתהליך
- קל לממש מדיניות הוגנת ויעילה
- קל לממש יכולות מיוחדות, כגון תהליכים שרצים רק שאף תהליך אחר לא מוכן לריצה

פרק 5
תכנות תהליכים בו-זמניים

מטרות בתכנות בו-זמני

- רצון לנצל מספר מעבדים פיזיים במחשב
- רצון לנצל את המעבד בזמן שהתוכנית ממתינה להתקן איטי, כגון קריאה מהדיסק
- מתן אשליה למשתמש שמספר פעולות קורות במקביל, כמו טיפול בקלט בזמן שהתוכנית ממשיכה להציג אנימציה או קול ברצף, גם אם יש מעבד אחד
- מימוש שרתים שבהם כל לקוח מטופל על ידי חוט נפרד
- פרט לראשונה, דוגמאות למודולריזציה טמפורלית : רצון להפריד מודולים שיכולים לרוץ בכל מיני סדרי ביצוע

תיאום

- כל חוט מבצע פעולות אחרות
- אם ניתן לבצע את הפעולות של חוטים שונים במקביל או בכל סדר שהוא, שגר ושכח
- ברוב המקרים, יש לכפות אילוצים על סדר הפעולות על מנת להבטיח נכונות
- דוגמה: שני חוטים שכל אחד מבצע i=i+1
- אם הראשון קורא את i לאוגר, מקדם את האוגר, השני קורא את i לאוגר, מקדם את האוגר, הראשון כותב את i לזיכרון, השני כותב את תוכן האוגר ל-i, המשתנה קודם ב-1 ולא ב-2

מנעולים

- מנעול הוא עצם שמבטיח **מניעה הדדית**
- שני מצבים: חפשי ונעול
- ניסיון נעילה של מנעול חפשי מצליח ושגרת הנעילה חוזרת
- לחוט מותר לשחרר רק חוט שנעול על ידו
- ניסיון נעילה של מנעול נעול ממתין לשחרורו
- חוט שבידו מנעול (נעול) מונע מחוטים אחרים לנעול אותו :

```
lock(m)
i=i+1
unlock(m)
```

אירועים

- מנגנון שבעזרתו חוט יכול לחכות לקבלת הודעה על אירוע מחוט אחר
- המנגנון שיתואר תקף בלינוקס/יוניקס, בחלונות מנגנון דומה אך שונה בפרטים
- **`wait(c,m)`** משחררת את המנעול **`m`** ונכנסת להמתנה לאירוע **`c`** **באופן אטומי**
- **`signal(c)`** משחררת את אחד החוטים שממתין ל-**`c`**
- **`broadcast(c)`** משחררת את כל החוטים שממתינים ל-**`c`**
- ל-**`signal`** ו-**`broadcast`** אין שום השפעה אם אין חוטים ממתינים
- אחרי התעוררות ולפני חזרה מ-**`wait`** המנעול ננעל שוב, אולי לאחר המתנה (לא אטומי)

דוגמה: קורא וכותב


```
forever
  lock(m)
  while (state==full)
    wait(c,m)
  כתוב לחוצץ
  signal(c)
  unlock(m)
  פעולות אחרות
```

```
forever
  lock(m)
  while (state==empty)
    wait(c,m)
  קרא מהחוצץ
  signal(c)
  unlock(m)
  פעולות אחרות
```

למה while ולא if?

```
forever
   lock(m)
   while/if (state==full)
      wait(c,m)
   כתוב לחוץ
   signal(c)
   unlock(m)
   פעולות אחרות
```

- מבטיח שחוט לא יבצע פעולה אלא אם התנאי מתקיים
- שימוש ב-if היה עלול לגרום לביצוע הפעולה אם הודעה על האירוע לא תמיד מבטיחה שהתנאי מתקיים
- if נכון כאן, אבל עלול לגרום שגיאות בתוכניות מורכבות
- while מאפשר בדיקת נכונות מקומית, if מצריך בדיקה גלובלית

מספר קוראים וכותבים

- כאשר יש מספר חוטים שכותבים וקוראים, **signal** עלול להעיר חוט שהתנאי שהוא מחכה לו לו מתקיים (כותב מעיר כותב אחר)
- החוט שהתעורר יגלה שהתנאי לא מתקיים ויחזור לישון
- האירוע אבד וחוט שמחכה לתנאי שכן מתקיים עלול שלא להתעורר לעולם
- **broadcast** פותר את הבעיה כי כל הממתינים מתעוררים
- עלות גבוהה יותר מ-**signal**

```
forever
   lock(m)
   while (state==full)
      wait(c,m)
   כתוב לחוצץ
   broadcast(c)
   unlock(m)
   פעולות אחרות
```

פתרון יותר יעיל עם 2 אירועים

```
forever
  lock(m)
  while (state==full)
    wait(c_e,m)
  כתוב לחוצץ
  signal(c_f)
  unlock(m)
  פעולות אחרות
```

```
forever
  lock(m)
  while (state==empty)
    wait(c_f,m)
  קרא מהחוצץ
  signal(c_e)
  unlock(m)
  פעולות אחרות
```

יעיל יותר ונכון

קיפאון

חוט א׳:

```
lock(m1)
lock(m2)
unlock(m2)
unlock(m1)
```

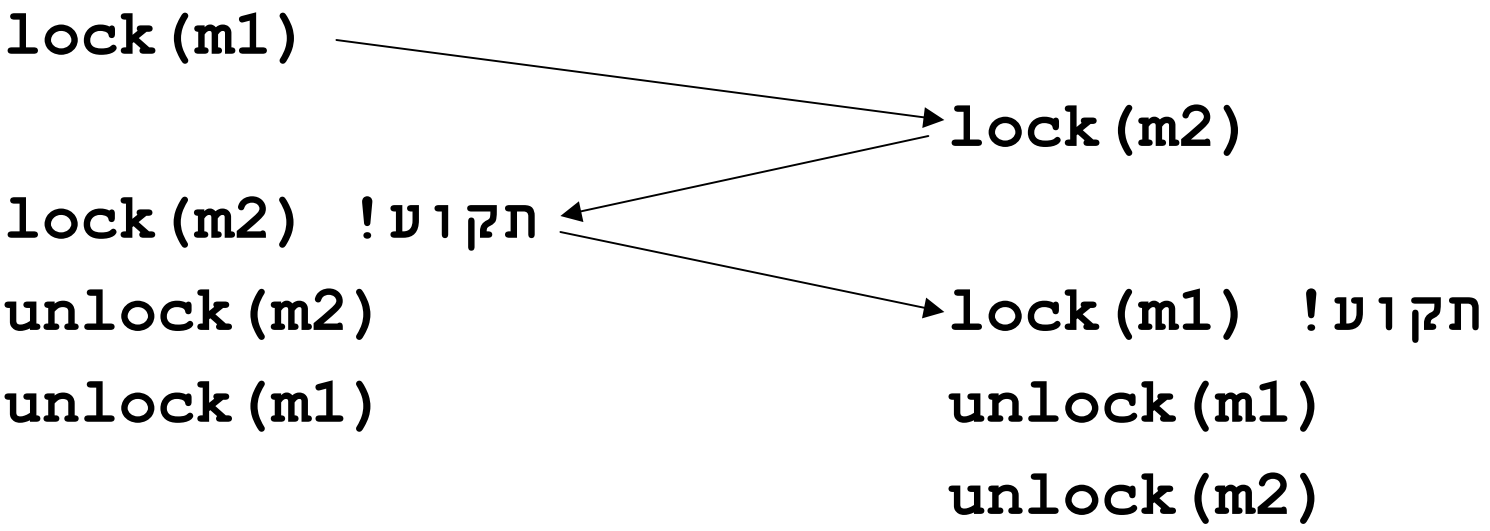
חוט א׳:

```
lock(m1)
lock(m2)
unlock(m2)
unlock(m1)
```

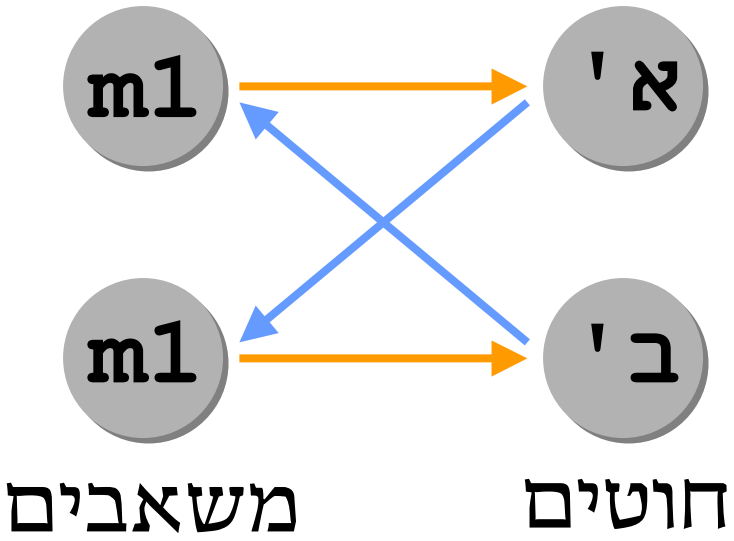
קיפאון: דוגמה לתזמון

חוט ב׳:

חוט א׳:



מודל פורמלי: גרף דו-צדדי



חוט מחכה למשאב

חוט נועל משאב

הימנעות מקיפאון

- אם חוט נועל מנעול אחד לכל היותר בכל נקודת זמן
- אם ניתן להגדיר סדר חלקי על משאבים וחוט תמיד נועל לפי סדר:
  - סדר שכבות התוכנה אם בכל שכבה נועלים משאב אחד לכל היותר
  - סדר איברים ברשימה (לא דו–צדדית)
  - סדר איברים בעץ מהשורש לעלים או מהעלים לשורש
  - סדר שרירותי (כתובות של משאבים, למשל) אם ניתן לנעול בכל סדר
- לא תמיד קל להגדיר סדר ולהבטיח נעילה לפי סדר (למשל כאשר משתמשים בתור עדיפויות ממומש כ–heap)
- חוטים עלולים להיכנס לקיפאון לא רק בגלל שימוש במנעולים אלא גם בגלל שימוש במשאבים בלעדיים אחרים או אירועים

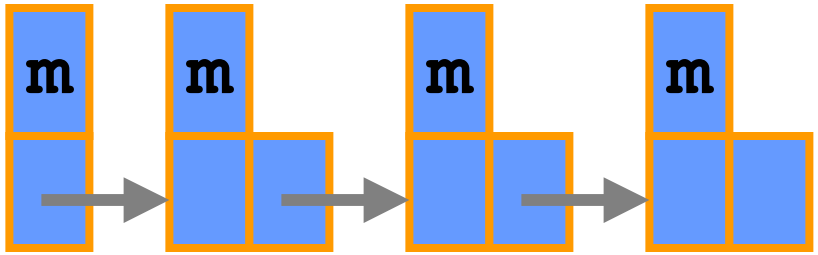
היחלצות מקיפאון

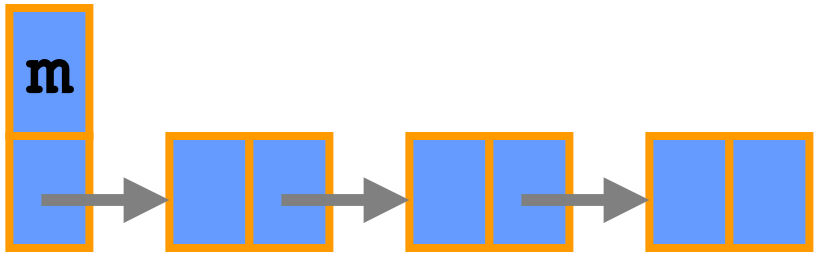
- מערכות מסוימות מסוגלות לזהות מעגל בגרף, להעיף את אחד החוטים, ועל ידי כך לשבור את הקיפאון
- לא ישים במערכות הפעלה, כיון שבדרך כלל תוכנית לא מסוגלת להמשיך לרוץ כאשר אחד החוטים שלה עף
- מערכות מסדי נתונים מעיפות את אחד החוטים הקפואים או כאשר יש חשד לקיפאון (אי שחרור משאב תוך פרק זמן סביר)
- גם בחלונות וגם ביוניקס/לינוקס ניתן להגביל את זמן ההמתנה לאירוע (בחלונות גם למנעול)
- עדיף לכתוב תוכנית שלא יתכן בה קיפאון!

אבחון וניפוי קיפאון

- קיפאון הוא תקלה נפוצה בתוכניות בו–זמניות
- קיפאון הוא גם תקלה קלה לאבחון: החוטים נתקעים וניתן לבחון את הסיבה לקיפאון (המעגל בגרף) במנפה
- בגלל שהחוטים נתקעים הקיפאון קורה בדרך כלל קרוב לשגיאה בתוכנית, להבדיל מסוגי תקלות אחרות (בעיות זיכרון)
- ריצה ללא קיפאון אינה מבטיחה, כמובן, חיות בכל ריצה

גרעיניות





- מנעול אחד לרשימה שלמה
- חוט אחד לכל היותר מטפל ברשימה בזמן נתון
- נעילה אחת לכל פעולה על הרשימה

- מנעול לכל איבר ברשימה
- מספר חוטים יכולים לטפל ברשימה בו–זמנית
- נעילה בכל מעבר באיבר; יותר תקורה בגלל בו–זמניות

אין פתרון מיטבי גלובלי; הפתרון העדיף תלוי ביישום

הגינות: מנעולי קוראים/כותב

- להדגמת נושא ההגינות נממש מנעול קוראים/כותב
- צריך לאפשר או למנוע גישה למבנה נתונים (כמו מנעול)
- ארבע פעולות: נעל/שחרר לקריאה, נעל/שחרר לכתיבה
- מותר למספר בלתי מוגבל של קוראים לגשת למבנה הנתונים בו–זמנית בלי כותבים, או לכותב אחד בלבד בלי קוראים
- (ממומש כחלק מממשק החוטים של לינוקס ויוניקס)

פתרון פשוט

```
read_lock(rw)
  lock(rw.m)
  while (rw.i < 0)
    wait(rw.c,rw.m)
  rw.i++
  unlock(m)

read_unlock(rw)
  lock(rw.m)
  rw.i--
  if (rw.i == 0)
    signal(rw.c)
  unlock(rw.m)
```

```
write_lock(rw)
  lock(rw.m)
  while (rw.i != 0)
    wait(rw.c,rw.m)
  rw.i-- /* == -1 */
  unlock(m)

read_unlock(rw)
  lock(rw.m)
  rw.i++ /* == 0 */
  broadcast(rw.c)
  unlock(rw.m)
```

הפתרון אינו הוגן

- גם אם חוטים נועלים מנעולים ומקבלים אירועים לפי סדר הכניסה להמתנה (לא בהכרח נכון), הפתרון אינו הוגן
- כל זמן שיש קוראים, קוראים נוספים יצליחו לנעול וכותבים ימשיכו להמתין
- אם קוראים ממשיכים להגיע כל הזמן, הם ירעיבו את הכותבים
- יש מספר פתרונות הוגנים אבל צריך להגדיר את הדרישות
- למשל, המנעול עובר לסירוגין בין קוראים וכותב (אלא אם אין חוטים מהסוג השני), קוראים שנכנסים להמתנה בזמן שאחרים קוראים מחכים לסיבוב הבא

גם העצמים הבסיסיים אינן הוגנים

- מערכות הפעלה מספקות מנעולים ואירועים שמונעים הרעבה אבל אינם מבטיחים, בדרך כלל, שירות לפי סדר ההמתנה
- סיבה 1: הגינות מוחלטת קשה למימוש ועלולה לפגוע ביעילות
- סיבה 2: במחשב מרובה מעבדים מספר חוטים עשויים להיכנס להמתנה בו–זמנית ממש; אין סדר בין בקשות השירות
- כפי שראינו, גם אם מנעולים ואירועים היו הוגנים, תוכניות מורכבות עלולות להיות לא הוגנות ואף להרעיב חוטים
- בדרך כלל כדאי להניח שהעצמים הבסיסיים מונעים הרעבה ותו לא, ולממש הגינות בתוכנית עצמה אם יש צורך בכך

שיקולי יעילות

- עדיף להימנע מלהעיר חוטים שנכנסים מייד להמתנה (כלומר משימוש ב–broadcast במקום באירועים ספציפיים יותר)
- חלונות תומכים בשני סוגי מנעולים: משתני מניעה הדדית וקטעים קריטיים
  - משתנה מניעה הדדית ניתן לשימוש מתהליכים שונים ויש לו תקורה גבוהה של קריאת מערכת
  - בקטע קריטי ניתן להשתמש רק בחוטים של תהליך אחד ויש לו תקורה נמוכה, לפחות כאשר המנעול חפשי
- ביוניקס ולינוקס מנעולים ואירועים נגישים רק לתהליך בודד וניתן לממש אותם ביעילות

עדיפויות והיפוכן

- מערכות מסוימות תומכות בעדיפויות לחוטים
- לעיתים המשמעות של עדיפות היא שחוט בעדיפות גבוהה לעולם לא ימתין למעבד שמריץ חוט בעדיפות נמוכה
- היפוך עדיפויות:
  - חוט בעדיפות נמוכה נועל משאב
  - חוט בעדיפות בינונית רץ לאורך זמן רב
  - חוט בעדיפות גבוהה מתעורר, מבקש את המשאב הנעול, ונכנס להמתנה; למעשה החוט הזה ממתין שהחוט בעדיפות בינונית יסיים
- על מנת למנוע היפוך עדיפויות, חוט שנועל משאב צריך לרשת את העדיפות המירבית של כל מי שמחכה למשאב

אירועים בחלונות

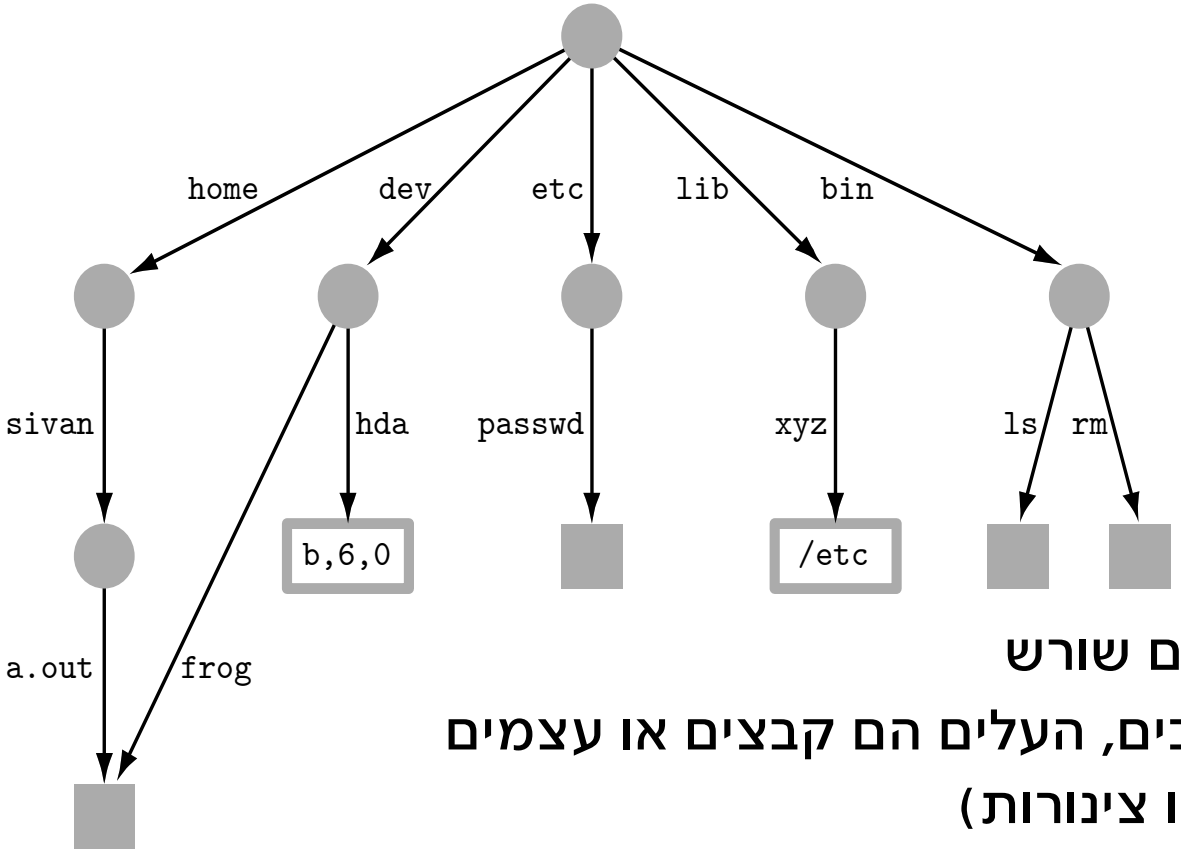
- שני סוגי אירועים: איפוס ידני ואוטומטי
- שני הסוגים הם משתנים בינריים מיוחדים וניתן לקבוע את ערכם ל–1 או 0
- המתנה לאירוע מחכה שערכו יהיה 1 אם איננו כזה, ואינה ממתינה אם ערכו 1; אינה משחררת או נועלת כל מנעול
- השתחררות של חוט מהמתנה לאירוע אוטומטי מאפסת אותו, כך שרק חוט אחד מתעורר; האירוע זוכר שערכו 1 אם אין ממתינים ברגע שבו מעלים את ערכו
- אירוע ידני מאופס רק באופן מפורש

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 ותקליטורים)
  - במערכות הפעלה מסוימות (מקינטוש, חלונות 2000/NT) קובץ יכול להיות מורכב ממספר רצפים; אנו נתעלם מאפשרות זו, שמייצגת בעיקרון סוג שונה מעט של מדריך.

מרחב השמות



- עץ או גרף חסר מעגלים עם שורש
- צמתים פנימיים הם מדריכים, העלים הם קבצים או עצמים אחרים (התקנים למשל, או צינורות)
- 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 2000/NT

- 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 האחוד הזה מתפענחים כאילו התחילו ב- "\??" ועצמים נדיפים כאילו התחילו ב-"\BaseNamedObjects"

פעולות על מרחב השמות

- יצירת עלה מסוג מסוים ומצביע אליו ממדריך כלשהו
- יצירת מדריך ומצביע אליו
- יצירת מצביע נוסף לעצם קיים
- יצירת מצביע סימבולי
- מחיקת מצביע; העצם נמחק כאשר אין מסלול מהשורש אליו ואינו בשימוש על ידי תהליך
- שינוי הרשאות גישה לעצם
- שתילת ועקירת מערכת קבצים שלמה בצומת מסוים
- פענוח שם למזהה פנימי והודעה למערכת ההפעלה שהעצם בשימוש (**open**), והודעה על סיום שימוש בעצם (**close**)
- קריאת תוכן מדריך או חיפוש במדריך

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 במדריך

מצביעים סימבוליים

- עלים במרחב השמות שמכילים שם של עצם אחר
- כאשר מנגנון הפענוח מגיע למצביע סימבולי, הערך של המצביע משורשר לסיומת השם המתפענח, 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 2000: עצם ששמור בדיסק (נקודת פענוח)
- 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

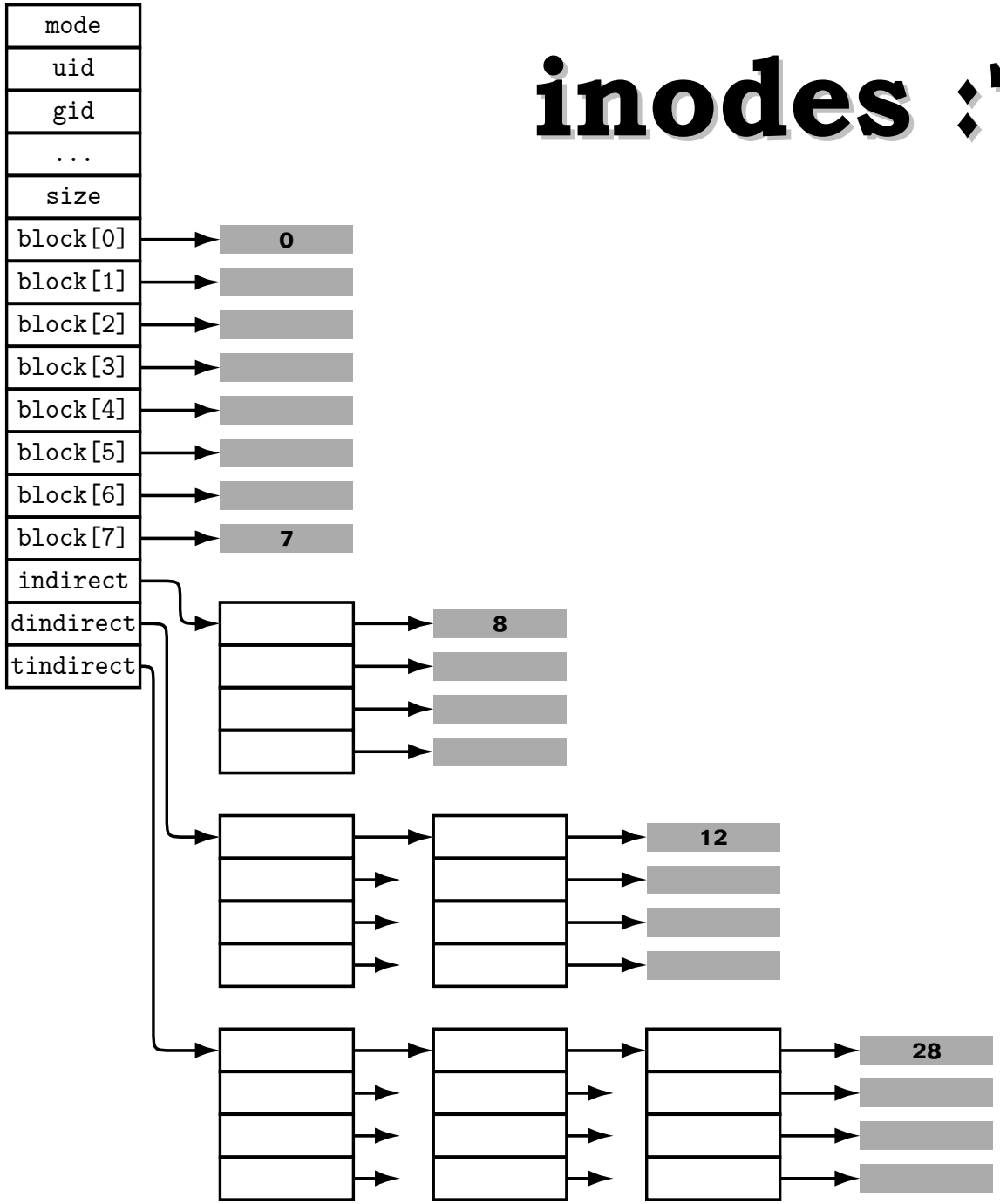
סמנטיקה של גישה לקבצים

- קונסיסטנטיות בגישה ממספר תהליכים:
  - כתיבות וקריאות הן אטומיות, כלומר ניתן לסדר את כל הכתיבות והקריאות בסדר שלם וקריאה תמיד רואה את הקובץ לאחר הכתיבה האחרונה; פעולות לא מתבצעות בצורה הדרגתית
  - כתיבה לקובץ נראית מייד לתהליכים אחרים
- עמידות 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 חיצוני); לפעמים אפשר לדחוס כמה זנבות לקובץ אחד

מנגנון מיפוי: inodes



מיפוי קבצים ב-NTFS

- 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
- קובץ מיוצג על ידי מספר הבלוק הראשון שלו
- בכל איבר בטבלת ההקצאה שמור מצביע לבלוק הבא של הקובץ
- הבלוקים הפנויים משורשרים לרשימה גם הם
- מיפוי בקובץ ארוך דורש זמן פרופורציוני למספר הבלוקים בקובץ (מעבר על רשימה מקושרת)
- הקובץ בדוגמה שמור בבלוקים 54, 23, 47

```
ex.doc → 54

0:
1:
…
23: 47
…
47: -1
…
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 ספריה
- 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

- 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

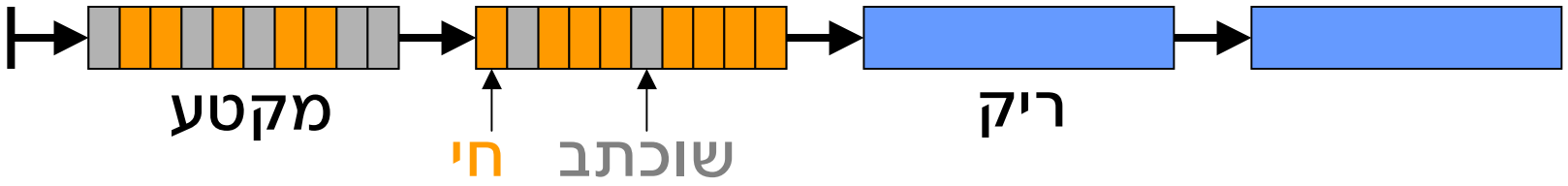
FFS: פרטי ההקצאה

- 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

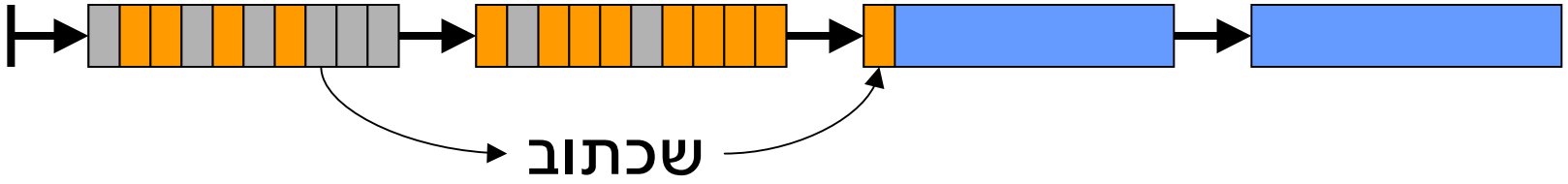
- ביצועים מצוינים בסביבות שבהן יש הרבה העברת נתונים מכיון שהזרוע זזה מעט (הקצאה בקבוצות גלילים) ומכיון שלא ממתינים הרבה לסיבוב הדיסק (הקצאה מיטבית של בלוקים והעברה של אשכולות שלמים
- ביצועים פחות טובים בסביבות שבהן יוצרים ומוחקים קבצים בתכיפות: להקצאת בלוקים אין השפעה אבל תכונות אחרות של FFS מגבילות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

גישה אחרת: LFS



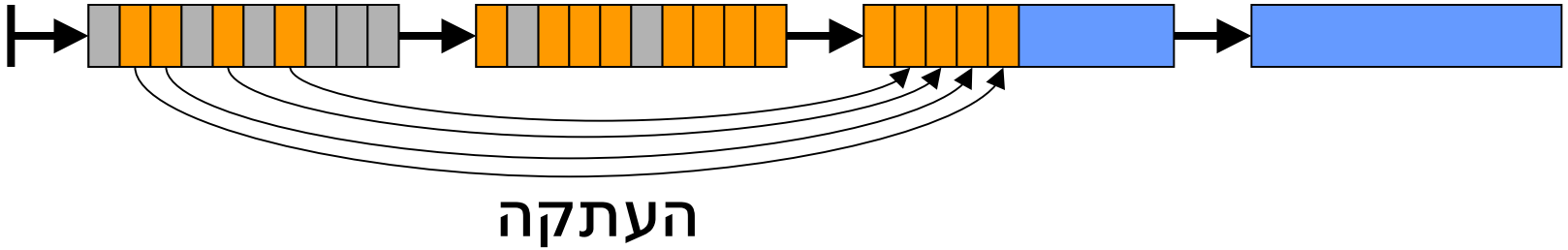
- הדיסק מחולק למקטעים בגודל קבוע (כמיליון בתים) שמשורשרים לרשימה
- נתונים נכתבים על מקטעים כאילו הרשימה הייתה מגילה אינסופית; נתונים נכתבים ומשוכתבים רק בסוף הרשימה
- תחילת המגילה מכילה בלוקים "חיים" (חלק מקבצים) ובלוקים של קבצים שנמחקו ובלוקים ששוכתבו
- סוף הרשימה ריק
- תהליך מיוחד דואג שתמיד יהיה מקטעים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

שכתוב בלוק ב-LFS

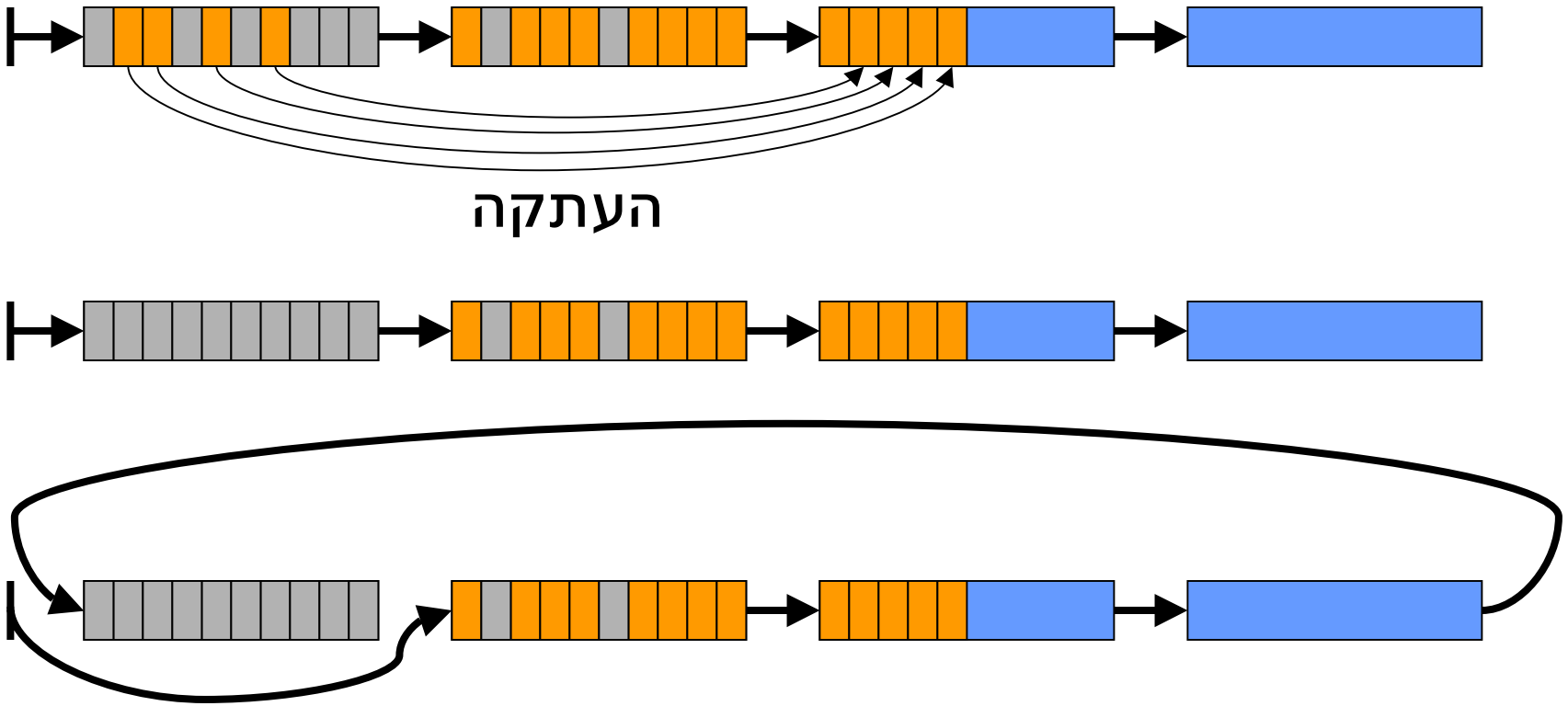


- בלוקים משוכתבים לסוף המגילה
- כאשר משכתבים בלוק, צריך לשכתב גם את ה–inode או בלוק המצביעים שמצביע עליו, משום שמקומו השתנה
- שרשרת השכתובים נעצרת ב–inode משום שמדריכים מצביעים ל–inodes לוגיים שטבלה ממפה למיקום בדיסק
- בלוקים אינם נכתבים אחד אחד אלא בקבוצות, כאשר מצטבר מקטע שלם או כאשר חייבים לכתוב בלוק לקובץ

ניקוי מקטעים ב-LFS



ניקוי מקטעים ב-LFS



❖ 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

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 מכתיבה 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 זא /2000, רוב הגרסאות המסחריות של יוניקס, וגרסאות חדשות 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

פרק 7
רשתות תקשורת ו-IP

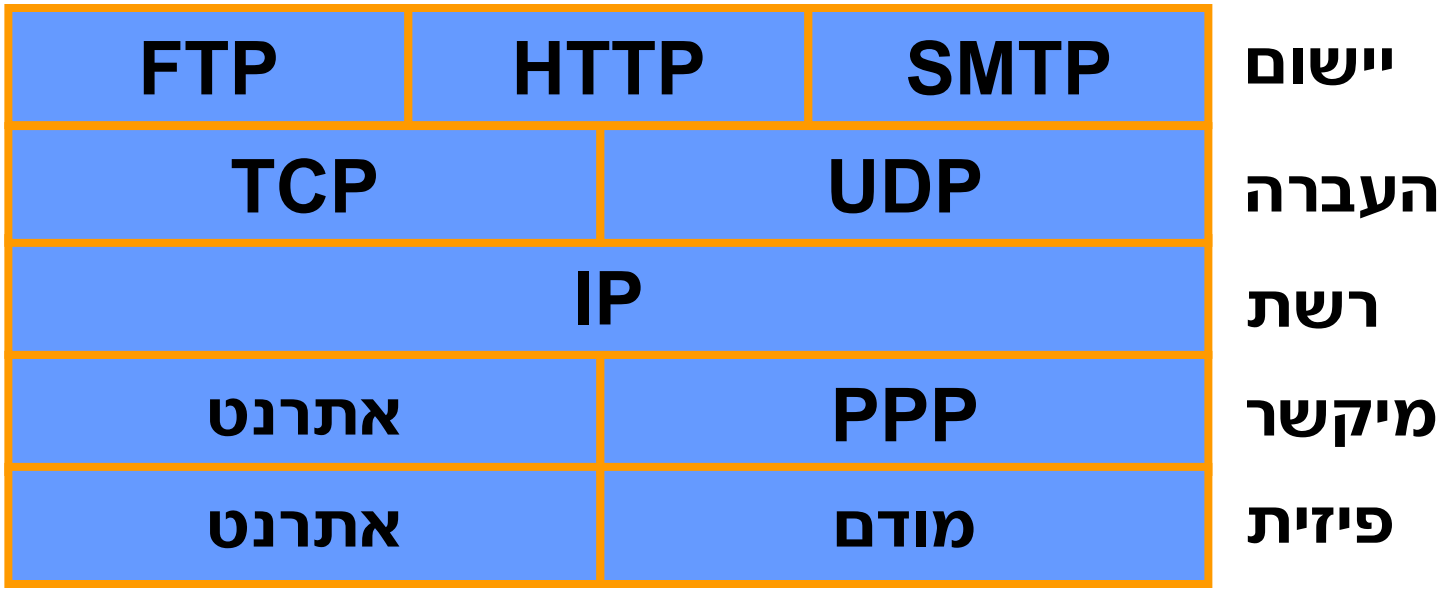
הרצוי לעומת המצוי

- ברמת היישום דרוש קשר אמין בין תוכניות שרצות על מחשבים שונים שאינם מחוברים ישירות זה לזה, אלא בעקיפין דרך רשת תקשורת מורכבת (כמו האינטרנט)
- ברמת החומרה ניתן לתקשר רק בין מחשבים שמחוברים פיזית, התקשורת לא תמיד אמינה, והיא מתבצעת לפעמים במנות (חבילות מידע בדידות בגודל קבוע או מוגבל)

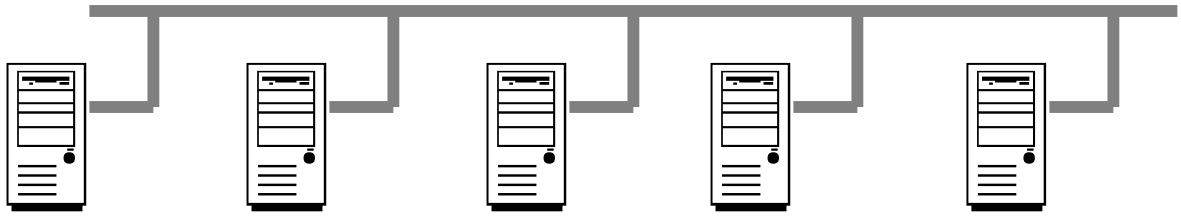
על הפער מגשרים בשכבות

- השכבה ה**פיזית**: המאפיינים הפיזיים של ערוץ תקשורת (רמות מתח, סוג מחבר, מספר חוטים, תדרים, וכדומה)
- שכבת ה**מיקשר** מגדירה פרוטוקולים להעברת מנות מידע בין מחשבים שמחוברים פיזית בתווך
  - פרוטוקול מגדיר את מבנה המנה, שיטת מיעון, ועוד
- שכבת ה**רשת** מגדירה פרוטוקולים לניתוב מנות בין מחשבים ברשת שאינם מחוברים בהכרח פיזית זה לזה
- שכבת ה**העברה** מגדירה פרוטוקולים לתקשורת בין תהליכים במחשבים שונים
- שכבת ה**יישום** מגדירה פרוטוקולים ליישומים ספציפיים, כמו העברת קבצים (FTP ו–HTTP), דואר אלקטרוני (SMTP), ועוד

פרוטוקולי האינטרנט

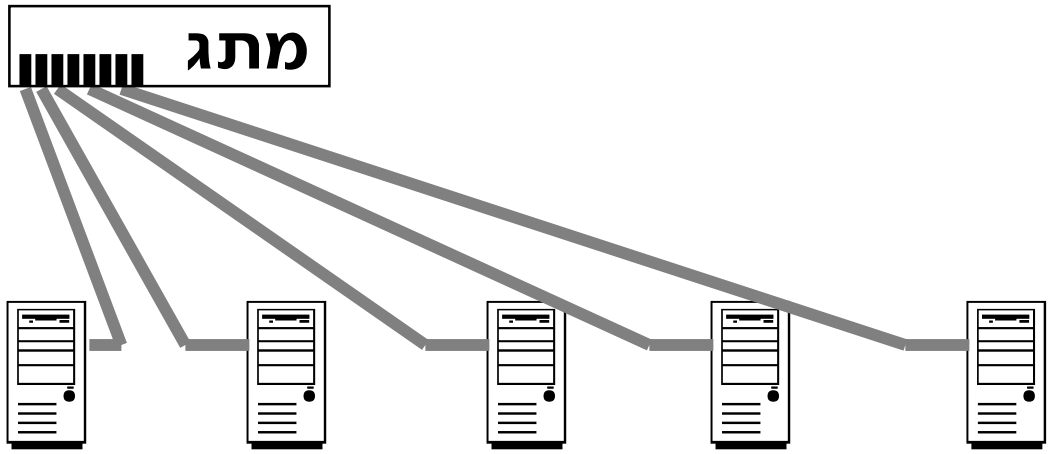


השכבה הפיזית ושכבת המיקשר: אתרנט



- מקטע מחבר מספר מחשבים בבניין יחיד (מגבלת מרחק)
- לכל מחשב במקטע יש כתובת אתרנט
- כל מחשב יכול לתקשר עם כל אחד מהמחשבים האחרים במקטע
- מנגנון לגילוי שגיאות
- ניתן לקלוט רק אם מחשב אחד בלבד משדר
- במקור קצב של 10 Mb/s עם חיווט מסורבל, כיום בד"כ 100 Mb/s עם חיווט נוח (דומה לחוטי טלפון); חומרה לקצב של 1 Gb/s יקרה עדיין ודורשת בד"כ סיבים אופטיים: בשימוש בעיקר בשרתים

מקטע אתרנט ממותג



- מבחינת המחשבים המחוברים אין הבדל בין מקטע ממותג ורגיל
- המתג מעביר כל מנה ליעדה
- מאפשר תקשורת בו–זמנית בקצב מלא בין זוגות מחשבים

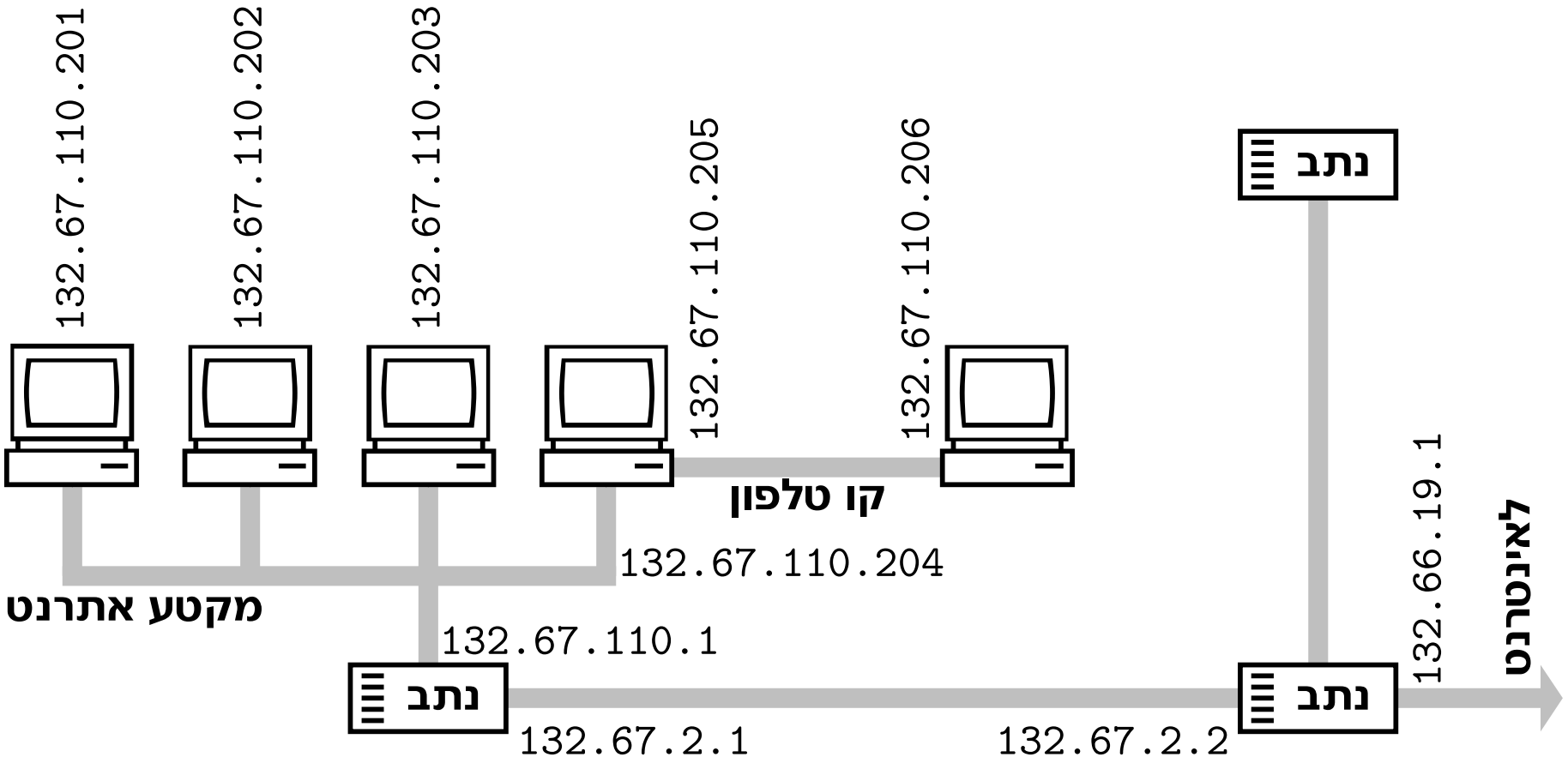
מודמים

- התקן שמקשר את הפס לרשת הטלפון או שמקשר בקר תקשורת טורית לרשת הטלפון
- ממיר רצף סיביות ספרתי לאות אנלוגי וחזרה
- קצב של עד 56 Kb/s, לא צפוי להשתפר
- פרוטוקול המיקשר הוא בדרך כלל PPP, פרוטוקול שמטפל גם ביצירת הקשר ובאימות זהות המשתמש שמבקש להתחבר

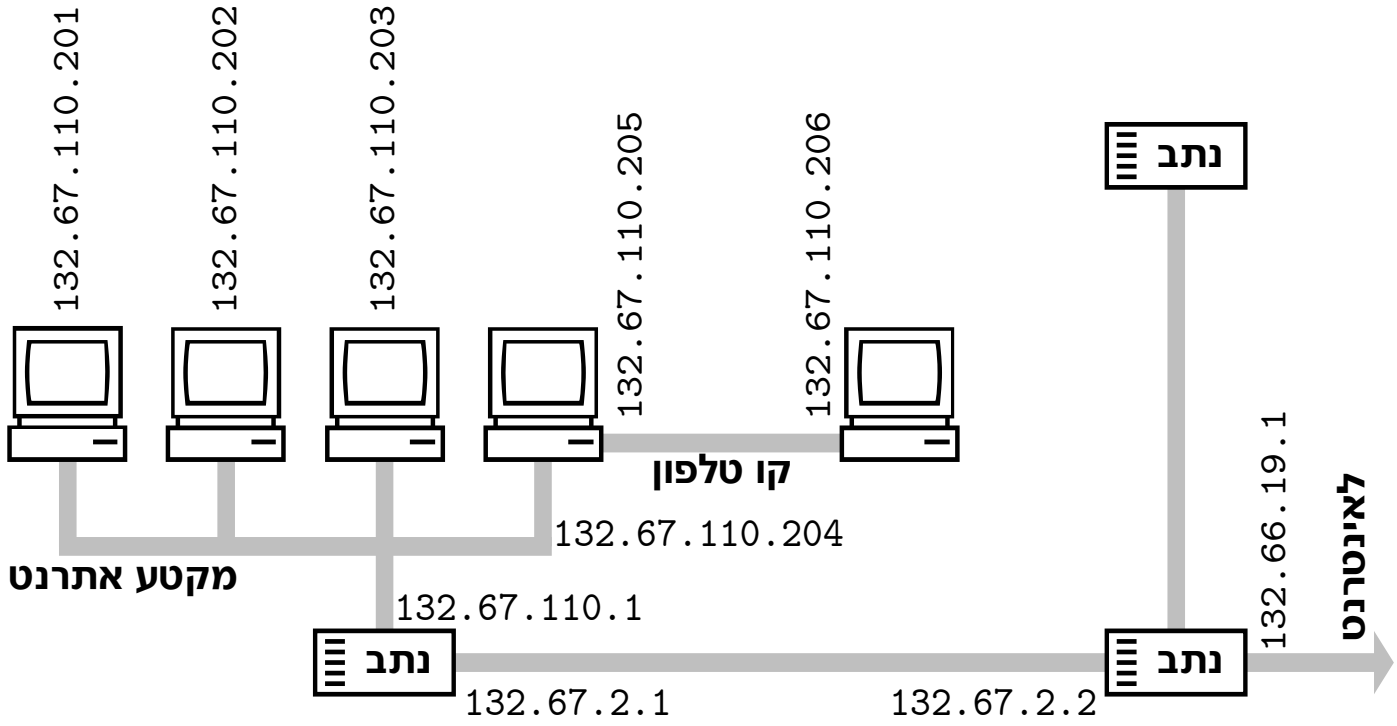
רשתות תקשורת אחרות

- רשתות מקומיות דועכות: token-ring, decnet
- רשתות אלחוטיות: הרשת הנפוצה ביותר מקשרת מחשבים בחלק של בניין בקצב של 11 Mb/s
- מודמים לקווי טלפון סיפרתיים: ISDN (קו טלפון ספרתי בקצב של 2 x 64 Kb/s), ADSL (תקשורת ספרתית על קו טלפון אנלוגי, מאות Kb/s עד מספר Mb/s, לא סימטרי)
- מודמים לשימוש בתשתית הוידאו בכבלים: עד מספר Mb/s, ערוץ משותף לכמה צרכנים
- רשתות תקשורת לאזורים נרחבים

פרוטוקול הרשת IP



פרוטוקול הרשת IP



- היפר–גרף שבו המחשבים הם צמתים, קשתות מחברות זוגות צמתים והיפר–קשתות מחברות מספר צמתים
- לקצוות של קשתות יש שמות בני 32 סיביות
- עקרונית שמות הם ייחודיים (מעשית לא)

עקרונות ניתוב מנות ב-IP

- מנה מפלסת את דרכה ברשת מהמקור ליעד
- כל מחשב במסלול מחליט על הקשת הבאה שהמנה תעבור ועל המחשב הבא במסלול (באחד מקצות הקשת הבאה) על פי היעד שמצוין במנה; המסלול אינו נקבע מראש
- מחשב שמבצע החלטות ניתוב כאלה נקרא נתב; בדרך כלל אין לו תפקידים משמעותיים אחרים, אך כל מחשב כיום מסוגל לנתב
- נתב מקבל החלטות ניתוב בעזרת **טבלת ניתוב**
- כתובות IP מוקצות בצורה שמאפשרת להשתמש בטבלאות ניתוב קומפקטיות

טבלאות ניתוב

- כל כניסה בטבלה מתארת החלטת ניתוב לקבוצה של כתובות
- קבוצת הכתובות מתוארת על ידי תחילית
  - כתובת יעד
  - מסיכה של 32 סיביות מכריזה איזה סיביות ביעד שייכות לתחילית
  - דוגמה: יעד 127.0.0.0 ומסיכה 255.0.0.0 מתארות את כל הכתובות שהסיביות הבכירות שלהן 01111111
- ההחלטה מיוצגת על ידי
  - שם של קשת (מנהל התקן והתקן בשכבת המיקשר)
  - כתובת IP של המחשב הבא במסלול, שצריך להיות מחובר לקשת
  - אם הכתובת חסרה, ניתן להגיע לכל היעדים בקבוצה ישירות דרך הקשת
- הכניסה עם התחילית הארוכה ביותר שמתאימה ליעד של המנה קובעת את הניתוב

הקצאת כתובות IP

- כתובות IP מוקצות בצורה היררכית שמשקפת את מבנה הרשת
- טבלת ניתוב מכילה כללי ניתוב ויוצאים מן הכלל
  - כניסה בטבלה מייצגת כלל ניתוב עבור היעדים שהתחילית מתארת
  - כניסה עם תחילית יותר ארוכה (ותואמת) מייצגת יוצא מן הכלל
- השאיפה היא לטבלאות עם מעט כללים ומעט יוצאים מן הכלל
- השאיפה מתקיימת מבחינת נתב מסוים אם כל אחד מהמחשבים שמחוברים אליו הוא המחשב הבא הנכון עבור קבוצת יעדים שניתנת לתיאור במספר קטן של תחיליות

דוגמה להקצאת כתובות IP

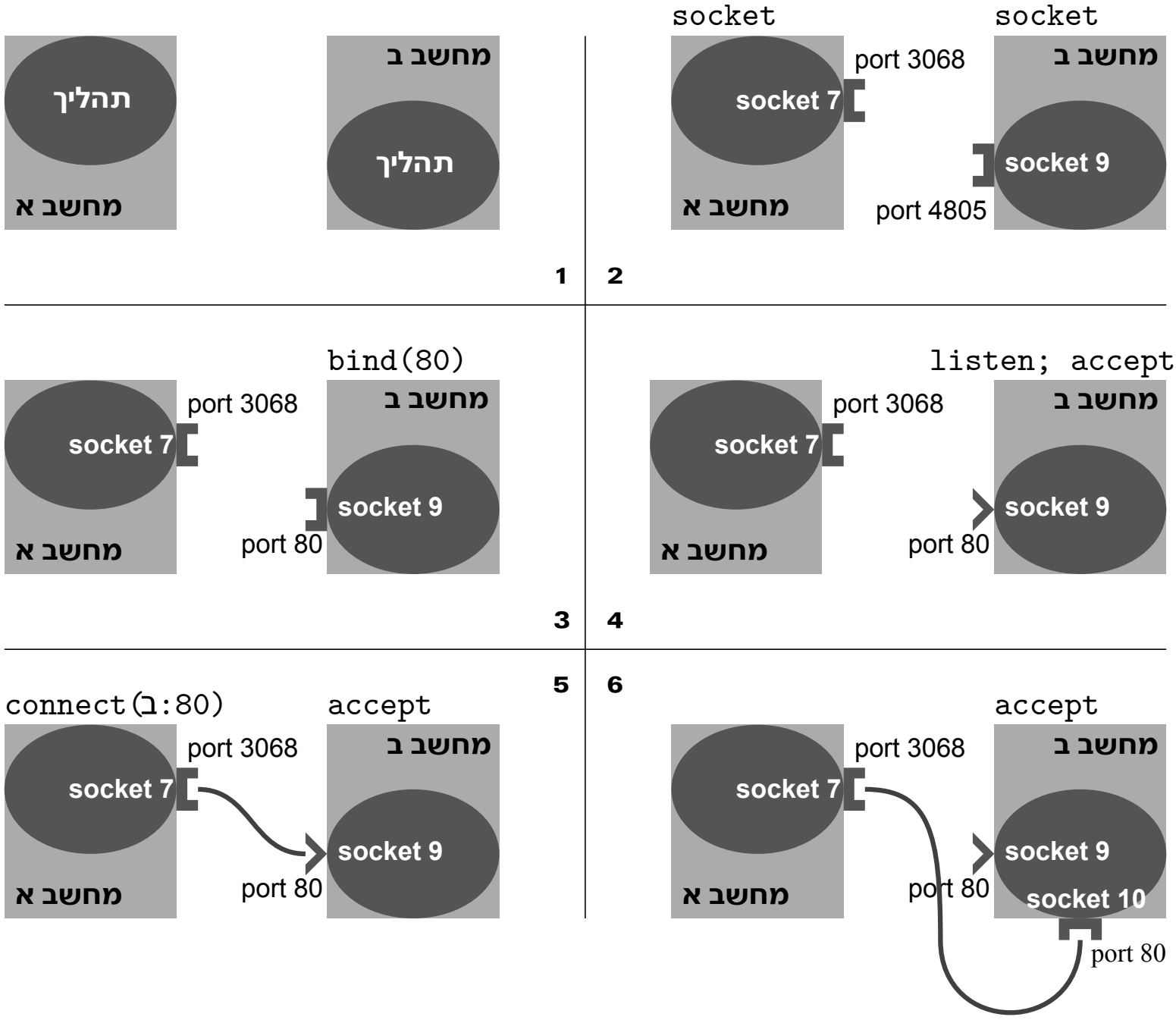
- הרשת האוניברסיטאית הישראלית קיבלה מהגוף המנהל של האינטרנט תחום כתובות מסויים
- הרשת האוניברסיטאית מקצה לכל מוסד תת–תחום מתוך הכתובות שהוקצו לה
- אם כל מוסד מקבל תחום כתובות עם תחילית אחת ואם לכל מוסד יש נתב אחד בכניסה לרשת שלו, אז כל היעדים של מוסד מיוצגים בטבלאות הניתוב במוסדות אחרים בכניסה אחת בלבד
- כל מוסד מקצה תתי–תחומים קטנים יותר לפקולטות ובתי ספר, שמקצים כתובות למחשבים בודדים

תחזוקת טבלאות הניתוב

- במחשבים אישיים וביתיים תחזוקה ידנית (בעזרת הפקודה route ותסריטי אתחול) ועדכון הטבלה בזמן התחברות לרשת (חלק מפרוטוקולים כמו DHCP ו–PPP שמקצים כתובות דינמית)
- בנתבים: תחזוקה ידנית או השתתפות באלגוריתמים מבוזרים לקביעת טבלאות ניתוב (על ידי מציאת מסלולים קצרים, למשל)
- טבלאות הניתוב צריכות להתעדכן כתוצאה משינויים ברשת כמו נפילה של קווי תקשורת
- בכל מקרה, חשוב שטבלאות הניתוב לא ייצרו מעגלי ניתוב שמנות עלולות להסתחרר בהם; הסתחררות כזו מבזבזת משאבים ומונעת ניתוב מנות ליעדן

מבוא לשכבת ההעברה: שקעים

- כתובת IP מציינת מחשב; על מנת ליצור קשר בין תהליכים צריך כתובת לתהליך
- תהליך יכול ליצור שקע, עצם שניתן לשלוח ולקבל דרכו מידע
- שקע חדש מקבל כתובת שרירותית, אבל ניתן לקשור אותו לכתובת מסוימת; הכתובת נקראת **שער**
- בפרוטוקול UDP ניתן לשלוח מנדעים (מעין מברקים) לשער מסוים בכתובת IP מסויימת
- בפרוטוקול TCP ניתן לחבר שני שקעים בקשר קבוע ואמין


תהליך
מחשב א
מחשב ב
תהליך
1
socket
socket
socket 7
port 3068
מחשב א
מחשב ב
socket 9
port 4805
2
bind(80)
socket 7
port 3068
מחשב א
מחשב ב
socket 9
port 80
3
listen; accept
socket 7
port 3068
מחשב א
מחשב ב
socket 9
port 80
4
5
connect(ב:80)
accept
socket 7
port 3068
מחשב א
מחשב ב
socket 9
port 80
6
accept
socket 7
port 3068
מחשב א
מחשב ב
socket 9
port 80
socket 10
port 80

זרימת מידע במחסנית הפרוטוקולים

- קריאת מערכת מעבירה מידע מתהליך (שכבת היישום) לשכבת ההעברה דרך שקע
- קריאת המערכת מזהה את פרוטוקול ההעברה על פי סוג השקע וקוראת לשגרה מתאימה של אותו פרוטוקול
- השגרה הזו מפרקת את המידע למנות או מצרפת אותו למידע קודם או מתייחסת אליו כמנה, מוסיפה למנות תחילית עם שדות של הפרוטוקול, ומעבירה אותן לשגרה של שכבת הרשת
- השגרה של שכבת הרשת מחליטה איך לנתב את המנה, מוסיפה לה תחילית IP, ומעבירה אותה לשגרה של פרוטוקול המיקשר המתאים
- השגרה של שכבת המיקשר מתרגמת את כתובת ה– IP לכתובת פיזית של מחשב שמחובר לקשת היוצאת, מוסיפה תחילית, ומעבירה לבקר לשליחה

פרוטוקול ARP

- פרוטוקול לתרגום כתובות IP לכתובות פיזיות של מחשבים במקטע אתרנט
- ממומש כחלק משכבת המיקשר, לתרגום כתובות רשת לכתובות מיקשר; פרוטוקולים דומים דרושים למימושי מיקשר אחרים
- מחשב שצריך לבצע תרגום כזה שולח הודעת שאילתה לכל המחשבים במקטע (הודעת broadcast)
- מחשב שמזהה את כתובת ה–IP שלו בשאילתה כזו עונה בהודעה עם כתובת האתרנט שלו (MAC address)
- שכבת המיקשר מטמינה את התרגומים הללו בטבלה לזמן מה על מנת לחסוך בהודעות ARP (שמעכבות משלוח מנות)

פרק 8
פרוטוקול TCP

תכונות הפרוטוקול

- קשר דו כיווני אמין וסדור בין תהליכים במחשבים שונים
- העברה מהירה ככל האפשר של מידע
- מניעת עומס מיותר על הרשת, עומס שעלול לנבוע ממשלוח מנות שיאבדו וייווצר צורך לשלוח שוב
- יצירת ופירוק קשר באופן מפורש ומוסכם על ידי שני הצדדים
- תכונות מתקדמות שלא נדון בהן, כגון הודעות דחופות

בזכות התכונות הללו, פרוטוקול ההעברה הנפוץ ביותר

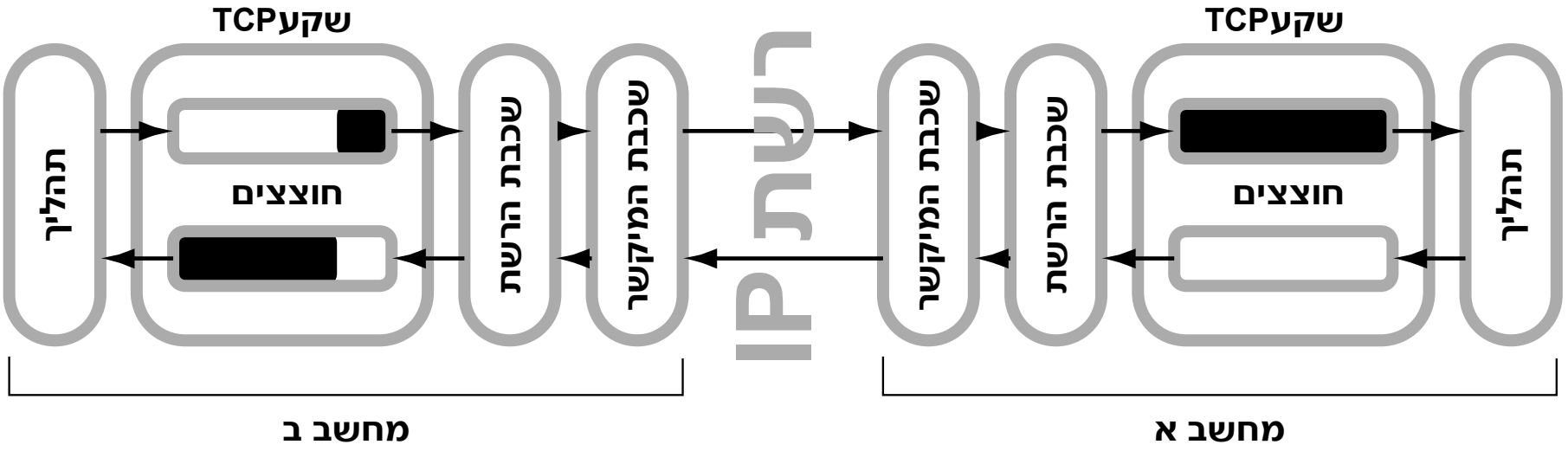
המימוש מורכב, מכיון ש...

- פרוטוקול הרשת IP אינו אמין ואינו סדור, גם אם פרוטוקולי המיקשר אמינים וסדורים
- נתב שמקבל מנות בקצב גבוהה מזה שהוא יכול לטפל בהן (למשל בניתוב מערוץ מהיר לאיטי) פשוט מוחק אותן
- IP שולח מנות בדידות וכל מנה מנותבת בנפרד; כאשר טבלאות הניתוב משתנות, מסלולים משתנים, ומנה מאוחרת ברצף המידע עשוייה להקדים מנה מוקדמת

נגזרות מהדרישות

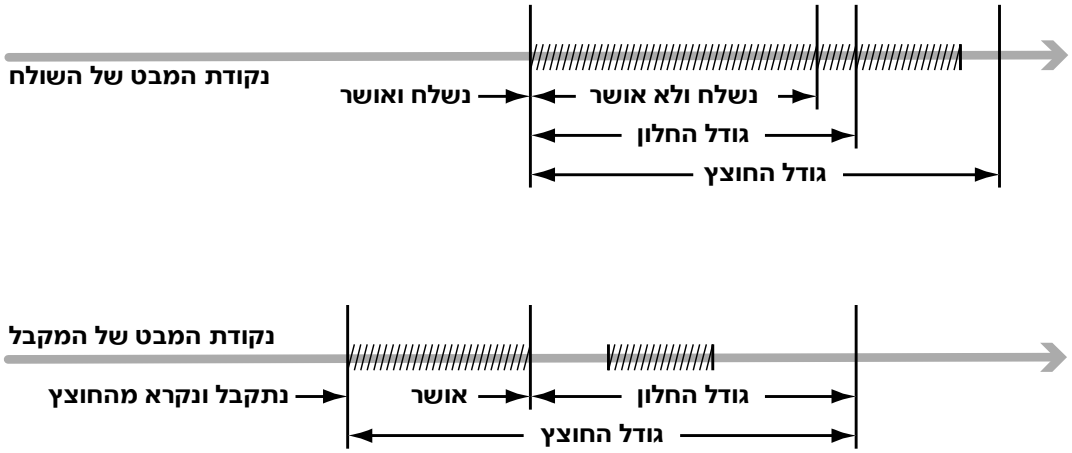
- מספור סודר על מנות על מנת שניתן יהיה לשחזר בצד המקבל את הרצף לפי סדר
- משלוח חוזר של מנות שאבדו
- אישורי קבלה על מנת שניתן יהיה לזהות מנות שלא הגיעו אחרי פרק זמן סביר ולשלחן שוב
  - מכיון שאי אפשר לדעת האם מנה אבדה או רק מתעכבת הרבה, מנות עלולות להגיע יותר מפעם אחת!
- חוצצים למשלוח מנות באופן יעיל

חוצצים ב-TCP



- החוצץ בצד המקבל מאפשר לשולח לשלוח כמות גדולה של מידע לפני שהוא מקבל אישור למנה כלשהי; בהעדר חוצץ כזה
  - השולח יכול היה לשלוח מידע שאולי אין היכן לאחסן: עומס ללא צורך
  - לשלוח מנה רק כשהקודמת מאושרת: לא מנצל את רוחב הפס של הרשת
- החוצץ הזה מאפשר גם לקבל הודעות שהתהליך המקבל עסוק
- החוצץ בצד השולח מאפשר לתהליך לרוץ בשטף ולנצל את הרשת

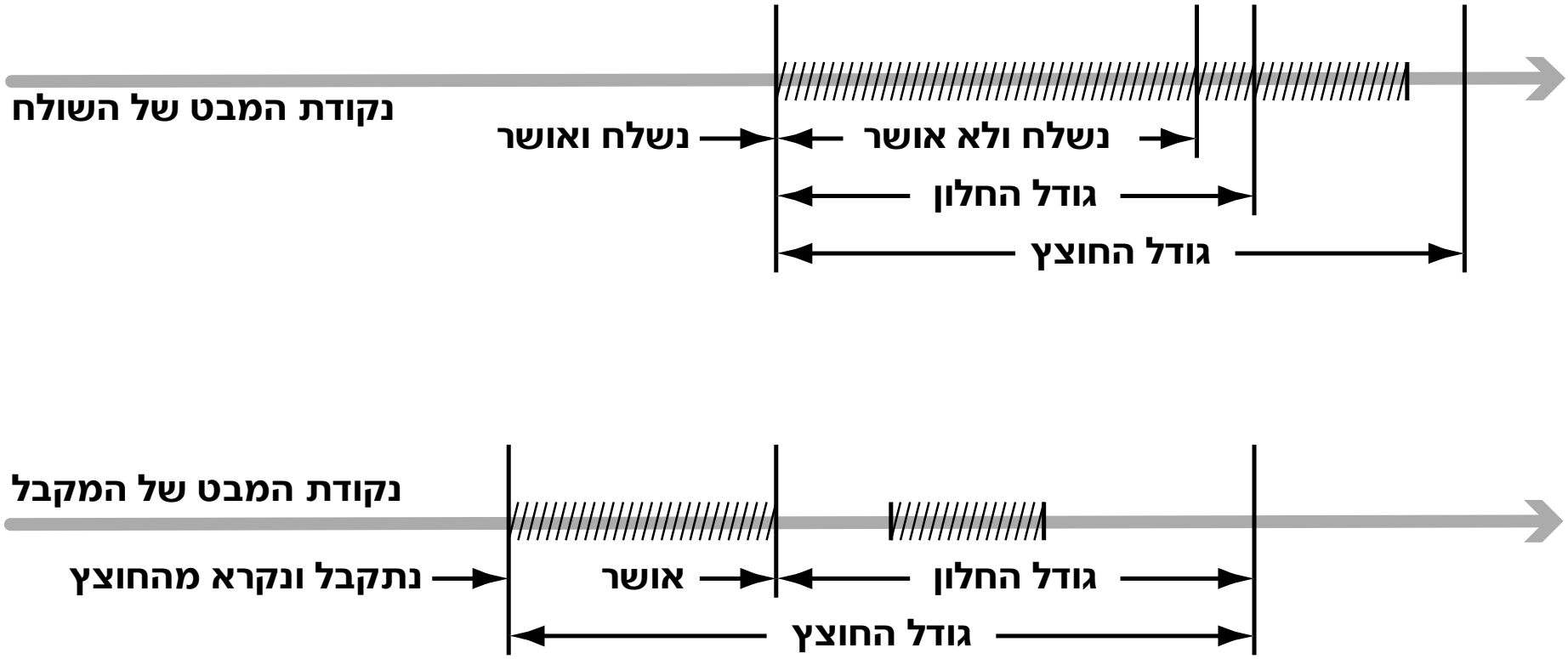
תחילית TCP



❖ לכל מנת מידע מצורפת תחילית עם השדות הבאים

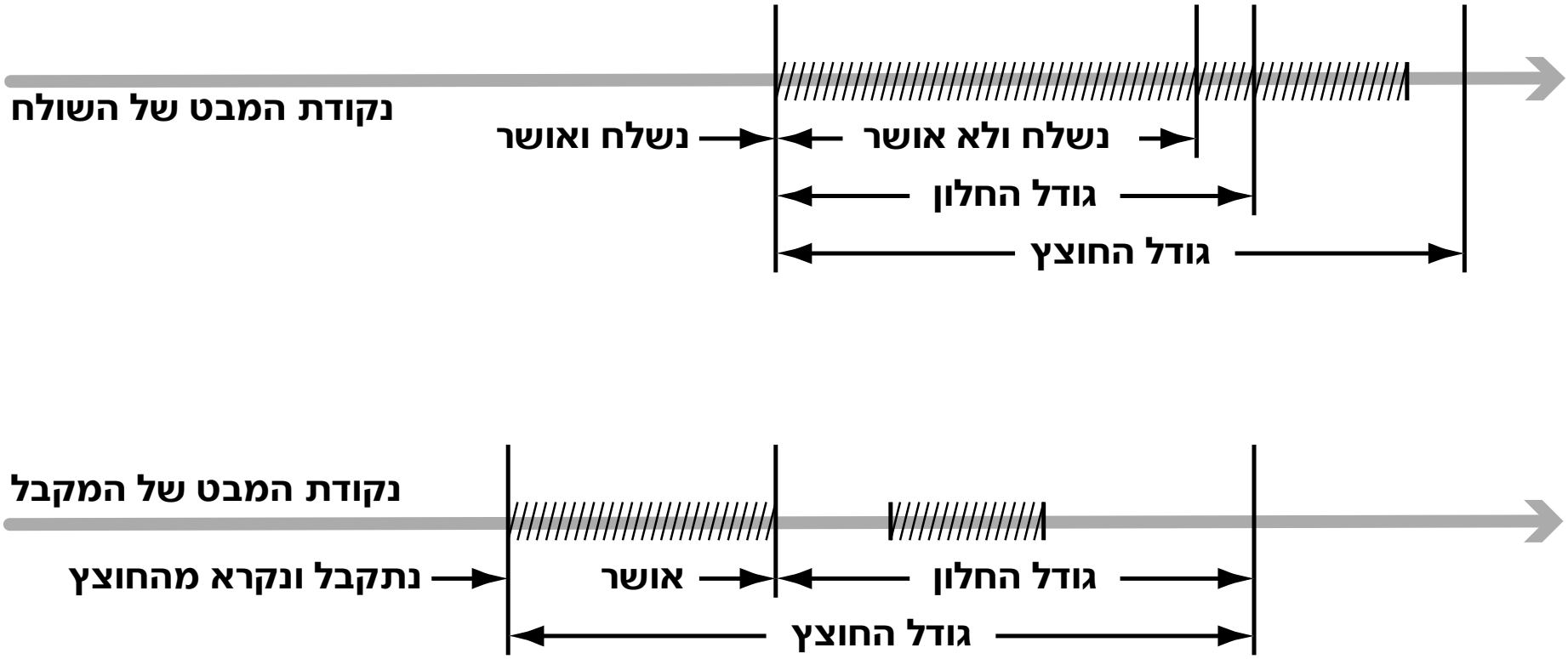
- כמות המידע שהמנה נושאת, בבתים, 0 הוא ערך מותר
- מיקום המידע ברצף מודולו 2^{32}; השדה הזה מהווה את השדה הסודר
- מיקום הבית האחרון שנקלט ללא חורים על ידי הצד ששולח את המנה; אין דרך לאשר קבלת מנה שנושאת נתונים שנתונים קודמים להם חסרים
- גודל **חלון** מודיע לצד השני כמה בתים שלא אושרו מהמשך הרצף חוצץ הקבלה מסוגל להכיל; שדה של 16 סיביות, בבתים או יחידות גדולות יותר
- סכום בדיקה לגילוי שגיאות ושדות שמציינים מצבים מיוחדים

חלונות מחליקים



מצב פשוט: מצב הקשר נראה זהה לשני הצדדים (לא טיפוסי)

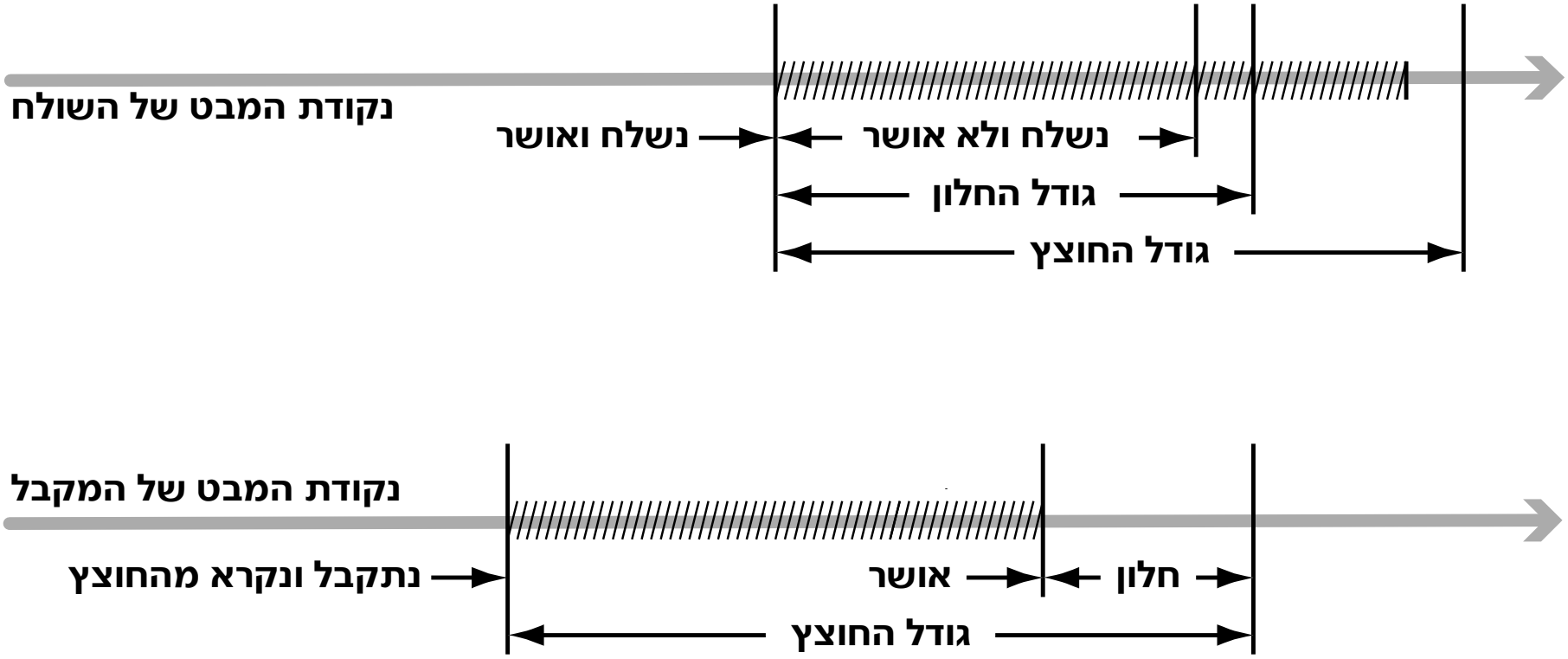
חלונות מחליקים



מצב פשוט: מצב הקשר נראה זהה לשני הצדדים (לא טיפוסי)

מה קורה כאשר מתקבלת מנה שממלאה את החור?

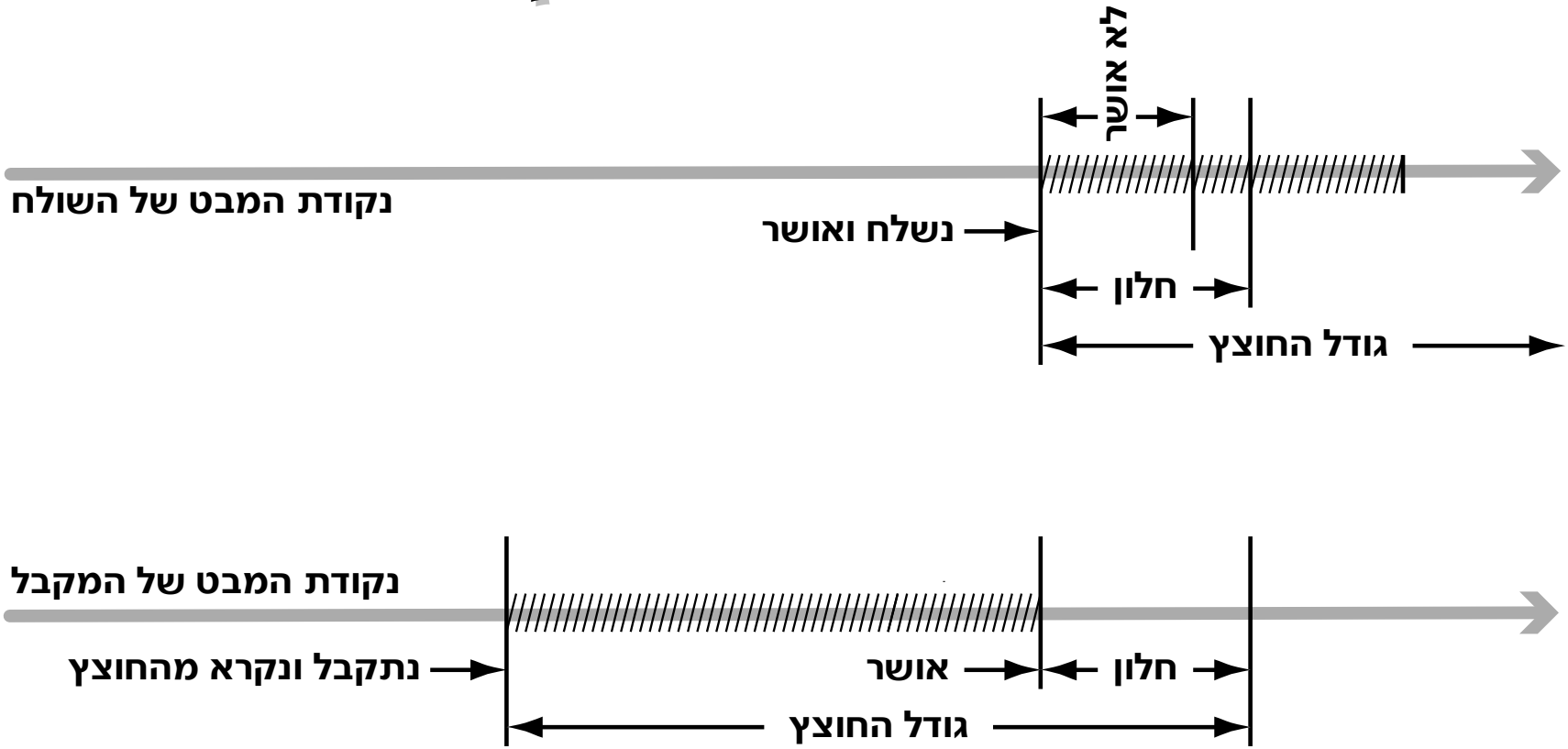
החור התמלא



המקבל שולח אישור (עדיין לא התקבל)

מה יקרה כאשר השולח יקבל את האישור?

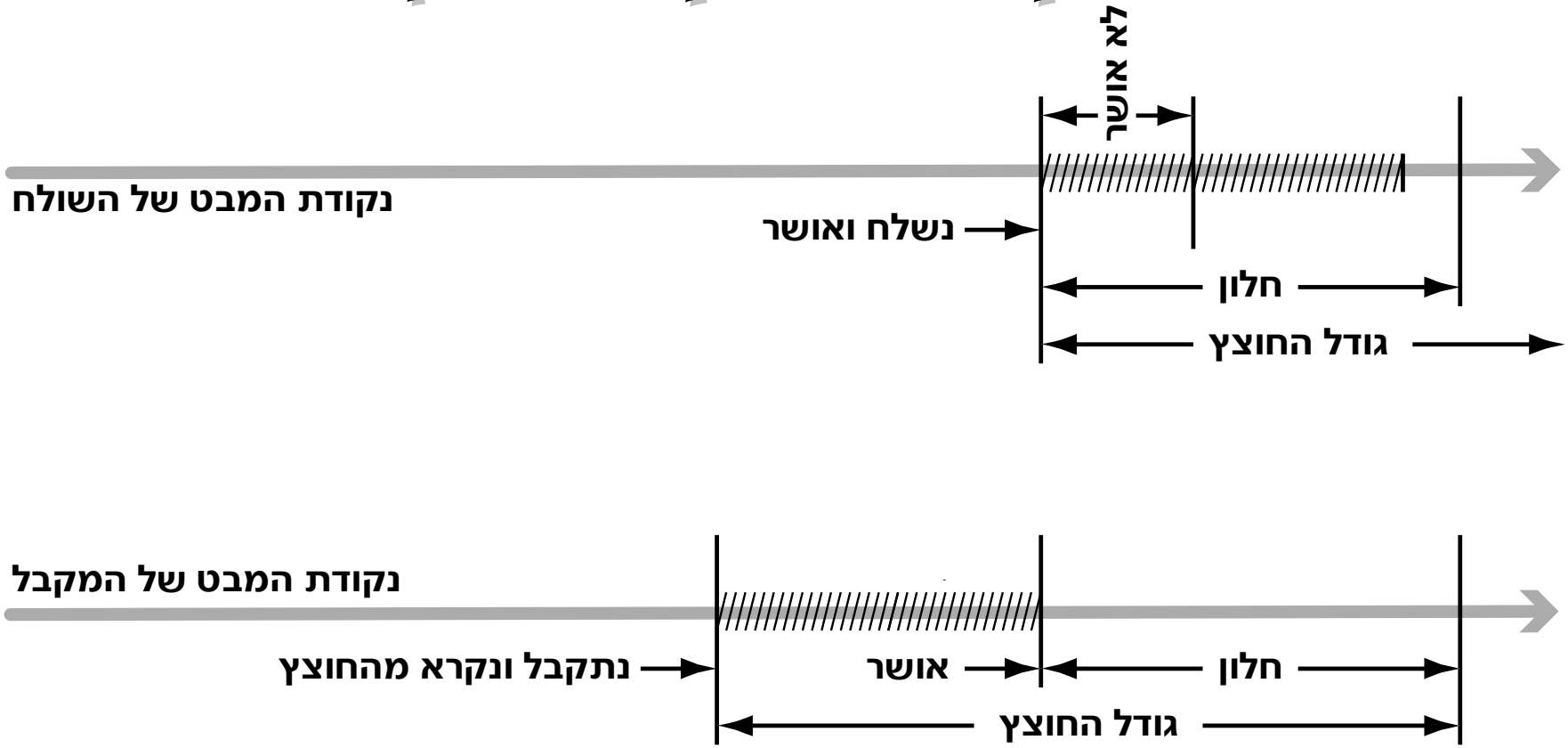
האישור התקבל



כאשר המקבל יקרא מהחוצץ, הוא ישלח עדכון חלון

מה יקרה כאשר השולח יקבל את עדכון החלון?

עדכון החלון התקבל



מה יקרה כאשר השולח יעביר נתונים נוספים לשקע?

אובדן מנות

- שירות יקיצה מזכיר לשולח לשלוח שוב מנות שלא אושרו (שירות שידור חוזר, retransmit)
  - שליחת מנה כאשר השירות כבוי מפעילה אותו לנקודת זמן בעתיד
  - כאשר הוא פוקע שולחים שוב את המנה הישנה ביותר שלא אושרה ומתזמנים יקיצה לזמן רחוק יותר
  - פרק הזמן ליקיצה גדל פי קבוע בכל פעם, עם גג
  - הגדלת פרק הזמן מיועדת למנוע שידור חוזר תכוף מדי בערוץ איטי
  - כאשר מנה מאושרת, מתזמנים שוב יקיצה עבור המנה הישנה ביותר שעדיין לא אושרה

גם אישורים הולכים לאיבוד

- התוצאה של אישור אבוד הוא חלון סגור
- שירות יקיצה בשם **persist** מזכיר לשולח לברר האם החלון נפתח
- השירות מופעל כאשר יש מידע לשלוח והחלון סגור (או קטן מדי)
- בזמן יקיצה, השולח שולח בית בודד אם החלון סגור וכמה שאפשר אם הוא פתוח, ומתזמן יקיצה נוספת
- כאשר המנה הזו מגיעה, המקבל מאשר אותה עם גודל חלון עדכני
- שירות היקיצה מכובה כל אימת שמגיע עדכון חלון שמאפשר לשלוח מנה שלמה
- המקבל מאשר גם מידע שכבר נתקבל ואושר, מחשש שהאישור המקורי אבד

אופטימיזציות 1:
סינדרום החלון האוילי

- חלון כמעט סגור מלמד בדרך כלל שהחוצץ בצד המקבל מלא במידע שהתהליך המקבל לא קרא עדיין
- במצב כזה אין טעם לשלוח מידע נוסף: זה לא יועיל למקבל ומנות קטנות מעמיסות על המחשבים והרשת
- על מנת למנוע משלוח מנות קטנות:
  - המקבל אינו מכריז על חלון קטן (פחות ממנה סטנדרטית או פחות מרבע גודל החוצץ שלו) אלא על 0
  - השולח משהה משלוח מנות קטנות עד ליקיצת **persist**
- התנהגות השולח מעכבת תקשורת ללא צורך כאשר למקבל חוצץ קטן מאוד: כדי למנוע זאת, שולחים מייד אם גודל החלון לפחות חצי מהחלון הגדול ביותר שהוכרז עד כה

אופטימיזציות 2:
האלגוריתם של Nagle

- השולח משהה משלוח מידע חדש שאינו ממלא מנה שלמה כאשר לא כל המנות שנשלחו אושרו
- כאשר נשלחות כמויות מידע גדולות, העיכוב במשלוח קטן ומונע עומס
- כאשר ערוץ התקשורת מהיר (רשת מקומית) ואישורים מתקבלים כמעט מייד, אין בדרך כלל עיכוב כלל
- האלגוריתם הזה פוגע בביצועים של יישומים אינטראקטיביים מסויימים וניתן לכבות אותו על ידי קריאה ל-setsockopt

אופטימיזציות 3:
השהיית אישורים ועדכוני חלון

- מנה שאינה נושאת נתונים אלא רק אישור ועדכון חלון מעוכבת עד ש:
  - נוצר צורך לשלוח מנה של נתונים, או
  - חלון הקבלה גדל למנה שלמה, או
  - חלפה חמישית שניה
- האופטימיזציה הזו מונעת משלוח מספר עצום של עדכוני חלון כאשר התהליך הקורא מבקש נתונים מועטים בכל קריאת מערכת
- ההשהיה מוגבלת על מנת למנוע מנות כתוצאה מיקיצות retrastmit ו–persist בצד השני

אופטימיזציות 4:
מניעת עומס

- עומס על נתבים וקווי תקשורת גורם לאובדן מנות
- יקיצות persist ו–restrasmit מעידות בדרך כלל על עומס (ורק לעיתים רחוקות על השחתת מידע בתווך)
- השולח מקטין את קצב שליחת החבילות כאשר יקיצות כאלה גורמות לו לחשוב שהרשת עמוסה
- המנגנון מבוסס על התנהגות כאילו החלון קטן ממה שהוא באמת, אך לא נתאר אותו באופן מפורט

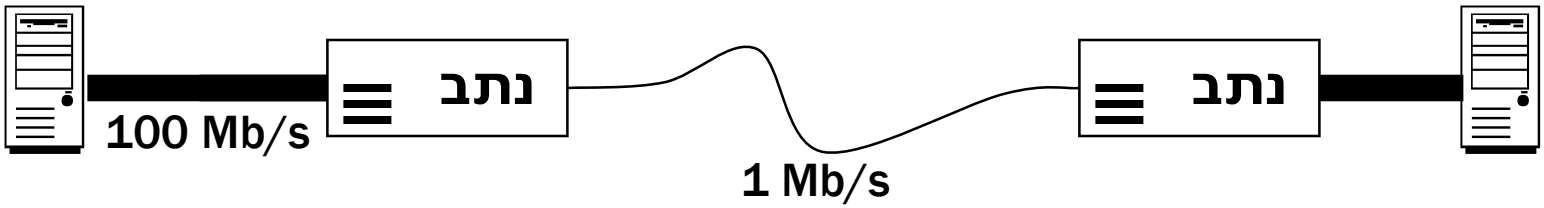
אופטימיזציות 5:
שליחה חוזרת מהירה

- השולח מסיק שמנה בודדת אבדה כאשר הוא מקבל שלושה אישורים זהים ברצף
- ההנחה כאן היא שמנה אבדה ומנות עוקבות התקבלו וגרמו למשלוח אישורים
- במקרה כזה השולח משגר שוב את המנה שכנראה אבדה ואינו מקטין את קצב השליחה (מהאופטימיזציה הקודמת)

אופטימיזציות 6:
קביעת גודל חוצצים

- בהנחה שהתהליך הקורא לא מעכב מידע בחוצץ, תפקיד החוצץ הוא לאפשר לשולח לשלוח מידע לפני קבלת אישורים
- מה צריך להיות גודל החוצץ? קצב הערוץ כפול פעמיים ההשהיה
- למשל,
  - ערוץ לוויני שמנה חוצה ברבע שניה ברוחב פס של 100 Mb/s
  - אישור למנה מגיע חצי שניה אחרי שליחתה
  - השולח צריך לשלוח 50 Mb שהם 6 MB לפני קבלת אישור
  - מאותו רגע הוא מקבל אישורים בקצב השליחה
  - החוצץ צריך להיות בגודל 6 MB על מנת לנצל את רוחב הפס של הערוץ
- חוצץ של 64 KB מאפשר לשלוח רק במאית מרוחב הפס

קביעת גודל חוצצים – חוצץ גדול מדי

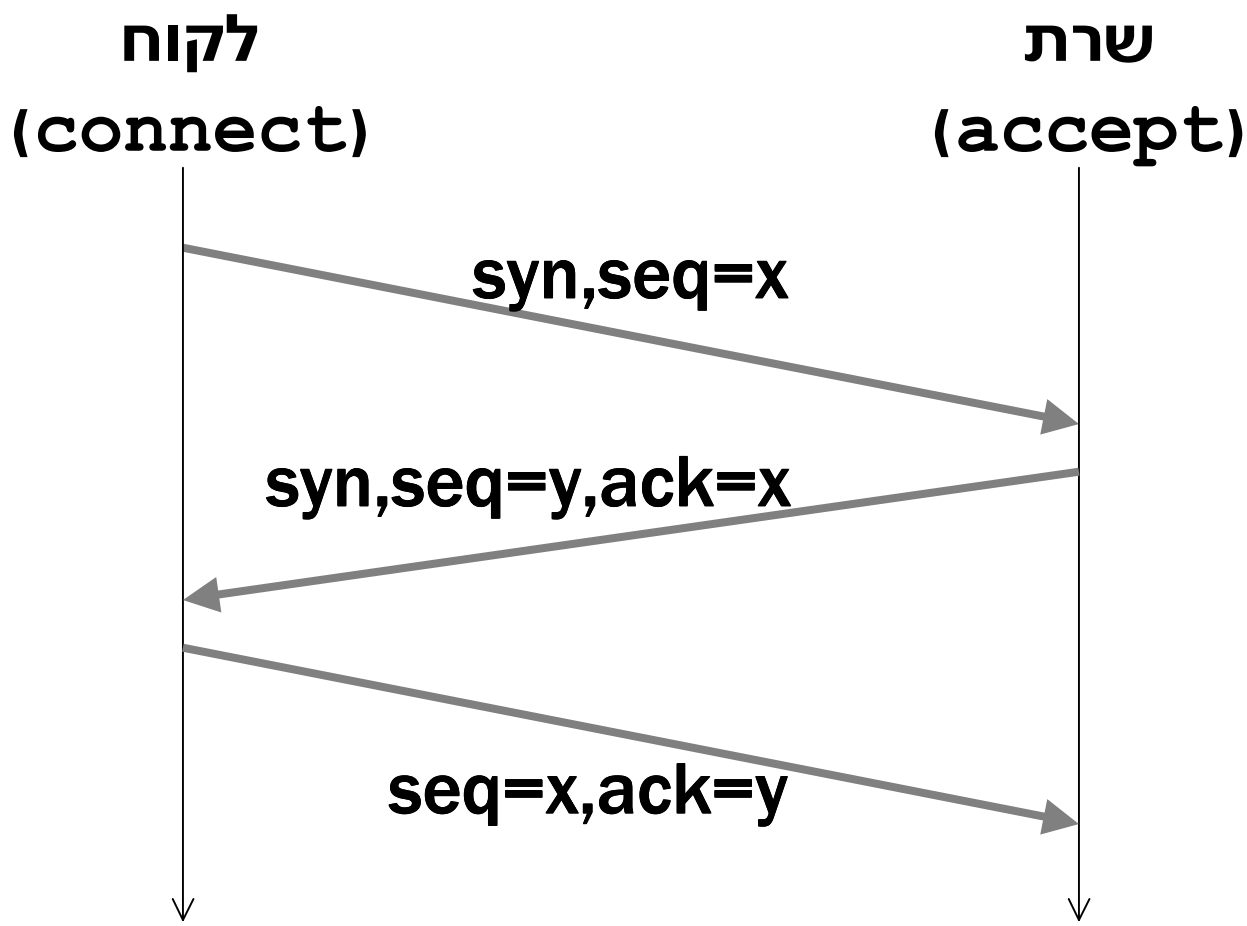


- מאידך, חוצץ קבלה גדול מדי מאפשר לשולח להציף נתבים במנות שהנתב אינו יכול להעביר
- החוצץ צריך להתאים בגודלו לרוחב הפס האפקטיבי של הערוץ מקצה לקצה ולא לרוחב הפס של מקטעים בדרך

אופטימיזציות 7:
התחלה איטית

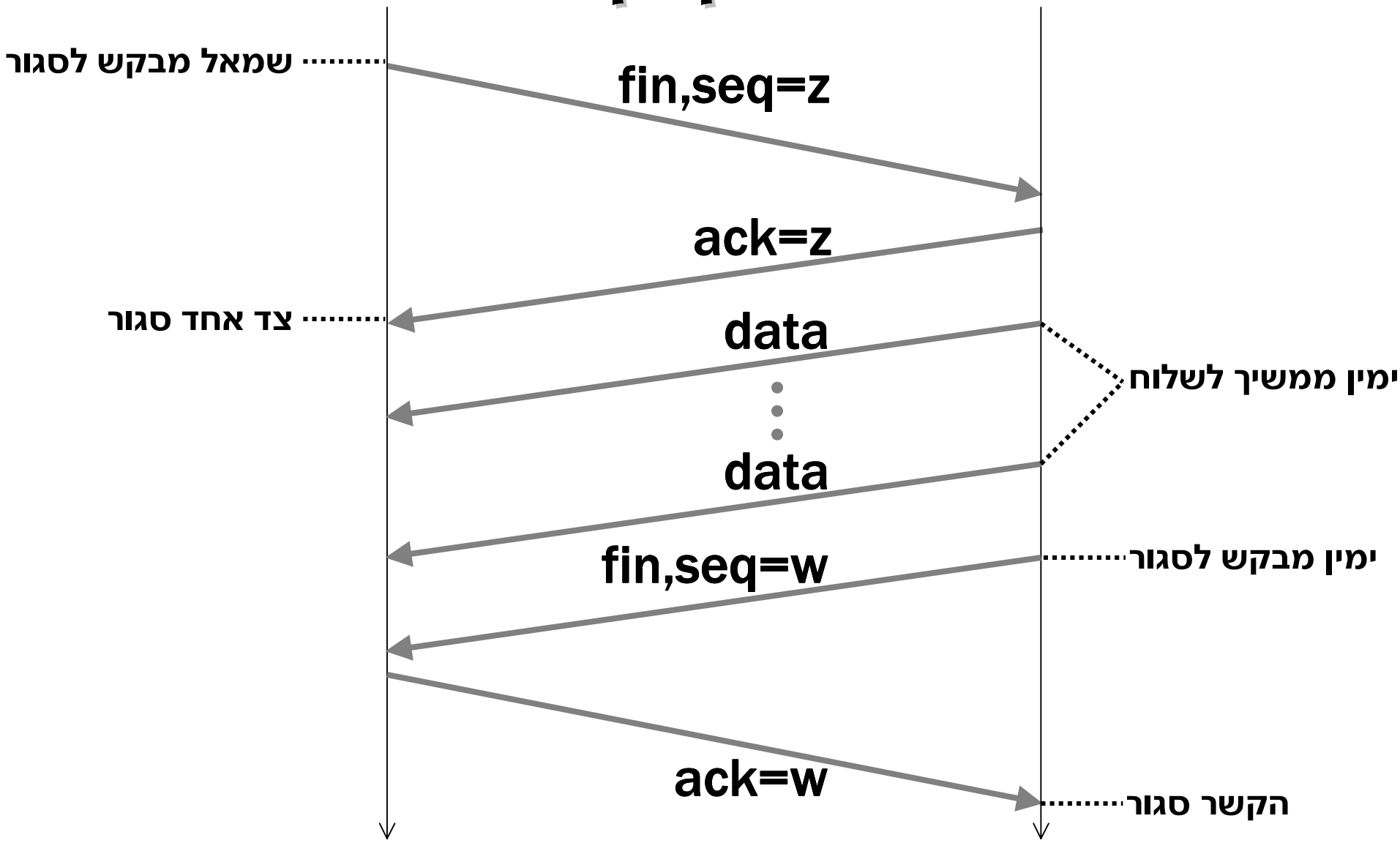
- במצב יציב של הקשר מנה נשלחת כל אימת שמגיע אישור שפותח את החלון למנה נוספת
- בתחילת הקשר החלון פתוח לרווחה והשולח עלול לשלוח חלון מלא בבת אחת לנתב קרוב
- אם הנתב לא יכול לשלוח את המנות במהירות, חלקן יאבד
- על מנת למנוע את זה, השולח משתמש בחלון עומס שמגביל את כמות המידע שנשלח ולא אושר
- חלון העומס מתחיל במנה אחת וגדל באחת כל אימת שמתקבל אישור
- קצב שליחת המנות גדל כך אקספוננציאלית לקצב של הערוץ

יצירת קשר



הרצפים מתחילים ממיקום אקראי כדי להקשות על התחזות

פירוק קשר


שמאל מבקש לסגור
fin,seq=z
ack=z
צד אחד סגור
data
ימין ממשיך לשלוח
data
fin,seq=w
ימין מבקש לסגור
ack=w
הקשר סגור

מנות מקשר מפורק

- מנות שהתעכבו ברשת עלולות להגיע לאחר שהקשר פורק
- אם האישור ל–fin אובד, מנת fin נוסף תגיע בהמשך למחשב שאישר קבלת fin ושמבחינתו הקשר סגור
- אי לכך, מערכת ההפעלה שומרת על מבנה הנתונים שייצג את הקשר זמן מה לאחר סגירתו כדי שניתן יהיה לטפל במנות כאלו
- כדי שניתן יהיה לטפל במנות כאלו, מערכת ההפעלה אינה מתירה ליצור קשר נוסף עם אותם פרמטרים (כתובת ip ומספר שער מקומיים ומרוחקים) באותו פרק זמן
- מערכות הפעלה רבות אינן מאפשרות למחזר את הפרמטרים המקומיים; ניתן להורות להן למחזר על ידי קריאה ל–setsockopt

מנות מקשר שנשכח

- שמירה לזמן מה על מבנה הנתונים של הקשר לאחר סגירתו מאפשרת לטפל ברוב המנות שמגיעות אחרי סגירה
- אבל לעיתים מנות עלולות להגיע זמן רב לאחר סגירה
- בנוסף, תקלות במחשבים אחרים (או ניסיונות שיבוש מכוונים) עלולות לגרום לקבלת מנות של קשרים שלא היו קיימים מעולם
- מערכת ההפעלה עונה למנות כאלה במנת מיוחדת שמסומנת בסיבית reset
- קבלת מנה כזו צריכה לגרום לשולח להפסיק לשלוח מנות בקשר

מודעות לקיום קשר

- כאשר אין צורך להעביר מידע בקשר TCP, מנות אינן נשלחות כלל
- התנהגות זו גורמת לחוסר מודעות להתנתקות הקשר עקב נפילת אחד הצדדים ועקב התנתקות של הרשת
- ביישומים שבהם דרושה מודעות לקיום או אי–קיום הקשר, ניתן להורות למערכת ההפעלה לשלוח מנה מדי פעם לבדיקת קיום
- המנגנון מופעל באמצעות שירות היקיצה keepalive

פרק 9
מערכות קבצים מבוזרות

מערכות מבוזרות

- קבוצה של מחשבים וציוד אחר (מדפסות למשל) מחוברים ברשת תקשורת.
- אנו רוצים לאפשר למשתמשים לנצל חומרה דרך הרשת ולא רק חומרה מקומית.
- בדרך כלל לא נפעיל את כל החומרה תחת מערכת הפעלה אחת.

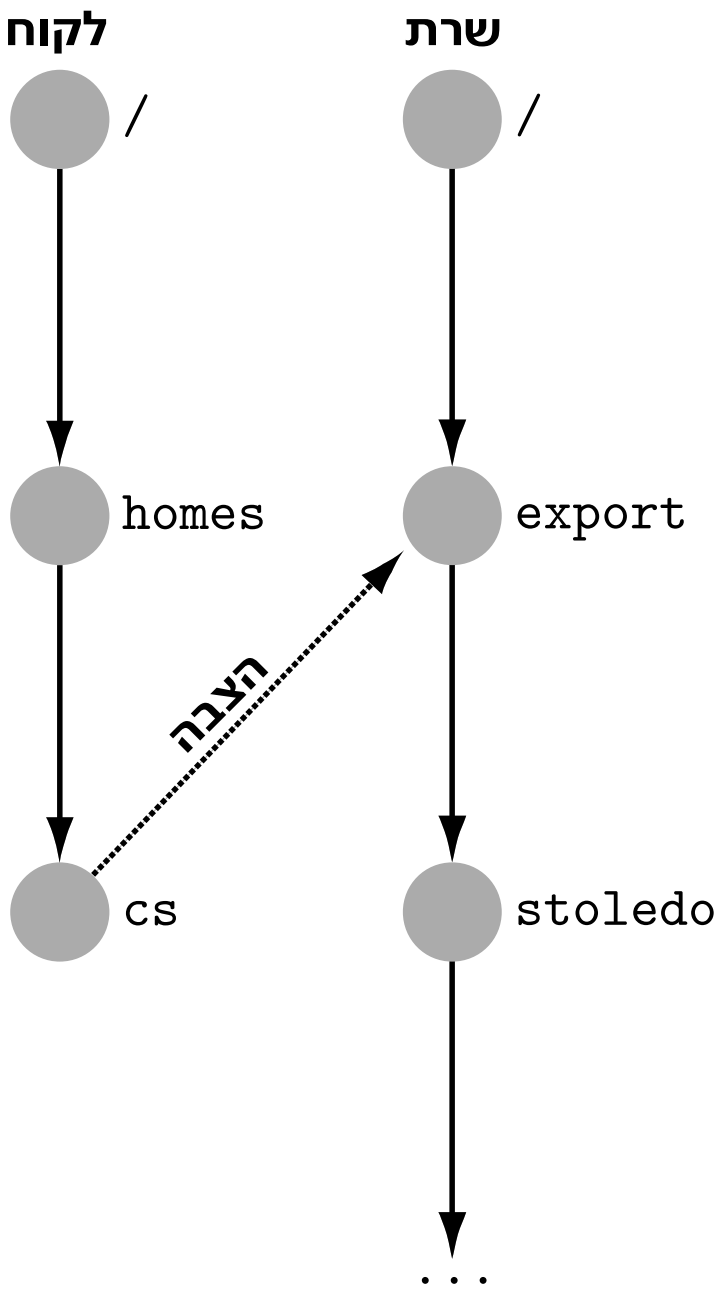
למה לא מערכת הפעלה אחת?

- במחשב יחיד כל רכיבי החומרה פועלים או שהמחשב מכובה/מקולקל.
- במערכת מבוזרת יתכנו מצבים רבים שבהם חלק מהרכיבים מקולקלים או כבויים ועדיין אנו רוצים להשתמש בשאר.
- במחשב יחיד ההגנה של מערכת ההפעלה מוחלטת.
- במערכת מבוזרת עלולות להסתנן לרשת הודעות שלא סוננו על ידי מערכת ההפעלה.
- צריך להתחשב באיטיות רשת התקשורת.

מערכות קבצים מבוזרות

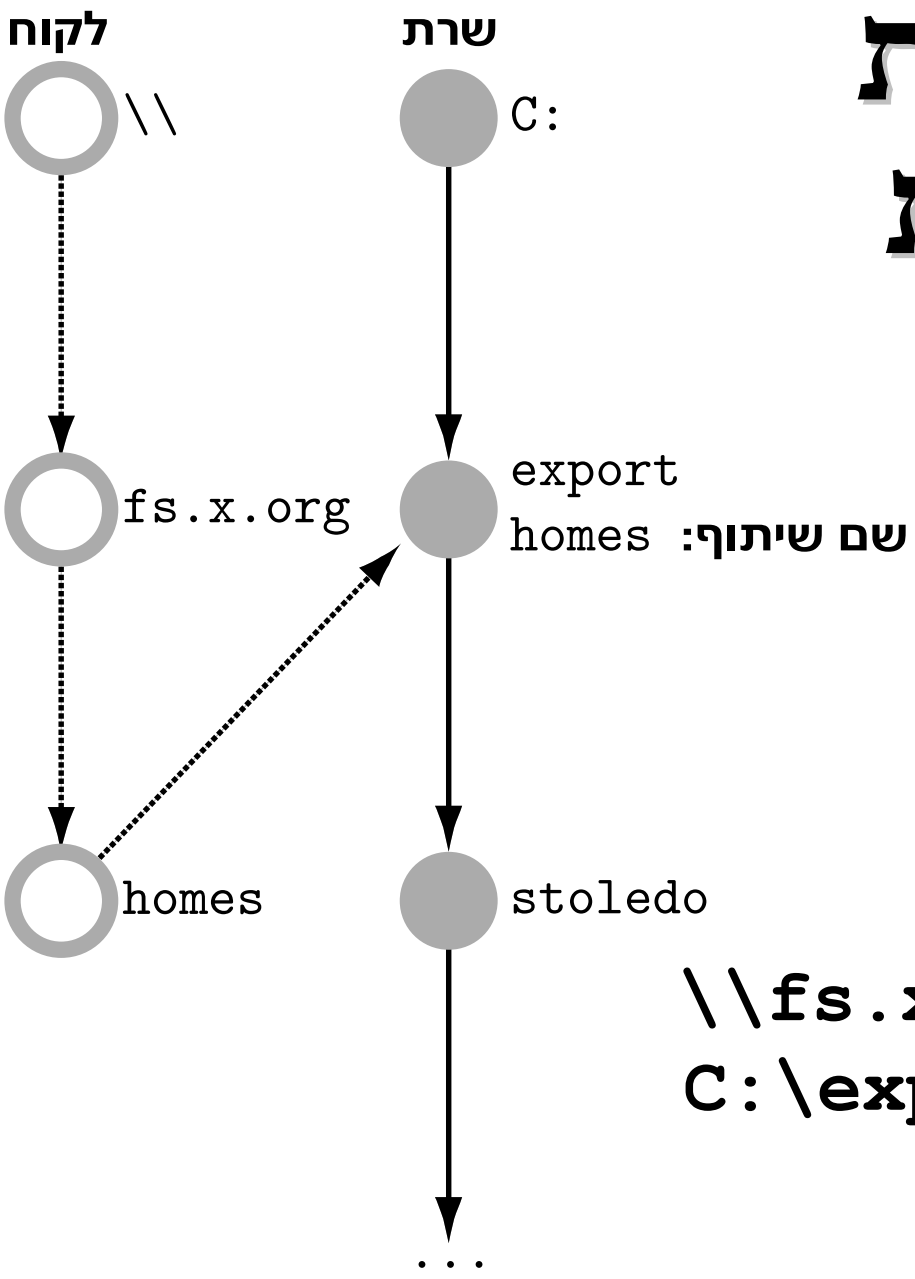
- מאפשרות לגשת ממספר מחשבים (לקוחות) לקבצים שמאוחסנים במחשב אחר (שרת).
- מאפשרות לפזר עומס חישובי בין לקוחות.
- מאפשרות למשתמשים גישה לקבציהם ממקומות גיאוגרפיים שונים (מחדרים שונים ועד יבשות שונות).
- לעיתים קל יותר לנהל קבצים בשרת מרכזי (למשל, קל יותר להבטיח שקבצים יגובו או שתהיה יתירות).
- לעיתים זול יותר לרכוש נפח דיסקים עבור שרת מרכזי. למשל, יתכן שאין צורך בדיסק שלם עבור כל תחנת עבודה. מאידך, לעיתים דיסקים לשרתים יקרים יותר מדיסקים לתחנות עבודה.

הצבה שרירותית במרחב השמות



מערכת הקבצים ששורשה ב-
/export בשרת מוצבת על גבי
/homes/cs בלקוח

שיקוף שם שרת במרחב השמות



בלקוח, השם

\\fs.x.org\homes\stoledo

מתייחס ל-**C:\export\stoledo**

בשרת **fs.x.org**

עוד על שמות קבצים מרוחקים

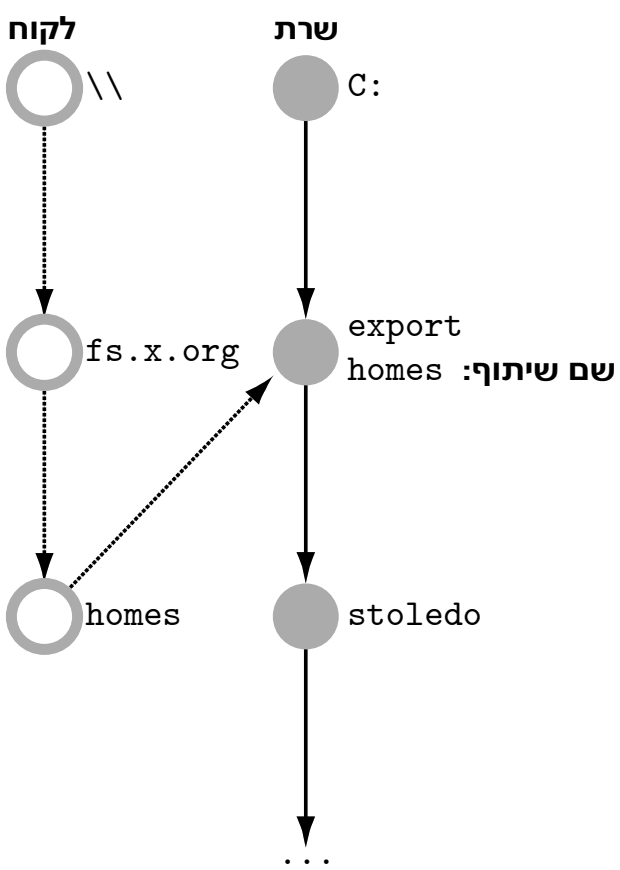
- מערכת AFS בעולם היוניקס:
  `/afs/ibm.com/stoledo/a.out`
- בחלונות ניתן להציב קבצים מרוחקים רק באות כונן; החל מחלונות 2000 ניתן להציב בנקודה שרירותית.
- שרת חלונות משתמש בשם חיצוני (share name) עבור תתי מרחבים שהוא מספק.
- שיקוף שמות שרתים משמש גם בגישה לקבצים באינטרנט דרך URL–ים, לגישה לדואר אלקטרוני בפרוטוקולי POP ו–IMAP, וכדומה.

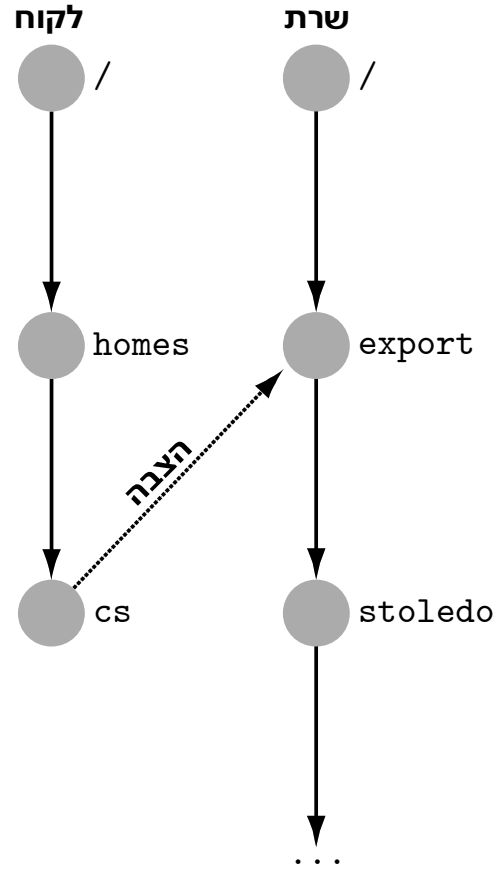
השוואה

הצבה שרירותית	שם קובץ משקף את השרת
שם הקובץ מציין את רק את תוכנו	שם הקובץ מכיל גם את מיקומו
הצבה דורשת לפעמים התערבות של מנהל המערכת	המשתמש יכול לגשת לכל שרת שיש לו הרשאות לגשת אליו
בלבול אם לא ניתן להציב במקום מסוים או אם קבצים מוצבים בנקודות שונות בלקוחות שונים	שם קובץ קבוע
החלפת שרת ועדכון ההצבה ללא מודעות של המשתמשים	משתמשים מודעים להחלפת שרת

פענוח שמות קבצים מרוחקים

מערכת ההפעלה של הלקוח מזהה נקודת הצבה של מערכת קבצים מרוחקת או תחילית שמציינת שמות מרוחקים, שולחת את סוף השם לשרת לפענוח, ומקבלת חזרה מזהה לקובץ.

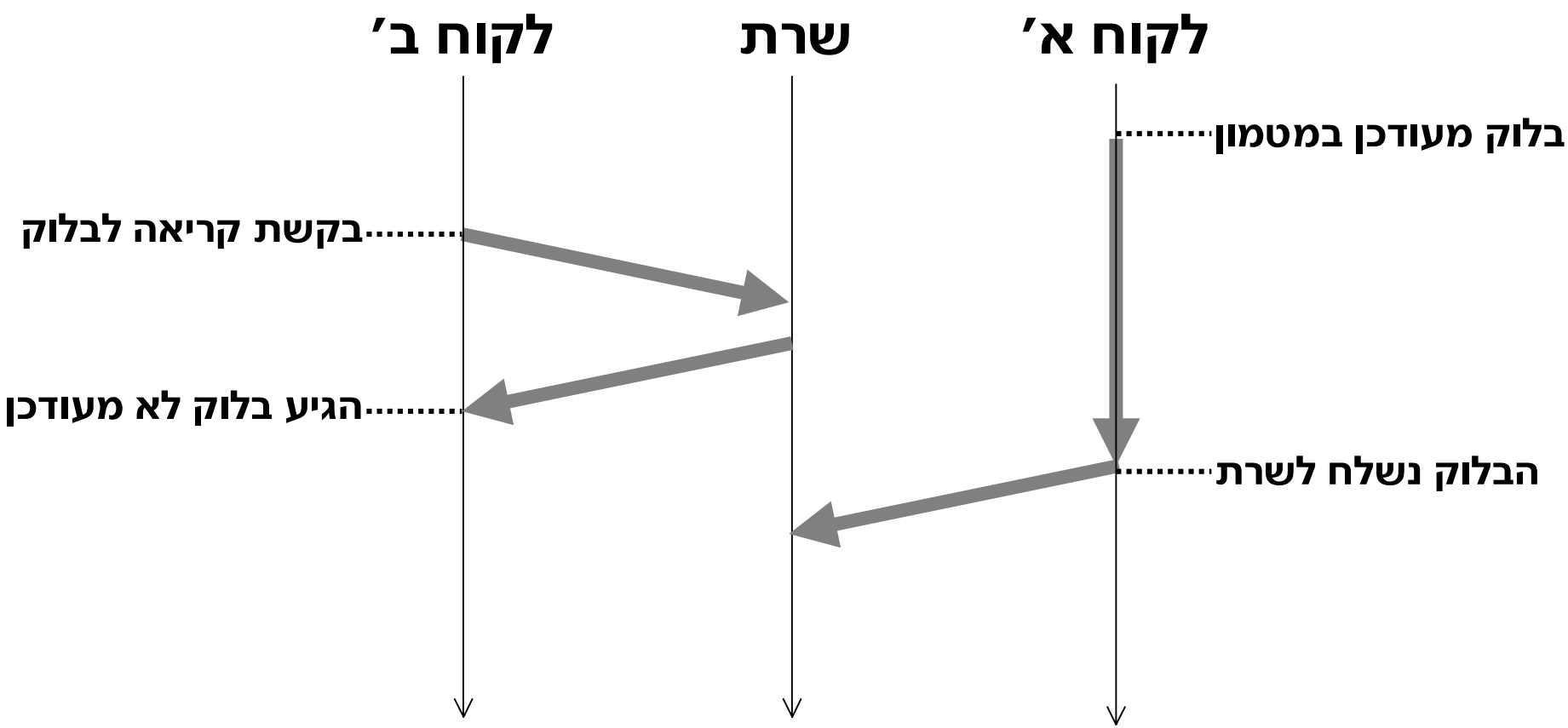




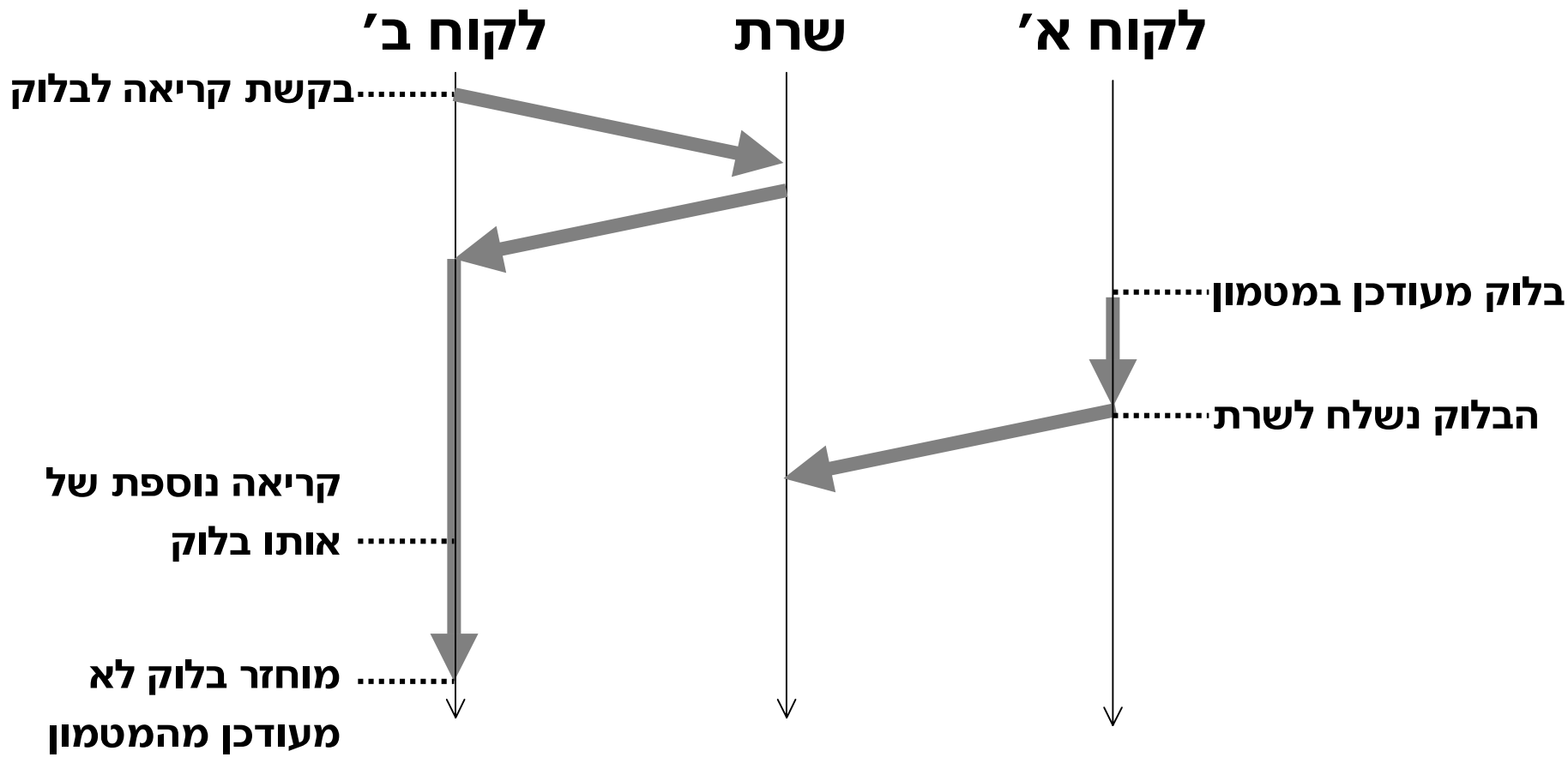
הטמנת נתונים בלקוח

- הלקוח מטמין נתונים (לקריאה חוזרת או ולחציצת כתיבות) על מנת להתגבר על שתי בעיות:
  - **תקשורת איטית בין השרת והלקוח**
  - **עומס על השרת ממספר לקוחות**
- דוגמה: רשת של 100 Mb/s לעומת דיסק מקומי עם קצב העברה של של 20 MB/s = 160 Mb/s
- דוגמה: רשת של 1 Gb/s ושרת עם מערך דיסקים עם קצב העברה כולל של 100 MB/s = 800 Mb/s

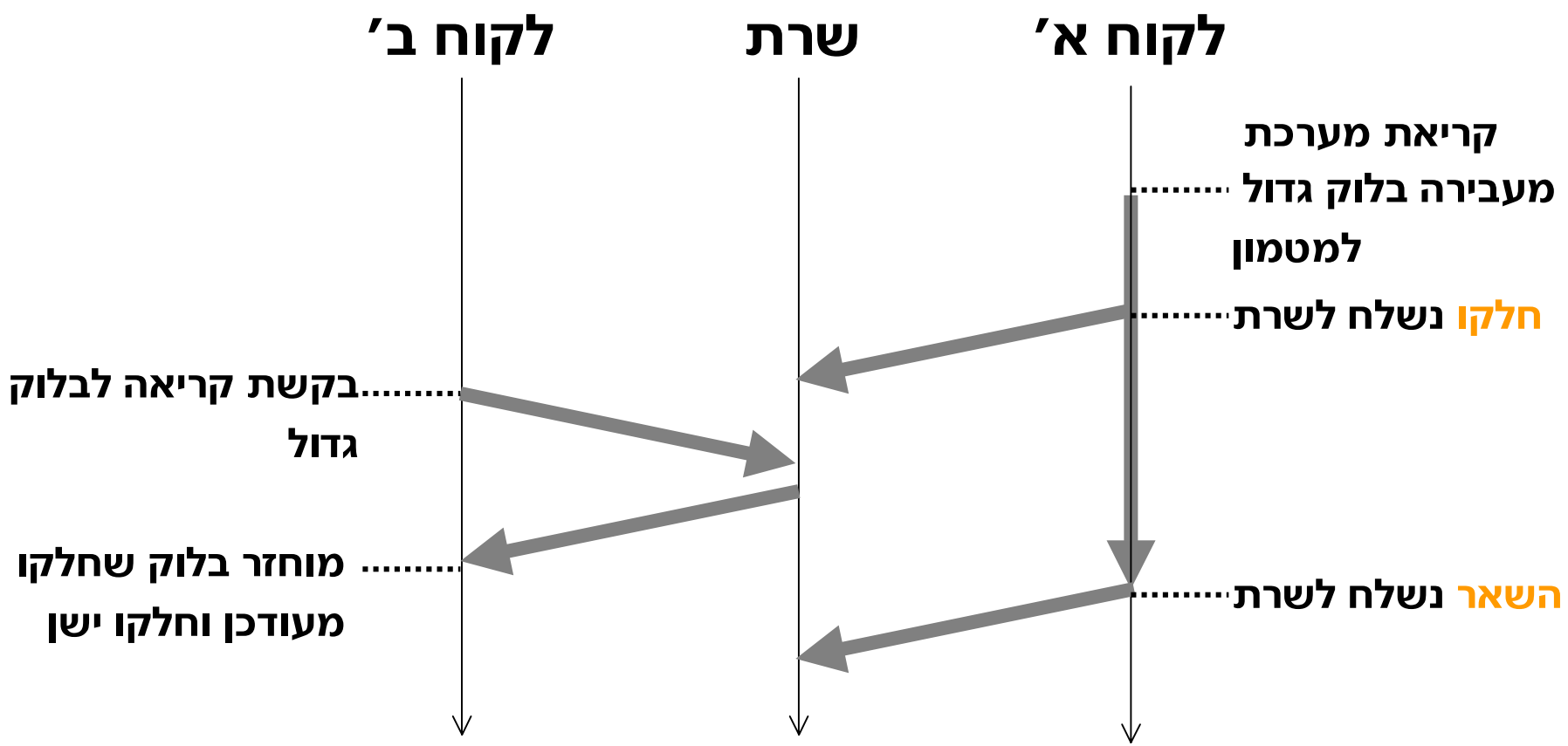
הטמנה עלולה לפגוע בסמנטיקה (1)



הטמנה עלולה לפגוע בסמנטיקה (2)



הטמנה עלולה לפגוע בסמנטיקה (3)



פתרונות לבעיית הסמנטיקה

- לא להטמין נתונים כלל
  - כל בקשת גישה מועברת לשרת
  - עלולה לצוץ בעיית ביצועים קשה, כמעט לא בשימוש
- session semantics במערכת הקבצים AFS:
  - הקובץ מועתק ללקוח כאשר פותחים אותו
  - פעולות בלקוח מתבצעות על העותק המקומי
  - עותק מעודכן מוחזר לשרת כאשר סוגרים את הקובץ
  - ביצועים טובים אלא אם ניגשים לחלקים של קבצים
  - סמנטיקה חלשה לשיתוף נתונים (צריך לחכות לסגירה) אבל ברורה
- פרוטוקול מיוחד לשמירה על קונסיסטנטיות (נתאר בהמשך)
- התעלמות מבעיות הסמנטיקה על מנת למרב ביצועים: NFS

הפרוטוקול חסר המצב של NFS

- במערכת הקבצים המבוזרת NFS השרת אינו זוכר דבר אודות לקוחות פרט לזהות המחשבים שמותר לו לשרת, ולקוחות אינם זוכרים דבר אודות השרת פרט לזהותו
- כתוצאה מכך, נפילה או התנתקות מהרשת של לקוחות אינה משפיעה על השרת כלל
- נפילה של שרת אינה משפיעה על לקוחות אם הוא עולה חזרה תוך פרק זמן סביר
- הפרוטוקול חסר המצב משיג שרידות גבוהה במערכות מבוזרות עם מרכיבים לא אמינים

פתיחת קובץ NFS

- כשתהליך פותח בלקוח קובץ, הלקוח שולח לשרת בקשת פענוח עם שם הקובץ
- השרת מוודא שהלקוח מורשה ושהמשתמש רשאי לגשת לקובץ
- השרת מחזיר ללקוח מזהה שכולל את מיקום הקובץ בדיסק (inode) ואת זמן יצירת הקובץ
- המזהה תקף עבור גישות לשרת כל זמן שלא נוצר בשרת קובץ חדש באותו מיקום; המזהה תקף גם אם השרת נפל ועלה חזרה

פעולות על קבצי NFS

- בקשות קריאה מקבצי NFS פתוחים מועברות לשרת (עם המזהה והמיקום בקובץ שממנו קוראים)
- הלקוח שומר במטמון בלוקים שנקראו ונכתבו לאחרונה
- בקשת קריאה לבלוק שנשמר במטמון מחזירה את תוכנו מהמטמון אלא אם הוא ישן מאוד (חצי דקה)
- בלוקים נכתבים לשרת בהדרגה אבל לפני ש–close חוזר
- אין צורך להודיע לשרת שקובץ נסגר
- כל התקשורת בין השרת ולקוחות מבוססת על בלוקים בגודל קבוע
- השרת מאשר קבלת וביצוע כל בקשת שירות, כולל כתיבת בלוקים ופעולות על מרחב השמות

וידוא ביצוע ב-NFS

- הלקוח חייב לוודא קבלת אישור על כל בקשת שירות
  - NFS רץ בדרך כלל על UDP שאינו אמין (אין בעיה כזו ב–TCP)
  - יתכן שהשרת נפל ולא קיבל את הבקשה או שקיבל ולא הספיק לבצע
- אי קבלת אישור לאחר זמן סביר גורמת למשלוח בקשה חוזרת
- השרת עלול לקבל בקשה יותר מפעם אחת
  - יתכן שהבקשה הראשונה או האישור עליה פשוט התעכבו ברשת
  - יתכן שהשרת ביצע את הפעולה בבקשה הראשונה אבל האישור אבד
- מכיון שהשרת חסר מצב הוא אינו זוכר שביצע כבר את הבקשה
- השרת עשוי לבצע את אותה פעולה יותר מפעם אחת
- במידת האפשר, הבקשות צריכות להיות אידמפוטנטיות

פעולות לא אידמפוטנטיות

- רוב הבקשות מגדירות פעולות אידמפוטנטיות, כתיבה למשל
- בקשות מסוימות לא ניתן להגדיר כאידמפוטנטיות, למשל יצירת קובץ תוך וידוא שאינו קיים (open עם דגלי O_CREAT, O_EXCL)
  - הביצוע הראשון ייצור את הקובץ
  - הביצוע השני ייכשל כי הקובץ קיים ויחזיר קוד שגיאה ללקוח
- הלקוח צריך להיות מוכן לקבל הודעות שגיאה לא נכונות ולטפל בהן בצורה סבירה
  - בדוגמה, לברר את תכונות הקובץ, מי יצר אותו ומתי, או
  - לדווח למשתמש על הבעיה אבל גם להסביר בדיווח שאולי הוא שיקרי
- עוד הבדל בסמנטיקה: קבצים פתוחים עלולים להימחק כי השרת אינו מודע כלל לכך פתוחים

קיבוץ כתיבות

- אישור על כתיבה מורה שהשרת כתב את הבלוק למדיה יציבה
  - הלקוח יכול למחוק את הבלוק המעודכן מהמטמון
  - כתיבת בלוק בודד עלולה להצריך יותר מגישה אחת לדיסק (אם מגדילים קובץ) ולא ניתן לעכב אותה כדי לא לעכב את האישור
  - כתיבה מיידית כזו גורמת לבעיית ביצועים בדיסקים של השרת
- בגרסה 3 הלקוח יכול להרשות לשרת לעכב את הכתיבה
  - כאשר הלקוח רוצה לכפות כתיבה בזמן סגירת קובץ או על מנת לפנות מקום במטמון, הוא שולח הודעת **commit**
  - השרת כותב את כל הבלוקים ומחזיר את זהות הבלוקים שנכתבו
  - הלקוח צריך להיות מסוגל לטפל בכתיבה חלקית (עקב נפילה של השרת, למשל) ולשלוח את הבלוקים החסרים שוב
- מאפשר לשרת לבצע מספר כתיבות בסדר יותר יעיל
- פרוטוקול עם מצב, אבל מצב שמותר לשכוח

סוגי הצבות של מערכות NFS

- הצבה קשיחה
  - הלקוח לא מוותר לעולם
  - קריאת מערכת חוזרת רק אחרי שהשרת מבצע את השירות
  - תהליכים שמבצעים פעולות מול שרת שנפל נתקעים
- הצבה רכה
  - קריאות מערכת חוזרות עם קוד שגיאה אם השרת לא עונה תוך זמן סביר
- הצבה ניתנת לפסיקה
  - קריאת מערכת חוזרת אחרי שהשרת מבצע את השירות או שהתהליך מקבל איתות (למשל **control-c**); בהעדר איתות ממשיכים לחכות

שימוש בסוגי הצבה שונים

- הצבה קשיחה: בעיקר כאשר הלקוח לא יכול למעשה לתפקד ללא השרת, כמו במקרה שבו כל מערכת הקבצים של הלקוח מרוחקת
- הצבה קשיחה מתאימה גם לסביבות אצווה (batch) שתוכניות רצות בהן זמן רב, ורצוי שפשוט יחכו לעליית השרת כשהוא נופל
- הצבה ניתנת לפסיקה: מתאימה לסביבות אינטראקטיביות שבהן תוכניות לא מתוכננות לטפל בנפילת שרת; המשתמש יחליט
- הצבה רכה: מתאימה לסביבות אינטראקטיביות שבהן תוכניות מתוכננות לטפל בנפילת שרת, למשל על ידי הודעה למשתמש ושמירת עותק של הקובץ בדיסק מקומי (בזמן "שמור")
- הצבה רכה עלולה לגרום כשלים כשהשרת עמוס מאוד

חוזי שכירות קצרי מועד ב-NQNFS

- לקוחות מבקשים "חוזה שכירות" על קבצים שהם ניגשים אליהם
  - חוזה קריאה עם הטמנה מתיר ללקוח לקרוא מהקובץ ולהטמין בלוקים
  - חוזה כתיבה עם הטמנה מתיר ללקוח לקרוא ולכתוב ולהטמין בלוקים
  - חוזה ללא הטמנה מחייב את הלקוח לתקשר עם השרת בכל גישה לקובץ
- אם יש חוזה כתיבה עם הטמנה ולקוח מבקש חוזה, השרת מבטל את החוזה הקיים, מודיע על כך לבעליו, ומעניק חוזים ללא הטמנה
- אם יש חוזה עם הטמנה כלשהו בתוקף ולקוח מבקש חוזה כתיבה, השרת מבטל את החוזים הקיימים ומעניק חוזים ללא הטמנה
- אם יש חוזה כתיבה ללא הטמנה בתוקף ולקוח מבקש חוזה הוא מקבל חוזה ללא הטמנה
- אם יש רק חוזי קריאה ולקוח מבקש חוזה קריאה, הוא יורשה להטמין
- תוקף כל החוזים מוגבל בזמן (כדקה) ויש לחדש אותם

חוזי שכירות ושרידות

- חוזים אבודים (הלקוח והשרת לא מסוגלים לתקשר) עלולים לגרום לשרת לסרב להעניק חוזים חדשים או לפגיעה בקונסיסטנטיות
- שרת שצריך לבטל חוזה ואינו מצליח לתקשר עם הלקוח פשוט ממתין לפקיעת החוזה; שני הצדדים יודעים אז שהחוזה לא בתוקף
- לקוח שהיה בידו חוזה כתיבה עם הטמנה והחוזה מבוטל (מפורשות או בשל תפוגה) חייב לשלוח מייד את העדכונים לשרת
- שרת שנופל ועולה ואינו יודע איזה חוזים יש אולי בתוקף אצל לקוחות פשוט מחכה לזמן הפקיעה המקסימלי לפני שהוא מתחיל להעניק חוזים חדשים
- סיכום: הטמנה רק אם אין העברת מידע בין לקוחות דרך קבצים, יש מצב בשרת ולקוחות אבל שיכחון אינו אסון (כמו **commit**)

פרק 10
הגנה ואבטחה

על מה נדון

- הסכנות האורבות למערכות מחשב: פריצות ותקיפות
  - פריצה: מי שאינו משתמש לגיטימי מצליח לבצע פעולות או משתמש לגיטימי מצליח לבצע פעולות שאינו מורשה לבצע
  - תקיפה: גורם חיצוני גורם נזק בלי להשיג תועלת ישירה
- אימות זהות: מי שם?
- הרשאות: איזה פעולות מותרות לכל משתמש
- מנגנוני רישום: מי עשה מה
- הגברת הרשאות והתחזות מותרת

נזקים מפריצות

- קריאת מידע על ידי מי שאינו מורשה (מבחן, למשל)
- שינוי או הוספת מידע בקובץ (שינוי ציון, שינוי יתרה בבנק)
- מחיקת מידע (תיק פלילי, עדויות לעבירה)
- שימוש במשאבים ללא תשלום (כוח חישוב, תקשורת)

סיבות לפריצות

- גישה פיזית לחומרה
  - גניבה של מחשב או רק של דיסקים או סרטים
  - ציטוט לקו תקשורת או לתקשורת אלחוטית
  - חדירה לחדר מחשב שממנו ניתן להשתמש במחשב ללא אימות זהות
- התחזות (גניבת זהות של משתמש לגיטימי)
  - מבחינת מערכת ההפעלה, מי שעובר את מנגנון אימות הזהות (סיסמה למשל) הוא משתמש לגיטימי
- סוס טרויאני
  - תוכנית שגורמת למשתמש לגיטימי לבצע פעולות שמותרות לו מבלי שהוא מודע לכך שהוא מבצע אותן
  - הפעולות הללו משרתות את הפורץ (למשל שולחות לו מידע) ו/או מזיקות למשתמש הלגיטימי
  - לעיתים התוכנית מבצעת בנוסף שירות מועיל למשתמש ולפעמים הוא רק סבור שהיא תועיל לו

סוסים טרויאניים

- וירוסים שמגיעים כתוספת לדואר אלקטרוני
  - המשתמש מריץ תוכנית או פותח קובץ שמכיל גם תוכנית
  - היכולת להריץ תוכניות מתוך תוכנת דואר אלקטרוני או מתוך מעבד תמלילים שימושית לעיתים אבל יוצרת פרצת אבטחה
- תוכנות מסחריות ששולחות מידע אודות המשתמש לחברה
  - התוכנה מבצעת את תפקידה אבל גם אוספת מידע ללא ידיעת המשתמש
- ניצול תכונה לא מתוכננת של תוכניות
  - תולעת האינטרנט ניצלה פגמים בתוכנות שרת על מנת להחדיר למחשבים קוד שרץ תחת הרשאות **root**.
  - פגמים דומים התגלו בתוכנות רבות

הקטנת החשיפה לסוסים טרויאניים

- הימנעות משימוש בתוכניות ממקור לא ידוע
  - סיכויים פחותים לסוס טרויאני בתוכנה של חברה ידועה משום שגם לחברה יש אינטרס למנוע הימצאות סוס טרויאני בתוכנה (פגיעה בשמה)
- במקרים קיצוניים, הימנעות מהרצת תוכנה שלא ניתן לבדוק את קוד המקור שלה
  - תוכנות פופולריות עם קוד פתוח נקראות על ידי רבים והסיכוי לסוס טרויאני קטן
- הרצת תוכנות שרת שעלולות להיות חשופות לתקיפה (שרתי HTTP למשל) עם הרשאות מינימליות לביצוע תפקידן
- ביקורות על גישה לקבצים ועל מידע שיוצא לרשת

תקיפות מניעת שירות

- שימוש אינטנסיבי במשאבים שגורם למניעת שירות או מתן שירות איטי למשתמשים לגיטימיים
- בדרך כלל התקיפה מתבצעת על ידי שימוש בשירות שניתן לכל מחשב ברשת, לא רק לקבוצת משתמשים קטנה, למשל
  - שרתי HTTP מספקים קבצים לכל דורש
  - שרתי דואר אלקטרוני מוכנים לקבל דואר מכל מקור
- הצפת שרתים כאלה בבקשות שירות מאיטה או מפילה את השרת
  - נפילות בגלל פגמים שלא מתגלים תחת עומס רגיל או בגלל מיצוי של משאבים, כמו קבצי יומן אירועים שממלאים את הדיסק
- קשה למנוע כאלה תקיפות כי הבקשות לגיטימיות באופיין

אימות זהות: סיסמאות

- מחרוזת שרק המשתמש הלגיטימי יודע
- אמצעי זיהוי נוח ואמין
- אנשים נוטים לשכוח סיסמאות שאינם משתמשים בהן בתדירות
- אנשים רושמים סיסמאות קשות לזכירה או שמתחלפות תדיר
  - פורצים עלולים למצוא או לגנוב את הרישומים הללו
- אנשים בוחרים סיסמאות צפויות אם נותנים להם לבחור
  - שמות פרטיים, ימי הולדת, וכדומה

פיצוח סיסמאות

- ניחוש (אם הן קלות)
- בדיקת סיסמאות רבות באופן אוטומטי על ידי תוכנית
  - אפשרי אם ניתן להפעיל את תוכנית ההתחברות באופן לא אינטראקטיבי
- ציטוט לסיסמה כאשר היא עוברת ברשת או בקו תקשורת
- גישה לקובץ שסיסמאות רשומות בו
- שימוש במידע שדולף ממנגנון בדיקת הסיסמה (מעבר לכן/לא)
- הרצת תוכנית שמתחזה לתוכנית ההתחברות הרגילה של המחשב וקולטת את שם המשתמש והסיסמה שלו

התגוננות מפיצוח סיסמאות

- להכריח משתמשים לבחור סיסמאות ארוכות וקשות לניחוש
  - ניתן גם להכריח משתמשים להחליף סיסמאות לעיתים קרובות
  - עלול להיות בומרנג: אנשים נוטים לרשום הסיסמאות קשות
- מנגנון בדיקת הסיסמאות צריך לענות רק כן/לא ולא לאפשר ניסיונות התחברות חוזרים מהירים
- סיסמאות צריכות להיות מועברות בקווי תקשורת ולהיות שמורות בקובץ בצורה מוצפנת
  - מערכת ההפעלה שומרת רק פונקציה **f(p)** של הסיסמה **p**. בכל פעם שהמשתמש מקליד את **p** מחשבים מייד את **f(p)** על מנת לבדוק את הסיסמה, ומוחקים מייד את **p** מהזיכרון
  - אם הפונקציה **f** ידועה (**`crypt`** ביוניקס/לינוקס) קובץ הסיסמאות צריך להיות מוגן מפני קריאה על ידי משתמשים רגילים
- מנגנון הפעלה מיוחד לתוכנית ההתחברות (**alt-ctrl-del**)

אימות זהות: אתגרים וסיסמאות חד-פעמיות

- סיסמה חד פעמית:
  - דף מודפס שבכל התחברות משתמשים בסיסמה אחרת שבו
  - מחשבון מיוחד שמייצר סיסמה חדשה כל פרק זמן קצר (כדקה)
  - המחשב יודע מהי הסיסמה הבאה בסדרה או הסיסמה לכל נקודת זמן
  - ציטוט אינו מאפשר התחברות
- אתגר
  - מחשבון שעונה על חידות שהמחשב שמתחברים אליו מציג
  - המחשב יודע מה התשובה הנכונה לכל חידה (תלוי בזמן ובמחשבון)
- בשני המקרים: אימות זהות על ידי הוכחת בעלות על חפץ פיזי
- בדרך כלל בשילוב סיסמה רגילה למניעת פריצה על ידי גניבת החפץ
- שיטה יקרה (מחשבונים), מסורבלת ליישום ושימוש, אך בטוחה

אימות זהות ביומטרי

- זיהוי אדם על פי תכונות פיזיות: מאפייני קול, טביעת אצבע, פנים, צורת הרשתית
- לחלק מהתכונות דרושים אמצעי קלט מיוחדים (לטביעות אצבע ורשתית), לאחרים רק מיקרופון או מצלמה
- אחוז טעויות מסוים בדרך כלל (מניעת התחברות ממשתמש לגיטימי או אישור התחברות למתחזה)
- ניתן לעיתים לשטות במנגנון האימות על ידי הצגת הקלטה של המשתמש הלגיטימי או תמונה שלו (או בובה שלו)
  - יש דרכים להתגבר על כך, למשל על ידי כך שמבקשים מהמשתמש לומר משפט אקראי ובודקים גם את תוכן הדברים וגם את חתימת הקול

הרשאות

- עצמים ברי הגנה: קבצים, מדריכים, שקעים, התקנים, תהליכים (בחלונות גם מנעולים ואירועים)
- לכל סוג עצם יש פעולות שניתן לאסור או להתיר למשתמשים
  - קריאה וכתיבה של קובץ, קריאה וכתיבה מהתקן או שקע
  - להרוג תהליך או לשלוח לו איתות
  - להודיע על אירוע, לחכות לו, לנעול ולשחרר מנעול (בחלונות)
- למשתמש הכל-יכול (root, administrator) מותר הכל

מטריצת גישה:

	אמיר	חזי	סיון	ענת
a.out	read,wr.			
intro.doc	read	read,wr.	read	read
תהליך 213		kill		

ייצוג מטריצת הגישה

- **אין** במערכת ההפעלה מבנה נתונים אחד שמייצג את המטריצה
- הרשאות מיוצגות באופן מפוזר
  - יחד עם כל עצם נשמרת רשימת בקרת הגישה שלו (שורה במטריצה)
  - תהליכים עשויים להחזיק הרשאות נוספות שנקראות יכולות
  - המטריצה היא איחוד רשימות בקרת הגישה והיכולות
- רשימות בקרת גישה שמורות בצורה דחוסה מכיוון שהן עלולות להתארך מאוד במערכות מרובות משתמשים

דחיסת רשימות בקרת גישה

- קבוצות משתמשים
  - ניתן להגדיר קבוצות שרירותיות של משתמשים (תלמידי שנה ב׳, למשל)
  - איבר אחד ברשימה מתיר גישה עבור כל המשתמשים בקבוצה
- רשימות הרשה/סרב
  - רשימת בקרת הגישה מכילה רשומות הרשאה ורשומות סירוב לפעולות
  - הרשימה נסרקת לפי סדר בזמן בדיקת הרשאות
  - האיבר הראשון שתקף לגבי המשתמש והפעולה קובע הרשאה או סירוב
  - רשומות מאוחרות קובעות כללים (להרשאה/סירוב), רשומות מוקדמות קובעות יוצאים מן הכלל
- שיתוף רשימות זהות בין עצמים (במערכת הקבצים של חלונות 2000)

בקרת גישה במערכות ספציפיות

- בחלונות 2000/NT (אבל לא ב-FAT): רשימות הרשה/סרב כלליות
- ביוניקס ולינוקס רשימות הרשה/סרב עם שלוש קבוצות בדיוק, ושלוש פעולות מותרות או אסורות לכל קבוצה
  - קבוצת כל המשתמשים, קבוצה שרירותית, וקבוצת המשתמש בעל הקובץ
  - קריאה ,כתיבה, והרצה של קובץ כתוכנית או פענוח דרך מדריך
  - ניתן לקבוע ברירת מחדל שאוסרת כל אחת מתשע הגישות (umask)
  - דוגמה: `-rw------x` מתירה לבעל הקובץ קריאה/כתיבה, אוסרת על כל סוגי הגישה לחברי הקבוצה, ומתירה קריאה לכל מי שאינו בקבוצה

יכולות

- מזהה משאב בחלונות ובלינוקס/יוניקס הוא למעשה מצביע ליכולת שמתירה פעולות מסוימות על עצם מסוים
- ההרשאות נבדקות בזמן פתיחת העצם והיכולת נשמרת במבנה נתונים של מערכת ההפעלה; התהליך מקבל מזהה ליכולת
- היכולת ממשיכה להיות תקפה גם אם ההרשאות של העצם משתנות
- ניתן להעביר יכולות מתהליך לתהליך: ממילא תהליך יכול לבצע עבור תהליך אחר גישה לקובץ וכדומה
  - ביוניקס/לינוקס העברה של מזהי קובץ פתוח דרך שקע

מדיניות הרשאות

- כללים שקובעים מה מותר למי
- את המדיניות צריך לממש בעזרת
  - ברירות מחדל ליצירת קבצים חדשים (umask)
  - מנגנון ירושת ההרשאות בין מדריכים בחלונות
  - מנגנון קביעת בעלות על קבצים חדשים (סיבית setgid במדריכים)
  - בקרה על הרשאות של קבצים קיימים
- דוגמה: קבצים חדשים במדיכים של פרוייקטים נגישים לכל חברי הפרוייקט, קבצים חדשים במדריכי הבית נגישים רק למשתמש
  - המימוש ביוניקס ולינוקס מתואר בספר

מנגנוני רישום

- מי עשה מה (או אפילו רק ניסה)
- מצביע על ניסיונות גישה לא לגיטימיים ומאפשר לדעת מי קרא או שינה קבצים ומתי
- בחלונות ניתן לצרף לעצמים רשימת רישום גישה שמורה איזה פעולות של איזה משתמשים יש לרשום (עלול לייצר כמויות מידע גדולות!)
- אין מנגנון דומה בלינוקס ויוניקס אבל יש תוכנות שמזהות שינויים בקבצים חשובים
- יש בלינוקס/יוניקס מנגנון רישום לאירועים חשובים (לא נסיונות גישה) כמו נסיונות התחברות של **root**

התחזות מותרת

- לעיתים צריך לבצע מטלה מסוימת עם הרשאות של תהליך אחר
  - תוכנת שרת שמריצה תוכניות במועדים קבועים עבור משתמשים צריכה להריץ כל תכנית עם ההרשאות של המשתמש שביקש להריץ אותה
  - הרצה של תסריטים דרך שרת אינטרנט (CGI, ASP)
  - תוכנות שרת שמאפשרות התחברות למחשב (login, ssh, telnet)
  - תוכנות להעברת קבצים (ftp)
- לתהליך של המשתמש הכל יכול מותר להתחזות לכל משתמש
  - בחלונות ניתן להעניק יכולת התחזות לתהליכים של משתמשים אחרים
  - ביוניקס אפשר להתיר לתהליך שמריץ תוכנית להתחזות לבעל התוכנית

הגברה

- לעיתים צריך לעדן את מנגנון ההרשאות
  - להתיר למשתמשים לשנות קובץ, אבל רק שינויים מסוימים
  - להציב ולהסיר מערכת קבצים מתקליטור אבל לא מערכות קב׳ אחרות
  - ...
- הגברה ביוניקס/לינוקס
  - משתמש בעל משאב שרוצה לפקח על הגישות אליו משתמש בתוכנית שמשתמשים אחרים מריצים על מנת לגשת למשאב
  - התוכנית בבעלות המשתמש ושמורה עם סיבית **setuid** דולקת
  - התהליך שמריץ את התוכנית יכול לעבור בין זהות המשתמש המריץ ובין המשתמש בעל התוכנית, שהוא גם בעל המשאב; הרשאות המריץ הוגברו

פרק 11
מבנה מערכת ההפעלה ואיתחולה

פרק 12
ממערכות הפעלה למערכות מחשב