פרק #2 מערכת ההפעלה והחומרה

מחשב כפי שמוכר לנו כיום כולל:
מעבד (הכולל בתוכו אוגרים, ולצדו זיכרון מטמון [cache])

רכיבי ציוד המנוהלים ע"י בקרי ציוד\התקן (device controllers)

כל הנ"ל קשורים דרך פס\ים (bus) משותפ\ים לזיכרון, שהפניה אליו מנוהלת ע"י בקר הזיכרון.

<u>תזכורת:</u> אוגרים הינם זיכרון קטן, יקר, ומהיר, שנכלל במעבד.
דוגמות לאוגרים שיעניינו אותנו:
IR, PC, PSW,
IDTR
אוגר הבסיס ואוגר הגבול (לבדיקת פניה לזיכרון),
אוגרים לשימוש כללי
בעת ביצוע *החלפת הקשר* (context switch) יש לשמור בזיכרון את ערכם של אוגרי התכנית שריצתה מופסקת, ולטעון מהזיכרון את מצב אוגרי התכנית הנטענת (כפי שנשמרו עת ביצוע אותה תכנית נקטע בעבר).

ציור 2.1

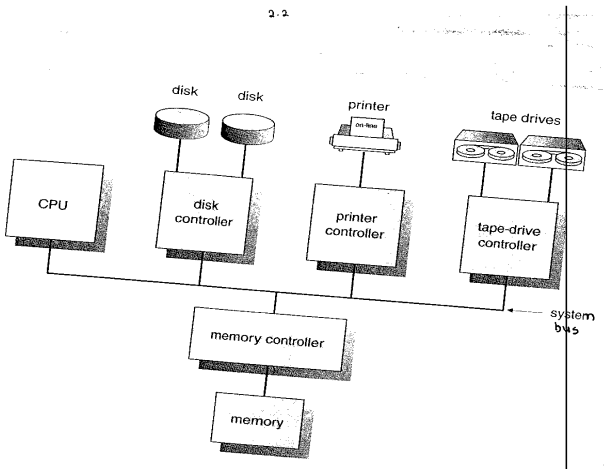

Figure 2.1 A modern computer system.

בקר, הינו מעבד פרימיטיבי, האחראי על סוג ציוד ייחודי (מסוף, דיסק, רמקול) ויודע להעביר מידע בין חומרת הציוד והזיכרון. הבקר עשוי להיות מופקד על כמה יחידות מאותו ציוד. לבקר יהיו אוגרים וזיכרון משלו. הוא יעביר את המידע ראשית מהציוד לזיכרון שלו, ומשם לזיכרון הראשי.

המעבד והבקרים פועלים במקביל (וכך מנוצלים ביתר יעילות) ועל-כן מתחרים על מחזורי זיכרון.

נזכור שמשך הפניה של המעבד לאוגריו הוא מחזור יחיד של השעון; לעומת זאת פניה לזיכרון (דרך הפס) עשויה להמשך מספר מחזורי שעון.

המשמעות: השבתת המעבד עד קבלת הנתונים להם הוא זקוק.

צרה צרורה!

פתרונות חלקיים: אחסון נתונים להם המעבד זקוק באוגרים, או לכל הפחות בזיכרון מטמון מהיר: cache שהפניה אליו אינה דרך הפס.

אתגר\בעיה: מה יישמר באוגרים ובזיכרון המטמון?

<u>1. עלייתה (ונפילתה) של מ.ה. (System Boot)</u>

עם הדלקת המחשב מורצת תכנית הנקראת **טוען העלייה** (bootstrap program/loader).

תכנית זאת מזהה את החומרה הנכללת במערכת המחשב (המעבד ואוגריו, הזיכרון, רכיבי הציוד ובקריהם), בודקת את תקינותה, מאתחלת אותה,

ותודות לכך מסוגלת לטעון את גרעין מ.ה. (השמור בדיסק), ולהעביר לו את הפיקוד.

ב- PC מכונה תכנה זאת בשם BIOS = Basic I/O System.

ה- BIOS (או טוען העלייה, באופן כללי) שוכן בזיכרון לקריאה בלבד (ROM = Read Only Memory), שנצרב ביצור, ועל-כן שאינו יכול להינזק.

(כיום הוא יישמר ב: EEPROME = Electrically Erasable Programmable ROM, או flash [הזול יותר] ע"מ שניתן יהיה לעדכנו.

המונח firmware מציין תכנה השוכנת ע"ג רכיב חומרה שאינו הזיכרון, בד"כ ROM).

עם סיום האיתחולים, יטען טוען העלייה מהדיסק לזיכרון את **גוש העלייה** (ה- boot block) של מ.ה..

גוש העלייה ייטען את יתר גרעין מ.ה..

לעתים גם תהליך זה יעשה בשני שלבים:
לדוגמה ה- BIOS טוען את ה- MBR = Master Boot Record = סקטור אפס בְדיסק שהינו boot disk (גודלו 512 בתים).

ה- MBR טוען את ה- LILO = Linux Loader שיטען את יתר מ.ה.

רק אחרי השלמת כלל התהליך ניתן לומר שמ.ה. רצה.

עת מ.ה. מתחילה לרוץ היא מבצעת איתחולים משלה (לדוגמה: עדכון טבלת הפסיקות, אתחול\טעינת מבני הנתונים להם היא תזדקק [כגון מערך ה- PCB])
ומתחילה להמתין לאירועים (וכל עוד אלה לא מגיעים היא נחה).

אירועים עשויים להיות משני סוגים:

א. **פסיקות חומרה**: הנשלחות למ.ה. ע"י הציוד המאותת לה שעליה לעשות משהו שכן משהו קרה בציוד (המשתמש הזיז את העכבר, הבקר סיים לכתוב נתונים על הדיסק).

ב. **פסיקת תכנה**: הנשלחת למעבד\למ.ה. ע"י המעבד עצמו שמריץ תכנית אשר:
   1. זקוקה לשירות של מ.ה. (כדי לפלוט נתון, כדי לברר את התאריך\שעה). = trap
   2. בצעה שגיאה (התכנית ניסתה לחלק באפס, לפנות לכתובת לא חוקית).= abort
   3. בצעה פעולה תקינה אך כזו שהמעבד 'נתקל בבעיה בביצועה' וזקוק להתערבותה של מ.ה. (התכנית פנתה לכתובת חוקית, אך כזו שלא מצויה, בינתיים, בזיכרון). = fault

<u>2. גרעין מ.ה. (O.S. Kernel) לעומת תכניות המערכת (System Programs)</u>

עת אנו דנים במ.ה. נהוג להבחין בין **גרעין מ.ה.** לעומת **תכניות המערכת**.

<u>2.1 גרעין מ.ה.</u>
ליבת מ.ה. המצויה כל העת בזיכרון ואחראית על:

1. טיפול בתהליכים: ייצורם, וניהולם, בפרט הקצאת המעבד להם, קבלת קריאות המערכת מהם.

2. טיפול בציוד: תפעולו באמצעות **מנהלי ההתקנים** השונים (device drivers) = רכיבי התכנה שבפועל מתפעלים את הציוד, בתיווך בקרי הציוד

3. ניהול הזיכרון: הקצאתו לצרכי מ.ה. ולתהליכים (בפרט במערכות זיכרון מדומה).

4. ביצוע קריאות המערכת (system calls): פונקציות שמספקות שירותים שונים לתכניות הרצות במחשב.

השאיפה בד"כ היא למזער את גרעין המערכת, ולכלול בו רק את המינימום ההכרחי.

יתר השירותים למשתמשים יינתנו ע"י תכניות המערכת (system programs) בהן נדון בקרוב.

הגרעין לא נחשב לתהליך, אלא למבקר\מנהל התהליכים.

<u>אופנות גרעין לעומת אופנות משתמש</u>
עת המעבד מבצע קוד של הגרעין הוא נמצא באופנות גרעין (kernel mode) ורשאי לבצע כל פקודה (מיוחסת) שהיא, ולפנות לכל כתובת בזיכרון.

עת המעבד מריץ קוד אחר הוא יהיה במצב פחות מיוחס: בד"כ במצב משתמש (user mode) ופעולתו אז הינה מוגבלת (הוא לא יורשה לטפל בווקטור הפסיקות).

עת המעבד במצב גרעין הוא מריץ רק קוד שנכלל בגרעין (מבחינת כתובתו) והדבר מושג ע"י ששינוי במצב המעבד מחייב שינוי במונה התכנית (PC).

(במעבדי אינטל המעבד עשוי להיות מעבר לשתי אופנויות לה גם במצבים:

1. מצב מנהלי התקן (שמועדים יותר לבאגים מהגרעין)
2. מצב תכניות המערכת (שהוזכרו מעל, וידונו בהמשך).

בהתאמה: גם מקטעי הזיכרון מסומנים כך שתכנית תוכל לפנות רק למקטעים ברמת הגנה כמו שלה, או פחותה.

יוניקס, לינוקס וחלונות לא מנצלים את שתי האופנויות הנוספות).

2.2 תכניות המערכת (System Programs)

תכניות אלה כוללות שירותים רבים שאנו רגילים לקבל 'מהמחשב שלנו' אך שאינם ניתנים ע"י גרעין מ.ה., אלא ע"י תכניות נלוות. כמשתמשים במחשב אנו נחשפים יותר לתכניות המערכת, אשר רצות במצב משתמש, וכמו כל תכנית פונות לגרעין מ.ה. (באמצעות קריאות מערכת) עת הן זקוקות לשירותים שונים

ה- shell על פקודותיו השונות, שכל אחת מהן הינה תכנית מערכת.

מנהל שולחן עבודה (desktop manager), המתבסס על תכנת חלונות (windows manager).

תכניות שונות לטיפול בקבצים: cat, cd, chmod, cp, diff, find, grep, mkdir, pwd, rm, tail

תכניות המספקות מידע: man, whatis, cal, date, look, who, finger

עורכים: ed, vi, emacs, troff

תמיכה בשפות תכנות: cc, make, lint, ar, ranlib, yacc

תקשורת: mail, write (to another user), talk

תכניות אחרות: דפדפן, מסד נתונים, מעבד תמלילים, משחקים...

כאמור, תכניות המערכת, מזומנות ע"י מפרש הפקודות (ה- shell הטקסטואלי או הממשק הגראפי).

מימוש תכניות המערכת עשוי להתבצע באחד משני אופנים:

א. מפרש הפקודות יכיל את קוד הפקודות לביצוע, שתכללנה כפונ' שונות בתכניתו. (כך נוהגים במק(?))
ב. מפרש הפקודות יפעיל *תכנית נפרדת* אשר תממש כל פקודה. (כך נוהגת יוניקס).

יתרונות וחסרונות כל גישה:

א. בגישה ב' נקל להוסיף פקודות\תכניות-מערכת חדשות (ואין צורך לשנות את מפרש הפק').
ב. בגישה ב' סוגיית העברת הפרמטרים לתכנית האחרת מורכבת יותר.
ג. טעינת תכנית חדשה לשם ביצוע כל פקודה הינה איטית יותר מהסתעפות לפונ' בַתכנית הנוכחית.
ד. יש פחות אינטגרטיביות ואולי אחידות בין תכניות המערכת השונות.

ביוניקס נהוג להבחין בין שני סוגים של תכניות מערכת:

א. commands = ls, date, pwd (פרק #1 בדפי ה- man)

ב. utilities = עורך, מהדר, מפרש פקודות.

3. <u>מבנה מערכת יוניקס</u>

א. החומרה והבקרים שלה.

ב. מתופעלת ע"י גרעין מ.ה., בפרט ע"י מנהלי ההתקנים הנכללים בגרעין.

ג. מעבר לכך אחראי הגרעין על כל ניהול התהלכים (יצורם, הקצאת המעבד להם, מתן שירות להם, סיומם) והזיכרון.

ד. מעל הגרעין מצויים מפרש הפקודות, אשר מאפשר הרצה של תכניות, בפרט של תכניות המערכת. אלה וגם אלה רצות במצב משתמש, ופונות לקבלת שירותים מהגרעין.

ה. ברמה האחרונה מצויים המשתמשים הפונים למערכת באמצעות מפרש הפקודות.

4. <u>מיקרו-גרעינים (שחורים) Microkernels</u>

אחד העקרונות הרצויים בבניית מ.ה. הוא מזעורה ומימושה באופן מודולארי (ולא באופן מונוליתי).

גישת המיקרו-גרעין מדגישה כיוון זה (על חשבון אחרים).

על-פי הגישה יכיל הגרעין רק מינימום שבמינימום (וכל היתר יהפוך לתכניות מערכת).
והשאלה: מה יוותר ומה יצא הינה בויכוח.

וכמו כן מה שיוותר יחולק למודולים נפרדים אשר יעבירו הודעות זה לזה (באמצעות מכניזמים של העברת הודעות).

התפקיד המרכזי של הגרעין: לאפשר העברת הודעות בין תכניות שונות הרצות במרחב המשתמש.

הטיפול בציוד יתבצע ע"י מודולים שירוצו באופנות משתמש, ויקבלו הודעות מתכניות הזקוקות לשרותיהם.

יתרונות:

א. שינויים במערכת לא מחייבים שינויים בגרעין.

ב. הגרעין קטן ולכן תחזוקתו יחסית קלה.

ג. בשל קוטן הגרעין נקל יותר להשיג נשיאות.

ד. מכיוון שהגרעין קטן מעט עבודה נעשית בו: יתרון בטיחותי.

5. קריאות מערכת (System Calls)

הזכרנו שכל הטיפול בציוד, ובמבני הנתונים של מ.ה. נעשה ע"י הגרעין. עת תכנית מעוניינת לבצע ק\פ, או לטפל בנתונים (לברר תאריך, מספר תהליך) עליה לפנות לגרעין שיעשה עבורה את הק\פ, או ימסור לה את המידע.

הדרך באמצעותה התכנית פונה לגרעין היא ע"י זימון פונ' הנקראת קריאת מערכת. בעקבות זימון הפונ' נשלחת פסיקת תכנה (מבוצעת פקודה אסמבלר מיוחדת), אשר (ככל פסיקה) גורמת למ.ה. להתעורר, ולספק את השירות הנדרש.

פרק #2 בדפי ה- man של יוניקס מתאר את ק.מ. (בניגוד לפרק #3 המתאר את פונ' הספרייה).

עת אנו כותבים תכנית ב- C/C++ אנו מזמנים פונ' ספרייה (כדוגמת fopen או cin), פונ' הספרייה היא שמכינה את הארגומנטים לק.מ., ושולחת את פסיקת התכנה. מבחינתנו לא ניכר ההבדל בין ביצוע ק.מ. ישירות (time) לבין זימון פונ' ספרייה המזמנת ק.מ. (scanf), לבין זימון פונ' ספריה שכלל לא מזמנת ק.מ. (strlen, rand).

פונ' הספרייה מגדירות לנו API = Application Programming Interface = כיצד תכנית מבקשת שירות ממ.ה.

שלושה API נפוצים:

א. Win32 למערכת חלונות.
ב. POSIX API (כפי שהזכרנו).
ג. Java API לתכניות הרצות על JVM.

פונ' הספרייה מגבירות פורטביליות.

פונ' הספרייה תעביר את הארגומנטים לגרעין:
באוגרים,
ע"י דחיפתם ע"ג המחסנית,
ע"י אחסונם במקום כלשהו בזיכרון. כתובת המקום תועבר באוגר.

רק כדי לסבר את העין נציג תכנית דמיונית, וק.מ. שהיא מזמנת: התכנית קוראת נתונים מקובץ א', וכותבת אותם על קובץ ב'.

א. על התכנית להציג פרומפט למשתמש (ק.מ.) ואז לקרוא את שמות הקבצים מהמקלדת (ק.מ.)

ב. עתה התכנית תפתח את קובץ הקלט לקריאה, ואת קובץ הפלט לכתיבה (ק.מ.).

ק.מ. עשויה להיכשל (קובץ לא קיים, אין הרשאות, לא ניתן לקרוא מהציוד) ולהחזיר אז ערך > 0.

ג. עתה בלולאה התכנית תקרא מקובץ א', ותכתוב על קובץ ב' (ק.מ.). נניח שכל נתון נקרא התכנית כופלת בערך אקראי. srand(time(NULL)) מצריך ק.מ. (בשל time), אך rand() לא מצריך.

ד. בתום הפעולה יש לסגור את הקבצים (ק.מ.).

ה. על התכנית לסיים (ק.מ.)

סוגי קריאות מערכת

ניתן לחלק את ק.מ. לחמש קבוצות עיקריות:

א. בקרת תהליכים.
ב. טיפול בקבצים
ג. טיפול בציוד
ד. אחזקת מידע
ה. תקשורת

א. בקרת תהליכים (Process Control)

1. יצירת תהליך.
2. סיום תהליך באופן תקין.
3. סיום תהליך באופן שגוי (שפיכת core, ושליחת הודעת שגיאה).
4. טעינת והרצת תכנית אחרת במקום הנוכחית.
5. שליפה\עדכון תכונות תהליך (קדימות, זמן ריצה מותר).
6. הריגת תהליך.
7. המתנה פרק זמן X.
8. המתנה לאירוע כלשהו (סיום בן).
9. שליחת איתות לתהליך אחר.
10. הרצת תכנית פקודה, פקודה.

ב. <u>טיפול בקבצים</u>
1. יצירה\מחיקה של קובץ.
2. פתיחה\סגירה של קובץ לק'\כ'.
3. ק\כ מ\על קובץ.
4. תזוזה למקום רצוי בקובץ.
5. שליפת\עדכון תכונות קובץ (chmod).
6. העתקת\העברת קובץ.
7. כנ"ל עבור מדריכים.

ג. <u>אחזקת מידע</u>
1. קריאת תאריך\שעה.
2. מספר המשתמשים, גרסת מ.ה., שטח הדיסק הפנוי...

ד. <u>תקשורת</u> לשם העברת מידע בין תהליכים המצויים באותו מחשב או במחשבים שונים
1. ברור כתובת ה- IP של המחשב.
2. פתיחת\סגירת ערוץ תקשורת.
3. קבלה\שליחה של מידע מ\על הערוץ.
4. הקצאת\שחרור זיכרון משותף.
5. יצירת\סגירת כלים אחרים להעברת הודעות.

6. <u>פסיקות</u>

כזכור: פסיקה הינה איתות למ.ה. שעליה להיכנס לפעולה שכן חל אירוע כלשהו, ויש לפנות לטפל בו (וכמובן לשם כך על המעבד להפסיק את מה שהוא עושה עתה).

הפסיקה עשויה להגיע מהחומרה, או מהמעבד עצמו (מהתכנה).

החומרה שולחת פסיקה **אסינכרונית** ע"י העברת איתות חשמלי לבקר הפסיקות. בקר הפסיקות יאגור את קוד הפסיקה עד שהמעבד יפנה לטפל בפסיקה.

בין ביצוע כל שתי פקודות עוקבות בודק המעבד האם יש פסיקה הממתינה לו, ואם כן יפנה לטפל בה.

לכל פסיקה יש מספר המזהה אותה, אך ייתכן שמספר מרכיבי ציוד ישלחו פסיקה עם אותו מספר (ואז יש לברר מי מביניהם שלח את הפסיקה).

וקטור הפסיקות: מערך ה- Interrupt Descriptor Table (IDT) מכיל לכל מספר פסיקה אפשרי את כתובת הפונ' לטיפול באותה פסיקה.

כתובת המערך, ותפקיד כל כניסה בו נקבעים ע"י החומרה. (כניסה 0#: שגיאה בחילוק, 14#: page fault, 32#: timer, 128#: ק.מ.). ה- BIOS מאתחל את המערך, ומ.ה. דורסת אתחול זה.

בפסיקה יש לטפל כמה שיותר מהר (הזזת עכבר עד 150מ"ש המשתמש לא ירגיש). בפרט בעת טיפול בפסיקה לא מבוצעת החלפת הקשר. רק נשמרים בצד האוגרים אותם דורסים, וערכם משוקם בתום הטיפול בפסיקה.

פסיקת תכנה (מכונה באינטל: חריגה) נשלחת ע"י המעבד לעצמו, כדי לאותת על אנומליה.

שלושה סוגי פסיקות תכנה:

א. <u>trap</u> פסיקה זאת תשלח ע"י תכנית המזמנת ק.מ.. התכנית תבצע פקודת מכונה ייעודית. הפקודה תביא להפעלת פונ' של מ.ה. אשר תברר מהו השירות הדרוש, ותספקו.

ב. <u>abort</u> פסיקה זאת תשלח עת התכנית ביצעה שגיאה (seg. fault, ניסיון לביצוע פקודה מיוחסת במצב משתמש) ויש להעיפה.

ג. <u>fault</u> התכנית בצעה שגיאה 'שלא בעוונותיה'. על מ.ה. לדאוג 'להסדרת העניינים', ויש לחזור על ביצוע הפקודה שכשלה.

7. <u>סיגנלים (Signals)</u>

פסיקה הינה הדרך של הציוד (בפרט המעבד) לאותת למ.ה. להתעורר, ולהיכנס לפעולה.

הדרך של (גרעין) מ.ה. לאותת לתכנית על אירוע חריג כלשהו הוא ע"י שליחת סיגנל לתכנית.

(גם תכניות עשויות לשלוח סיגנלים אלה לאלה)

<u>דוגמות</u>

א. גודל קובץ עלה על המותר.
ב. התהליך ביצע פקודה לא חוקית.
ג. המשתמש הקיש ^c או ^z.

תהליך יכול לקבל סיגנל רק עת הוא <u>רץ</u> במצב <u>משתמש</u> (אחרת הסיגנל ימתין לו). סיגנל ממתין נקרא pending signal.

בכל פעם שהתהליך מתחיל לרוץ במצב משתמש נבדק האם ממתינים לו סיגנלים.

סיגנל הנשלח לתכנית בשל תקלה בהרצת התכנית ייקרא סיגנל **סינכרוני**. סיגנל הנשלח לתכנית ע"י תכנית אחרת הינו **אסינכרוני**.

כמה דוגמות לסיגנלים:

א. SIGFPE נשלח לתכנית שניסתה לחלק באפס *בשלמים*
ב. SIGSEGV נשלח לתכנית שפנתה לכתובת שגויה.
ג. SIGTINT בקשה מהתכנית להסתיים (המשתמש הקיש ^C).
ד. SIGKILL על התכנית להסתיים מייד!
ה. SIGSTOP על התכנית לעצור (עבור דבגינג).

(man -7 signal יציג לכם את הסיגנלים השונים).

עת התכנית מבקשת לבצע פעולה שעלולה להיכשל (פתיחת קובץ, הקצאת זיכרון) לא יישלח לה סיגנל.

סיגנל נשלח עת התקלה חלה בכל מקום שהוא בתכנית, ואינה אמורה להיבדק\להיות-מנוטרת ע"י התכנית.

תהליך המקבל סיגנל בד"כ יועף, (אולי תוך שפיכת core). אולם תהליך עשוי להגדיר שהוא:
א. מתעלם מהסיגנל.
ב. תופס אותו, ומטפל בו (כפי הבנתו), ע"י הרצת פונ' ייעודית הנקראת signal handler.

מהסיגנלים: SIGKILL, SIGSTOP לא ניתן להתעלם, ולא ניתן לתפסם.

8. <u>הגנה על הזיכרון</u>

כדי להבטיח פעולה תקינה של המחשב יש להגן על שטח הזיכרון המוקצה למ.ה. מפני פגיעה ע"י תכניות המשתמשים; באופן דומה יש להגן על שטח הזיכרון המוקצה לתכנית א' מפני תכנית ב'.

בפרק #9 נתאר מכניזמים מעודנים להשיג זאת. עתה נתאר מכניזם פרימיטיבי.

נניח שלתכנית מוקצה שטח זיכרון רציף.
שני אוגרים יוקצו לתחימת שטח הזיכרון:

א. <u>אוגר בסיס</u> (base register): יכיל את כתובת הזיכרון הנמוכה ביותר הנכללת *במרחב הכתובות* של התהליך.
ב. <u>אוגר גבול</u> (limit register): יכיל את גודל שטח הזיכרון המוקצה לתכנית.

חומרת המעבד בודקת שכל כתובת אליה התכנית מתכוונת לפנות מצויה בתחום הנ"ל. אחרת נשלחת חריגה, בעקבותיה סיגנל, והתכנית מועפת.

(כמובן שהטיפול באוגרים נעשה רק ע"י פקודות מיוחסות).

ב- DOS בימיה הראשונים לא הייתה הגנה שכזו, התוצאה הייתה שתכניות המשתמשים יכלו לעשות שמות במ.ה., במידה שחייבה ביצוע reboot.

9. <u>הגנה על המעבד</u>

תקלה אחרת ממנה יש להישמר היא שתכנית המשתמש תאחז במעבד עד בלי די.

הפתרון: עיתאי (timer) אשר יישלח פסיקה אחרי פרק זמן קבוע מראש (ביוניקס: 0.01 שנ').

עת חלה פסיקה (כל פסיקה) השליטה עוברת\חוזרת למ.ה. אשר עשויה להקצות את המעבד לתכנית אחרת, או לאפשר לתכנית להמשיך לרוץ 'עוד קצת', או להעיף את התכנית.

לפני הקצאת המעבד לתכנית (זו או אחרת) תשתמש מ.ה. במעבד למשימותיה שלה: עדכון מבני הנתונים שלה (כמה זמן התכנית רצה), החלטה מי תהיה התכנית הבאה שתזכה במעבד וכולי.

כמובן שהטיפול בטיימר יעשה ע"י פקודות מיוחסות.