www.cs.huji.ac.il/~os

המרצה: ד"ר עדי יסלסון

מה, מערכת ההפעלה היא התוכנה הראשונה שעלה על המחשב. היא מנהלת את כל
שאר התוכנות במחשב. ניתן לחלק את המחשב לשכבות:

ישנם מחשבים שבהם מערכת ההפעלה היא חלק מהחומרה, כמו למשל בטלפון
נייד או במכונית. ישנם מחשבים שבהם מערכת ההפעלה היא חלק מהתוכנה.
ישנם מחשבים שבהם מערכת ההפעלה היא חלק מהחומרה וחלק מהתוכנה.

תפקידי מערכת ההפעלה:

1) מתן שירותים למשתמשים ולתוכנות.
2) ניהול המשאבים. מה זה משאב? כל מה שצריך כדי להריץ תוכנה.
מה, מערכת ההפעלה עוזרת לתוכנות לתקשר עם המשאבים.
3) הפשטה של משאבים - כך נראה שיש למחשב מספיק משאבים כדי להריץ
כמה תוכנות במקביל (למשל כמה מעבדים). מה שיוצר את ההפשטה הזאת
נקרא scheduler. מנהל את הגישה למשאבים (הכוונה כאן למעבדים של
המחשב). מי יכול לגשת למשאב מתי.
4) מנהלת תכניות לריצה. התכניה קוראת ל- open.
מערכת ההפעלה מנהלת את הפעולות של פתיחת הקובץ.
5) מנהלת תכניות לריצה. התכניה שלה. התכניה קוראת ל- read.

מבחינת שאלה היא יכולה להתחיל ברשימה שאר בפעם הבאה היא
תתחיל במקום אחר. standard entry points
לא כלום ברשימה כן כלום בזכר מתחלה במקום אחר. יש לה אופן
מ"מ ההפעלה לא אומרת לחיות על פי, אלא שני. היו כאלו על פי
שלבית והיו מבלות אותם כתיבה בקבוצה שעושה מי שמפעל אותה.
תכנית שאין לה התחלה, עיבוד מסוף, אלא יש לה אופן נמרים שיהיו
2) מה ההבדלה, מניאף של אין דברים שהתכנו, היא תכנית ריאקטיבית. כי
את כתבה אומץ בכל אופן אותה וכך אותה?
ובקבוצות השבות:
1) אפשר לתת רעיון של איך שיהיה לקבוצות במקרה
כמה תסריט רעיון שלך איך שיהיה לקבוצות במקרה
ובין הפעמים.
2) התכנית השלישית כלה.
ורעיון לשפר בין השאלות אחרים.
אחכה שהוא יביא מהלכים הנשון ניביא את הנתונים המקרה הראשון
ואם את הפיסוק ואם את המקרה במקרה. במקרה שהבקר של הפיסוק
אפשרי כי הפיסוק הוא כביכול חומרה שונה מהמקרה אפשר להפעיל
מ"מ ההפעלה אבקשה את התכנית ומותחת תכנית אחרת לביצוע. כלה
שבקבוצ מעלין הפקודות מהלכה. אבל במקרה שאני עושה שהם
אפשרי שאפשר לה עיבוד הזה במקרה שמבוצעים מהלכים אבקשה אפשר
3) מ"מ ההפעלה אבקשה את read. במקרה לקבוצ בלפחות ברק שהקובץ
עליו היינו רוצים שכך תכנית תוכל לעשות את כלה.
תוכל לקרוא של הפקודה. למשל halt כי שהוקבה שאבקשה את המכונה
היא הקראות שהוקבה של מ"מ ההפעלה. לא היינו רוצים שכל תכנית
שהיא מ"מ, ההפעלה ויכלה לבקש. אלא שהתכנית עושה לה systems call
ענין מהיות שהכנית תכלה לקבוצ לקבוצ. במקרה כלה יש שהקובץ איזשהו

(2)

מה מוכרח להיות שונה?

1) סופר שהוא לתכניות - למשל read. יש לזה כל אינו סיבות. בין השאר אולי גם רוצים שתכניות יפריעו אחת לשנייה. למשל אם תכנית עושה halt זה מפיל כל שאר התכניות. מלך אחד תכנית לא תדע שתכנית אחרת תיגש לקבצים פרטיים שלה, כל אלו ההפשטה עליו תאפשר את זה.

2) ניהול משאבים - יש במחשב משאבים שהתכניות מתחרות עליהם. לרוב מחשבי מאקס. כל התכניות רוצות לרוץ וגם רוצות את הזיכרון מחשב אחד זה אפשרי כמו - ביוס. אבל עושים תורות (כמו ילדות מאז שהילדות עשתה באותה בובה). על ההפשטה מנהלת אותה כמון.

על הנתונים שהם בסיכון במקל ובעזה לחלק.
רוב המשאבים שמתחרים עליהם בסיכון. אישהו צריך להחליט מי יכנס עכשיו לחדר ומי לא; מתי צריך להחליט איך תכנית שלנו עם הנתונים שהם בסיכון ומי לא.

- ניסיון לגשת לכתובת מחוץ לתחום
- חלוקה באפס
- ניסיון גישה לא חוקי לזיכרון

מתי אפשר לחזור לפקודה?

exception - פקודה שגורמת לעבור לטיפול במערכת ההפעלה של המעבד עד לפקודה הבאה.

פסיקה - interrupt - אירוע של התקנות של ה-HW. לא קשור לפקודה אחרת. היא מגיעה אל הליבה של ה-CPU למשהו אחר - לא לפקודה הבאה.

פסיקות

פסיקה היא כשהמעבד לא יבדוק את מה שהיה אמור לבצע מהמשך אלא יעבור לטפל בפסיקה. אז מה קורה? מי מטפל? מע' ההפעלה התפתחו המון מאז הנשימה. אז הסיפ היו שהיה משהו אחר שמטפל בפסיקה ולא מאפשר לזרוק בזמן שהכניסו מתחה לפעולה אלא לסיים את הפעולה ואז אפשר היה לתת לתכנית אחרת לרוץ

③

- גישה לזיכרון
- הקשר אבטחה
- סוג פעולת קלט
- הגשת נתונים אחרת תלויה

פסיקות שקשורות לתהליכים
מאפשר לעצמך אפילו מספר
תיקיות של הדפדפן ומה מובנים
סוג פעולת קלט.

- בקשת I/O
- sleep
- halt
- בקשת תקשורת

של מעבד ההפעלה.
המתכנת ה system-calls כך בעצם המשתמש
יכולות לבקש מאחרות ההפעלה. אנחנו מכיר
system-calls -כל מיני דברים שהתקיות

רשלנות פסיקה הורים ל וקטור הפסיקות ומוצאים את הכתובת
הפונקציה ל- ISR ואפליקציה את הביט של MK ואז המעבד יכנסה
לעבודה של מה שהיה בזה. לאחר בתחילת של פסיקת ליכתוב מעבד
הפעלה וכלה לעשות כל אחד, ביקורת כפי יסדר את זה ואז לחצנו
תהליכיות עבור זה להמשיך.
עוד דוגמה, כשיש תכנית שפלה לישון מישהו לדיסק את. כשיש
פסיקה שהתקבלה שזה היא מודיעים אותה ואומרים שהיו וכלה
ערוץ. בעצם אומר זה יש מפה לתכניות שיכלה כבעצם.

ולכן מהמעבדון של שבלץ פסיקה זה להפעיל את MK ב.ב. כשמאחר
הבוקנג של מעבד ההפעלה שמטפלת מפסיקה אל מתניעים אחר המעבד
עמידה הקודם שלה - שומרים מחדש את האוגרים של המשתמש וקפיצה
זה נכון השאר מכניסים אחר הביט של ה- MK.

יכניסה לה'ות המון systems calls והן כלן מאותשות ע"י פקודה trap
שמיא לא מיוחד כי אנחנו רוצים שתכניות יוכל להשתמש בזה. ומעבד
הוא שיש רק trap אחד אפל מפה שיכולות שניים שמעבד ההפעלה
וכלה לעתה.

משתנים system calls לא מלאים ישירות לקוד של מנ' ההפעלה.
הסיבה, שאפשר לקבוע להן הוקציות מתוך התכנית או הוקציות בתוך
ספריות שאנחנו מתחברים אליהן. הן שלבים כמו:

- אופציה לקבוע הוקציות מסוכנות (למשל read)
- סוגי המשתמשים עושים כמה דברים:
  1. שומרת מצבה
  2. שומרת ארגומנטים
  3. עושה trap - אומרת למערכת הוקציה מה היא
- trap אומר לניווט למנ' ההפעלה
- סוגי מנ' ההפעלה עושה כמה דברים
  1. מחלצת את המצבה
  2. מחלצת ארגומנטים
  3. switch (מצבה)

מטפלת במה שצריך
לפי המצבה

יום ד' אישי
שמח !!!

תזכורת מיבוא

עד מה נספר היום?

- מטריקות - איך מודדים ביצועים?
- עומסים - מהם עומסים על המחשב?
- שיטות
- תורים

מטריקות

מטריקה זה מדד. מה נרצה לעניין אותנו במקרה של מערכת הפעלה?

- ניצול (כמה זמן המעבד עובד ולא סתם בטל)
- throughput - כמות עיבוד למן - הספק
- overhead - עלות הפעלה
- זמן תגובה (מהבקשה ועד שקיבלנו את התגובה)
- idle time

* זמן תגובה הוא אחד המדדים החשובים שעל פיו מעריכים מערכת (הוא מתייחס לעבודות שעולות לה. אם המע' עושה זמן ההתאוששות יהיה נמוך. ואם הוא מבלבל זמן על שטויות אז כנראה מאוד תסבך.
* ההספק נמדד ביחידות של פעולות לזמן. זמן הסתם הינו רוצים שההספק יהיה כמה שיותר גבוה.
* התקורה איזה קשורה היא לניצול כיצד אנו על המשאבים שיש במחשב, כאומן היינו רוצים שהתקורה תהיה נמוכה
* נפנה מסתכלים על ההספק מ- idle-time - ניצול.
אם המחשב לא עושה כלום, כלומר הניצולת שלו נמוכה, אז מן הסתם היינו רוצים שההספק במחשב פחות טוב.

הקשר בין כמות תשובה למספק

כמות תשובה ומספק הם פרמטרים אשר נמדדים באותו יחידה.

- **תפיסה** - אם תכנות רצה את אחרי השנייה אז כמות התשובה וה מספק זה בעצם אותו דבר. המספק הוא מספר התכנות המאוזן שיחזיק כמו כמות התשובה הוא מאוזן כמו יחידת התשובה.
אפשר שם ש התכנות אחתלות, לאחר היינו מאלגרית מהותן כמות אל המספק הוא אותו המספק (אותו מספר תכנות מתבצע באותו הכמו) אפשר כמו כמות התשובה הוא כמובן אפשר יותר.
- **הבדלה** - אם יש הרבה רווחים בין התכנות, לאחר הם לא נמוכה, אל אפשר להוסיף תכנות שירוצו במרווחי הכמו ואל המספק ישתנה אפשר כמו כמות התשובה פאן.

זה תיסרון כללי של המספק - הוא תלוי בהיצעות של התכנות.

מדוד שמיבועים של מאלגרית תלויים באופי הפרמטרים שהמאלגרית פריבה שלהתמופה איתם. אלגוריתם ניתחנו באופן הבא:

כך כללי נוכל לראות כאשר כמן ניתן להחליף את האלגוריתם של הקלט ולראות כיותר.

אפשר אף, הם פשוט היו תכנית כיאקסימית - הם כלים שרצה מאו ומאר. ולפעמים המתכנית שפריבה לרוץ הם הקלט של מאו ההם פשוט. והכמן המאוזן תלוי בקלט - לפעמים כפעמים. לפי שרוצים להסביר אפ התוצאות נמצות מהפעמים או מהמאלגרית.

- התפלגות (נמני בלבד, אפשרי קבצים)

התפלגות של נמני בלבד נראית אשכו כלה:

יש הרבה מאוד תהליכים בקרים מאוד ואפס מאוד תהליכים ארוכים. נשים ♡ שלוקי לכן התפלגות אקריטית בהתפלגות אקספוננציאלית e^{-x} היסו כ- x הסתברות ההסתברות ל- x ואלו מאוד שמאפיין נמני בלבד שלה מה שנקרא **בלי זכרון**, אפשר קבצים ההסתברות x ל- x בדוגמת פיזיקוואטית, שמלי e^{-x}. לה אומר שכך יש סיכוי לן לנית לפגוש תהליכ ארוך.

ההשכבה של לה:

תהליכים: ← תהליכ אקרי היסו קצר (איכרו שניות)
← שניה של שעות מצפה היסו תלק אחרי ארוך, טוען רבה הלמן הנצפה עובד. על התהליכים הארוכים.

אפשר לחשוב אחתהליכים כמה נחמד שלא קצר אפשר שנייה בלמן היסו כלל תיפגע בתהליכ הארוך.

מאז, אינטואיציות אך שקבלה הרבה יותר קשה לפעמים הישני. יש שני פתרונות: - התכנות אינטואיציות מפנסים של. קושי ויש ארוחים - התכנות אינטואיציות בשפן אחיך ובדרכה מסוכרת הפתרונות לה בעצם אין אינטואיציה מהאקסטרמיה.

שיטות לתזמון תהליכים: מדיניות של מתזמן

- השיטה הישירה היא פשוט לנסות באופן אקראי ולמצוא את כמות ההתאמה של המתזמן וקו מיני מפקים אחרים.
שיטה היא קצת מופלאה, אם רופים שהשוות שני מתזמנים שונים - אחד סיים ואחד שהמציאו אל צריך לנסות את המתזמן שלנו ולה עכ טריוויאלי. יתר על כן, אפ יש לו מאין אל שום שבר לא שובב.

- סימולציה - תכנית אחשב שכותבים שנוח פיתוי של מתזמן והאיתית לאחר במקרה של אתמן אפ שר. על עשות תכנית שמקשלת תיאור של הבקשות ואומרת מה המתזמן יחליט. לאחר הסימולציה נוקבים אחרי מה שהיא עושה במציאת הנכון.
זה מהה יתר יאים כי אפשר לשנות מהמקיפו אפ לא פחות נאין. לזה מפוקח עיאחרי כן אם פיתוי בפל באופן אוטומטי אנו שנתנו שום פרט.

- אנליטי - שמנות אופן אתאמי של המ' שמנות אשוואות שמתארות את מה שקורה.
היתרון הגדול היא הדינמיות והבנה של מה שקורה. עם סימולציה מפני עדמה נוחות נבלפה אותה. בשעון לקפ אל מהסק מסקנות צריך לעשות הכללה מפיקות

תרגיל

התכנית ישמות בתור וכ פשה התכנית הישוה רלה במעב?

נשתנית משימה היכן בדיוק בזמן שהיא נמצאת בסטטוס
של הפיסק אמנם ייתכן שזאת תכונה רצויה
עדיפה שפיסק כך זה לו יש תלות. ובמע' מורכבת יותר
ובעלים להיות כמה פיסקים.
אחרי שתכונה משימת ייתכן שהיא תרוקן ערוץ שכ.
וייתכן שהיא שמשתמש רואה בזאת נוראי בתכונה, אשר
אותה ומצייר במע'. אנחנו מניחים שלמשתמש אין תלות.

אחרון נותר את המע' הפשוטה שפשוט יש בה שרת שמייצר
תכונה תדשור ויש לה רק תור אחד.

אחרון נניח שהשרת מייצר תכונה רק מיידית בקצב λ
בהתפלגות אקספוננציאלית והעבודה נותן שרות בקצב μ.
ההתור 'תפוס' לנו). נסמן את היחידות $\frac{\mu}{\lambda} = \rho$.

נסתכל על שני מכרים: 1 | D | D 1 | M | M

1 – אחד אחד
M – מספר שווה
D – מספר שונה

היה ברור לנו שהם מאוזנים. קל לראות שהמאזן הזה יתן יתרון מאוד מינימלי בניחוש עבור מספר התשובה המאוזן היא 0½, מפני שיש בדיוק אותו מספר של אפשרויות לכל אחד מהם, אפשר אם נסתכל על כל שאר האפשרויות שמאזנות מקבלים, אפשר לראות שיותר ארוכות ויש שיותר קצרות? נניח שאפשרות מהירות ביותר? אפילו מה שם לכה לא סטטיסטי וזוכה? מה שם יש שאלות

אז עדיין מספר התשובה הוא 0½.

אם $n = 10$, $\lambda = 5$ אז ההיגיון לראות כך

מספר התשובה הוא 0½.

והיגיון לראות כך:

אם $n = 10$, $\lambda = 1$

שלושה, אז

מה התפלאות זה גם מה שיותם נתן עבורי.

קצת השנות נקבה ב.ג.ש.ב הם ראומן מאוזנים וזוגיים

כאינו שאר פונקציות אחסקידת אבי. אלא לאן ההאבה שלה היא כלה:

כפי שהנקודה יהודה יתר. אולם השאלה היא איך נוכיח זאת? באמצעות אקסיומת האינדוקציה מכיר שכאן ההאבה עלה בידו של אספר טבעי כלשהו. אספר זה יסומן ב n, ומספר זה יהיה בעל התכונה שהוא הקטן ביותר מבין כל המספרים הטבעיים שעבורם ההאבה שגויה.

נסמן ב A את קבוצת המצבים בהם ההאבה שגויה. כלומר A היא קבוצת כל המצבים q עבורם $\delta(q,1)=0$.

מצב 0 - שלפני ההאבה ה n שלה קטן.

מצב 1 - ובלשה אחרת שבה, לאחר שאין קטן מהותנו בפעם הבאה ההאבה שגויה:

לא תלוי בקלט ההאבה של ההאבה (אפשרי

הכיוון ההפוך מאופן ש: בקצה השרות

מצב 2 - שאם נכנסה אחד ההאבה בצעד השרות של ההאבה

אותנו לנית שהתקבל אינסופי, נקי שהקבוצים יהיו סופיים.

השיטה יתר ההאבה מכיר

ההסתמכות לחלק החוכחה יתר כי קיים

אפשר לנסח את ההסתברות שאפלון את האבחים לאפסים.
ההסתברות לזה היא באפס 2 היחס שבין ההסתברות של
האלטרנטיבות האלה:

- היינו ב-1 ועברנו ל-2
- היינו ב-3 ונשארנו ב-2
- היינו ב-2 ונשארנו שם.

אותו שכתבים את המשוואה אפשר את הגרף הזה:

הגרף הזה תלוי על סדרי איכותיות בהמון תחומים. למשל
היתר מסוים הפועל אחרת לפי הגרף הזה. כאשר הנישלטר
מפואר הוא עלות תיאוש מסוים שמי שלקוח אלגוריתם.

משפיע למשפר בידוים של אפרכם שיק לאדם את הדבר
שהוא צוואר המקבוק ולהתרגל אותו מרכיב טוב יותר.

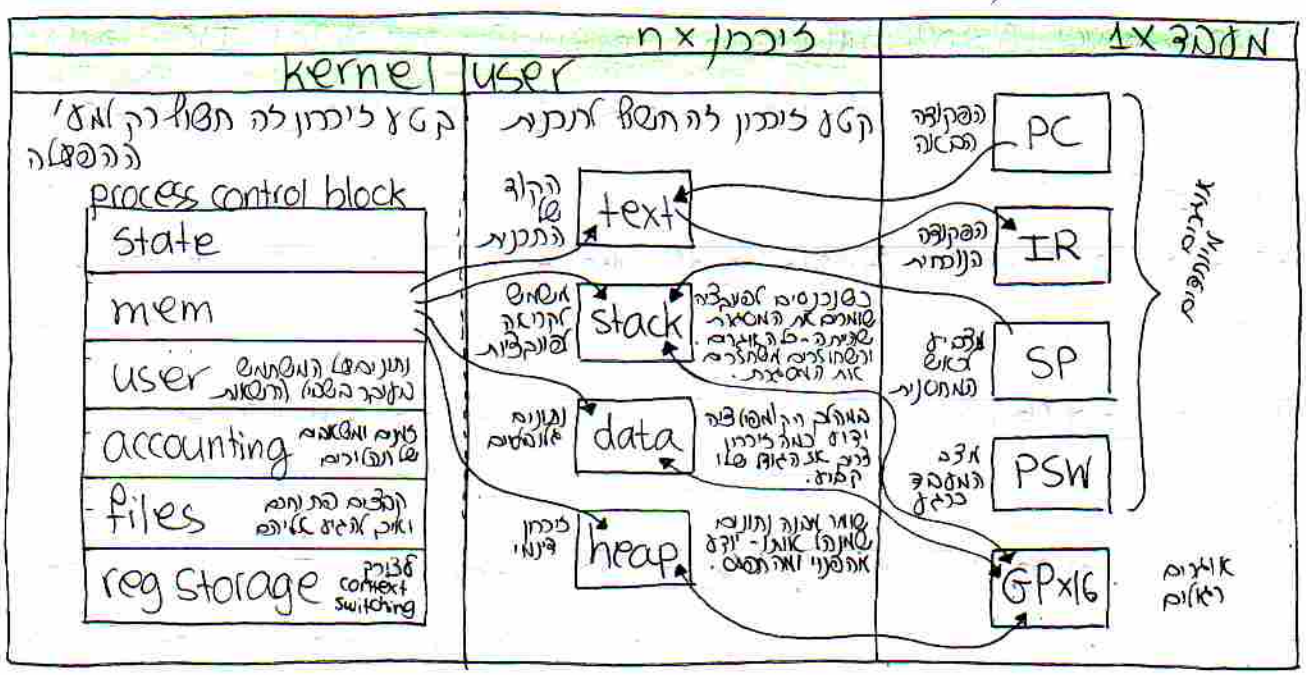
(8) 15.3.07
OS

תהליכים

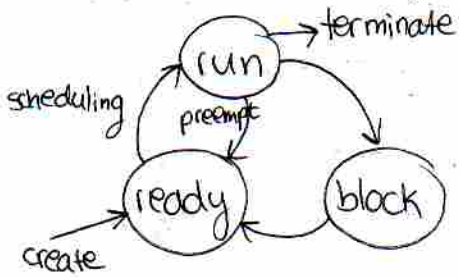
מה זה תהליך? נניח שמפעילים תכנית בריצה (באופן תמידי, אוויר). תהליך קשור לתכנית אבל לא זהה. ובדרך כלל יהיו כמה תהליכים שמריצים את אותה תכנית. למשל אם כמה אנשים מחוברים ל-server וכולם מריצים emacs אז כל אחד מהם מפעיל תהליך משלו אבל הקוד הוא אותו קוד. אם סוגרים את כל האפליקציות של התכנית עדיין לא יתכן שכל התהליכים ייעלמו. אבל אפשר להיפטר מתהליכים בודדים ומהתכנית עצמה אם נהרוג אותם.

אבסטרקציה של המחשב. הכוונה שתכנית את המחשב ראתה כאילו היא היחידה. כלומר, אין מושג שעלים על כמה תהליכים בו"ז והתכנית לא מודעת לכך.

אפשר להיפטר תהליכים בדרך של הרישום (בלבד התכנית).



מצב המעבד – תהליכים מסוים יכולים לגשת במעבד מצב מוגן. מצב בו התהליך את כל ההוראות במעבד (אם יש לו כך מעבד אחד). כאשר מתחילים את מצב ready, כאשר מוכנים לרוץ (אם היה אפשר אבל אם היו רבים). תהליך נמצא במצב ריצה (אם היה ready, כאשר מוכנים לרוץ (אם היה אפשר אבל אם היו רבים).



Threads

תהליכון נותן תחושה כאילו:
כל מה שהמכונה רואה כשהיא מתחילה לבצע תוכנית מסוג, ומהו רגע התכנית מאבדת את מהלך שלה.
הרעיון של תהליכונים הוא שתכנית תהיה מסוגלת לבצע מקומות. כל אחד שיש כאן מעבדים, ומה רגע אפשר ב"ל בקטעים שונים של התכנית.

בשביל מה לה טוב?
(תשם לאשם את שרת web. ה שהת היה
אין תכנית כאחת שמקשרת בשהה יהיה
נתונים (קובץ או בסיס). אז היה אפשר
מהאי OK ובשלה מיש היה שלח אות לה לאי שבוק. ואז היה פנוי לקבלת
בקשות נוספות. הבעיה היא שלבת בשהה אין שלהם שפים אחסוך. וזה
הכבה יותר יעיל שעבר לקבל בקשות מאין שמבואים פל. אפשר פשוט
אות פ הנתונים של הבקשה. אז היה לכתה רוטינה שיקראת כשהוא והיה
יודאי מהבקשה הנתה שהיה פולחת לפה בדיוק כדיפה שהם שהפ כאן שדיק.
אז כן פאי שאיפה בקשה תשה יקרים תהיכון שיטפל בה. אז מאבדת
ובעיה פרטי בהמה תהליכונים מאותו כאן ואז, היה שלה שנוצרת אותם. בשתהיכון
אחד מהבים OK היה נחסם אפשר לה לא לזוז את פ ההתרים.
למה לא כפאי שפשוט יהיו הרבה תהליכים? כי יכולים להיות תהליכונים שרוצים את
אותו פל ואז לה יעיל יותר. אפשר אם מאאת ההתליכונים לעובשורים אחד לשני.
בשום דרך אז אפשר פפשות כאה תהליכים. אלא שני: לזכור תהליכון חדש
לה פחות יקר מיצירה של תהליך חדש.
מה מיכיון משותף לתהליכונים ואה לא?

- text - משותף - לה אותו הקוד
- stack - נפרד, שהו פ אחד כל מאזם אחר
- data - משותף
- heap - משותף

אם תהליכון הקלה כיכון מראה הוא תהליכון אחר יכול לגשת אליו?
- לתרונית כן כי הליכון משתמש אמנם בתהליכון אחר כדי שיוכה לו סוינטר לליכון מסוים אחרת הוא לא יודע למה לגשת. אל הפיתרון לזה הוא המשתנים הגלובליים.

תמונת הליכון בשיש תהליכונים מתוך תהליך:

התיאור בלה אנית שתהליכונים הם ידורים של מ', ההפ בלה- היא שטרת לפ יפה ומתלמות אותם. אמנ לה לא המימוש היחיד שאפשרי.
אפשר לגשת פימוי של בדבר תהליכונים מרמת המשתמש, ומ' ההפעלה לא יפעת מלה, מהמתם יש רק תכנית את שלה.

Kernel threads	User threads
• ניתן לשמור מידע על כל תהליכון בנפרד, כולל מצב הרגיסטרים, מחסנית ומזהה.	• ניהול ותזמון התהליכונים נעשה ברמת הספרייה ולא ברמת מערכת ההפעלה.
• התהליכונים שלהם - כל קריאת system-call גורמת לחסימת כל התהליך.	• יש תהליכון אחד בתהליך כל עוד הקרנל מתייחס אליו כאל תהליך אחד ולא כמספר תהליכונים.
• אפשר לנצל את המקבילות שבמערכות מרובות מעבדים.	• יש אפשרות לתזמון ייחודי.
• תזמון של מ' ההפעלה מתבצע ברמת התהליכון אחד.	• מתבצע מחסנית כי מ' ההפעלה תשמור רק תהליכון אחד.

ריבוי תהליכים multiprogramming

מה אותנו רוצים להגיע מריבוי תהליכים? - יש כמה שיקולים:

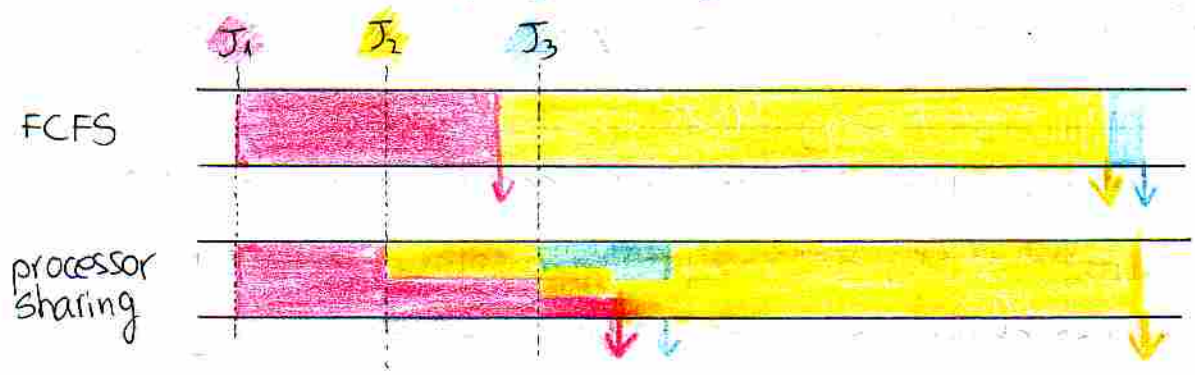
- תגובתיות responsiveness - מבטיח שמ' תגיב מהר ככל האפשר לפעולות משתמש.

ניתן שתכניות מיוחדות, כשתכניות מיוחדות אם ה"מעבד פנוי ויכול לה עד סוף ולאחרונה. אחרת, כשיכול אפשרות ההאו בתור רגיל. לה נקרא FCFS - first come first serve.

כיצד ה"מעבד יכול לתת לכל תהליך את הזמן לרוץ? אנו משתמשים ב processor sharing

כיצד ניתן לשפר את התגובתיות? ניתן לחלק את הזמן לפרוסות זמן (quantum) ולאחר כל פרוסת זמן לעבור לתהליך הבא בתור.

ריבוי תהליכים multiprogramming



יתרונות של ריבוי תהליכים

1. **תאוצת'ת** - אם יש תהליכים בקלי הוא אלו ניתן לחכות עבור אלה שפנו שיסיימו (וזה יכול להיות ארוכים מאוד) אלא הוא אחתון עדיף איך שכן התאוצה מהווה יותר.
2. **ניצול של כלים שונים** - התכנית יכולות להשתמש במשאבים שונים שאין השורים לאלפה (כאן פיסק, רשת וכו'), אל אם תכנית מתחה יכולים אחר תכנית אחרת יהיה עליך את האלפה מלאן הלה מחכה לכלים אחרים שהוא פנוכה. אותו אנקשים למן שאחת היה מעולכס. הניצול מאפשר את התפוקה.
3. **נוחות למשתמשים** - מבחינת לה שאפשר לפתוח כמה מיני תהליכים באותו זמן. כשאל אותנו עושים יותר מאשר אחד בעת למן. יש לנו הרבה תלויות פתוחים לאותנו כלים שכלים יהיו מוכנים לנו כשנסגר אליהם. תוך אלה הם יכולים לעדכן עדכון מוצלח- מהלון אחד לאחפן, מהלון אחר לעדיף הקופץ מלפוף תלון לעדכן מסמך. וכו...

ס הפעלים הטעמים לאלה באים לעמיד

1. **תקורה overhead** - הע' פיכה לעשות יותר עבודה אחה שהיו היתה פיכה לעשות אם היה תהליך אחד. בשביל לעשות את כל ה- switching -context אל ההפעלה פיכה להחשתמש באלפה ואלו מהשאבים אחרים, אל לתכניות נשארם פחות משאבים.
2. **הפרעה בין-תהליכים** - הכוונה היא לכך שאם יש 2 תכניות אל הקלפה הלו כי 2 פחות מהיר, אלכו לפרעות אחרות. לעומת, נשאל פיכולים נכון שהתכנית הראשונה תהיה מהירה יותר מהשנייה? לא בהכרח. אם התכנית הראשונה מבצעת הרבה קריאות מהדיסק, אז היא תחכה הרבה זמן עד שהדיסק יחזיר את המידע, ובזמן הזה התכנית השנייה יכולה לרוץ.

הפתרון האופטימלי:

האם יש הפתרון האופטימלי (מספרי הסדר הנמוך). לא אכן
פיתרון של SJF אבל משפר את זמן התגובה ושאלת אם השאלה
נכונה היה שהיפוכם.

זמן ההמתנה וזמן התגובה מאוזנים.

יתרון בקבצי מסוים וגם סתאום היו מספרים אשהו בזמן הריבוי התלמון
בזמן הריבוי היא פונקציה. לא היה בשלב תלמון. בגלל היה קרוב שהנתונים
עשק, כאות הפיסקים שיהפכו לתתיות מכוסות, בשלב שאשהו נפסק

cache- ה context-switch שיש אחרי cache. איפ אחרי ב- cache
שלפ ביצוע היא ב- cache. איפ אחרי שיש context-switch ה- cache
שלפ גם את מפת בכל הנתונים שהתבצעו קריה. אל תהיקוים אחרי - context
switch טומים פחות מאשר אחרי שה- cache התמלא בהם.

③ סינכרון - הכוונה היא לסינכרוניות של אין, ההפעלה. יותר בקשה שתתקן אין
אסינכרונית, ירצים לקרות יותר מאיים וכו'.

תזמון Scheduling

לה הנתונים שאו אין, ההפעלה אחריה אה לקרוא ואתו מאה.

① האבחה הפשוט ביותר היא שהאלגוריתם FCFS. כלומר בכבר נתונים כל
התהליכים שיש לבצע.

- הטיפול הפשוט ביותר הינו FCFS - פשוט נגיש אותם לפי הסדר.
לה קשה שקומה תמיד, פשוט מאוד, קל מניח שאם הגעה ראשון ממשי-
אף תהליכ לכן ירצ לטעון שקיבל אותו.
- בעיה שאותנו יקרים מה אורך התהליכים. אל אפשר למיין לפי אורך
ולהגיש אבקצ'ארוב - SJF (Shortest Job First). ככה הפקולטות
הארוכות שיין יבתו והתה לאן אבל הפקולטות הקצרות יתבצעו מהר
מאוד ולכן זמן ההמתנה וזמן התגובה מאוזנים.

נראה את ההבדל בין ביצועי on-line של SJF. קל להבין שמדובר כאן בהתאמה אופטימלית.

לאלגוריתם ה- SJF נוסיף תכונה אופטימלית.
אנו נאפשר כאן להחליף תהליכים ברגע שמגיע תהליך חדש שאורך הריצה הנדרש שלו קצר יותר מאשר הזמן שנותר לסיום התהליך הרץ כעת.

אם נחלק את הקצר עם הארוך למן התהליכה המאוחר יהיה ערוך את כך. ניתן שים בפף תהליכים:

- כמה התהליכים נתנים מראש } זמני ריצה נתונים מראש } off-line ו- real-time.
- כמה התהליכים נתנים מראש } אלה רבו הנתונים מאוחר כיצד. כך מתאים מאד סמיתית סופית ל- on-line. וזו יקבע מה יש בה כדי, אולם ויהו לכן יקבע מה למן כך יקח ומטה שיהו לכן יקבע מה בולק שבתות לפף רגע.

כמו שמשתמשים גם ברעיון preemption (הפקעה) - הפסקת תהליך מאולץ ברירה.

② נוריד את ההנחה שאותנו ב- off-line. ניתן שתהליכים אינם נמנים שונים אולם יפוץ רגע למן ירוץ.

על מה שעושים לה SRT (shortest remaining time). נשאיר תהליך תפיש לה שמי אפשרויות:

- הכמן שנותר לתהליך שרץ קצר מכמן התהליך התפיש. במקרה כזה עדיף לסיים את התהליך הרץ.
- הכמן שנותר לתהליך שרץ ארוך מכמן התהליך התפיש. במקרה כזה עדיף להפקיע את המעבד מהתהליך שרץ, לעדכן אותו, להריץ את התפיש ואת להמשיך.

③ צריכה להיות שיטה כמו הריבוי לא יודעים. אין הבדל אי אפשר עשות
צריכה מידע לפני הבאן. אמר! מה אפשר לעשות?
- כמו במקרה הקודם שרצה עשוי לפני סוף הריצה ולזה
- הפקקה של הקצאות שווה(RR) round robin - לא מצליח ישום של
processor sharing שזו מהות ניתן לחישוב (אי אפשר לחלק
לו אפילו יותר מאשר לכל נתון).

מסתבר כאן הנקודה שבאות יש תהליכים קלים ותהליכים ארוכים.
אם נמצא את מיקום על תהליכים ונוכל כולם תוכל אל הם רוצה יהיו
בלאן של האחרון שלא הרבה יותר ארוך מאשר אם היינו מחלקים אותם לפני
אחד אחרי השני. אלו הסיבה שהקצאה הלוח טוב ניתן אפשרות היא
RR הוא אלגוריתם לכן מוכן Oblivious. הוא לא משתמש בשום מידע.
אפשר לשפר את הביצועים אם נושים אשהו מכם יותר.
- תלמון של שיפורים: הקצאות ירוחה שונה ל", ע מיני אפשרים
• שימוש במאפיינים שונים. I/O פעולות CPU
• סיכום לאן ריבה (accounting) - אפשר לפקוח אי הוא קלה / ארוכה
שם. כך. שלף נתן שלף את מי שנראה הכי קלה שם כך. יקרא שתהליכים
מתקדמים פחות או יותר בקצב אחיד כי בריצה שתהליכים לא יותר אל אחרים
ונתנים על שם שיטה (אחד יותר) ואלגוריתם שרצו פחות שם כך.

multi-level feedback queues

הרעבה — Starvation

ורוץ לנצח שהמעבד תפוס וזאת. למשל אם מגיעים אליו תהליכים אחרים ומה מגיעים לפניהם. ורוץ לנצח אבל שלה. לא נורא כי כך - מה בלוה בקבוצה של שלום עד. אם המעבד לא נמצא בנוכחות מספר אחד שהוא יגיעו התהליכים שרצים אחרי. ורוץ מספרו של פעמי. יש אנשים שלא רוצים שיש הרעבה לה לכן נוכל אלגו לה אפילו טוב בקבוצה של שלום יב. למשל, נניח שהשלה פלוה, אחר פרוסס לאיזשהו תהליך ב-CS וכולם הסתדרו של כוס ב-. אבל אז שהאב, תוסה לעשות את כולם ואל אף אחד לא יגיש תלונה או שהוא תמתר כאה קורבנות שהיו לו תיכן לחם לעמוד בפלץ וכשאר יקבלו שרות אפשרי.

בהתפלגות פנים אדם אנחנים שרוב הזמן האב, אבלעת תהליכים אוכלים. אבל אם נשלוח לעבוד תהליך אחד אחרי לפניה האון כמן יהיה תהליכים קצרים.

אישה אחת של תיזמון שאפשר לעקוב בה היא שאחוז מהקצאות.

- הקצאות לפי "חשיבות". האחת האלגו' הזה fair share scheduling - שם אחד אקסף את מה שקורה שיכולנו. כאן fair share כלה equal share. אחר בגלל הקפיטליסטים, לא נוכל לעשות ככה. אז שנרצה יותר יקפץ יותר. יש אב ההפשה קובעת פרמטרים שלפיהם היו את עדיף - למשל תהליך שה סוגי כתוב אקפץ תיזמון יהיה יותר מתהליך שה סוגר מורכב. או תהליכים צריכי עדיף על תהליכים פשוטים.
- תיזמון הוא לעשות epochs (תקופות מסוים). הרעיון הוא שלפני תהליך כאן קבוע ואחרים שהתהליך הזה כל הזמן יכול לרוץ עדיף (תוך יותר ותוך פחות, אחר אין אחד לא "שאר רעב כלל).

שמו שכזה, אותנו מעניין במעלכות הפעלה והאלה הם
פרוייקטים ☺

לפני תחילת הפרויקט עברנו על תהליכים ואנחנו
נעבור היום על נ ראשי השונים שנלמדו אלה.

Concurrency

בשנוי האופציה ניתנו אפשרות הפעולה. נניח שיש לנו רשימה מקושרת

יש צורך לשים new שנוצר להיכנס שניהם ויש פוינטר ל- current.
מה עושים?

```
new.next = cur.next
cur.next = new
```

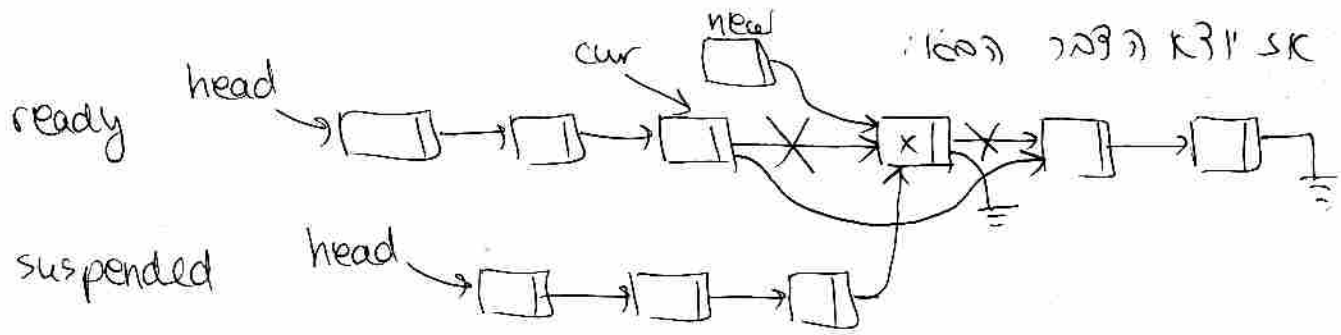
מראה לכן פשוט ונכון וכן שהם עלינו. אפשר מה קורה אם בפועל אחר
הפקודה הראשונה יש context-switch ואחרים תהליכים אחר שהם הזה
כולה ועשה משהו עם הרשימה?

נסמן את משהו כפה יותר מסוכן.

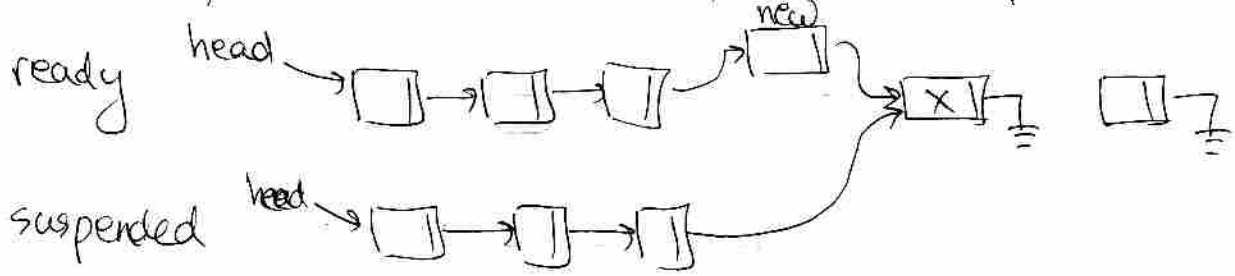
יש שתי רשימות - ready ו- suspended. יש אחר שנוצרים
להיכנס ל- ready (תהליכים שעושה את לה אפסק באמצע. אלא
נזכר הפעלה הזאת:

והתהליכים שעושו אחרים נולדה לה אחד את x ל- suspended והזה
עושה את לה. אלא הזה אחרים אחר x א- ready פעם ב' ו לאחר
הפוינטר שלפני ל אחר שנוצרו אותו לרשימה ה- suspended.

אנו יודעים להסיר:



ודרושה התהליך המשתמש באותו קטע נתון יוצא



מה נראה לנו מה שניתן לעשות!!

את הקטע שבו נמצאים את האיבר שצריך להסיר ונקרא critical section מקווה אטומית שמבצעת את כל הפעולה:

P_1

```
go_in_1 = true
while (go_in_2 == true)
    CS
go_in_1 = false
```

P_2

```
go_in_2 = true
while (go_in_1 == true)
    CS
go_in_2 = false
```

אנחנו צריכים להשתכנע שלא ייגרם לנזק שהקטע הקריטי של שני התהליכים מתבצע בו זמנית.

קריטריונים של פתרונות:

1. נכונות - כך שתהליך אחד יכול להיכנס ל-CS בזמן נתון
2. התקדמות - אם כמה תהליכים רוצים להיכנס ל-CS אז אחד יצליח אחרי זמן סופי (כדאי שלא ייווצר deadlock)
3. הוגנות: אם תהליך מנסה להיכנס ל-CS, אחרים יקבלו מספיק זמן סופי שם בזמנים.
4. כללי: פיתרון ל-n תהליכים.

(14)

האלגוריתם של פטרסון - פתרון הדדי שמבטיח שאין דרישת שתפתור את בעיית ה- deadlock שיש שם (אם שניהם נכנסים לתוך אז אף אחד לא יוצא). הוא נוסף אחרונה אלמנט שני שמאפשר אחרי תור בין התהליכים אפילו דרוש לעשות את זה במידה הבאה כי יכול להיות שמאחרי יהיה רשום שנה הפעם של 1 אחרי 1 אחרי לכן רוצה לפרוץ.

P_1	P_2
go_in_1 = true	go_in_2 = true
turn = 2	turn = 1
while (go_in_2 & turn == 2) ;	while (go_in_1 & turn == 1) ;
CS	CS
go_in_1 = false	go_in_2 = false

לכן מספיק להיות שם אחד שעשה ה- context-switch לכן ירצע ל"דדלוק deadlock.

את האלגוריתם של פטרסון אפשר לבדוק ולהוכיח אחרי לכן שהוא מקיים את הדרישות.

אלגוריתם המאפייה של לאמפורט

לכל תהליך יש מספר סידורי i שמאפשר לשתי תהליכים.

```
1) am_choosing[i] = true
2) for j = 0 ... N
   2.1) if (my_T[i] ≤ my_T[j])
        2.1.1) my_T[i] = my_T[j] + 1
3) am_choosing[i] = false
4) for j = 1 ... N
   4.1) while (am_choosing[j]) ;
```

* מתחילה am_choosing אחרת) ואחרי j-0 ה-ticket אחרת)

```
4.2) while ((my-T[j] > 0)  &
          [(my-T[j] < my-T[i])  ||
           (my-T[j] == my-T[i] & j < i)])
4.2.1)     ;
5) CS
6) my-T[i] = 0
```

זוהי אלגוריתמיה שלא נזדקק בכלל להנחה. אולם אפשר שם לקפח
עלינו הנחה מהנחות. יש פקודה test & set (test & set) כאשר
פעולה על bit אחד שהיא עושה בו שני דברים. היא מחזיקה
את תוכן השדה ושמה בו 1 בדרך אטומית אל לא יהיה אפשרות
שיהיה interrupt בעת הפקודה הנ"א.
אילו היה לנו מבנה קוד של נעילה דומה ל- g עשוי כך:

```
bit g = 0
while (test & set (g));
CS
g = 0
```

שאז יש אי-שוויון מכתיבות ל- $g=0$. ואז אי-שוויון תנוך עושה
ל test והופך אותו ל- 1 ואז כל התהליכים מתמלאה
אלא הוא עושה את ה- CS. ושם מ- $g=0$. וכך כמה
זה מעשיק. אז זה פתרון מאוד נפוץ ואתה אולי יש לו שמו
מלא. הבעיה היא שהדוגמה לא מראה אופיה זה ושניה
שהם משומשת לפני הלגוריתמיה שעינו ודש busy waiting. זה
נתון מאוד לא נעים אם יש אלפי אלף, כי אנחנו מבזבזים משאבים על
לא עשות כלום - על לחזור ולקרוא בלולאה. הוא עניין שנוגע לזה איך
עוד ההפעלה פסוק לעשות עתניק הבא.

סמפור הוא משתנה שלם אשר ניתן לאתחל אותו אך ורק באמצעות שתי פעולות אטומיות: P ו-V.

- P - בדיקת ערכו של המשתנה.
- V - שחרור.

מימוש אפשרי הוא למשל:

S = 1	אתחול
if (S==) wait	P
S++	V

וכאן אפשר לעשות בקטע קריטי כזה:

```
sem: mutex
P(mutex)
[CS]
V(mutex)
```

הבעיה היא שהמימוש של הסמפור לעיל יהיה בקטע קוד קריטי, ולכן נסתפק כאן בלהגיד את הדברים הבאים: סמפור זו מה שקראנו בשיעורים הקודמים בשם busy-waiting.

פתרון משולב

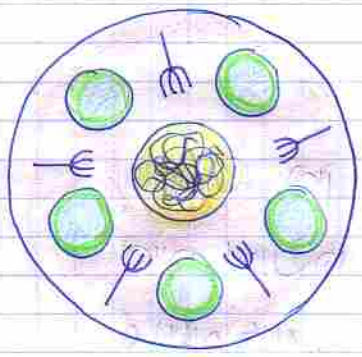
סמפור מאפשר לנו לפתור בקלות בעיות סנכרון. אך קיים מצבים בהם בעיות שאותן הוא פותר. לעומת זאת יש לנו בעיות אחרות שלא ניתן לפתור בעזרת סמפורים. למשל RQ_S, CashS, ובמקום שנגדיר ב-ready קודם יושם P(S_RQ) כך שהמעלה וכל משתנים ו-V(RQ_S). אחרי הפעלה לתור ה-suspended ומפה הפעלה. סמפורים P-ו V קוראים lock ו-unlock.

יש שני סמפורים שימושיים. הראשון - busy wait צריכים לבדוק
מתי כדאי להשתמש בכל אחד. שני - השתמשו במנעולים בזק
לא להיצמד לאחד מהם יותר מדי. בדיק לפי מה שיתר מתאים.
וכה מביא להבחנה של סוגי מנעולים שונים. למשל סמפור עד מנה
מתנים כך לא מה ואם שר מתת יראה התלוים סמפור בינ.
הדגיה הוא אם רוצים לשנות מנה מתנים. אז אחד של כתבה
דוק להיות אקסקלוסיבי. מוצץ שיהיו משנה את מנה המתנים
אסור שאחרים תהיה גישה אליו. (כמה שאנחנו מדברים בתרגול)

31.5.07
OS -

אותו שלב נוסף ב- concurrency. היו נעסוק בבעיות (deadlocks)
שנוצרות כתוצאה מהסנכרון בין התהליכים ונלמד כיצד לפתור אותן.

פעמים רבות מישהו פילוסופים מוצלחים שכח אוכל שהם עומדים לאכול. הם חייבים לחשוב ולאכול. הם חייבים מזלגות וחומרים סמוכים. הם יושבים סביב שולחן עגול שמאכלים לידם סמוכים. לכל פילוסוף יש צלחת ומין הצלחות יש מזלגות.



אומר להם ובימו את המלצר שמגיעים מאחו בצמן יודע לתת לפילוסוף אין שני מזלגות ואף אחד לא מוכן את המלצר ומה רק מחכים אחד עד השני ואחרי הם יוכלו מרעב.

פתרונות אפשריים.

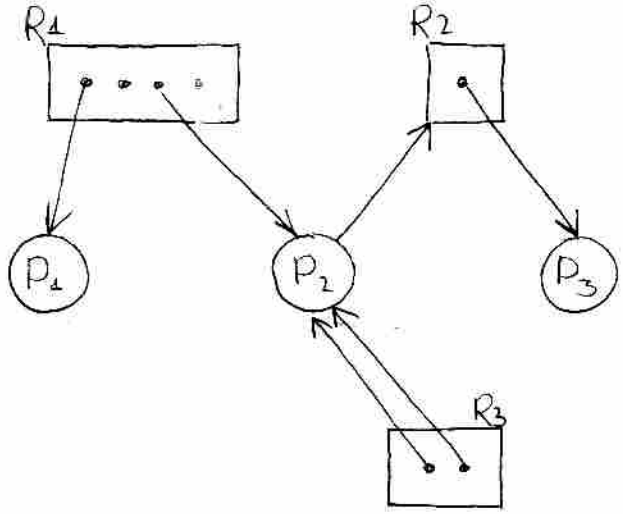
- ע פילוסוף ירים את המלצר הימני. אך הוא עסה לחזור את השמאלי. אפי הוא עכ שם הוא ינית את הימני ואחרי כמן תקועים עסה שוב. עשים ♡ שהתלך של בצמן הקפוצלי הוכו מאוד מעשה. אפשר לא היו אוכלים עד שמעסים שוב הם היו נתקעים מעוראה אינסופית של עקיבת מהנית. לזה נקרא livelock
- אפשר לחשוב כך ראכלה פילוסופים לחכות אחד לאוכל
- ע פילוסוף ינית לצמוף בלחה שהוא. יהיו ראלה שיוקחים קפה את המלצר השמאלי, ויהיו ראלה שיוקחים קפה את המלצר הימני. אם מישהו לא מצליח לקחת את המלצר הפשוט את הוא עכ עוקה את המלצר השני. נבה מעוט מסוף יהיה לפחות פילוסוף אחד שיצליח לקחת שני מזלגות.

כעת שאר הפתרונות בעלות הפתרונות יפה כן. יש תהליכים ויש משאבים.
נניח שאר הם לוקחים מלצר אחד של הקבלה משאבים.

- מסכן מיידית משאב מיידית אופן של משאב
- נקבע סדר מסוים מיידית אופן של משאב
- תכף מתהליכ למשאב שהתהליכ מחכה ימשאב

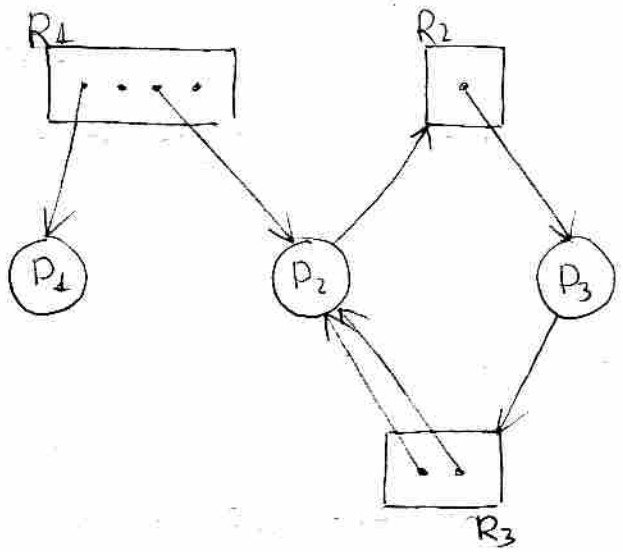
- תת א קיים מצב שבו תהליכים אחד שמחזיק מקבלה שהליכים.

למשל,



האם יש כאן תהליך? כן! P_2 אחד שניתן אחרות אם P_3 סיים משימתו ואז P_2 יוכל להמשיך ולכן היה טוב ויפה.

לעומת זאת, מצלם הבא יש תהליך:



תנאים לקיום תהליך:

- הקצאת המשאבים היא בלעדית (2 תהליכים לא יכולים לגשת את אותו משאב בו זמנית)
- אין הפקעה של משאבים
- Hold-and-Wait (תהליכים מחזיקים משאב ומבקשים למשאב אחר מתוך כך אחרים לא יכולים לגשת אליו)

* מצלם מתוך ההקראה, אנחנו נדבר בהמשך איך לזהות תהליך. או תאיך נשים לב שיש תהליך. קודמה בהמשך הדבר.

אין מעגל כי R_1 ו-R_3 יכולים לשמש את P_2 ואילו P_1 ו-P_3 משתמשים בשניהם.

שיטות להתמודדות עם הבעיה:

- <u>מניעה</u> (prevention)
  - נסתכל על תנאי 3. אולי אין סיכוי שיתקיים אלא במידה. האם ניתן לשנות?
    * אפשר לשחרר את כל המשאבים שמחזיקים ומבקשים. אבל זה עלול להיות יקר ולא מועיל שכן אז אפשר יהיה לקחת אותם ולהקצות אותם לתהליך אחר ולא ברור שזה יהיה לטובת.
    * הפתרון הכי ברור (ואולי זה עובד במקרים מסוימים) שאפשר להשתמש בפוסק בשביל שיהיה ניתן לכבות את התהליך ואת כל מי שמחזיק במשאבים שלו (אחרי שהוא מנקה את הבלגן שלו).
  - נסתכל על תנאי 4.
    * שיטות סדר. נותנים לתהליכים ולמשאבים מספרים ומאלצים אותה על סדר עולה. כלומר תהליכים מחזיקים משאבים שיש להם מספר נמוך יותר ומבקשים משאבים עם מספר גבוה יותר. אז לא יכול להיות מעגל.
- <u>התחמקות</u> (avoidance)
  - מידע נוסף - הוא מידע "עתידי" - כל המשאבים איזה מתוקצבים
  - אפשר לבדוק בקלות רבות אם נמצאים את המצב במידה ומידע

כאמור נשארה השאלה איך אלגוריתם מוצא מסלול.

אלגוריתם הדייקסטרה (Dijkstra)

- כל תהליך P יש בהכרה שלו פרישה מינימלית $\vec{T_P}$ שאומרת כמה פחמים הוא יוכל לקבל בהשוואה לשאר התהליכים.
- לכל תהליך יש וקטור נתיב $\vec{C_P}$
- המצב שיש לתהליך מוגדר ע"י $\vec{A}$

מהגדרת מצב מסוג שמתואר ע"י $\{\vec{T_P}, \vec{C_P}\}$, $\vec{A}$ תהליך יכול לבצע וקטור $\vec{R}$. ישנם 2 אופציות:

- לקבל את כל השלכים שהוקצו לו.
- לשלח את התהליכים עד שיפסיק לקבל מהם (אין להשלכות).

אין לנו שום אם הוא יהיה מסוגל. נניח שיש לנו $\vec{R}$ התהליכים יכולים לבצע. אנו מחפשים את המינימום שיש לנו תהליכים שמו כל תהליך יבצע סדרת - שהרי לה תעבוד ביחד כדי שאותו תהליך יוכל לבצע בשלמות.

א - נניח שנתון לנו את הוקטור. אזי $\vec{C_P} = \vec{R} + \vec{C_P}$

$\vec{A} = \vec{A} - \vec{R}$

תזכורת: ראשונים אלו הנתונים הראשוניים של $\vec{C_P}$ ו- $\vec{A}$

אלגוריתם מסוגלים

- אם הוקטור הוא חוקי, כלומר $\vec{T_P} < \vec{R} + \vec{C_P}$ (כלומר שפחות מכמות האלרגיות אחת ש אי שווין) אז הוקטור הוא חוקי.
- אחרת.
- אם $\vec{A} - \vec{R} > 0$ (כלומר אין מספיק אנשים בתחנה) אזי אי אפשרי שהוא את התהליכים ברשותו ולכן יתרסם אותו.

אין שאלה, בהשוואה לשלושת הסעיפים הקודמים לסעיף זה שאלה כבדה.

ב - נמצא סדר שבו התהליכים יוכלו להתחלף. נניח P קבוצת התהליכים.

```
while ( P ≠ ∅ ) {
    found = False
    foreach p∈P {
        if ( C_p ≤ A ) {
```

אחרת- קח כל והחזר את כל המשאבים // $\vec{A} = \vec{A} + \vec{C}_p$

$P = P - \{p\}$

found = True

}

}

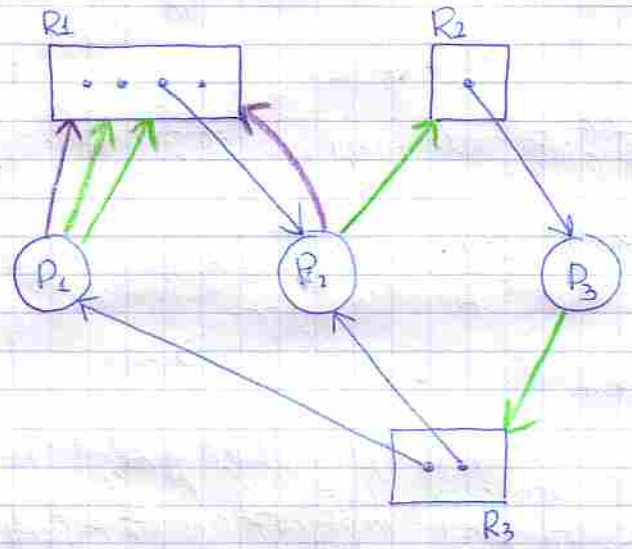
if (found == False)

return FAIL

}

return OK

תאר דוגמה שלנו.



$C_p \rightarrow$

$M_p \rightarrow$

$R \rightarrow$

$A = (3,0,0)$

ליד שם תהליך הכוונה היא את כל המידע הנמצא אצל התהליך אבל אנחנו לא נדרשים לזה במשימה זו ולכן נתעלם מהנתונים על התהליך

נניח בשלב מסוים שתהליך P_1 מבקש את $\vec{R}$ את הוא יוצא מהשורה אם מקבל לו P_2 את P_3 ואת

איך התשובה של סדר התהליכים ב- foreach ? שם ניתן מבקשים את כל המשאבים ניתן אלגוריתם סיבוכיות $O(n^2)$ ומה לגבי הוא עובד. נניח שלא ידוע מהיות שמניים סיפור אפשרי ונתון נכון שלא קיים כלה!

תקשורת

מה זה תקשורת ולמה בכלל את זה?

תקשורת זה כשתהליכים מדברים אחד עם השני. בדרך הפשוטה שבעולם זה היה שולח מכתבים - כשיש כאן מחשבים אנחנו תהליכים עובדים על אותו המחשב. מאוחר זה משתמשים בדרך מסובכת יותר: אנחנו משתמשים בתקשורת בתוך מה שקשור ל- web - תקשורת בין תהליכים במחשב אחד שמתקיים מחשב אחר. אחד מהם הוא השרת (server) והשני הוא לקוח (client). יש הרבה תקשורת בין תהליכים מאותו מחשב.

תקשורת בין תהליכים זה בלבול רציני אבל מאוד הפשוט. היו כל הקטע הזה שמפריד תהליכים אחד מהשני. המטרה שלנו היא לגרום לתהליכים לתת שרות אבל שיש תהליכים אחרים - 'המתחרים' שמנסים לגרום בדיוק לדברים שהם לא. אפשר אפילו תקשורת אבל לא כזו שהוא בדיוק לדברים שהם רוצים, אלא הוא לא בדיוק לדברים אחרים!
איך תהליך יודע שיש תהליך אחר? איך הוא יודע איתו קשר?
איך הוא יודע לזהות אם הוא רוצה לדבר?

יש שתי דרכים עיקריות:

- לקבל את הפיד וזה ניתן במקרים מאוד מוגבלים. למשל כשתהליך עושה fork הוא ירצה לקבל פרטים על התהליך שנוצר. כשעושים ()fork = pid אז האבא מקבל את ה- pid של התהליך הבן. והבן יכול להשתמש ב- ()getppid כדי לקבל את ה pid של אבא שלו.
זה דבר לקבל את כל זה (במדויק הרבה יותר בהרצאה) הוא פשוט לא כזה משתמשים בזה הרבה. שימוש זה רק תקשורת שיש לו פתח כניסה ופתח יציאה. אם יש תהליך שיש לו שפופרת הוא עושה fork אז אותם תהליכים יכול לכתוב לשפופרת והשני יכול לקרוא

מאחר. הדברים שהצליחו יהיו מהות רק ליתקשרות בין בני אנושות.
ה"ני רוצים ליתקשר עם מישהו שלא פגשנו עדיין. אם רק תהיה יש
שם שונה ויש איפשהו אינטרנט - שמאפשר לנו לתת
לו שם ולקבל את כתובת המדויקת שלו. כמובן, מתוך נקודת ההתחלה
אחנו צריכים לדעת איך לגשת לשרת השמות כי אחרת הפלוגה הזו מתגלית.
יש כאן עוד בעיה - צריך שמות שונים. איך אחנו פותרים לזה?
קודם כל יש להסתייע אומרי שלא כל תהליך חייב להיות שם, אלא רק
לתהליכים שרוצים להיות נוכח מתקשרת.
מאחרה לה שות השמות היחודי (דוגמה) זה תהליכים הלקוחות.
יש כאן מאפיינים לאחרות שתי לקות:
- השרת קיים ויודע למן
- השרת לא יודע מאשר על לקוחות (המבקשים הלקוחים)
שרת שמות צריכים לתת שני שירותים:
- לקרוא (באר פונקציה שלוקחת אותה ומוסרת מלן פרטים)
- שאילתא לגבי שם נשם
אנו רשמנו לעיל קשר קודם בין אותה מהתהליכים צריך לקרוא ואנו
תהליך אחר יכול לבקש מאשת השמות כתובת של מה שנרשם.
שזה יש כאן בעיה. ניתן שיש שני תהליכים - לה שנרשם וזה שמתעניין
אותו. אנו לה שנרשם לא אמור להיות מעוניין כי כבר קיים שם כלה,
הוא צריך להתעלם את הנשם ומנסות לקרוא שזה. אבל אנו צריכים שמתעניין
אותו יכול לדעת את הנשם הנכון. אנו צריך שתהיה מינהם א"כי צריך
תשובה אחרת ישירה שתאפשר להם לקבוע את השמות. למשל,
לה שנרשם יכול לתשום את הנשם על המסך ורק אנו ה"ה שמתעניין
אותו יכול להקיש איזה שיקרא את הנשם מהמסך.
עוד אופציה היא להשתמש בשמות יחודים. למשל שם מחשב לה שרת
ה- web. אם יש מחשב שנותן לקוח שרת web אנו רשום עליו
אינשהו תהליך יכול לקבוע שהוא רוצה לקבל OS ואנו ע מה שצריך
ע-OS יכול אלו. אם מישהו אחר יכול שהוא OS אנו ע ה מקשור

שאילתה ל- http מחזירה לתהליך הדפדפן.

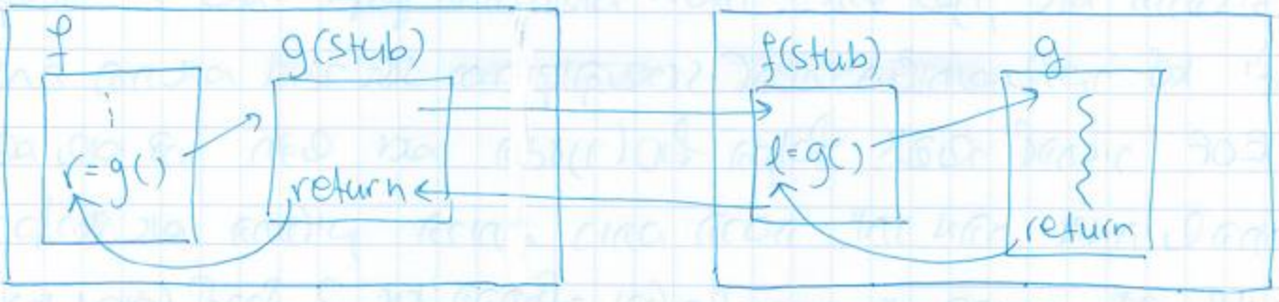
כמה דוגמאות של שרתי יודעים מוכרים:

- http - 80
- ftp - 21
- telnet - 23
- e-mail - 25

מנגנוני תקשורת -

1) RPC - remote procedure call - לאחר קריאה לרוטינה שנמצאת במחשב אחר, תהליך אחד מבקש את הקריאה אולם היא מתבצעת בהקשר של תהליך אחר.

הנה שני מחשבים/תהליכים:



f קוראת ל- g. מה קורה? stub שולח בקשה עם stub מתאים אחר ומיוחד קורא ל- g שם. היא בעצם, מתרגמת את הקריאה ל- f(stub). היא מתקשרת את מה שהיה ל- g(stub) והיא מכינה את הקריאה ל- g - r. התקשורת בין ה- stub-ים היא שקופה למתכנתים. הם לא מבינים מתשם על כך.

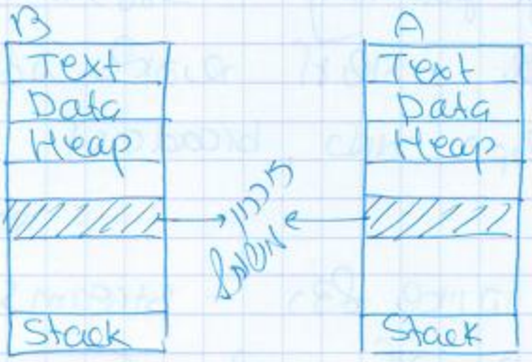
2) שיטת ההעברת - message passing - הקהילה היא שיש שתי פעולות שאפשר לעשות - send ו- receive וזה כולל אפשרות של broadcast, כלומר תקשורת של אחד לרבים אבל לא אחד לאחד.

3) streams - רצף שהוא תוכנה פשוטה לתקשורת. הקריאה והכתיבה לא מבדילים מהבדלים האלה. קוראים לזה pipes ו- sockets.

pipe משמש כמו – צינור שמחבר בין שני תהליכים.

כל פעם קצת שונה מהפיפ אלו יכול שמשמש כאחסון
לפרטים זמניים. מנקודת מבט התהליך הכותב הוא פשוט מקבל file
descriptor ומשתמש בקובץ ולא אכפת לו מה קורה מאחורי
הקלעים. כמובן שאין הבטחה יודעת – שכן לא מדובר בקובץ רגיל אלא
pipe וזה מאפשר פחות אילוצים. למשל, אין מיקום בקובץ רצף
ומנסים לקרוא מקובץ שהגענו לסוף שלו מחזיר EOF. אבל שני,
אם מדובר ב- pipe ומנסים לכתוב אחד והשני לא קורא כלום
שזה שניה לפני שהשני משתנן. אז מוסיפים את התהליך
הקורא לתוך ה- pipe ויהיה חשוב אחרון. אבל שני, אם התהליך
הכותב מתאים סוגר את ה- pipe אז אפשר לצפות שהקורא יבין
בשלב שהוא קורא ואם הצינור ריק והקובץ אפשר להחזיר EOF.
אבל קורה אם התהליך הכותב רוצה לכתוב יותר מהר שהתהליך
הקורא מספיק לקרוא? אז, ההפעלה יכולה לחסום את הכותב עד שמתפנה
מקום ב- pipe. אם הקורא סוגר את ה- pipe אז הכתיבה
של הקורא תיכשל כי זה כל הפואנטה מלה שהוא כותב היה שלה
לעבור לקורא!

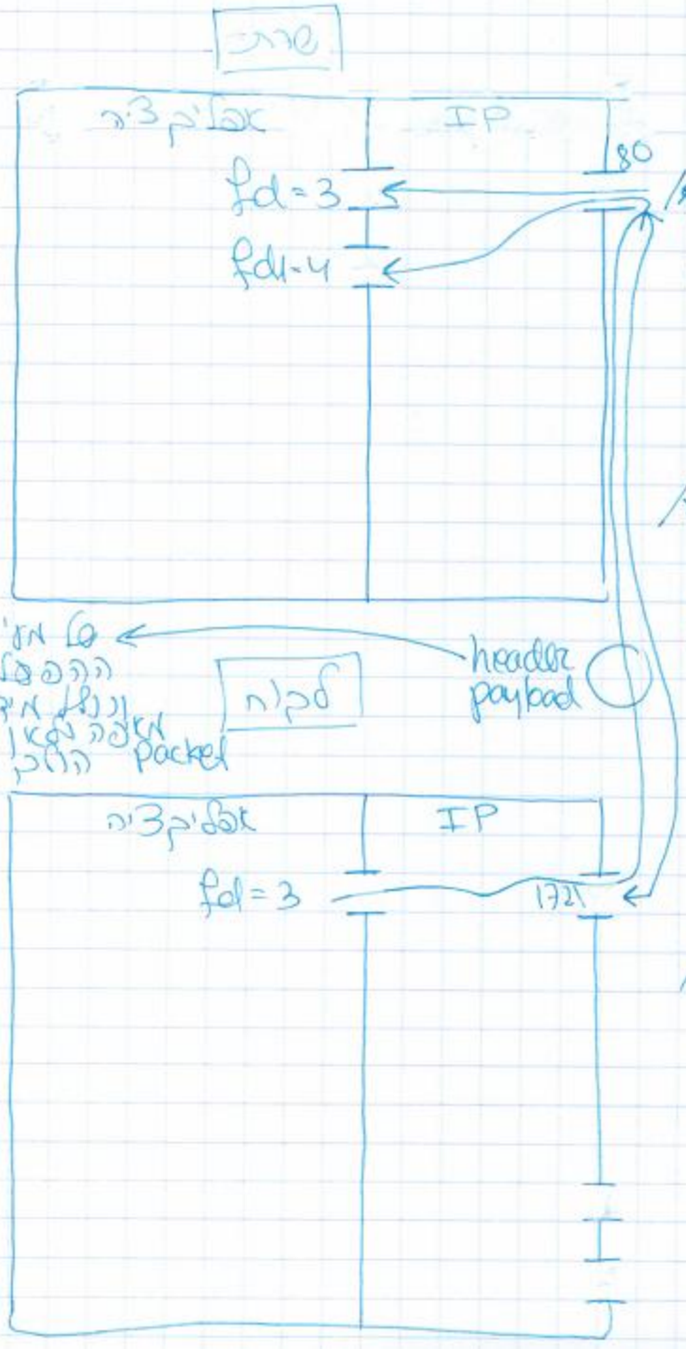
4) צינור אשכול – מהי הבעיה של תהליך מורכב אחרי אלוהים.
אפשר לדמיין מיליוני תהליכים של צינור אשכול
מאזור הזה קורים בדיוק אותם
דברים בשני תהליכים. כמו
שניתן ש- A עושה אותם דברים
באזור של B. זה נשמע קצת מוזר
כי למה לא נדבקו על ניהול צינור
אחד למשתמש צינור בודד אחד? אך הצינור הוא: אין שום
מילה שמשני תהליכים יהיה איפשהו לשותף מקום. מאחר כזה כמובן



(21) צריך לתת את הדעת לנושאים של סנכרון. ואז צריך להשתמש בכל מה שלמדנו על סמפורים וכו'. על מהות המימוש של אותה הכתובת נדבר קצת שלישי בלילה (תחת ההנחה שלא ירגו שם אותו).

socket הוא הכלה של שקע. שקע אפשר להקים רק מן, משפחה מושג שאותם אין רגע שם. התפוקציונליות הנאה מכוסה ע"י sockets שקע הם. שקע-תם ע"י נעלמו. socket זה שקע - אין אחרי כלה שאפשר להתחבר אליו. נדבר עליהם כהקשר של אי שרת-לקוח.

התימור באמצעות socket הכוונה לא סימטרי. השרת קיים אחרים והלקוח מתחבר לשרת.



אם ההפעלה נותנת ללקוח port אוטומטי, זה בדרך כלל port גבוה.

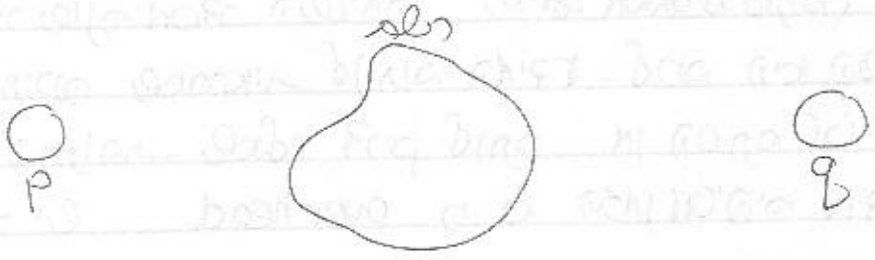
שלו שיר
אומת התקשורת מגדירה את מי יהיה עליה לענות את האפליקציה
שנכתב היא עושה (listen) ואחרת שהוא ותקשר עליו.
מרגע שהיא עושה accept היא מחזיר filedescriptor
חדש שעדכן קשור של הלקוח ל- fd חדש. זה fd "חלבי"
שמייצג את הלקוח הזה. זה טוב כי זה מאפשר לשרת כמה לקוחות
מאותו זמן. אחרי שיש לנו fd חדש ל לקוח
הראשון מיצגי תהליכון חדש שיטפל בלקוח ומנהו זה שיקרא
את מה שלקוח מחכה ויטפל בו, מיותים התהליכון הישן
של השרת חוזר ל- listen ומחכה לבקשות חדשות.

14.6.07
OS

(שיעור עם קלבו)

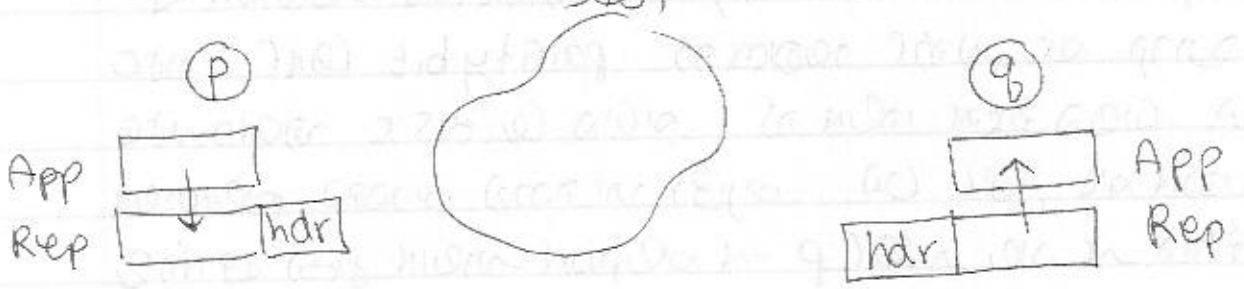
Networking

יש רשת - כרגע לא חשוב פנימה שתוכה אחוזתנו. ויש שני תהליכים - שרת מקבות וזה רוצים ליצור תקשורת ביניהם.



הבעיה היחיד בין p ל-q היא הרשת. מה לדעת שהמידע איבד בדרך אחד את השני. משמעות לזה יש שכבה של תוכנה representation ויש אחראי על ייצוג המידע. היא מוסיפה לא-מין איזה שהוא header שאומר איך המידע מקופל.

כשנשלח את ההודעה על פני הרשת, המידע ועוד עם ה - header עובר לצד השני ואותו תוכנה מורידה את ה-header שמוסיף על מי שמקבל. לזה נקרא כימוס (encapsulation). אז אחרי שה application layer מעביר את המידע מצד המקבל היא מעבירה אותו לאפליקציה של הצד.



בדרך יש הפעלה על אופן המידע שאפשר לשנות. את לוקחים את ה packet יחד עם ה - header ואולי מוסיפים אותו ל- packet -ים. עכ packet

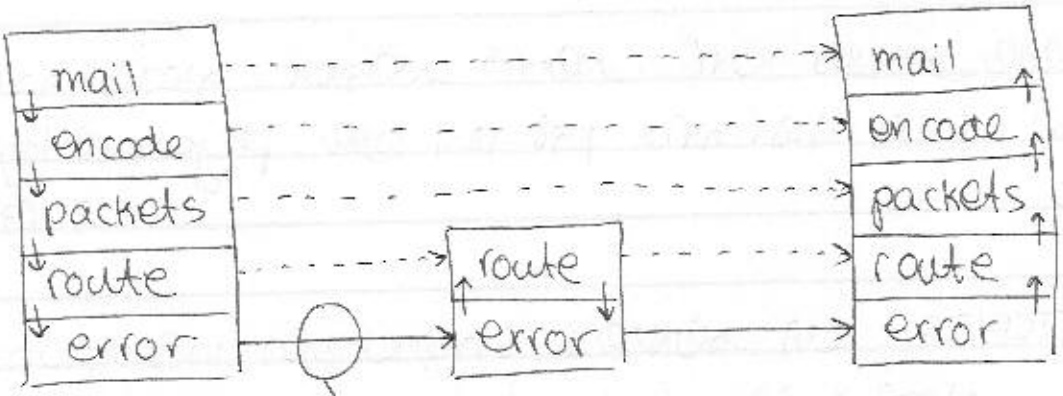
יש header אשר שאפשר את לתכנן אותם בדרך (למשל, יש שם מספר סידורי). בדרך השני מקפלים את כל ה packet-יחד ומוסיפים ה- header-ים שמגדירים אותם יחד לדרך אחת.

כל packet אחרי במשפט מדויק שלמעשה ממש יכול יכול לעבור מספר שלם מספרים שונים (במקום מספרים שונים). אבל יש שכבה של תוכנה שמתאימה לעובדה הזו כדי. לרוב היא נמצא במקום אחר שהיא מתלבטת שבאמת לעבור בין הסתם כל שכבה שהוא עובר יש over head כי כל פעמן מוסיפים מידע.

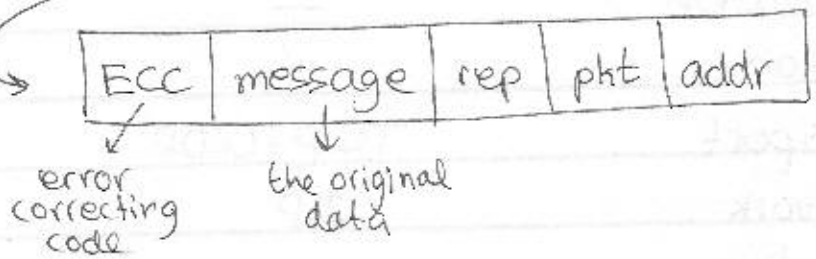
את המידע שולחים בדרך אלגוריתמים. כדי המידע חשוף לשכבות אלגוריתמיות שיבדוק ושינוי אחרי את המטרה.

⇐ משתמשים באלגוריתמים שאנחנו אם מצל קודם טעות. נאמן bit parity. לה מוסיפים ליהות אם קודם טעות מספר איזוי של ביטים. לה משהו מאוד פשוט. בפועל משתמשים בדברים יותר רציניים. מה נעשן, אם אנחנו שהמידע נכון משתמשים באלגוריתם מתקשים א- ק ישנות שנה את ההודעה.

בין השכבות של התהליכים יש קשר ויש השכבה שלמן מאושרת בצורה פרוטוקולים.



כל נתב מבצע שהמידע עובר דרכו קיבל את המידע של שכבה נמוכה ושכבה הגבוהה. בכל node עושים בדיקה של טעויות בקונאפ או בתחנת היעד של ה- packet



מה היתרון של התוכנה הבאה? כל שכבה עובדת מבוסס עצמאי. לא מתי של הפרד ומשול. למרות סטנדרטיזציה של הפרוטוקולים ואפשר שהמידע יגיע יותר.

ISO-OSI Layered Model

סטנדרט של ISO. הוא יותר מקפיד מאשר פרוטוקול. יש 7 שכבות:

- application - מתבסס ישירות על המשתמש
- presentation - השכבה שמתאימה לקידוד המידע
- session - מאפשר פרוטוקולי תקשורת בין תהליכים (כרום לא משימוש)
- transport - תיקון המידע ל packet - יפה ושימושה בלבד
- network - הנתוב של ההודעה ברשת.
- link - מאפשר לפקח טעויות ב- link, כמו ב hop אחד.
- physical - התווך - ethernet, wireless וכו'.

התוכנה רשתות אפשרית לחשוב. למשל מאחלת שיש לה physical layer שהם, הן יודעין לשלוח מידע לפי הגדרה הלה.

הפרוטוקול שבו רשת האינטרנט משתמשת הוא TCP/IP. כל מה שמשתמשים מן המפורט ולא מאוד שבע השכבות.

הנה ההשוואה:

ISO	TCP/IP
application	HTTP, DNS, Telnet, SMTP, FTP
presentation	—
session	—
transport	TCP-ו-UDP
network	IP
data link	—
physical	—

Flow Control

כי שמקבל מידע רוצה שיהיה לו buffer שיאחסן את המידע. אם ה buffer מלא ומגיעים עוד packet-ים אז הם (לפחות) יש איבוד מידע. לזה נקרא overflow.

אם הנתונים שרוצים לשמור מידע ושאר לעיבוד הקצה מלא יש congestion. משמעו שאין לה קיבולת לשלוח עוד הלאה.

משמרו ה control flow משתמשים משני טכניקות מסוימות - acknowledgements ו- timeout. לדוגמה שולח ה-q מקבל את ההודעה הוא שולח ל-q ack שאומר לו שהכל

קיים אך ההודעה אופשר למשוך עוד זמן. אפשר גם ה- ack גם ההודעה כלשהי שולח את ההודעה ויכול אומר איזה למען timeout שאם היה צריך לחכות עוד ack אז היה שולח את ההודעה שוב. ה timeout בדרך מוגדר תמיד כי תלוי רוצים לו יותר לחכות עוד. ניתן לשתמש ב- TCP כדי לוודא שההודעה מגיעה ובכל משתמשים במנגנונים האלה. UDP לא מבטיח שההודעות יגיעו אבל לא צריך לשלוח את המשפטים נשלחו. במילים אחרות, ב TCP ניתן להגיד את שמורת על סדר ההודעות כי שולחים את ההודעה ממשיכה רק אחרי שהקודמת הגיעה. אפשר שני השיטות האלה אחת את לא מנצלים את הרשת מאחר מהר. הפתרון היא pipelining. שולחים חלק של packets יחד שקיבלנו על ack. ומשתמשים את מדיניות sliding window. שמתחילים חלון שיכיל עומד איזו על כל ה- packet-ים. ההתקדם שלו תלוי את ה- ack הכי מתקבל שהוא ברגע. ומהות אם מגיעים הלא השלוח אלינו את התלוי קבועה.

הלקוח המקבל צריך לעקוב שתי דברים:

- שהוא קיבל את ההודעה
- שיש לו מקום ב- buffer.

את הדרך הלא טובה אפשר לעשות שני ה- packet-ים שנשלחים מלאו הכי. לה פשוט מתוסף header ולא מוריד קלה את התקלה.

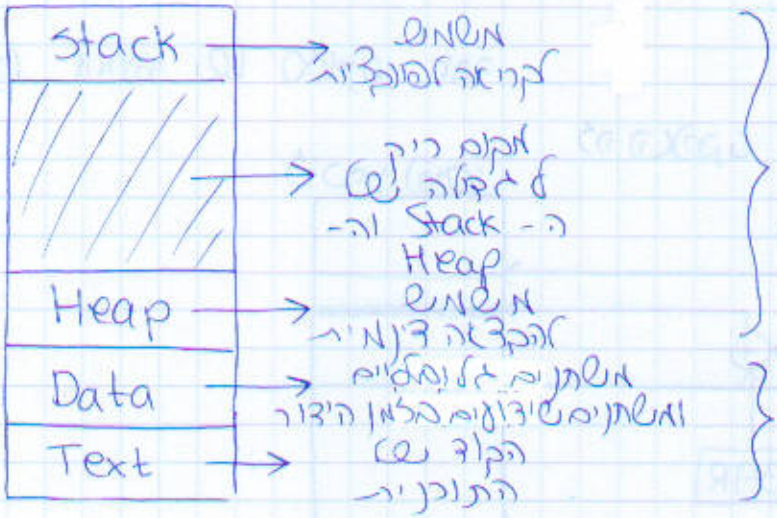
לה נקרא piggybacking

(25) 21.6.07
OS

ניהול זיכרון

זיכרון הוא אחד המשאבים הכי חשובים של המחשב ויש שם הרבה מקום לשיפורים. תלוי איפה נעשה התוכנה אנו, הדפסה רק מעלה את הצרכים.

❤ הצרכים הכי מיוחדים. אתה הרבותות של תהליך כמו שהמחברת / מחברת רואה אותו הכי רחוק ויפהפה (מסיבוכים אוקטלים שאתה הרבותות מתחיל כ-0 וגולך עד 3GB. תוארה של 32bit יכולה לתאוך מראתה רבותות של 4GB, אפל אנ' הדפסה רוצה לשמור לעצמה חלק אחרון). כל תהליך שלך במערכת חושבת שיש לו 3GB וזה בעצם לא קשור לכיבוי הפיסי שמאחורי שלו. אל את אתה הרבותות הלוגי של תהליך ממפים לאתה הרבותות הפיסי.
מתה הרבותות הלוגי מתוך לסטאנדרט

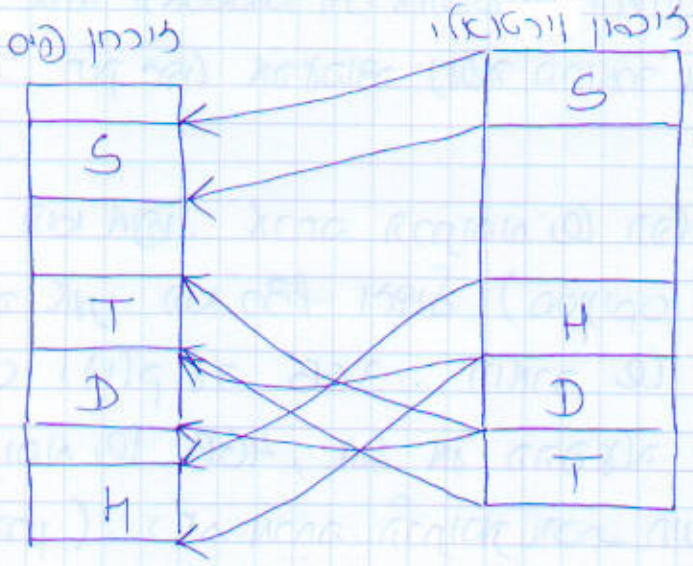


אלה שלשה בלוגי כי אתנו לא יודעים מאיפה זה בדיוק יהיה אבל אנחנו יודעים שהם אל אחרי מקום לשהו. אם המקום הזה נגמר אנחנו מבקשים

אלה לא משתנים מיוחדים כי לה יזכו בלמן הקומפילציה

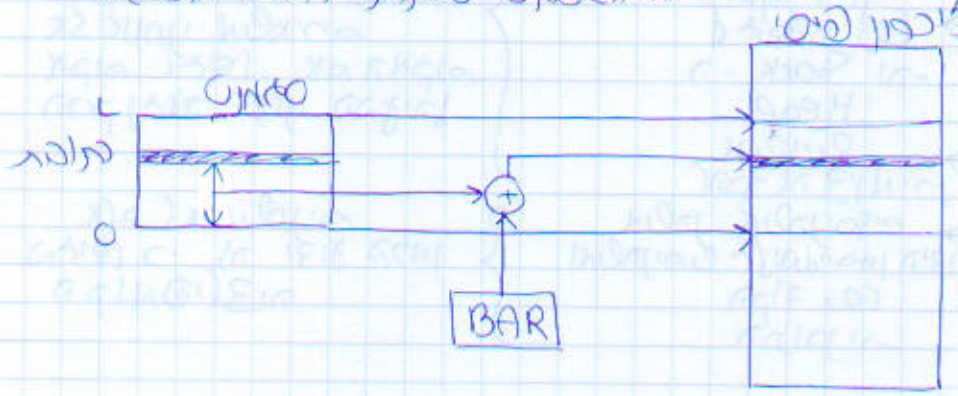
לתכנית לה מאוד חשוב שהזיכרון יהיה רציף. בשפות הכי אולמטיות שלה היו חשבון פוינטרים כ- C. למשל a[i] = a[i-13] + 57 היו מצפה [(int)sizeof * i + a] * לה חשבון סבבי של רבותות לינארי אם אתה לא יודע שהזיכרון רציף.

מכופל לא מאפשר את הגישה לכתובת פיסי רציף, אלא מאפשר כל סגמנט לאזור רציף, אבל מין הסגמנטים יכולים להיות ריווחים ביניהם בתפקיד של אין ההפלה.



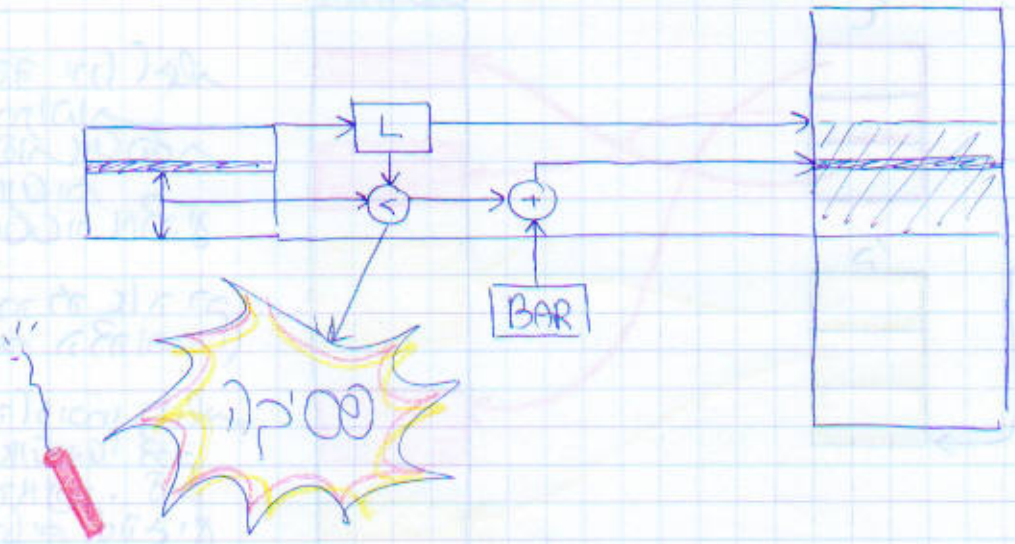
יש כאן בעיה מאוד קשה של Stack או ה- Heap גודלים באופן דינמי. כדי לעזור לזה בפועל כל מיפוי סגמנטים מוגדרים ומהמשתמש ניתן את הדגלים עם הסגמנטים הנכונים.

נתון מיפוי של סגמנט אחד.
זה האפקט שאנחנו רוצים להשיג:



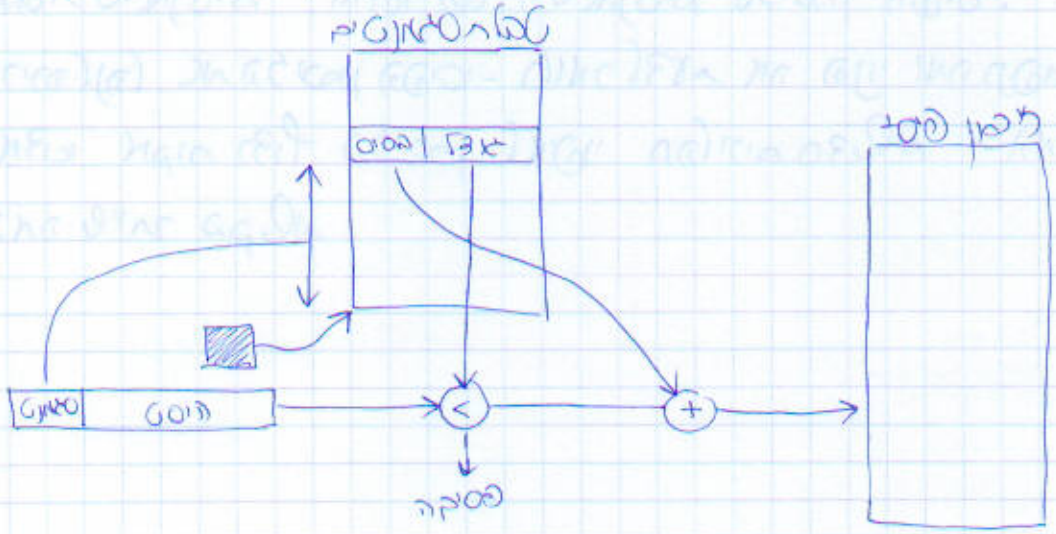
הקאפ"ר מייצר תחילת פיסית - המקום בזיכרון לתחילת הסגמנט ואת התחילת הפיסית הזאת אנחנו צריכים לאפות לתחילת פיסית. בשביל זה יש רגיסטר מיוחד בשם BAR (Base Address Register), שהתפקיד שלו הוא לאפשר את ההפלה לאפות תחילות. הוא פשוט שומר את ההיסט וכל פעם שפונים לגישה בכיבוי מחברים לזה את ההיסט וזה מקבל אותנו למקום הנכון.

אולם, לפעמים אנשים (או אנחנו תוכנה) כותבים תוכנית לא מאוימת שיש מהן פגיעת מכוונת שחורגת את סמנטיקת. את מטמון כזה יש ערך רגיסטר נוסף L (Limit) שמורה מקיבו את גודל הסגמנט. אם הכתובת מתחום האותר הוא בסדר, ואחרת יש פסיקה (segmentation violation).



מה לגבי שלמות אוטומטית ל' התוכנה כבר יעשה מכונה! אך ההפעלה לא מתרחשת ישירות מהשורה מכונה! באחריות של מ' ההפעלה היא שמטען לתוך מרחב הרגיסטרים הללו את הערכים הנכונים עבור התהליך שעומד לרוץ. ואז התהליך יכול להתחיל לרוץ מאופן ישיר את התוכנה (שם אין פסיקה כאמור).

אבל מהליך אין הטתם יש כמה סגמנטים. את מהליך יש טבלה סגמנטים שמרכזת בל סגמנט את הבסיס ואת הגבול שלו.



או שהמערכת תהליך שהיה שמור סמאפורים אחרת. אל תשתמש
אחרון ברגע את ההפעלה שיכול להגיע לחומרה אפשר שמור
הסמאפורים. משמע לה יש רישום איזה שמבקש על שמור
ורשום context switch על מה שעד ההפעלה שיכול עד שיור
הוא מעדכן את הטבלה הניהול רישום הלה.

האפשר יכול לגשת
לה תהליכי
שמאפשר את שמור
על הסמאפורים
שהרישום מגבלת
אלה.
(האפשרי לה אלה רק
הנאורים הלה ופה)
על גישה לזיכרון עד שור
האפשר אוטומט שכל
תהליך האחרון. על
אותנו רוצים עכשיו
לה תהליכי שאינם שמור
ל"ו הטבלה הלה ופה.

אבל תהליכים בפועל לא יכול פשוט לאפשר לכתובות של תהליך אחר
(אלא אם כן ישנם מנגנוני ההפעלה).

איפה מנגנוני ההפעלה שומרת את הכתובות של טבלת הסמאפורים של כל תהליך?
לכל תהליך יש PCB - מבנה נתונים שבו יש מידע שכבר שקשור אליו.
אז שם אפשר יהיה לשמור את הכתובת של טבלת הסמאפורים.
ניתן שתהליכים חדש יכול אז תמיד תהליך חדש וכמו
טבלת סמאפורים ומיפוי של הסמאפורים לכתובות הדפים. אז, ההפעלה
צריכה לנהל את הזיכרון הפיזי - לדאוג איזה פנוי ואיזה תפוס וכאשר
צריכה אפשר להקצות בזיכרון לתהליכים חדשים. ובמידה ויש לשחרר
כאשר בקשו.

הקצאת סטאטיים רציפים

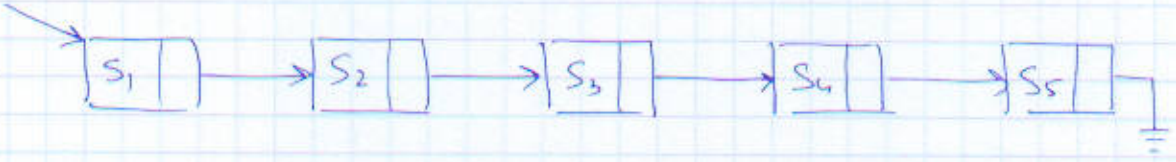
יש שתי פעולות שדרוש לתאוק מהן:
1. הקצאת מקטע מגודל S
2. שחרור

המציאות שלנו לעתים הלה הכל דינאמי. אפילו היה יכול מערכת את לא הייתה שום בעיה. אולם מאידך הדינאמי יכ כל הלמען הקצאות ושחרורים.

אך ההפלה אתיקה מנהלת מערכת - רשימה אשורשית של מקטעים פנויים וגדלים שלהם.

אם התהליך להקצות מקום מגודל S יש לעבור על הרשימה ולבדוק אם יש בכלל מקום פנוי בגודל S. אם אין כזה אי אפשר. אם יש, בדיק ההתליך אפשר לשים אותו. המידע ליצי הסטאטי המוקלה נשמר בטבלת הסטאטיים של התהליך.

השאלה היא איך מאורגים את הרשימה ואיך מתפשים מקום פנוי. נסתכל קודם על החשוש וההקצאה. נניח שיש לנו רשימה עם כמה איברים



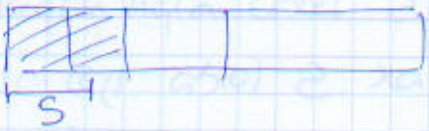
- First Fit - עובדים על הרשימה מההתחלה עד שמוצאים מקטע הפנוי הראשון שמתאים ומקצים ממנו.
- Best-Fit - עוברים על הרשימה מההתחלה עד הסוף ומחזירים את המקטע הכי קטן שמספיק בגודל המבוקש, כלומר כזה שהחתאמה תהיה הכי הדוקה.

אין תשובה אוטומטית מי מהם יותר טוב. לאלגוריתמים שונים מתאימים אלגוריתמים שונים. אולם באופן כללי, First Fit עובד בדרך כלל יותר טוב כי Best Fit יוצר פרגמנטציה של קטעים אשלך קטנים שהיו שכיר שוגע לשמוש בהם אשפו.

פרגמנטציה - המצב שבו הזיכרון מחולק למקטעים הפנויים מפוזרים ובדרך כלל שטח כזה יש מספיק זיכרון במקרה כזה אפשר להתרגז להגיד את הזיכרון, אבל לא עזרה מהמון למן!

הקצאה ע"י Buddy System

ניהול שמאפיין את הקצאת הזיכרון ס. את בקשה הקרובה ביותר מתלקים את הזיכרון לחלקות של 2 ומקצים את החלקה הנאותה ביותר שאפשרית. נניח הקצאנו 2^k. יהי j הינו ש- $2^{k-1} < s$. את החלק שנשאר לא משמוש נקרא פרגמנטציה פנימית.



יתרונות: התיפעול יעיל יותר - זריזות אלו עניין.
חסרונות: - קשה לנהל אם שטח הוא פנוי
- פרגמנטציה פנימית.

(29) 28.6.07
OS

ניהול זיכרון ע"י דפדוף (paging)

בלינוקס יש לנו 3G מתוך ה-4G שיש בכתובת של 32bit שמייצג את המרחב של התהליך.

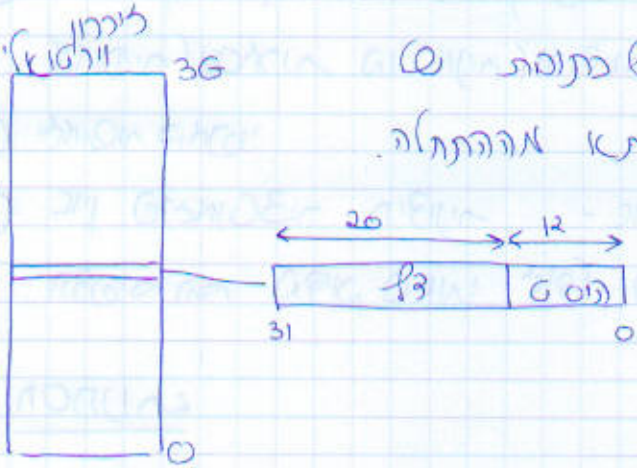
נעזרים לחלק את הכתובת הנ"ל לשני חלקים. 20bit ו-12bit שמייצגים "דף" ! שמייצגים הסט מתוך הדף.

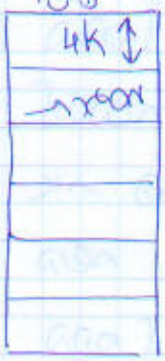
גודל דף הוא בגודל $4K = 2^{12}$ וישנם $2^{20} = 10^6$ דפים.

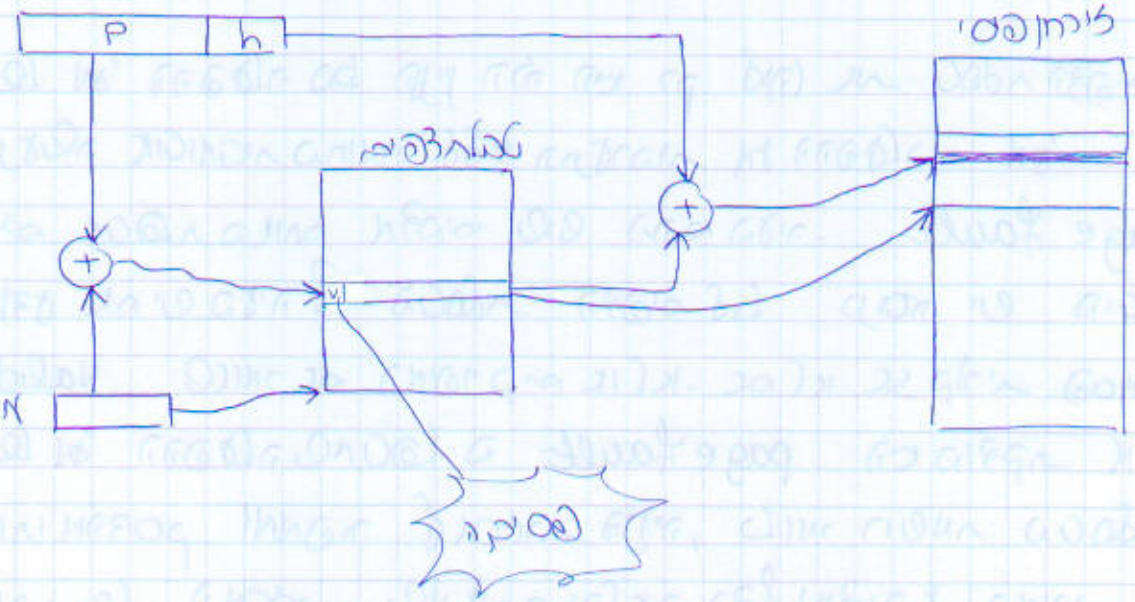
אם את הזיכרון הפיסי מחלקים לאזורים שקוראים להם מסגרות. וע"ז רעיון הדפדוף זה ההפרדה בין דפים למסגרות.

ההפרדה בין דפים למסגרות דומה מאוד כיוון שלשניהם יש אותו גודל אלא הכתובת יותר פשוטה, כי ע"י הקצאת דפים נאין את הצורך של חיבור מהתחום.

עם דף אחד נוכל להצביע לאזור מסגרת הים מאוכלס. בשביל זה יש לנו טבלת הדפים שאומרת איפה נמצא כל דף, עוד מאיזו מסגרת. את רשומות לשם שאחראי קצת מחזיקים אותה הדף ואת הדרים והסט השון.







מחלקה:

1. התחלקות מרחב כתובות לדפים. (גודל הדפים בכל המחשבים זהה, מאפשר לאפיון מיקומים ומשאבי מיפוי של הדפים)
2. שיתוף בזיכרון.
3. אין פרגמנטציה חיצונית - אין פרגמנטציה בין האפשרות כי כל הדפים הם בגודל אחיד. (אבל יכולה להיות פרגמנטציה פנימית.

הסתגרות:

0. תיאורטית יתכן שיש סגמנט שיתוקף אבל בפועל נשמרו להשאיר סגמנט פנוי להשאיר את הדף. אפשר פרקטית לא קורה.
1. אם הדף נמחק קרה מיפוי יש פסיקה (page fault) ואז התכנית כלה לא יותר - אישה לפסיק לה יקרה!
2. בשביל שיהיה לכתוב לא עולה לנו בשעה - זה גישה לטבלת דפים אם גישה לכתוב עדיין. כאשר לא יעיל בכל אלו פתרו את זה: טבלת הדפים נשמרת ב- cache אותו בשם TLB. ואז שומר את ההתמודות של הדפים שהשתמשנו בהם לאחרונה, אז בלי לפנות לכתובת רק אם פונים לדף חדש שעוד לא נמצאו ב-TLB.

התפקיד של מ' ההפעלה בפעולה הזו היא רק לעדכן את טבלת הדפים. ב השאר נעשה אוטומטית בחומרה בלבד התערבות מ' ההפעלה. מ' page fault ההפעלה היא אמורה לאפשר את המיפוי שיש שהיה בדיוק אבל אין דף! אז יש בעיה? בטבלת הדפים לכל כניסה יש סיבית valid שאומר אם המיפוי קיים או לא. אם לא אז קורה פסיקה ולקוד של מ' ההפעלה שמטפל ב page fault. הוא מודיע או תסדר, מביא את הדף מהכתובת ומאפשר לאפליקציה, ואומר רושמת בטבלה את הכתובת של הכתובת שם לתם והלאה הדף נמצא. כאמן, הוא גם מדליק את ה valid-bit.

אם אין מספיק מקום פיזי לכל הדפים של התהליך (כולל הדפים שנמצאים כרגע בדיסק) האם ניתן להריץ את התהליך? אולי ניתן להריץ תהליך שלא כל הדפים שלו נמצאים בזיכרון.

Demand-paging - מנגנון שמביא דפים לזיכרון רק כאשר מתרחש page fault. מתרחש page fault כאשר התהליך מנסה לגשת לדף שלא נמצא בזיכרון.

כשמתרחש page fault, מערכת ההפעלה יודעת שהדף לא נמצא בזיכרון ומביאה אותו מהדיסק. אם אין מספיק מקום בזיכרון, מערכת ההפעלה צריכה לפנות מקום ע"י הוצאת דף אחר החוצה. אם הדף שהוצא נכתב, יש לכתוב אותו חזרה לדיסק. אם לא, אפשר פשוט לדרוס אותו.

ביצועים: יש לנסות להביא לזיכרון רק דפים שבאמת נחוצים. אחרת, נבזבז זמן על העברת דפים מיותרים. בנוסף, יש לנסות לשמור בזיכרון דפים שנמצאים בשימוש תכוף, כדי למנוע page faults מיותרים.

working set = resident set

working set: קבוצת הדפים שהתהליך ניגש אליהם בפרק זמן מסוים.

resident set: קבוצת הדפים הנמצאים בזיכרון (משותפים לכל התהליכים).

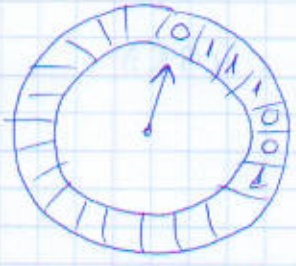
לפחות את ה- resident-set גם בזיה פשוטה מאוד.
לעומת זה - working-set לא רמה יותר מסובכת. הנחתה היסו שהעמוד
הקרוב הסו אינדיקטיבי לעתיד הקרוב - הלוקליות!
אנו מתעניינים בה-WS להיות קטנה ככל האפשר ושתעלה בדיוק
ב- d הכי קרובות האחרונות.
אם ה WS גדול מדי של התהליך אחרי הפלה - יש סכנה של
thrashing שזה אומר שכל הזמן יש page-faults
הפתרון הסו להקטין את ה- MPL (כמות התהליכים במע')
ולהחליט לתהליך אחד, אפשר לעשות swap-out - להוציא
תהליך ולהעביר את כל הדפים שלו מ-זיכרון לדיסק. ה scheduler
מחר את התהליך שנמצא מדפדפים הכי נמוכה ומשאיר אותו
מדיסק עד שהוא יחליט שאפשר להחזיר את התהליך.

Least Recently Used (LRU)

לוגיקה: לשמור את כל הדפים מרשימה משורשרת. עם כל גישה
נעביר את הדף לראש הרשימה. ה- LRU הסו נגד הרשימה.
הרשימה מאוירנת לפי הסדר האחרון שניגשו אליו.
את הראש של הרשימה הוא קיחה טוב של ה- WS.
איך לממש את זה? בעזרת משהו מתאים השימה משורשרת בו כיוונית
הנאנו שיהיו. כמו כאן נעים. אין תוארות שמושות את זה.

אלגוריתם השלוח Second Chance

זה מה שמושם מפועל. מוסיפים מיט אחד לכל דפים וכל
פעם שמשתמשים מכניסה התוארה מדליקה את המיט. מדי ההפעלה
ונלה לרפות את המיט. מחמינה לוקי מדי ההפעלה מאבדות את
(מסתכלים בכל דף ורשמים את ערך המיט. יש
מחוג שמצביע למשהו בשיש page fault אם הוא מצביע
ל-0 משתמשים במשהו, אחרת, משימים את הלוקי ל-0
ומוסרים הלאה (נותנים לדף הזדמנות שניה).



(31) 5.7.04
יום שלישי

מערכות קבצים

מה זה קובץ?

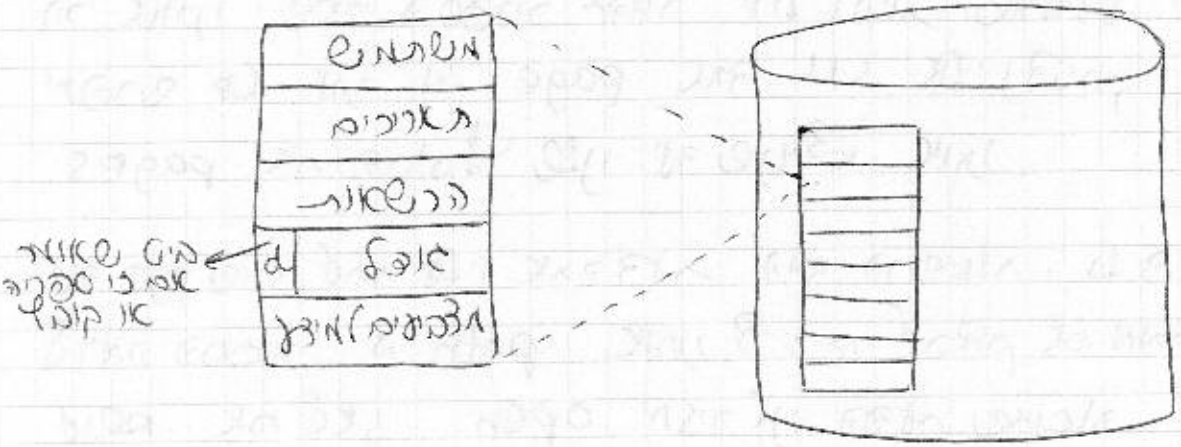
- מבחינתי יש רשימה של קובצים לא משנה עוד שלא היה קיים אלמלא מס' ההפעלה
- המידע שאותנו כותבים לקובץ נשמר לטווח ארוך
- לקבצים יש שם - בו ניתן לגשת אליהם

היום נתמקד ב: איך מממשים את מע' הקבצים.

מה יש בספריה?

- הרשאות משותפות
- שלא יהיה בלאגן
- מיתכון שמות!

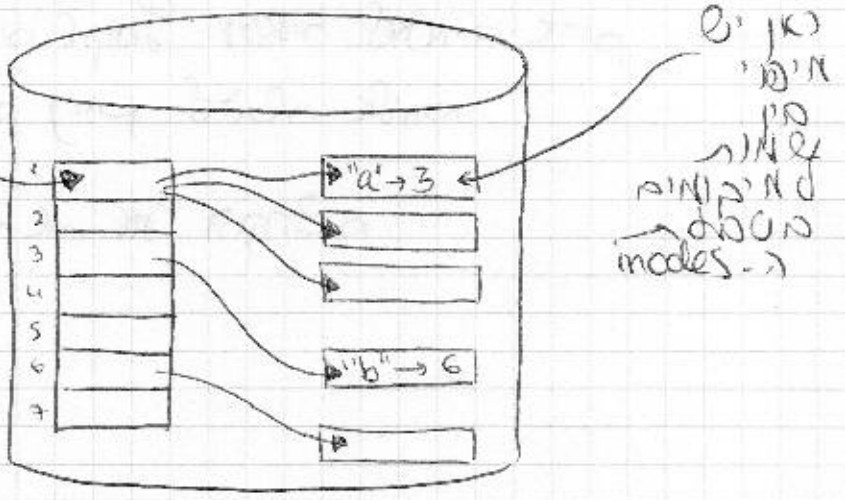
מע' ההפעלה מנהלת את כל עניין הקבצים. לכל קובץ יש מבנה נתונים שמייצג אותו. מינימום זה נקרא inode.
מבחינת הפקודה open(path, mode) יש מיפוי ל- path שמוביל למשתנים שבהם הנתונים של מע' ההפעלה.
יש הרבה קבצים, אז יש בזבוז טעם לתפוס את כל ה-inodes שמורה בדיסק.



נשים לב שהשם לא נמצא במבנה הנ"ל כי השם נמצא מספריה שממפה

אתנו מאיתים לקחל.

נניח שקוראים ל- open("/a/b", R). מה קורה?
בדיק להתיל ל- root. אין נקל אידה. ומאי? טוב, לה
נעקוב אחרי שהוכים אלו, צל אין מדיהה, בדיה להיות
מוכרה אפה הוא נמצא ואל יש לנו פוינטר קבוע אלו.



ה- inode של השורש יש פוינטר למקום שבו נשמר מידע
אלו. בפרט יש שם אינפו' בין ל הקבלים שמתוכ ה- root
והמקום של ה- inode שלהם.
אל נניח שה inode של "a" הוא ה- 3 אל הוכים ל- 3
ומתפשים שם את "b". וכהו.

מדיה אפשרה: כל התהיכ הזה פורש ד' הרבה גישות ל מכשיר -
ל פחות לפי האוכ של ה- path אותו עוד להיות שלח ידוע.
וכי אומרו שאם הספריה אפשרה יכול להיות שמידע שלה
יתפרש לא ידוע ה- page אחד וכל אל נפטיק לחפש מכאן
pages את הקבלל שלנו לד שנמצא אותו.

אוד נניח שיש לנו את פלו אחרי ומה הרשאות. כ פלח שעוכים
לראה בהאוה ל path אומרו לריכים לבדוק אם מותר לנו לעשות
פעולה. אם לא, open יחזיר לנו הודעת שגיאה.

נניח שהתוכנית שלנו נראית כך:

```
fd = open("/a/b", ...)
read(fd, buf, 100)
write(fd, buf2, 200)
```

ונראה איך יתנו הקריאות:

נשתמש בדוגמה הזאת כדי לשפר את התוכנית שלנו ולהבין מה מאחורי הקלעים כשמדברים על "שמירה". כלי נוסף שנשתמש בו הוא הפקודה strace.

ראשית: שמרו את הקובץ הזה בתור test.c והריצו את הפקודה הבאה: שימו לב שאתם מקבלים הרבה יותר פלט ממה שציפיתם. אל תבהלו, נעבור על הכל בהדרגה. אנו נתמקד בשלושת הקריאות המסומנות. אל תנסו לקרוא עדיין את כל הפלט, אלא רק את הקריאות של המילים. אל תבינו עדיין את כל הפרמטרים, אלא רק את שמות הפונקציות.

אחרת: מה שלא הבנו בשנה שעברה לגבי "שמירה"?

Memory-mapped files

אלטרנטיבה היא לתת לו שם קבוע מוגדר ע"י שורה בקובץ.
אם רוצים לשתף קטעים מסוים אפשר גם בלי קבצים,
ויהיה page-fault ואז ההגדרה יתן לקובץ. אם משנים
מעתק את הסיגנון זו בלי שהוא modified ומשתמשים
לשאר יקרים שיכיר לזהות אותו לפעם.
ומסיבה זו קשורות לו בדיוק שהוא בדיק מאוחר לקבצי
מקום ל- buffer cache, הרי זה יתן מאוחר כאה מקום
אותנו נעצרת משמעות

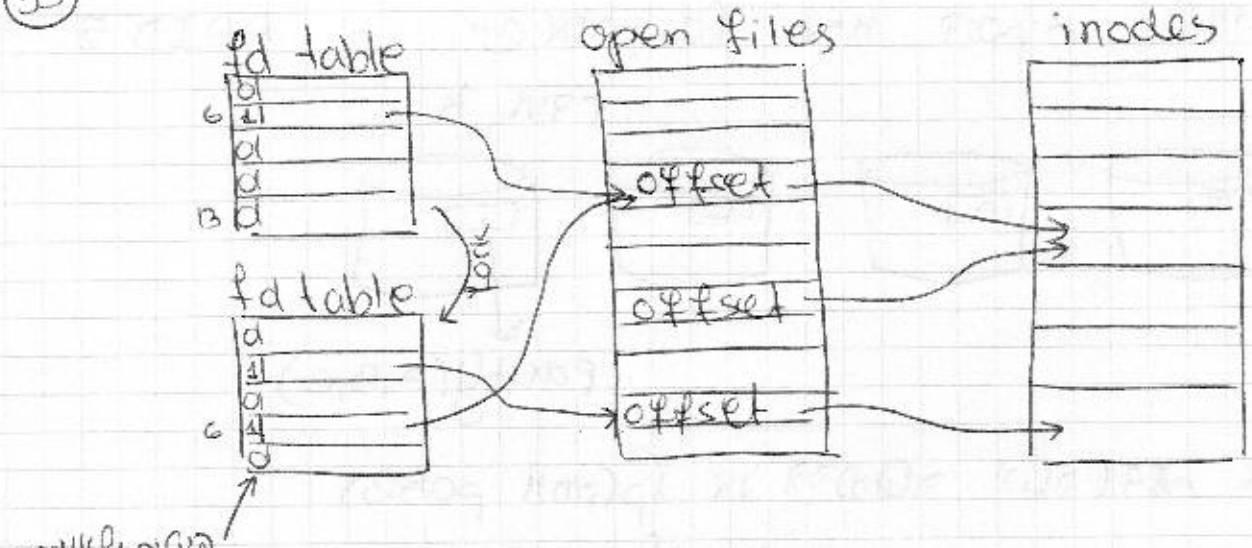
נאחר כאה דברים על השקרים שהמאיסו עלינו בשלה הקודמת.

קבצים מתותים ב unix

מיונקס יש לנו טמפה אחת אולא 3 8
- inodes - כל inode מייצג קובץ מאשר מוגדר על
ידיסק (כפי שהוא תכונה של הפיזיות
של הקובץ ולא של הקובץ עצמו)
- open files - כל כניסה בטבלה מייצגת פתיחה חשקה
של תהליך. היא מכילה שני דברים חשובים 8
  - offset (מאיזותה) 1-0)
  - פוינטר ל- inode הרלוונטי.

ומכאן תהיינה פותחים-אותו קובץ יכול
להיות להם offset שונה. כל שהם שני
כניסות ב- open-files אבל הם
מצביעים לאותו inode

- fd table - לכל תהליך מוסיף יש טבלה כמו זו שהוא מצביע
ל- open files.



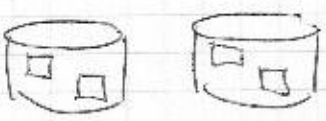
ה-fd-table ניתן לשכפול בעזרת הקריאה fork(), ושתי טבלאות ה-fd table שלו ומשרתים מצביעים לאותה offset. נשים לב שה-inodes וה-open files משותפים לכל הקבצים הפתוחים בתהליך אב ובנו. אולם כאשר אחד מהם מבצע lseek() - כלומר משנה את ה-offset - הדבר ישפיע על השני (כי הוא ישנה את ה-offset המשותף).

אחזור

כמה היטיב של דיסק הוא 5-10 שניות אפילו כדי לגרום לאחד שהדיסק לא יכול לתת את כל המידע שמבקשים.

כדי שמספר כזה לא יפריע לנו בפיסקים שאנחנו מתקינים:

RAID 1 - בעל מהוק יש mirror



יש שני דיסקים שהם אחד של השני.
אם אחד מתקלקל יש הדלק של השני.
יש לזה יתרון של איזון עומסים מקריאה.
אפשר להסתדר הכי שהנפח הוא רק 50%.

RAID 5 - יש מצב של k+1 דיסקים ושמירת k מידע.

כשדיסק מתקלקל את יכולה לשחזר את המידע כי הנתון לא אובד.

אז אם C התקלקל אפשר לשחזר אותו מלהיות ה - parity.

אחת הבעיות קיום דיסק אחד שמתקלקל. שכמעט שאחד ה parity הוא תמיד בדיסק אחד הכי עמוס בקבוק. את RAID 5 אוהב ש- parity יהיה בדיסקים שונים.

מוקד שמע ושרון!

12.7.07
20

חלק תרגיל מעשי

NFS - מי שלא יודע, אותנו משתמשים בלינוקס ע"י הרצת פקודת שיתוף שיש לפני login אז אותנו משתמשים ב NFS. מרוב שזה - ייתכן שהוא מפיסק הלקוח של המשתמש, שאר. אין לנו יכולת לדעת אפילו אותנו שמחובר לעבוד עליו מלפי, אלא אחרי שמשתמשים בכל הקבצים שלנו. זה המצב האידיאלי שהיינו רוצים שיהיה בכל מערכת הפעלה.

הרעיון - לקחת ב"מרוחק" את הקבצים.

אחרי הפקודה נוכל לגשת אל root של הקבצים של FS שם תקבל ריקה ב- WS. צמוד ל- inode של הת"ה ב- WS נתקה שנקראים נושאים בשרת רחוק שנו נושאים ע"י הרכיבים.

אפשר כאשר תהיה אפשרות לקטע קבצים מאחד השרת. אפשר גם אשר משאפיו בעץ תהיה הצורה. שמע, מסמיה /tmp יש קבצים שלפ"ה של התהליך ומה עד תהיה צריכה יכולת שהיה קבצים שונים אשר. נבנה כאן תהיה שהיה. ויתרות קבצים מאחד שרת. אפשר ע"י הרכיבים.

מגבלות של הרכיב:

- מגבלות תקשורת
- מגבלות אחרות (בשרת שלנו לא. שהיה)

/

בפעם הבאה שבה נבצע פעולה על הקובץ, שגי הפעולות שהלקוח יבקש (נאמר לקרוא בלוק בודד או בלוקים מרובים) ושמי האופציות יתרחשו באותה מידה.

פתרונות ב- NFS

① בשרת חסר-מידע (State less)

כדי להשיג את מטרת הקודמת יהיה עליכם לוודא שהשרת לא יחזיק מידע לגבי הלקוחות. כלומר, אם הוא שומר את המצב של הלקוחות הוא יכול לאבד אותו בקלות במקרה של קריסה.

② פעולות אידמפוטנטיות (כמו הטרנזקציות) - פעולות שאת תוצאתן אי אפשר לשנות בכך שנבצע אותן שוב ושוב. לדוגמא, עדכון מסד נתונים הוא פעולה אידמפוטנטית, אולם עדכון מסד הנתונים בפעולה אידמפוטנטית.

ומשתמש מרוכז של מחשב ושולח שאילתה את הפעולות והשניות על קבצים.

NFS לא תומך בקונסיסטנטיות. אבל יתכן מצב שבו תהליך אחד מעדכן קובץ ותהליך אחר אומנם קורא את אותו קובץ אך מקבל גרסה ישנה.

שרת תשובה

לא אוכל להתנות מראש שנותן את שרת תשובה.
אך רוב המשתמשים מוכן מוקדם יותר של לינוקס שבו אנשים
והיו מאפשרים ניוד של תהליכים (migration). וגם
אנחנו מרוצים מאוד שתהליכים את התפקיד שלנו הוא מרכזי
לעבוד.

במובן מסוים שאלות כמו שאלות:

- איך?
- מתי?
- איפה?
- איך? (ומשם רצף מהלכים שונים לפי מה שקורה בשרת, קריאות ופרטים פיסיים של העס. תהליך שמתמלא ת שונים כשמשפט מסוים ע', נאה פרטים: הדיסקיות, אחרת בניסוי, BSD וכו'. זע ס כי אפשר פשוט לעבור לאכוף אחרים אותם מהקרה אם התהליך פתח קובץ או ויתר השרת?
- כ- צוטט מטפלים מה ז' זה שמדפס ס תהליך בשר שלו מתחת לפיה שהוא הוא הוא הוא הוא system calls
מפעיל פה יפה.
- איך? יש פה כאה שיקולים.

① הקטנת הקריאה ⇐ מהיר התנהגות קטן

② הפשטה רוחב, זה מתיימר לספק את הזיכרון - ז' דפוי עשוי
לעשות. אחת מה מתאים אפשרי. כ- יוסע פיק. יש שני
דברים שפשר עשוי: ③ מהי תהליכים שיכול הרבה למן
ההזדמנות שמשתמשים בה בלה הכי שתהיה ת ל כ
לתת כאן שהוא מרוצים ז' עבור. מסתבר שתפעילו
תחת מהליכים ויש ז' עבור מפוקפק שלנו הוא מרוצים לא
לעבוד.

/

מתי התהליכים הם כאלו שבכלל שתהליך רק יותר נמוך ישנה
לו יותר מאוד קרוב, נתקע בפאזה להביא אותו.
ואם התהליך הזה עשוי להיות Stuck אצל כאלו שלא להביא
אותו לכזה תוצאה סופית אז אתם את התהליכים והזיקים
שיוצרים הטבע לעזרה.

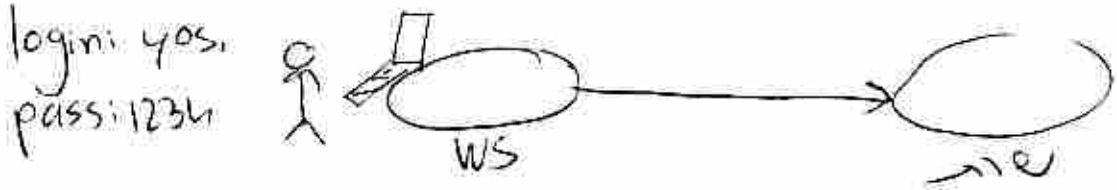
- לעומת אפשרות שמטופל ניצל בהזדמנות מצוינת ובמשימות
תוכנה יותר ואם היא יכולה לעזור בנושאים לשלוח אליה.
חוץ אופציה היא להסתכל על כל התחנות מבחוץ את הכי
טובה. אנחנו יכולים עם כזה שאין פעיל.

* עוד שנבחר את הכי טובה לה ולא לשתנות
* אפשרות תחנות עושות את לא מוכן או כן כולן
תכתירנה את אותה תחנת לאלה אפשרות שהיא באותו
מקום הכי פחות עמוסה.

מה שנ אצטמת עושה וכמו Twist עם ושכה דרמטית.
מעניין שם בפקד על תחנת לבוא כותרת תחנת לבוא את
ושלחת לה את הגלגל שלה אבחנות עומס. ג שם
תחנת לבוא אפשרות על 2 אותחנות שנשלחו לה מיצג,
מותרת את האופציות ושלחת לה לאחד. אפשר
לכאלו שנעשים אחשבן אחר מאוד מדגה אחידה על
כל תחנות הבודדה.

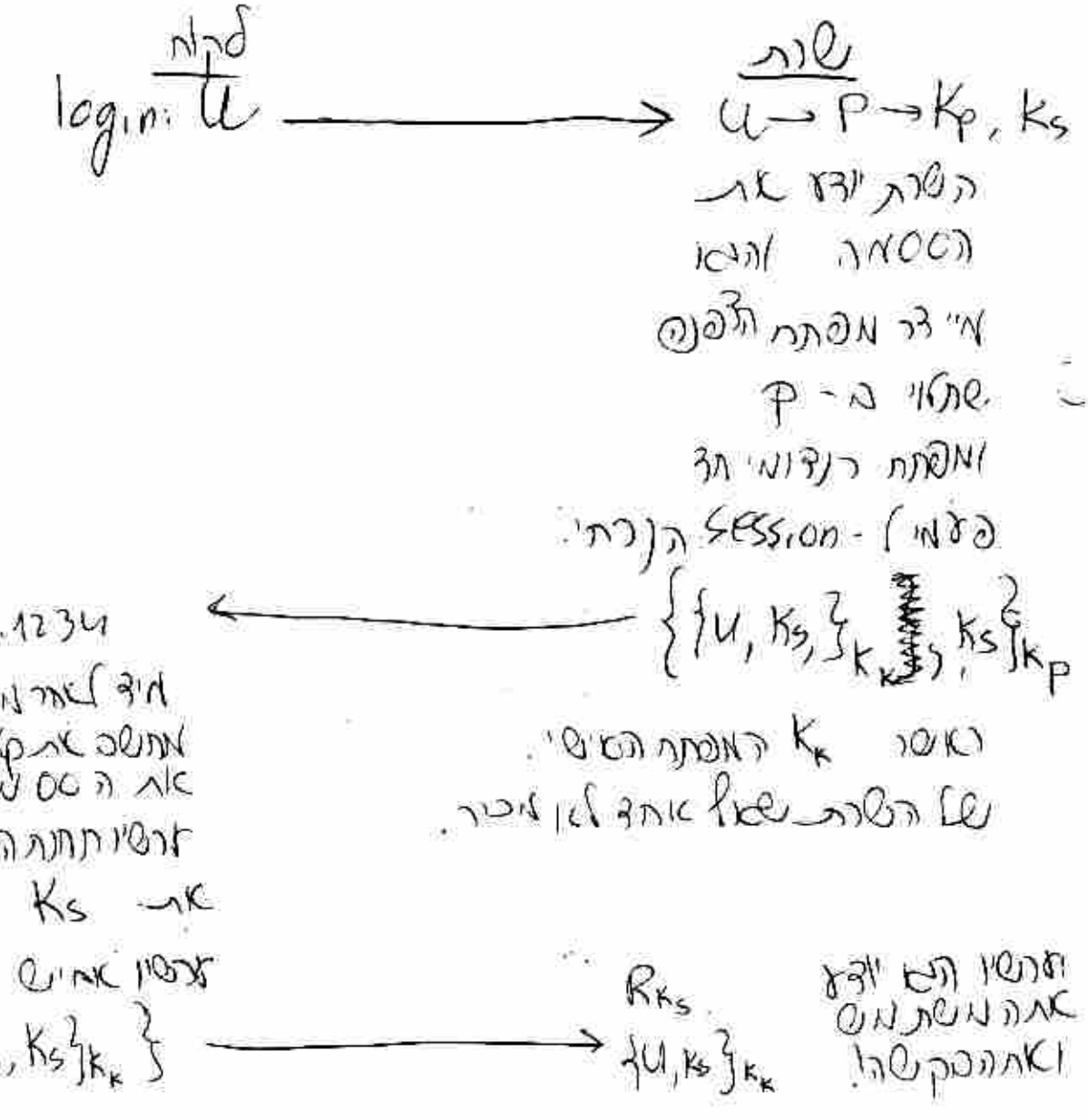
אימות משתמש (authentication)

מרבית שיטות האבטחה ברשת מבוססות על הנחה שהמשתמש הוא מי שהוא טוען שהוא.



כיצד נדע האם הוא אכן משתמש בשם login המתאים לסיסמא בשרת האימות? במילים אחרות, כיצד נוכל לוודא שאדם מישהו מסוים מתקשר עם השרת ולא מתחזה?

שיטת Kerberos



26.7.05
אס תפעלה

שיעור חלרה

- Buddy System - שיטה להקצאת סיאוגים רצפים בזיכרון (תהליך בדפדוף). אותנו נתמקד ב OS בעתיד.
  הרעיון הוא ששומרים אפשר נתונים מדורת על ומו שומרים סיאוגים.
  - את ההקצאות תמיד עושים בחלקות של 2.

דוגמה: הנה הזיכרון שלנו. הוא ריק

ורוצים להקצות נקה זאת □

את מחזיקים את הזיכרון כך שעוקבים את העבוד המינימלי שלה נכנס בו.

מקצים את לה. ← שפרימוטיבה פנימית.

ועל התאוים הוא:

ברשון אם מישה תתיבה כליאת □ את אותנו שר יוקצים אישה להקצות אותה. ואם מישה תתיבה כליאת ▯ אל אותנו יכולים לתרץ שוב את הבעל.

משתדור קוכה פשוט התהליך ההפוך. כמשתמשתמים שלה מפקיעים אם הבאת Buddy שלו משתחר ואם כן, את אשר לאחד אותם.

מטפלת הרוונטי (Interrupt Vector). יש פסיקות
יש רשימה של פוינטרים לקוד של מ' ההפעלה שמטפל
לכל הפסיקות והשגיאות יש מספר ומכיוון שמבוא מוגדר מטרש
- פסיקות אינטרנט - נגרם ע"י תוכנה בשימוש.
פשוט מביאים את הדף הנדרש מהדיסק.
- פסיקת page fault - נגרם כאשר לא נמצא
את הכתובת בזיכרון הפיזי. ב- page fault אפשר לטפל -
יש כאלה שמביאים את הדף הנדרש מהדיסק
אחרים פשוט זורקים את התהליך. לעומת seg. fault
שבו אין טעם לנסות שוב כי הבעיה היא
בקוד של התהליך, page fault, מחזירה את המצב היכן
שהיה לפני הפקודה שגרמה לשגיאה. לעומת segmentation fault
- פסיקת חומרה - נגרם ע"י התקנים חיצוניים
שליחת אותות ע"י התקנים חיצוניים. לעומת,
שלוח יכול לגרום לפסיקה חיצונית אחת קיסם.
- פסיקת מלכודת - trap - נגרם ע"י פקודה שאומרת למערכת
לעשות משהו ונצטרך לקוד של מ' ההפעלה. לעומת,
מהתכנית שמריץ תומרת לקוד של מ' ההפעלה.
על פסיקת מלכודת לא תמיד (exceptions) - זה בגלל
שהמעבד מנסה לבצע פקודה שאין מספיק מקום
לזכור ומאבד את השליטה למ' ההפעלה.
פרמטרים וחזרה:
- פקודת חזרה - השתמשים לחזרה מפקודה שנקראה ע"י
שגרת mode. זה יכול להיות אם אתה כותב מ' לדמות
או מתפעל או שמייצא תקלה במולטי-לוגר.
- חזרה ביכון - השתמשים לשחזר את התצורה שהיתה לפני
הפעלת הפסיקה. לעומת segmentation fault
זה פתח שקורה אם אתה מנסה לגשת
אחת הסגמנטים שלו. page fault, מבחינת המעבד היא
יצרה בעיה אחרת כמו seg. fault אלא מבחינת
מערכת ההפעלה זה שונה. ב- page fault אפשר לטפל -

- **פסיקות** - פסיקה זה אובייקט חומרה מאותחל אשר
מבלע את הפקודה הבאה שמגיע היה אחרי שבלע.
יש כאלה שמטפלים עוד לפני
- פסיקות חיצוניות - זה כשהכתב תומרה חיצוני למעבד מגיע
שקורה משהו וצריך לקוד שמ' ההפעלה. לעומת,
שלוח יכול לגרום לפסיקה חיצונית אחת קיסם.
- פסיקות מלכודת - trap - נגרם ע"י פקודה שאומרת למערכת
לעשות משהו ונצטרך לקוד של מ' ההפעלה. לעומת,
מהתכנית שמריץ תומרת לקוד של מ' ההפעלה.
על פסיקת מלכודת לא תמיד (exceptions) - זה בגלל
שהמעבד מנסה לבצע פקודה שאין מספיק מקום
לזכור ומאבד את השליטה למ' ההפעלה.
פרמטרים וחזרה:
- פקודת חזרה - השתמשים לחזרה מפקודה שנקראה ע"י
שגרת mode. זה יכול להיות אם אתה כותב מ' לדמות
או מתפעל או שמייצא תקלה במולטי-לוגר.
- חזרה ביכון - השתמשים לשחזר את התצורה שהיתה לפני
הפעלת הפסיקה. לעומת segmentation fault
זה פתח שקורה אם אתה מנסה לגשת
אחת הסגמנטים שלו. page fault, מבחינת המעבד היא
יצרה בעיה אחרת כמו seg. fault אלא מבחינת
מערכת ההפעלה זה שונה. ב- page fault אפשר לטפל -
פשוט מביאים את הדף הנדרש מהדיסק.
- מלכודת אינטרנט - נגרם ע"י תוכנה בשימוש.
לכל הפסיקות והשגיאות יש מספר ומכיוון שמבוא מוגדר מטרש
יש רשימה של פוינטרים לקוד של מ' ההפעלה שמטפל

תהיה ההתנהגות.
מסיגלים יש יותר אישות. לכן סיגל בניגוד לאשר לקבוע מה
לפי - נגוע אשר למסוף את כ הפסקות א- x ומטה.
כ הפסיקות מסוגות לפי סדר ואשר למסוף פסקות לפי
לעיל להציג לתנועת עם ואז בדיוק אז ההפעלה הניצה את
שונה את הפסקלפה ואז למשל מ אחרת של תלוקה מאפס ניכנס
לטיפול אינטרפט של עיסון מדוע הפסקלפה) או שאפשר
ואז סתר שם. כך נאפשר אותו מקום מתן האופק יעה לבדק
התהליכים. אפשר לטפל מסיגל, אפשר להתפלח מסיגל את
ליניק ש שלחת לתהליכים סיגל ואז לה תלוי במי שכתנה את
להמשיך ואז זה תלוי במעך ההפעלה את עושים. אך ההפעלה
במהלך של תרגיאת (חוץ מ- page fault) מכך א אפשר
שמידע את הקריאה.
הקיימים מעך ההפעלה ורשמיים חוברים לקוף המשתמש
שמסופן תוכרים לתכנית אחר ממשק בניגוד - למשל הפסקות
של שעון או של הדיסק. השקרואים ל- trap נוכרים
לפולקציה שיש מה switch עוקף את כ ה system-calls