

Introduction



© 2022 מערכות הפעלה

Operating Systems

Lecture 1 – Introduction

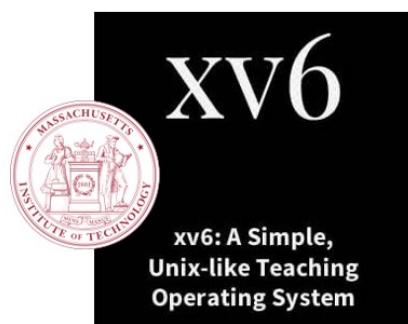
Dr. Marina Kogan-Sadetsky

נושא ראשון: רעיונות מרכזיים וabolוציה של מערכות הפעלה.

Assignments and grade structure

Course site: <https://moodle2.bgu.ac.il/>

Assignment	Subject	Weight
Programming 1	Scheduling	7.5%
Programming 2	Synchronization	7.5%
Programming 3	Memory Management	7.5%
Programming 4	Files	7.5%
Final Exam	All	70%



- Assignments and exams are mandatory
- Must pass final exam and assignments (grade ≥ 56)

2

יש 4 עבודות בקורס, כולל בשפת C, لكن יש לרענן את הזיכרון מקורסי קדם במידת הצורך.

יש צורך לעבור את המבחן בסוף כדי לעבור את הקורס.

הידע הראשי הוא במטלות ובהרצאות.

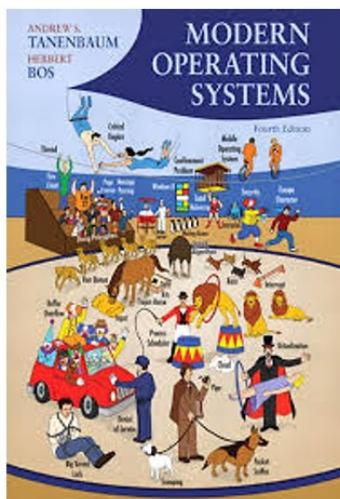
התרגולים יהיו סביב למערכת הפעלה 6Ax שנועדה להוראה אריה מערכת הפעלה אמתית לכל דבר. די נוח כי אין ל-6VX הרבה קבצי מקור ביחס לינוקס.

הערה: יש לקרוא את הקוד של 6ax לפני מטלה 1, שכן נוכל לכתוב קוד רק אחרי שנבין את רוב הקוד של המערכת.

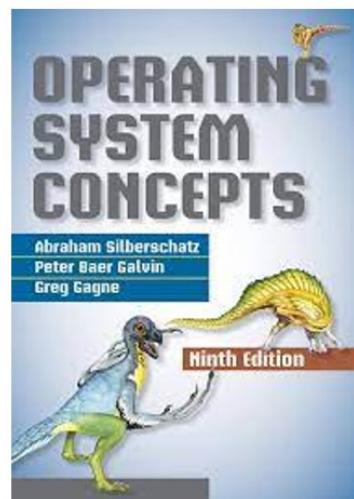
הकושי במטלות:

כאן לא רק שיש מטלות ארוכות אלא שאם צריך למשוך יוטר. לדוגמה סyncronization: למדנו מנגנון syncronization כמו אוטומטים מוניטורים וכו'. אין שימוש של זה ב-6VA ונאLUZ למשוך זאת בעבודה 2. בשביל זה נצטרך הבנה תיאוריתית של החומר.

Textbooks



A. Tanenbaum
Modern Operating Systems,
Prentice-Hall, 4th Edition, 2015



A. Silbetschatz
Operating System Concepts,
Addison Wesley, 9th Edition, 2012

מומלץ בחום להשתמש בספר משמאל, מספיק בשבייל הקורס חצי מספר זה.

הערה: הספר מצד שמאל הוא מהדורה הרביעית של הספר של טננבאום, ב-2022 יצאה מהדורה חדשה ומעודכנת של הספר.

Course Syllabus

1. Introduction <ul style="list-style-type: none">- history- concepts	6. File Systems <ul style="list-style-type: none">- Unix & Windows file systems- Distributed File System (NFS)
2. Process Management <ul style="list-style-type: none">- processes- threads- Unix implementation of Processes	7. Virtualization <ul style="list-style-type: none">- Virtual Machines- sensitive and privileged instructions- binary translation- memory virtualization
3. Scheduling algorithms	
4. Synchronization <ul style="list-style-type: none">- synchronization tools- deadlocks	
5. Memory Management <ul style="list-style-type: none">- virtual memory- page replacement algorithms- segmentation	

סילוב:

ב-2 הרצאות הראשונות נדון בהיסטוריה ורעיונות מרכזיים. לא נוכל להבין מדוע מערכות הפעלה נראות כמו שהן נראות מבלי שנלמד על ההיסטוריה.

לאחר מכן נעבור לפרק 2, שהוא ניהול תהליכיים ובפרט נלמד על המימוש ביוניקס.

בשלב השלישי נלמד על אלגוריתמי תיזמון תהליכיים, כולל בהינתן מספר כלשהו של תהליכיים, כיצד לחלק את המשאבים של המחשב בו זמן ובעזרת אсинכרוניות על פני כל התהליכים.

בשלב הרביעי נדון בסינכרונייזציה: מנגנוני הסינכרונייזציה אינם רק ברמת המעבד, אלא יש צורך גם בראם כדי לנעול את התהילה, כולל צורך לנעול את הקטע שלו בראם. כאן יש צורך בתמיכה חומרתית אך לא בהכרח תוכניתית. ברגע שהמעבד רואה את ההוראה באסמל'ו הוא יודע לפחות מבלי לערב את הקERNEL. למעשה הם אינם רק ברמת החומרה אלא דורשים תמיכה מצד סביבת הריצה וגם מצד שפת התוכנות עצמה.

דוגמה לכך היא מוניטוריים: זה פתרון ברמה תוכניתית, היא קיימת כי ג'אווה תומכת בהזה (דרך VM). כמעט ובכלתי אפשרי למשוך C++. כולל יש צורך בתמיכה של סביבת הריצה בשבייל זה.

בפרק החמישי דן בניהול זיכרון: זיכרון וירטואלי, החלפת דפים, חלוקה לסאגמנטים.

הפרק השישי דן במערכות קבצים, כולל שיטות לאחסן וארגון קבצים במחשב. אנו נדון במערכות הקבצים ב-Windows וביוניקס. כמו כן נדון במערכות קבצים אשר משתמשות ב프וטוקול תקשורת (NFS)

ווירטואלייזציה: מכונה וירטואלית.

שאלה: האם דרושה תמיכה של מערכת הפעלה על מנת להריץ אותה ב-VM?

תשובה: כן.

Introduction: outline

What is an Operating System?

Some history

OS concepts

What is an Operating System?

OS is a **constantly running program**,
that makes running of other programs possible.

1.

2.

OS has two main roles.
What are they ?



מה זה מערכת הפעלה?

בבסיס מערכת הפעלה היא תוכנית, שאינה רצתה בצורה קבועה, אשר תומכת בתהליכי הרצים בה. למעשה ל מערכת הפעלה 2 תפקידים ראשיים : שימוש כמכונה מורחבת וניהול משאבי המחשב.

השימוש כמכונה מורחבת היא בעצם התובנות מלמעלה (המשתמש) למטה (החומרה) על מערכת הפעלה והשימוש במערכת הפעלה נדרש לניהול משאבי המחשב היא הפוכה. בהרבה מקרים אנו נתבונן בנושא מסוים גם מעינו של המשתמש וגם מהענינים של החומרה, או יותר נכון, מתכני מערכת הפעלה.

What is an Operating System?

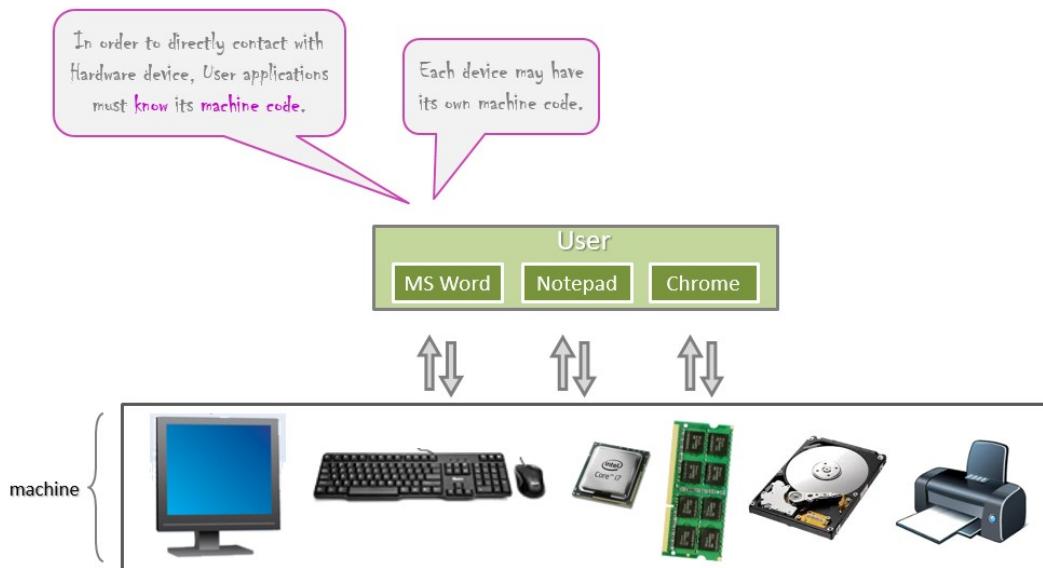
OS is a **constantly running program**,
that makes running of other programs possible.

1. An Extended Machine

2. ???

OS isolates users from being aware of
Hardware details. Indeed, OS is a
machine-independent abstraction of the
computer system.

Computer System Architecture



Operating Systems as Extended Machine: read from disk example

Read data from disk

- Read logical block address
- Translate logical block address to disk and sector numbers
- Convert sector number to: cylinder, track, sector, head
- Move disk arm to requested cylinder
- Wait for proper sector to appear
- ...

OS abstraction

return-code = read(fd, buff, nbytes)

fd – file descriptor – index to the table of process opened files

buffer to put the data from disk

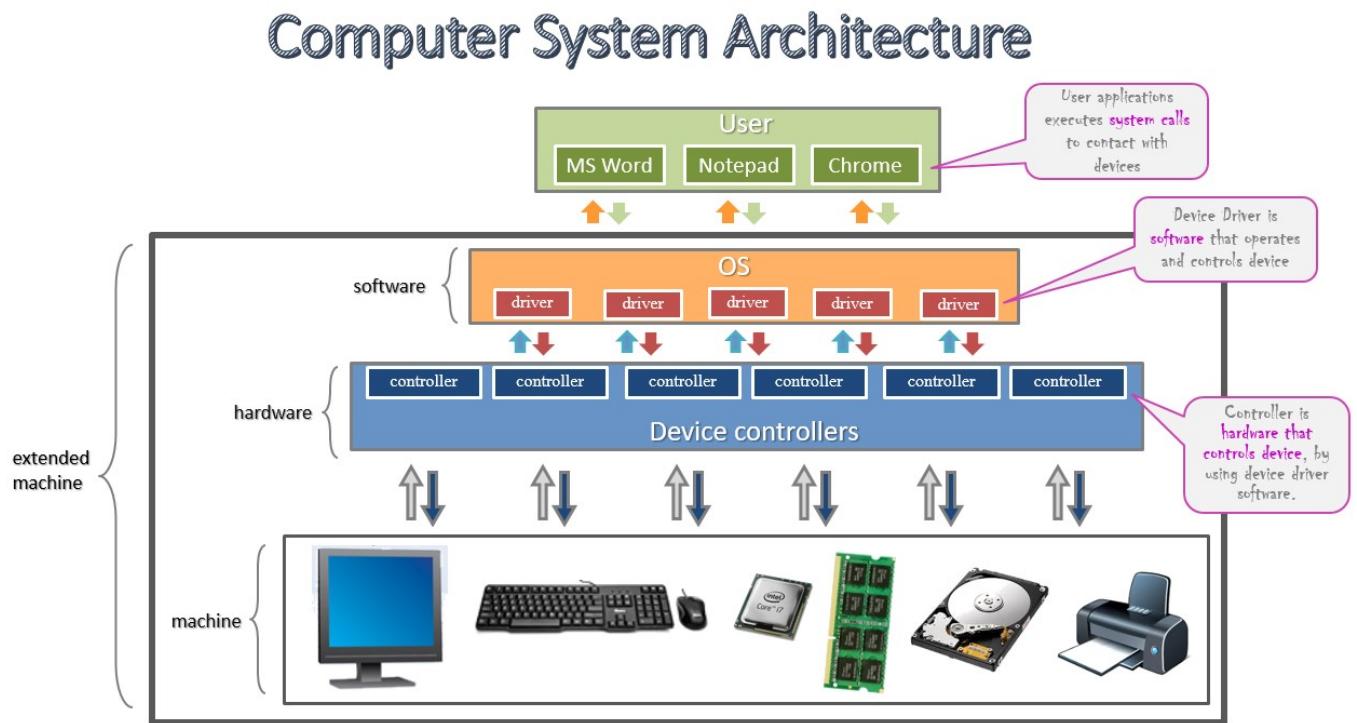
how many bytes to read

הארQUITטורה של רוב המחשבים ברמת שפת המכונה היא פרטיטיבית ומסובכת לתוכנות, שכן היא דורשת הבנה עמוקה של החומרה. לדוגמה, משתמש רוצה להריץ word, כתבו וודפדן כרום בו זמן. כל אחת מן התוכניות האלו דורשת גישה למשאים של המחשב, ולכן על מנת לאשת לחומרה, על התוכנית לדעת את הקוד בשפת המכונה שהרכיב יודע לעבוד איתה. אלו פעולות מסובכות לביצוע בשפת מכונה ולשם כך מערכת הפעלה מספקת משק נוח המאפשר לבדוק את המשתמש מהפרטים אודוט החומרה. כמובן היא מאפשרת אבטחה שאינה תלויות במכונה עליה מפעלת הפעלה רצה.

כדי להמיחס את המורכבות נתבונן בתהליך של קריית תוכן קובץ ב-word. ראשית על התוכנית בכלל להבין כיצד דיסק קשיח (באנגלית HDD) עובד. הדיסק הקשיח מורכב ממספר דיסקים וראש קרייה-כתיבה. על מנת שתוכנית תוכל לקרוא את הקובץ עליה להיות מסוגלת לדעת כיצד לקרוא כתובות של בלוקים בcone, כיצד לתרגם את הכתובות למספר דיסק וסקטורים, כיצד להמיר את מספרי הסקטורים לפרטים יותר קטנים וכך הלאה. מסובך.

למעשה העניין הנהו עוד יותר מסובך מאחר וייתכן כי ליצנים שונים יש מבנה שונה של דיסק קשיח, כמו הצד של הדיסק ממנו קוראים. זה פרט שימושה מיצן ליצן ואני לנו, כמשתמשים, את הרצון להתמודד עם בעיות אלו כאשר כל מה שאנו רוצים זה לקרוא מספר בתים מסוים. כאן מערכת הפעלה נכנסת לעניין - היא בעצם

מסתירה מהמשתמש (בדוגמה, כותב התוכנית ב-C) את הצורה שבה יש לגשת לכונן כדי לקרוא בתים מהקובץ.



ארQUITקטורת מערכת מחשב

ראשית יש להבין כי כאשר אנחנו מדברים על מכונה, הכוונה היא במעבד ובראמ. כל שאר התקנים כגון מסך, מקלדת, דיסק קשיח ומדפסת הם אינם חלק מהמכונה עצמה אלא **מחוויים התקני קלט/פלט**.

אם לא קיימת מערכת הפעלה ויש הרבה תהליכיים אשר מנסים לגשת לכמה התקנים, כל אחת מהתוכניות תctrר לדעת איך התקנים עובדים ואיך לעבוד במקביל. כאשר אנחנו רוצים לכתוב קוד שמטרתו להדפיס משהו על המסך אנחנו לא חושבים על איך המסךעובד, אבל התוכנית חייבת לדעת זאת כדי לדעת איך להדפיס את הפיקסלים על המסך. זה מסורבל.

לשם כך קיימת פיסת תוכנה אשר נקראת **דרייבר** (בעברית: מנהל התקן): זהו חלק מהמערכת הפעלה המספק ממשק, בדרך כלל של היצרנית, כדי לעבוד עם התקן. הדרייברים הללו יוצרים תקשורת עם הבקרים: מעין מעבדים (הלהקה למעשה אוגרים) של המכשירים שיודעים להמיר את הפקודה מממשק ההפעלה להתקנים.

נזכיר לדוגמה עם המשתמש שמרי'ז word, כתבן ודףן. כל אחד מהן צריך גישה למקלדת, לעכבר ולמסך. לשם כך הן מ Ritchot פקודות מערכת אשר פונות למערכת הפעלה. מערכת הפעלה נעזרת בדרייבר של התקן אשר פונה בצורה ישירה לבקרים של התקן.

מסקנה: כאשר קוד של דרייבר רוצה לטפל בפנייה אליו. הפניה היא מלא מידע בריגיסטרים. לדוגמה, נוטף לא יכול לפנות לSO באופן עצמאי.

איך מעבד יודע להבדיל ביניהם? בעזרת דגל, רגיסטר. כאשר CPU עובר מ מצב משתמש למצב קרNEL הוא משתמש בחתיכת קוד מיוחדת שנקראת TRAP. כאשר מעבד עובר מרחב המשתמש למרחב הקרNEL הוא משנה את הערך שלו.

What is an Operating System?

OS is a **constantly running program**, that makes running of other programs possible.

1. An Extended Machine

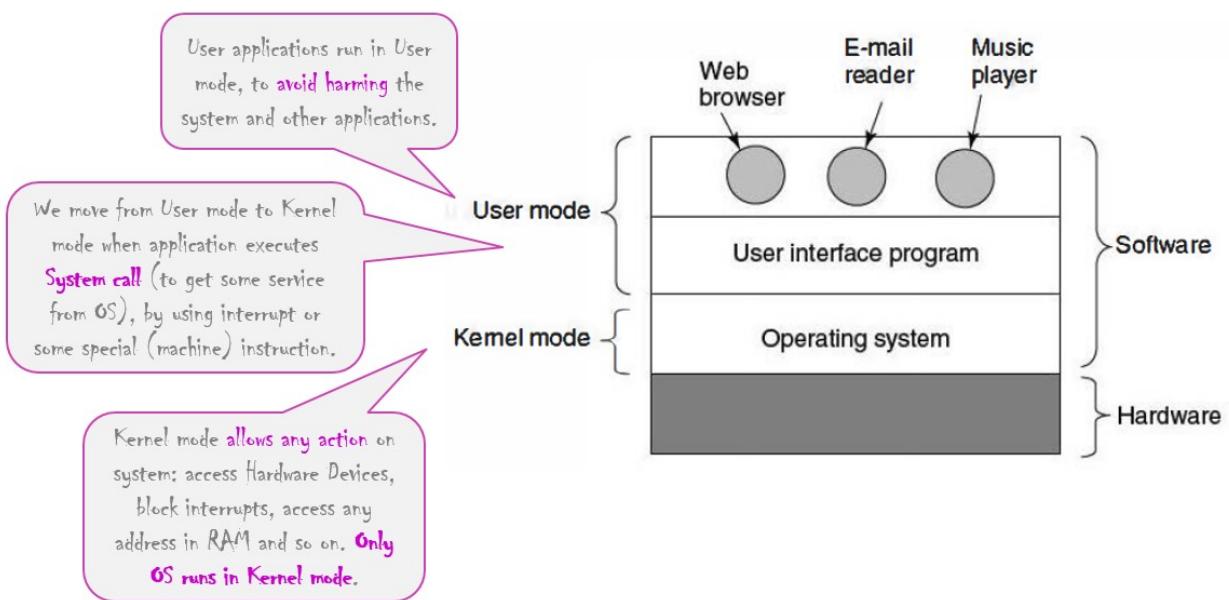
OS allocates resources (**CPU time, RAM, devices, etc.**) for all the processes that need these resources.

2. A Resource manager

OS scheduler function allocates CPU time for all the running processes

RAM allocation between processes is **dynamic** – different processes need different amount of RAM, at different times.

Layered Hardware-Software Machine Model



2. ניהול משאבי

בגישה זו, מערכת הפעלה מנהלת את כל הרכיבים של המערכת המורכבת, תוך כדי שהיא מספקת בצורה מסודרת ו邏輯ית הוצאות משאים של המעבדים, הזיכרונות והתקני הקלט-פלט בין כל התוכניות אשר רצחות אותן. בהינתן שמספר תהליכיים רצים בו זמנית, הרו' שכל תהליך ירצה לקבל זמן מעבד. עם זאת מספר המעבדים (או ליבות) הוא סופי ולכן יש מגבלת למספר התהליכיים אשר מסוגלים לרוץ בו זמנית.

במילים אחרות מערכת הפעלה צריכה לזמן בין התהליכים השונים. נניח כי 2 תהליכים מבצעים בו זמנית **MALLOC** ו-**MDPRINT** למסך. מי מדפיס ראשון, מי מקבל איזה תא זיכרון? צריך בשביל זה דרך לזמן.

בנוגע לזמן מעבד: אנו נראה מספר נקודות בהן מערכת הפעלה יכולה לתזמון תהליכיים כאשרណון בתזמון תהליכיים, אך לצורך הדיון על מנת לפתור זאת, נוכל להבחן כי תהליכיים יכולים לדרוש מידע, בין אם זה הודעה מתהליך אחר או מידע מהdisk. זהו מצב אפשרי שבו תהליך ירצה להמתין עד שמיידע זה יתקבל. בזמן זהה שבו תהליך ממתיין, מערכת הפעלה יכולה להתעורר ולהעניק את הזמן הריצה ל手続きים אחרים.

מערכת הפעלה מקצה משאבי לכל手続きים שזוקקים להם כאשר היא מקצת באופן דינמי את הראמ (手続きים שונים דורשים כמו זיכרון שונה ובזמנים שונים). יש למערכת הפעלה פונקציה, שנקראת OS Scheduler, שמקצת ל手続きים השונים שריצים זמן CPU.

ברוב המחשבים יש 2 מצבים אשר משחיקות תפקיד מכרייע בצורה שבה מערכת הפעלה עובדת. **מצב משתמש:** מצב בו לתוכנית אין הרשות מיוחדת ואינה יכולה לגשת בצורה ישירה למערכת. בצורה זו נמנעת פגיעה במערכת באפליקציות אחרות.

ברגע שתוכנית מעוניינת לגשת לאחד מהתקני קלט/פלט (כלומר ברגע שהוא מבצעת system call) במערכת אז משתמש ב-interrupt או בהוראות מכונה מיוחדות, היא חיבת לעבור דרך ליבת (kernel) מערכת הפעלה kernel mode.

מרחב הליבה (הkernel) מאפשר כל פעולה במערכת, כולל גישה ל התקני חומרה, כתובות בראם וכך הלאה. במצב זה רק מערכת הפעלה רצה (כלומר רק במקרה זה התוכנית שהיא מערכת הפעלה רצה)

Introduction: outline

What is an Operating System?

Some history

OS concepts

ההיסטוריה

הבנת ההיסטוריה של מערכות הפעלה היא קריטית על מנת לדעת איך מערכת הפעלה עובדת.

Evolution of Operating Systems

<https://www.livescience.com/20718-computer-history.html>

Operating Systems was **evolutionary developed**.

In order to survive, OS must be:

- **useful** (i.e., meets Hardware and User needs)
- **safe** (i.e., keeps user applications and computer resources safe)
- **efficient** (i.e., use fast efficient algorithms)
- **user friendly** (i.e., easy to use)

כדי שמערכת הפעלה תשרר רלוונטיות עליה להיות שימושית (עומדת בדרישות החומרתיות של המשתמש), בטיחותית (כלומר שומרת על התוכניות ומשאבי המחשב בטוחים), יעילה (כלומר משתמשת באלגוריתמים מהירים (יעילים) וכן ידידותית למשתמש).

First generation 1945 - 1955



Electronic Numerical Integrator And Computer (ENIAC)

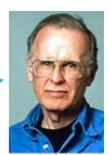
First computers were used for weapon simulations (research)

no OS, everything was done manually by operators - input is inserted manually by configuring lamps, output (registers and RAM content) is read manually by looking on computer's lamps state after program execution

no programming languages



1952: IBM researcher Nathaniel Rochester wrote the first symbolic Assembler for IBM 701 machine, which allowed programs to be written in short, readable commands.



1957: IBM team led by John Backus developed first high-level programming language called FORTRAN (FORmula TRANslation, because it was designed to allow easy translation of math formulas into code).



tens thousands lamps
➤ very expensive
➤ high-volume
➤ lots of heat
➤ lamps burned out recently
➤ non-reliable technology

הדור הראשון - שפורנות ריק: 1945-1955

למעשה עד פרוץ מלחמת העולם השנייה, לא הייתה התפתחות ממשית במבנה מחשבים דיגיטליים. מלחמת העולם השנייה האיצה את התהילך. כאשר המחשבים הראשונים שומשו עבור סימולציות נשקים (כלומר מחקר).

מחשבים אלו היו נטולי מערכת הפעלה כאשר בני האדם היו הולכה למעשה מערכת הפעלה של המחשב. כך בתמונה במאצע ניתן לראות מישחו שלמעשה מהוות מעבד אשר עובד עם הרגיסטרים של המעבד. הקלטים של המכוונה הוכנסו بصورة ידנית על ידי האדרת הנוריות. כמו כן הפלט של המכוונת היה למעשה מצב הנוריות לאחר ריצת התוכנית.

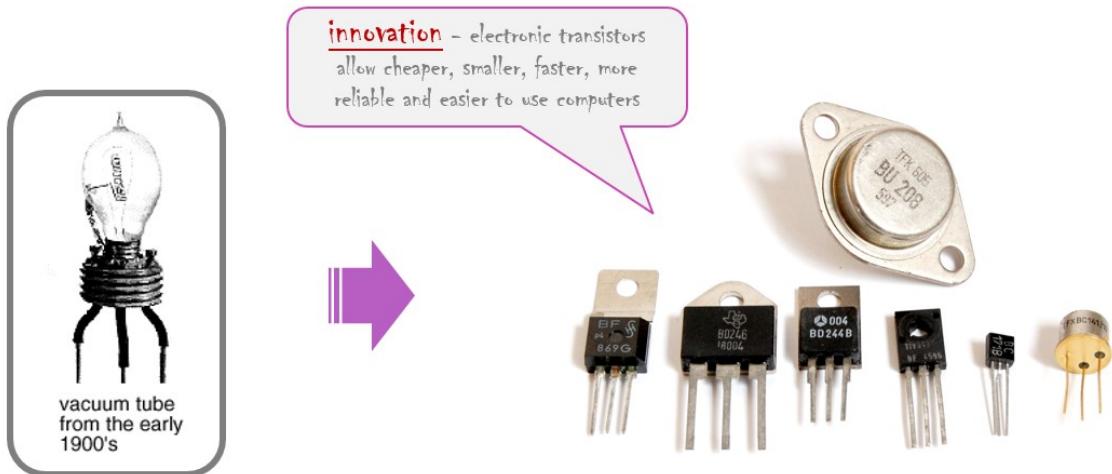
במילים אחרות: לא היו שפות תכנות. הוא לא היה יכול לזרוק הרבה חום ויהי יקר לתחזוק אותו (שכן הנוריות לא מחזיקות מעמד) היה שימוש במכוונה הזאת לפחות עשור.

שאלה: ניקח מחשבון: אנו יודעים כי מחשב אינו מחשבון אבל הוא עושה חישובים שגם מחשב יודע לעשות. אז למה מחשבון הוא לא מחשב?

תשובה: כי אי אפשר לתת לו תוכנית שיוכל להריץ.

ב-1952 נתניאל רוצ'סטר כתב את האסמבילר הראשון. ב-1957 ג'ון בוקס פיתח את שפת התוכנות העילית הראשונה.

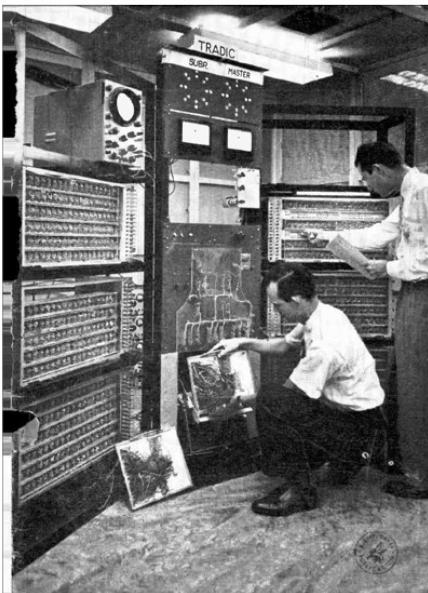
Second generation 1955 - 1965



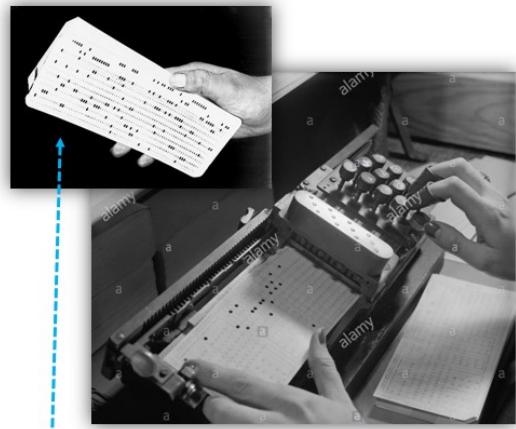
הדור השני (1955-1965): עיבוד אcoutות וטרנזיסטורים

המצאת הטרנזיסטור באמצעות שנות החמשים שינתה את התמונה. מחשבים נהיו אמינים מספיק בשבייל ליצור המוני ולמכירות עבור לקוחות. היפוי מחשבים אלו הייתה כי הם היו מספיק אמינים כדי שיוכלו להמשיך לתפקיד מספיק זמן על מנת לבצע מספיק עבודה. לראשונה יש הבדל ברור בין מעכבים, בונים, מפעלים, מתכנתים ואנשי תחזקה.

Second generation 1955 - 1965



1955: TRADIC computer, Bell Labs



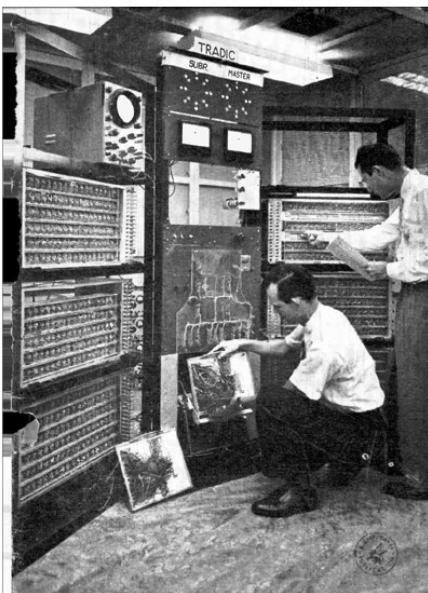
inventor of punch card: Herman Hollerith, 1861

כעת רק חברות גדולות, ארגונים ממשלתיים או אוניברסיטאות יכולו למכור מחשבים אלו, שמחירות עמד על מיליון Dolars.

כעת כוח האדם שעבד במחשבים דאג רק לקלט/פלט.

כדי להריץ "עובדת" (job), קלומר תוכנית או סדרה של תוכניות, המתכונת היה רושם אותה על דף (בשפה FORTRAN או אסמבילר) ואז מנוקב אותה על גבי כרטיסיות ניקוב. במילימ'ם אחרות, התוכניות הונרכו כרטיסיות ניקוב. כמו כן, תוצאת ריצת התוכניות הייתה בעזרת הדפסה. הkartesisיות האלו הומצאו עוד לפני 1861. במיוחד עבור אלגוריתמי מיון.

Second generation 1955 - 1965



1955: TRADIC computer, Bell Labs

bottleneck: input output devices speed (for example, cards reader maximum speed is ~1000 cards per minute, printer maximum speed is ~500 lines per minute), so CPU usage was pure.

solution: usage of magnetic tapes. Input was saved on special separate machine on tape, and then used on computer. Reading from tape was much faster than reading from cards. After the task was run, its output was saved on tape, and then was printed offline. So computer may run next task while some other machine (not computer) prints output of the previous task.



בעיה: להתקני קלט/פלט יש צוואר בקבוק.

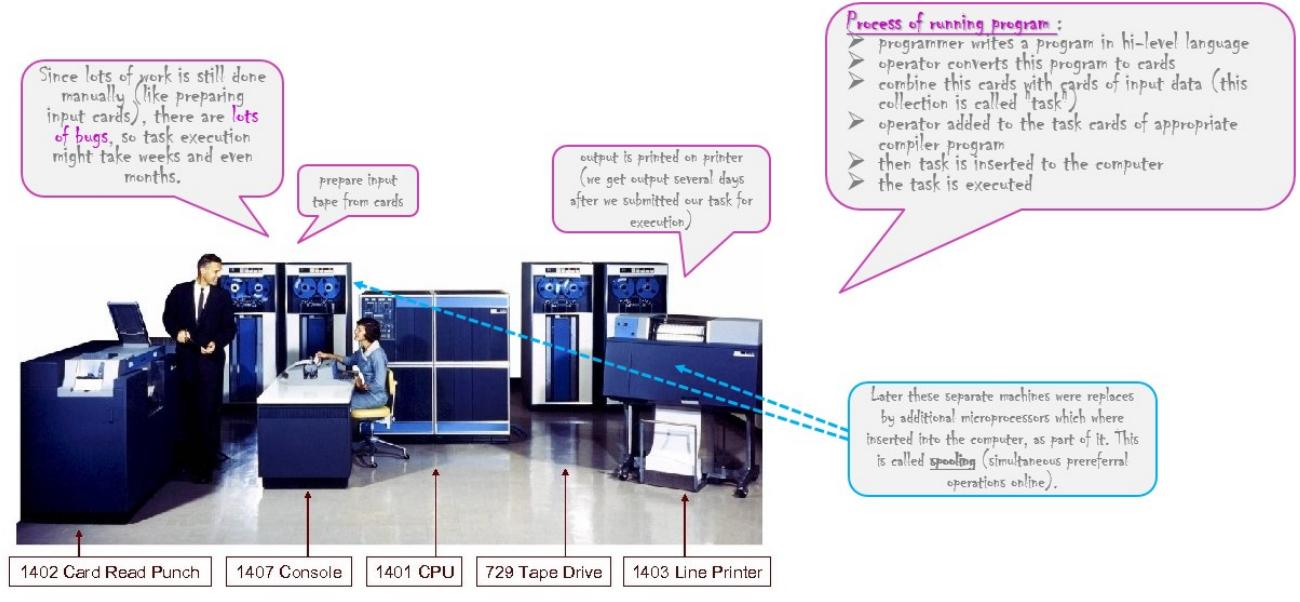
המתכונת צריכה להכין את הcartesisים, תוך כדי שהוא יכול לטיעות, להעביר את הcartesisים לאחד המפעלים, אשר

היה לוקח את פלט המכונה ומעביר אותה לחדר פלט שם המתכונת יכול לאסוף בשלב מאוחר יותר. למעשה רוב הזמן החישוב היה מבוצע בזמן שהמפעלים התרוכזו בחדר. זאת, יחד עם ההגבלה של 1000 קרטייסים (כ-500 שורות קוד) לכל יחידת זמן והעלות הגבוהה של הציוד היו מגבלת.

פתרון: שימוש במערכת אצoot בעזרת סרט מגנט. הרעיון היה לאסוף מחסנית מלאה בעבודות ב-"חדר קלט" ואז קרייתם מתוך סרט מגנט על ידי מכונות קטנות שאינן יקרות. הדבר אפשר למפעיל להריץ את התוכנית, לשמר את הפולט לתוך סרט ואז להדפיס אותו אחר כך. בצורה זו המחשב יכול להריץ את התוכנית הבאה תוך כדי שימוש אחרת מדפסה את הפולט של המשימה הקודמת.

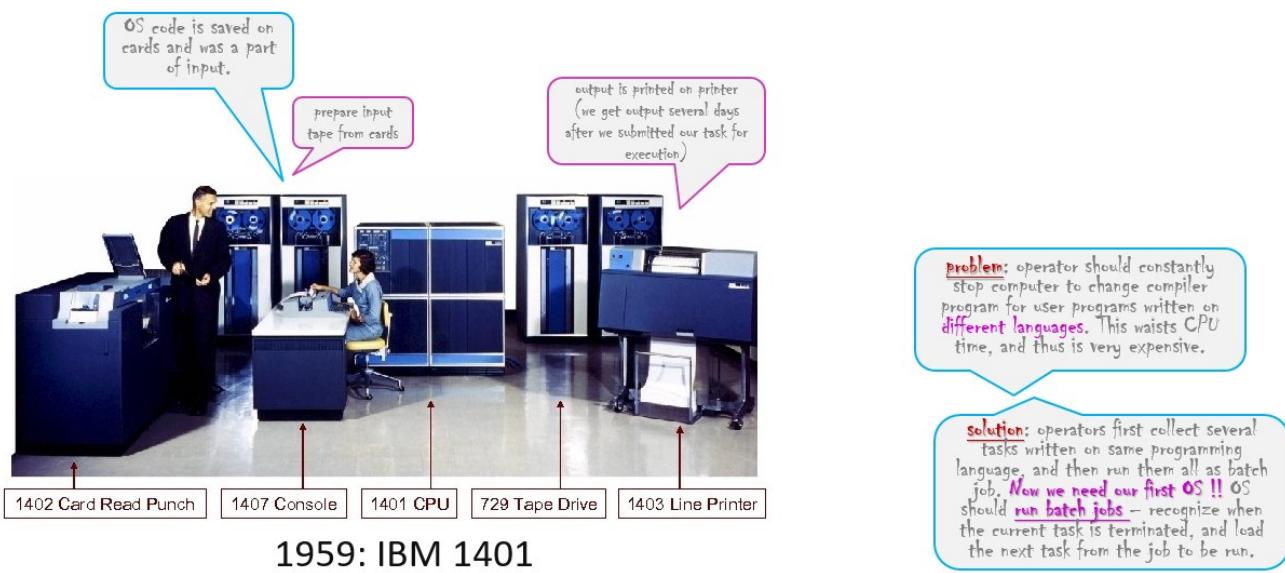
זהו למעשה הבסיס לריבוי תוכניות.

Second generation 1955 - 1965



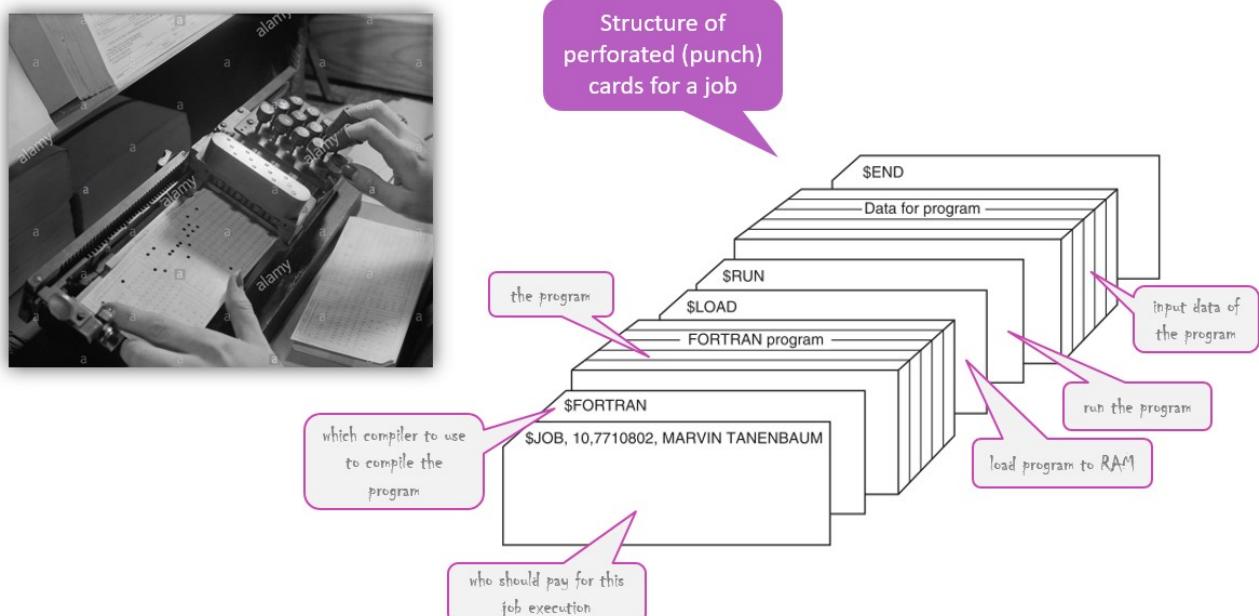
המכונה IBM1401 היא דוגמה ראשונה למחשב כזה. הוא היה טוב בקריאה לקרטיזיות, העתקת סרטים והדפסת פלט אך לא טובה מספיק בחישובים נומריים.

Second generation 1955 - 1965



עכשו צריך לדעת איך לנוהל מספר תהליכיים. יותר מזה אנחנו נctrיך שזה יהיה בرمת תוכנה. זה בדיק האב הקדמון של מערכת הפעלה - תוכנית מיוחדת אשר מ Ritchie את התוכנית הראשונה בסרט, מ Ritchie אותה וועברת לתהליין הבא.

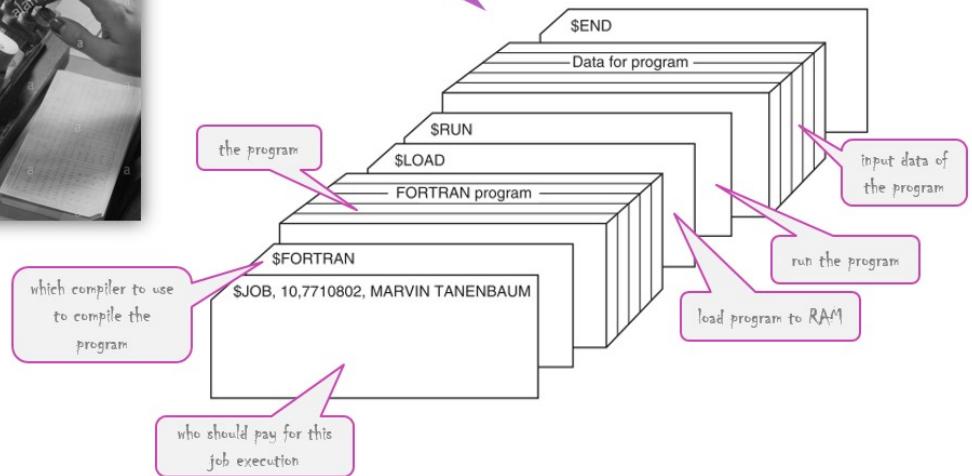
Second generation 1955 - 1965



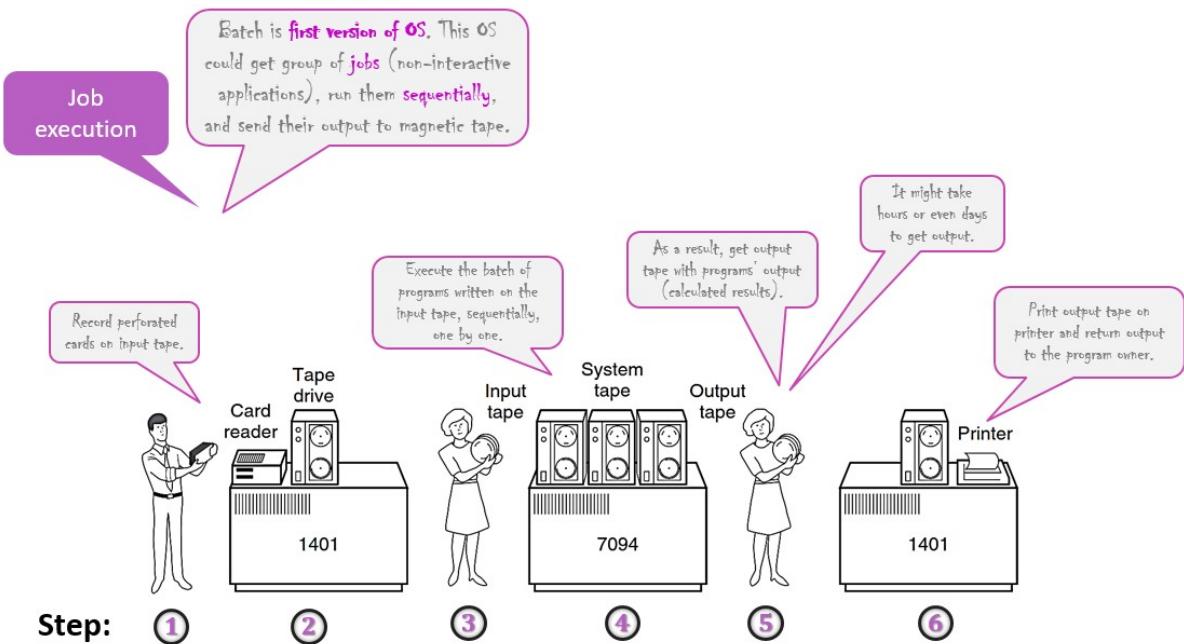
Second generation 1955 - 1965



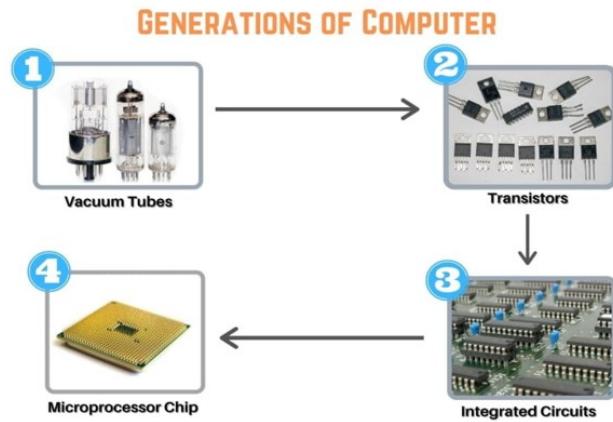
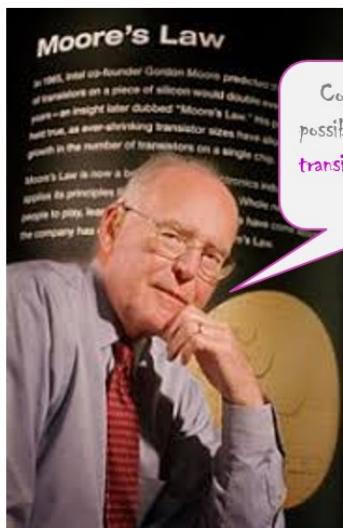
Structure of
perforated (punch)
cards for a job



Second generation 1955 - 1965

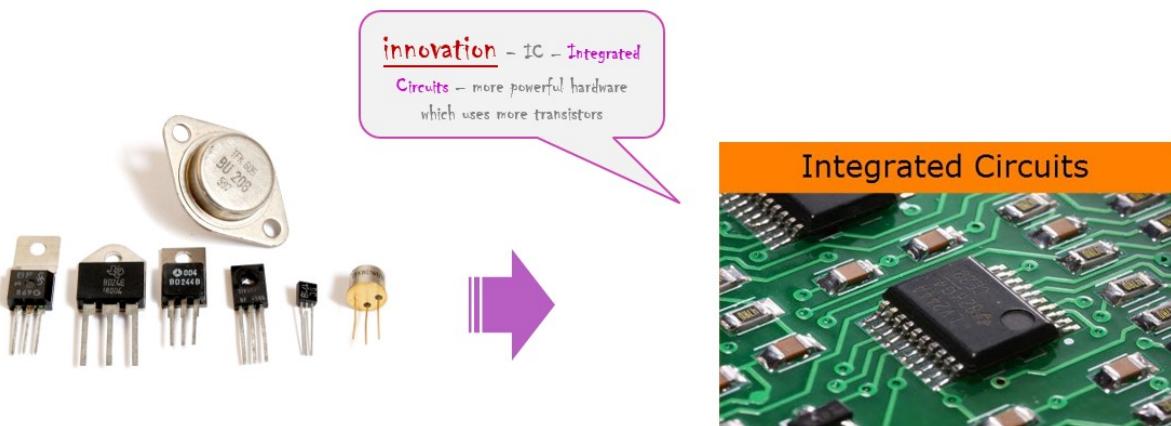


Second generation 1955 - 1965



חוק מור - בכל דור ניתן להכפיל את מספר הטרנזיסטורים באותו שטח עבור אותו מחיר.

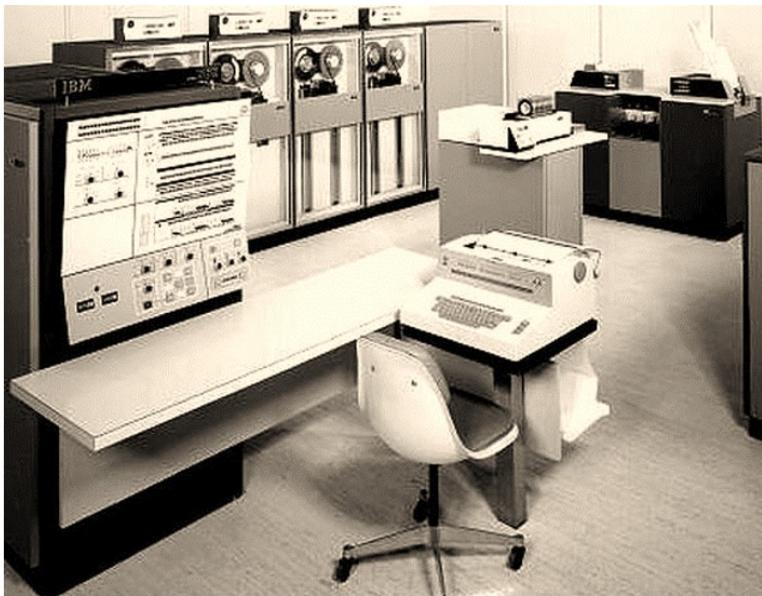
Third generation 1965 - 1980



דור 3 (1965-1980): מעגלים משלבים וריבוי שפות תכונות

החלפנו את הטרנזיסטורים במעגלים משלבים.

Third generation 1965 - 1980



1965: third generation IBM-360

Third generation 1965 - 1980



problem: tape is sequential, and thus reading and executing job tasks from tape could be only in the order they are recorded on the tape.



cutת הבעה היא בסרט המגנטי. נניח שיש לנו מספר תהליכיים שורוצים להרץ במקביל. הסרט המגנטי לא מאפשר זאת כי הוא כלי סידרתי.
עוד סיבה: נניח ותהליך משתמש ב-IO. הוא הולכת למעשה משחרר את המעבד לבצע משוח אחר. לא ניתן לעשות זאת בעזרת סרט

Third generation 1965 - 1980

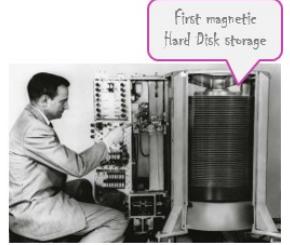


Now operator can choose in which order the tasks written on disk would be executed. Tasks may be chosen by their **priority** (who is the owner of the task, or/and importance of the task).

Scheduling tasks becomes a part of OS.

problem: tape is sequential, and thus reading and executing job tasks from tape could be only in the order they are recorded on the tape.

solution: magnetic disks, which allow random access.



First magnetic Hard Disk storage

למעשה הדיסקט הוא זה שהחליף את הסרט. זהו האב הקדמון של ה-CD. הדיסקט מאפשר גישה אקראית, ולכן בעזרתו ניתן לבצע תהליכים במקביל.

Third generation 1965 - 1980



problem: when read input or print output, CPU is idle (low CPU usage)

solution: **multi-programming**. When the current program executes IO, CPU is transmitted to another (READY) program. For this, we need to keep several programs simultaneously in RAM.



New features of OS and Hardware are required :

- **scheduling** – which programs should better run together (in perspective of RAM sharing, CPU sharing, IO sharing).
- **security** – programs should be isolated one of another, to avoid damaging of data and code of each other, and to avoid reading data and code of each other (confidentiality). Hardware solution is used to solve this issue (we know it as "Segmentation fault" or "Segmentation Violation (SEGV)" run time error).
- **CPU context switch** from one program to another – we should save the program state (registers values, etc.) to be able to get back to its execution
- When some process finishes its (current) IO, we need to let this process to continue its execution. For this, we need (**hardware**) **interrupts** system and its management.
- **privileged instructions** – let OS only to manage IO devices, CPU time, etc. in order to avoid conflicts between simultaneously running programs. This leads to **kernel (privileged) mode** and **user (non-privileged) mode** terms.
- But if user program cannot use IO, how it could get data from input, and output calculation results ? This leads to **system calls**.
- Although programs should be isolated one of another, sometimes we would like our programs to cooperate. For this, we need **cooperating mechanism** - message passing (piping, shared memory, etc) and **synchronization mechanism** between processes to solve race conditions.

We can see that only after idea of multi-programming OS becomes **really functional** and **really useful**.

כעת מערכת הפעלה והחומרה נדרשות לתמוך במנגנוןים הבאים:
זמן: איזה תהליכים כדאי להריץ ביחד.

אבלחה: בידוד כל התוכניות אחת מן השניה כדי למנוע פגיעה במידע וב קוד של כל אחד מן התהליכים. לשם כך יש צורך בתמיכה חומרתית (מה שאנו מכירים בימינו כ-"Segmentation fault" ("Segmentation fault")
ביצוע context switch מתוכנית אחת לאחרת - מערכת הפעלה תשמר את מצב התוכנית כדי לחזור להריץ אותה בשלב מאוחר יותר.

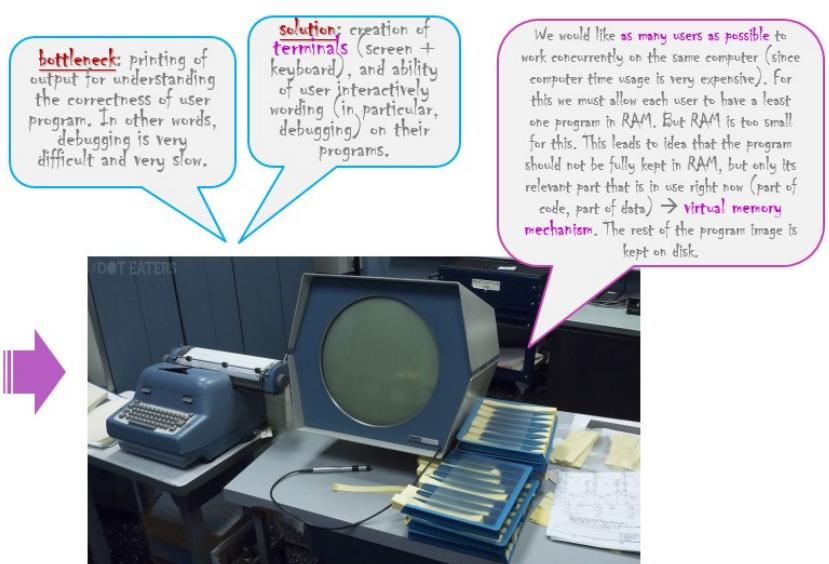
כאשר מספר תוכניות סימנו לעבוד עם התקני IO, יש לתת לתוכנית להמשיך בדרכתה. ככלומר יש צורך בתמיכה במנגנון פסיקות (interrupts) וניהול.

הוראות עם פרוילגיות - רק למערכת הפעלה יש את הזכות לניהול התקני IO, זמן CPU וכו' על מנת למנוע התנגדויות בין תהליכי הרצים במקביל. דבר שהוא לkernel mode ול-user mode.

כמו כן מערכת הפעלה אמורה לתרום במצב שבו תוכנית אינה יכולה להשתמש ב-IO - כיצד התוכנית יכולה לקבל מידע מהקלט ולהחזיר תוצאות חישוב? לשם כך מערכת הפעלה חייבת לתרום בקריאות למערכת.

לבסוף יש צורך במכנים שיתוף פעולה, ככלומר יכולת העברת הودעות בין תהליכי ומנגנוני סינכרוניזציה בין תהליכי כדי למנוע מרוץ תהליכים.

Third generation 1965 - 1980



New features of OS are required :

- **time quantum** – for how long to run one program, till we switch to run another (READY) program.
This allows users avoid long delays when they work on their programs.
- we need to save users code to some place while they work on it – creation of **file system**.

צוואר בקבוק: לא ניתן לדבג. כי צריך להדפיס כדי לראות.

פתרון: מסופים (מקלחת ומסך) והיכולת למשתמש בצע אינטראקטיבי עם התוכנית.

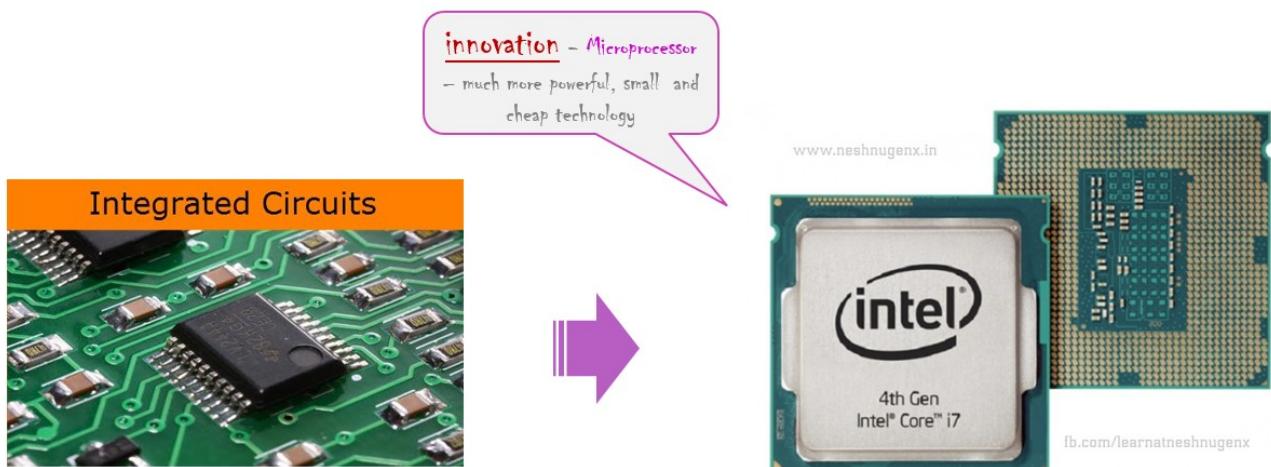
cut the computer needs to change to new features:

ראשית יש לתרום ב-time quantum, ככלומר הזמן שמשתמש מקבל כדי להריץ תהליך.

cut the matzmon לא מסיים את ריצת התוכנית, הוא יכול לעקב ריצת תוכנית בזמן שיש קרייאות IO. cut אפשר לתת מספר משתמשים להריץ תוכניות בคร שהוא מאפשר לעזור לזמן קצר תהליך כדי להריץ תוכנית אחרת.

cut אנחנו צריכים לשמור קוד של המשתמש במקום כלשהו - מערכת קבצים

Fourth generation 1980 - present



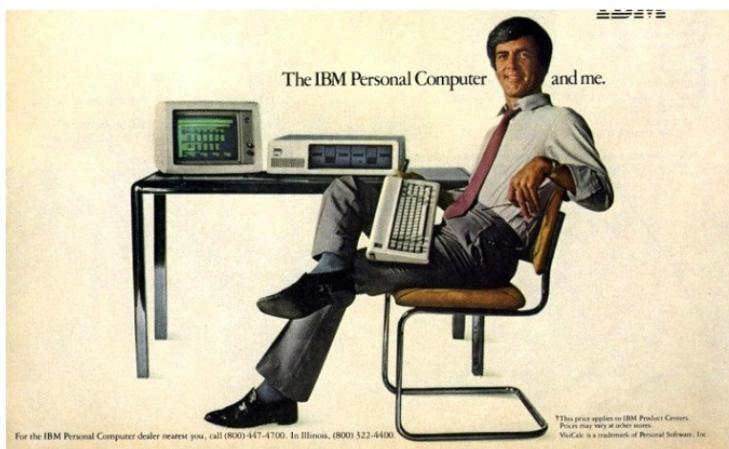
הדור הרביעי: המחשב האישי (1980-היום)

הפיתוח של מעגלי LSI, כלומר מעגלים המכילים אלפי טרנזיסטורים לסנטימטר מרובע של סיליקון הובילו לתחילה של עידן המחשב האישי. מחשבים אלו, ששםם המקורי היה מיקרו מחשבים, היו כה נגישים שלראשונה גם אנשים פרטיים יכלו לרכוש מחשב ולא רק ארגונים גדולים וחברות.

Fourth generation 1980 - present



Fourth generation 1980 - present



Initially, PC leads to extreme degradation of Operating Systems. The reason is the PC is used by **single user**, so there is **no need of multi-programming**, and thus there is no need in security features, resources sharing, and so on.

This is changed when PC becomes strong enough to serve as a **station** (server like) for **several users**, where each user has its own terminal (screen and keyboard). All excluded OS features comes back.

New features of OS are required :

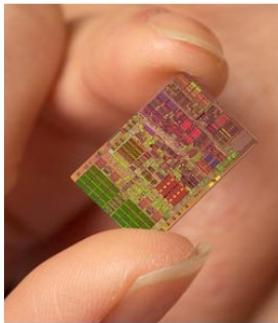
- Since PC is available (i.e., cheap enough) for many people, not only programmers want to use it, but also people that do not know programming languages and shell commands. OS needs a new feature of being **user-friendly**.
- At the same time, first networks are created. So OS needs new feature - ability to be **network OS**, i.e., use remote file system, remote message passing, and so on.

בהתחלת המחשב האישי הוביל לכך שמערכות הפעלה היו נוחות יותר. מאחר והמחשב האישי מיועד למשתמש אחד לא היה צריך בריבוי תהליכיים ולא היה צריך ביכולות אבטחה, שיתוף משאבים וכו'. הדבר השתנה ברגע שהמחשב האישי הפך להיות די חזק כדי לשמש מעין שירות למספר משתמשים כך שלכל משתמש יש את הטרמינל (מסך ומקלדת) משלו.

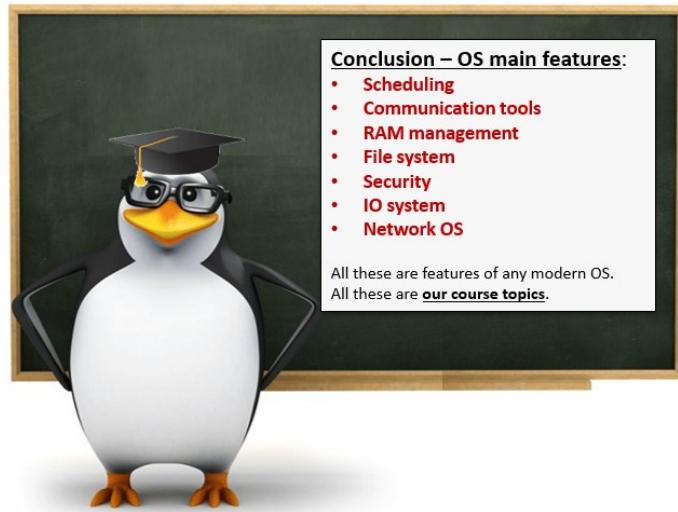
כעת מערכת הפעלה הייתה צריכה לתמוך ב-2 דברים חדשים:
ראשית על מערכת הפעלה להיות ידידותית למשתמש, מאחר וכעת המשתמשים של המחשב הם אינם רק מתכנתים אלא גם אנשים שאינם יודעים שפות תכנות ופקודות מסווג.
שנית, ריבוי המשתמשים הוביל ליצור הרשותות הראשונות. لكن מערכת הפעלה נדרשה להיות גם OS network, יכולת העברת הודעות מרוחק וכן הלאה.

Fourth generation 1980 – 2005 (?)

Some scientists claim that the **fourth generation ends in 2005**, when we realized that hardware improvement cannot lead to serious progress, since hardware produces lots of heat and it becomes impossible to chill it out. So, computers start using **multi-core processors** with shared caches. This leads to multi-thread programming, which uses multi-cores for **distributed parallel calculations**.



2005: first multi-core processor (IBM)



בשלב מסויים הבינוuai אפשר לדחוס עוד ואז עברו לمعالדים מרובי ליבות. למעשה יש אשר יגידו כי הדור הרביעי הסתיים ב-2005 כאשר המחשבים האישיים התחלו להשתמש בمعالדים מרובי ליבות עם מטמוני משותפים. הדבר הוליד לפיתוח מרובה תדרים אשר משתמשים במספר ליבות בשבייל חישובים מקבילים.

cutת מערכת הפעלה חייבת לתמוך בדברים המרכזיים הבאים:
תזמן, כל תקשורת, ניהול זיכרון ה-RAM, מערכות קבצים, אבטחה, מערכת קלט-פלט ויכולת עבודה בראשת.

Operating Systems timeline

עד עכשוו דנו בעקרונות של מערכות הפעלה אבל לא מערכות הפעלה עצמן (עם השמות שלהם)

First Operating System - 1955

GM-NAA I/O was the first batch jobs operating system for IBM 704 computer. It was created in 1956 by Robert L. Patrick and Owen Mock.

The main function of GM-NAA I/O was to automatically execute a new program once the one that was being executed had finished (batch processing).

Early mainframe computer OSs

- GM OS & GM-NAA I/O (1955)
- BESYS (1957)
- UMES (1958)
- SOS (1959)
- IBSYS (1960)
- CTSS (1961)



1956: IBM 704 (vacuum tube) computer with first (batch job) Operating System

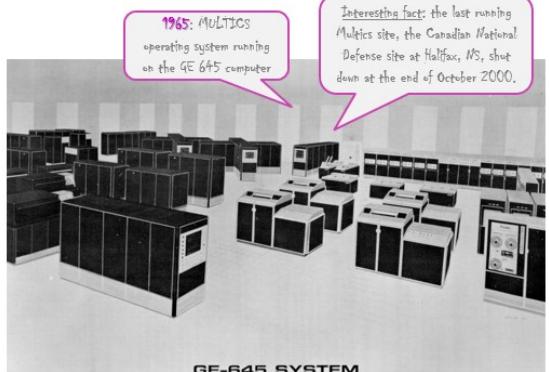
נתחיל ב-1955. מערכת הפעלה הראשונה טיפלה בעבודות על אוצרות. יכולת שלה הייתה להריץ תוכניות חדשות כאשר אלו שכבר רצויו סימנו. בשלב זה לא היו סטנדרטים, כלומר קו מנהה יחיד והמציאו המונע מערכות הפעלה. במצגת ניתן לראות רשימה של כל הפופולרים באותה תקופה. אף אחד מהם לא שרד.

MULTICS – (Multiplexed Information and Computing Service)

1965: Multics OS provides interactive access to many remote terminal users simultaneously, in a manner similar to MIT's CTSS system.

Notable Features of Multics:

- Segmented memory
- Virtual memory
- High-level language implementation
- Shared memory multiprocessor
- Multi-language support
- Relational database
- Security
- On-line reconfiguration
- Software engineering



GE-645 SYSTEM



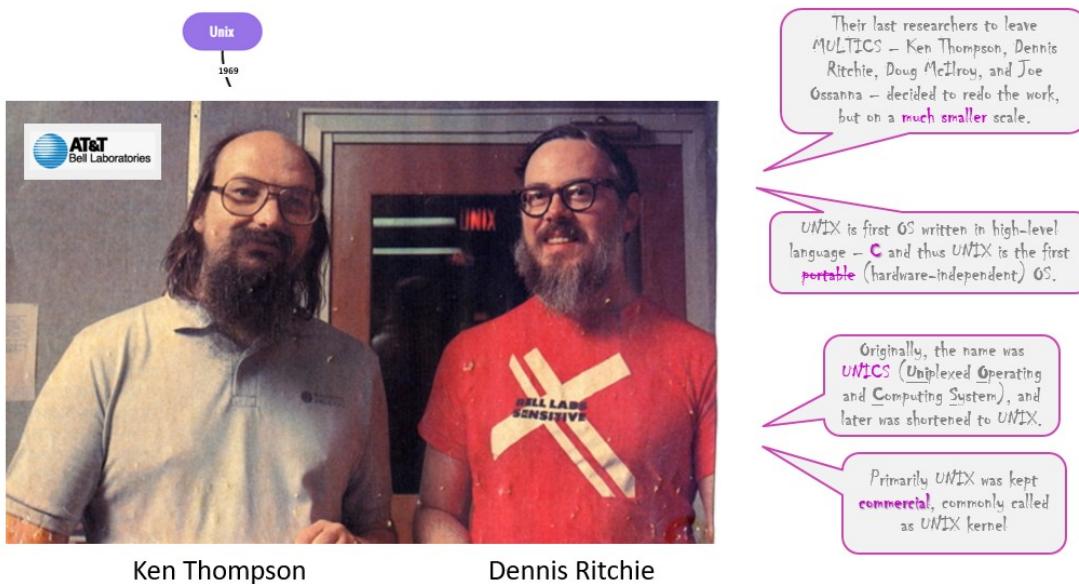
CTSS OS = Multics OS

הראשון לשימוש לב הרבה היא-multics שהייתה מקור השראה לIONIKS. היא נוצרה ב-1965 וסיפקה גישה אינטראקטיבית למספר משתמשים מסווג בצורה מקבילית.

היא הציאה לראשונה מיפוי זיכרון, זיכרון וירטואלי, מימוש של שפת תכנות עילית, מעבד מרובה ליביות עם זיכרון משותף, תמייה במספר שפות ולא רק אחת, בסיסי נתונים יחסיים, אבטחה, **אפשרות להרחיב ולהחליף רכיבים של**

פיתוח המערכת החל והיא נבנתה בקצב איטי למד", כיוון שהפתחים ניסו לדוחס הכל לשם. כתוצאה לכך הייתה at&t גנבו את הקוד והמציאו את יוניקס

Evolution of Operating Systems



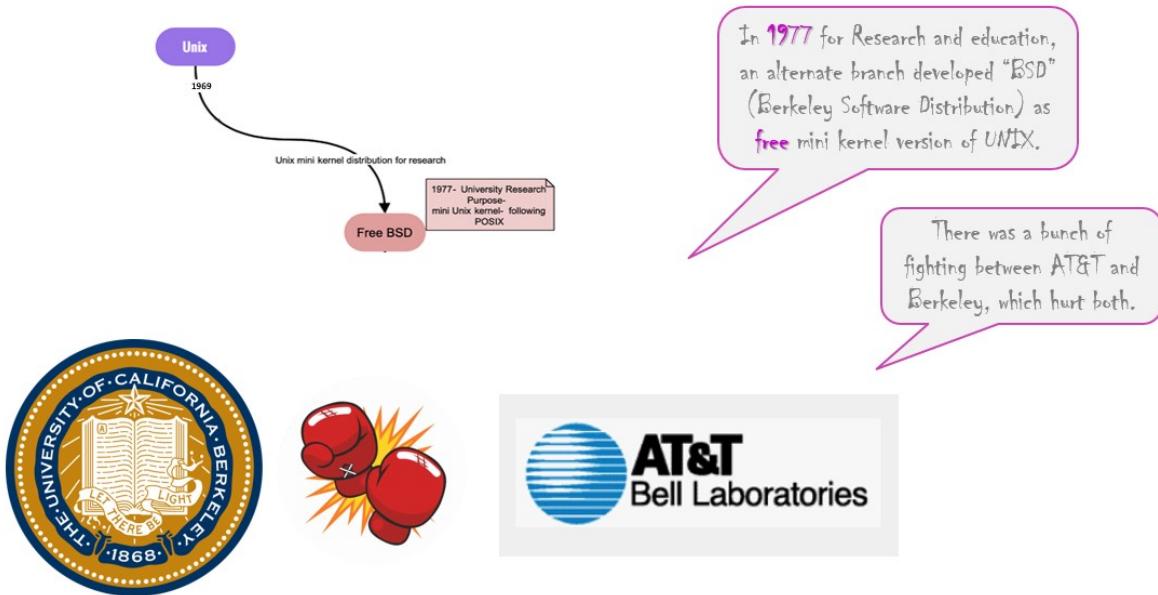
Ken Thompson

Dennis Ritchie

החוקרים האחרונים לעזוב את המיליטו לשכתב את העבודה שלהם אר באופן מצומצם יותר. ב-1969 הציאה חברת at&t אתUNIX. היא הייתה שפת התוכנות הראשונה שהשתמשה בשפת תכנות עילית - שפת C. יוניקס היא מערכת ההפעלה הראשונה שלא הייתה תלוייה במערכת הפעלה. למעשה זאת הייתה פעם ראשונה שמערכת הפעלה הייתה כתובה נכון.

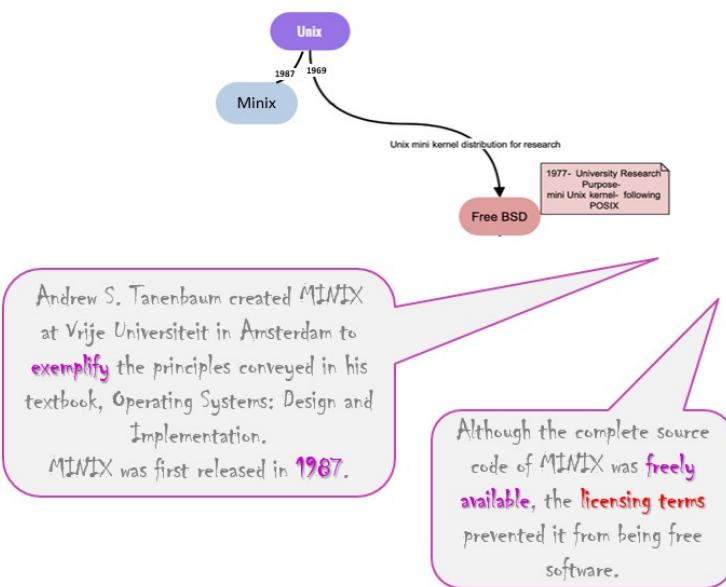
במקור יוניקס הייתה מסחרית.

Evolution of Operating Systems



ב-1977 פותחהBSD כמוינி ליבה של יוניקס בקוד פתוח למטרות מחקר והוראה. כתוצאה לכך היה מאבק משפטי בין at&t לבין ברקלי שפגעה בשנייהם.

Evolution of Operating Systems



Andrew Stuart Tanenbaum

בשנת 1988 tanenbaum כתב את minix על מנת להציג עקרונות שגלו בספריו הלימוד שלו. הארסה הראשונה שוחררה ב-1987 ולמרות شكود המקור שלו היה זמין ללא עלות, הרישיון מנעה ממנה להיות תוכנה חופשית.

History of PC (Personal Computer)



1950: Simon

Simon represented the **first experience** of building an automatic simple digital computer, for educational purposes. In fact, its ALU had only **2 bits**, and the total memory was 12 bits (2 bits x 6).



Edmund Callis Berkeley

cutת נယור להיסטוריה של המחשב האישי
היו המונן ניסיונות לבנות מחשב אישי. הניסיון הראשון היה ב-1950. המחשב סימון הציג את הניסיון הראשון לבניית מחשב דיגיטלי אוטומטי ראשון. למעשה ליחידה האריתמטית היו רק 2 ביטים וזכרון כולל של 12 ביטים (לא בתים)

History of PC (Personal Computer)



1948: IBM 610



1957: Olivetti Elea



1962: LINC



1970: Datapoint 2200

Datapoint 2200 is among the earliest known devices that bears significant resemblance to the modern personal computer, with a CRT screen, keyboard, programmability, and program storage.



1973: Xerox Alto and Star

first computer with mouse and GUI

במשך היו מספר ניסיונות כאשר הבולטים היו ה-2200 שהייתה בין הראשונים להציג משהו שזכיר מחשב אישי מודרני עם מסך, מקלדת, יכולת לתוכנת ו אחסון לתוכניות, וה-Alto ו-Xerox Star שהייתה המחשב הראשון עם עכבר וממשק משתמש אגרפי.

History of PC (Personal Computer)



1975: Altair 8800

Altair 8800 used Intel 8080 microprocessor
many digital resources mention Altair 8800 as father of all personal computers



ב-1975 הוצג המחשב Altair 8800, שהשתמשה בمعالג 8080 של אינטל ונחשבת לאבא של כל המחשבים האישיים.

History of PC (Personal Computer)



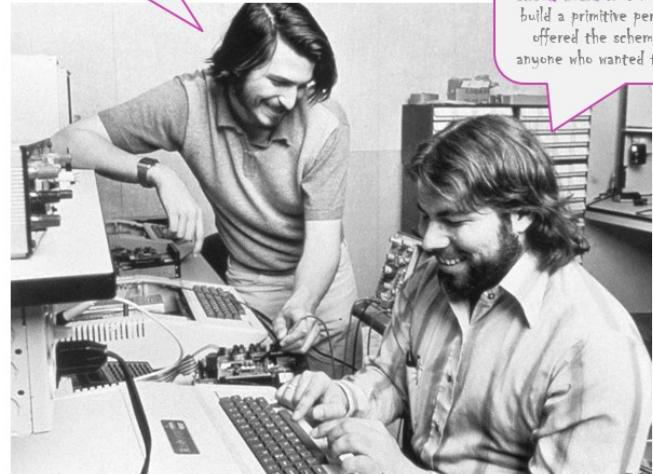
First (unused)
Apple logo

Apple I came with Steve Wozniak's monitor program, which included basic line editor, allowed to inspect memory, write to memory and run programs.



1976: Apple-1

Steve Jobs
recognized it could
be a **business**.



Steve Wozniak went to work at HP as an engineer, designing calculators. In 1976, in his spare time, he **designed a circuit board** that hobbyists could use to build a primitive personal computer. He offered the schematic at no cost to anyone who wanted to build a computer.

ב-1976 אפל הציגו את המחשב האישי הראשון שלהם. הוא כלל עורך "שורה" בסיסי שאפשר פיקוח על הזיכרון, כתיבה ל זיכרון והרצה של תוכניות

History of PC (Personal Computer)



1977: Apple II

Apple II was the **first truly successful** commercial personal computer

Apple computers used Operating Systems developed by Apple (based on UNIX).

Apple II came with 4 KB of RAM, but it could be increased to 48 KB. In addition, it came with a BASIC interpreter and the ability to support graphics and a color monitor. Over **2 million computers** had been sold till 1984.

IBM responded to the success of the Apple II with the IBM PC, released in August 1981.



1981: IBM 5150



1984: Macintosh (Apple)

Macintosh used its own **Macintosh Operating System** with **GUI**.

1985: Microsoft introduced **Windows OS** with **GUI** (that has many of the same capabilities of the Macintosh)

ב-1977 הציגה אפל את הדור השני למחשב שלה. הוא היה המחשב הראשון שבאמת היה הצלחה מסחרית. בשלב זה יוצר המחשב עבר לייצור תעשייתי. המחשבים של אפל השתמשו במערכת הפעלה פרי פיתוחם של אפל והתבססה על יוניקס. היא הגיעה עם 4KB של ראמ עם יכולת הרחבה ל-48KB, שזה היה המון באותה תקופה. כמו כן הוא תמך ביכולות גרפיות ובמסגר צבעוני.

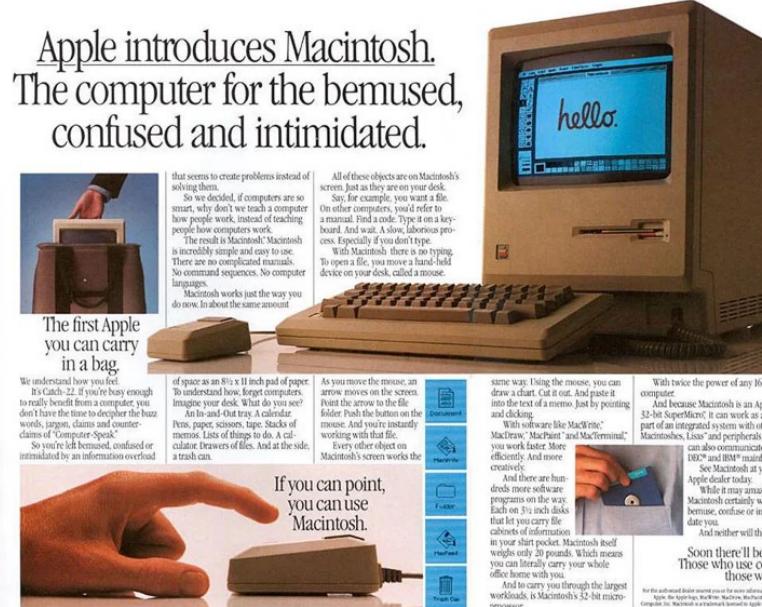
ב-1981 IBM הציגו כtagובה להצלחתו המסחרית של אפל 2 את המחשב האישי שלהם - ה-PC.

ב-1984 יצא המاك הראשון שהגיע יחד עם עברו ומערכת הפעלה עם משק גרפי.

ב-1985 מיקרוסופט הציגה את windows שהגיעה עם מסך גրפי שהציגה הרבה מהיכולות של המק.

self-read

Apple introduces Macintosh. The computer for the bemused, confused and intimidated.

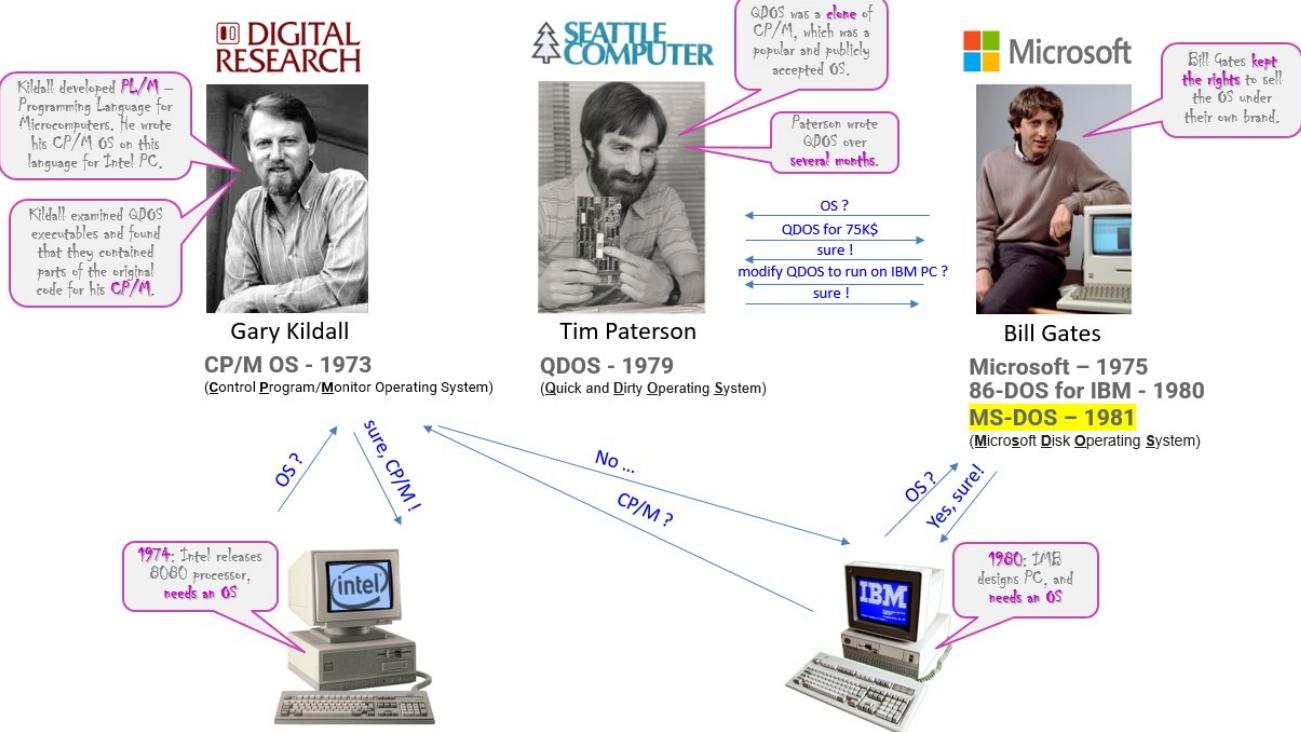


Macintosh's Personality: THE SERIOUS SIDE



THE FUN SIDE





פעם היה נהוג לسؤال חברות חומרה או חברות תוכנה. אם חברת חומרה הייתה זוקה לתוכנה על מנת להשתמש בחומרה שלה, היא הייתה פונה לחברת תוכנה לשם כך. אף הוי שוניים, שכן הוי גם חברת תוכנה וגם חברת חומרה - המערכת הפעלה שלהם הוי מתאימים רק למחשבים של החברה.

רעיון נוסף שהתפתח היה הרעיון של מערכת הפעלה למחשבים אחרים - הסיפור של ווינדוס

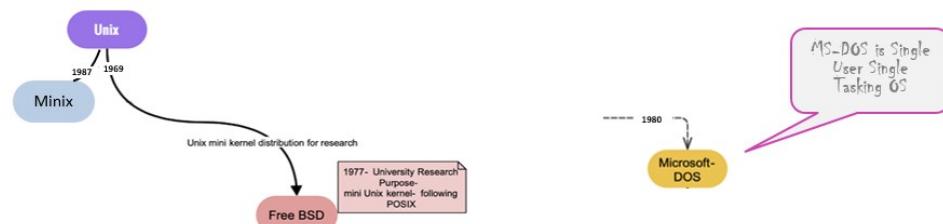
הסיפור של ווינדוס:

ב-1973 גاري קילדל פיתח שפת תכנות לマイרומעבדים במיוחד מחשייב אינטל. על בסיס השפה הוא בנה

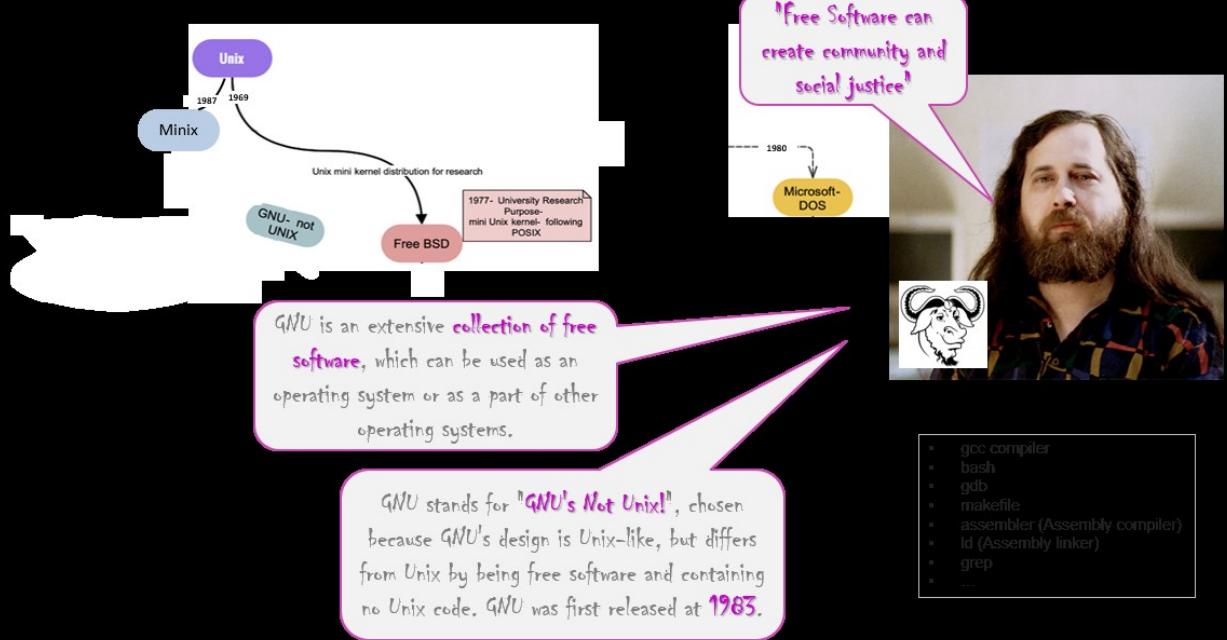
את מערכת הפעלה M/CP. ב-1974 אינטל הציגו את מעבדי 8080 והשתמשו במערכת הפעלה שגארி פיתח, בהסכםתו.

ב-1979 הציג טים פטרסון את QDOS שהיא שכפול של מערכת הפעלה שגאררי כתב אף הייתה פופולרית. במקביל כמה מייקרוסופט כחברת חומרה. חברת IBM גם פנו לאגاري בשכיל מערכת הפעלה אך הוא סרב לעשות אותם עסקה. וכך IBM פנו לביל גייטס. בהיותו איש עסקים מצוין, הוא הצליח להם למכור להם מערכת הפעלה עוד לפני שהייתה להםجوزת. לשם כך הוא קנה מטים את מערכת הפעלה שלו ושילם לו כדי להתאים אותה ל-IBM. כך ב-1981 יצא MS-DOS.

Evolution of Operating Systems



Evolution of Operating Systems



ב-1983 התחיל ריצ'רד סטולמן לעבוד על מערכת הפעלה שעה תחת האחרון שלו GNU. היעד של המערכת הייתה לבנות מערכת הפעלה חופשית כך שכל אדם יוכל לעיין וללמוד את קוד המקור של התוכנה.

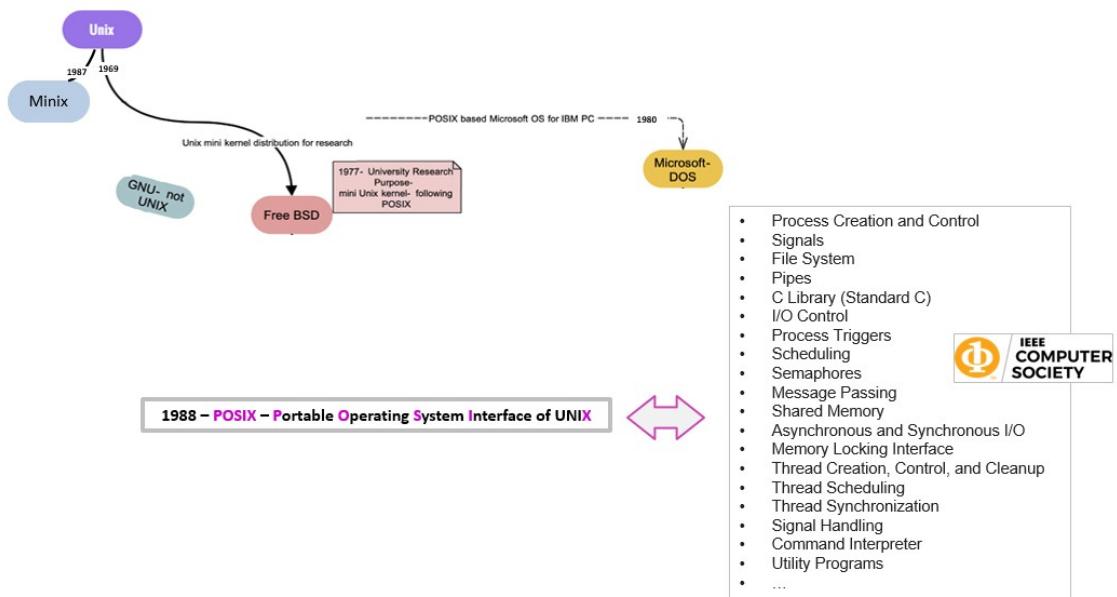
בהתחלת מערכת ההפעלה שהוא פיתח התבוסה על ITS שהיא מערכת הפעלה לארכיטקטורה PDP-10. בשלב מסוים הארכיטקטורה מתחה והוא העביר את הפיתוח לשפות C ו-POSIX ואף לדאוג כי יהיה תואמת לIONIKS.

הדבר דרש שכותב של כמעט כל הקוד חדש ומאפס (כי יוניקס לא הייתה תוכנה חופשית), אך היה שימוש ברכיבי תוכנה מצד שלישי כמו org. הגרנול של מערכת הפעלה התבוסה על Mach (שהיא עצמה הפכה להיות הבסיס של hurd GNU - הליבה הרשמית שאנו משחררת בימינו)

קיים פרויקט אחד שהיא בעצם הפקה של מערכת הפעלה של גנו.

[קריאה נוספת על אונגו](#)

Evolution of Operating Systems



ב-1988 היה נמאס מהבלגן וריבוי מערכות הפעלה במיוחד בריבוי של תוכנות.

רצו סטנדרט אחיד לכל מערכות הפעלה. כך נוצר אונגו - אוסף של דרישות ממוקם הפעלה על מנת לקבל את החותמת.

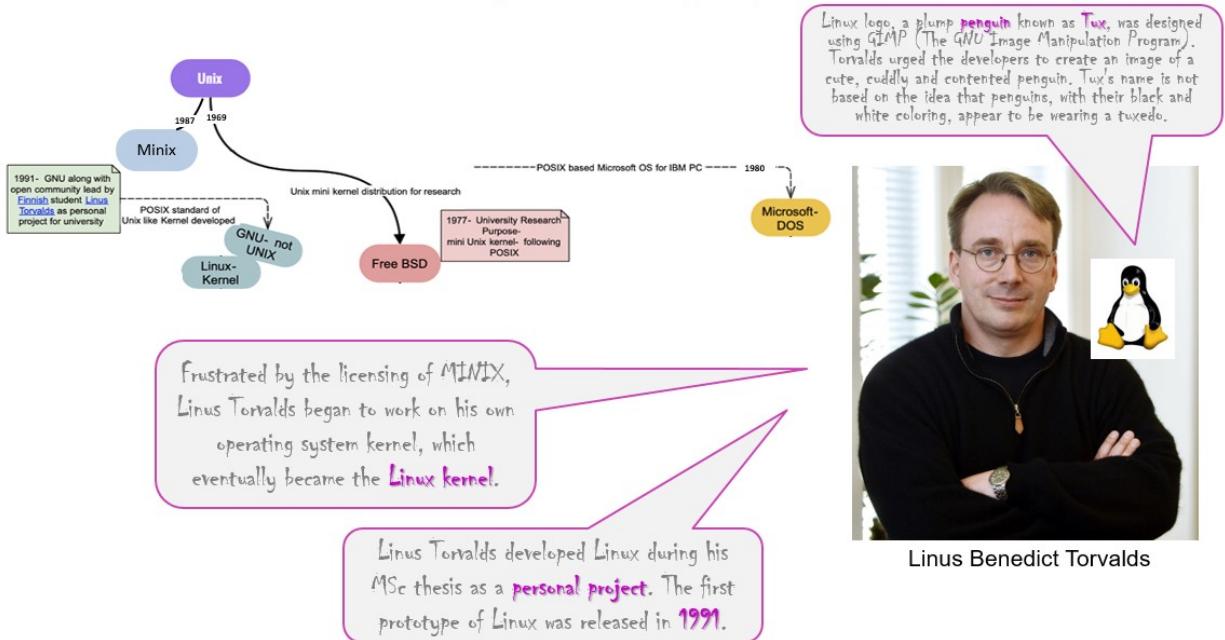
כתוצאה לכך המתכנת קיבל ממשק משותף לכולם וכך אפשר לקוד להיות נייד.

הערה:

למייט זכרוני מרינה אמרה שוינדוס עונה על הדרישות של POSIX. זה לא מדויק.
עד 10 windows הייתה לוינדוס תחת-מערכת שתמכה ב-XPOS על מנת לענות על הדרישות של הממשל האמריקאי. עם זאת בשלב מסוים אף אחד לא היה אכפת אז הם הסירו אותה. במקביל מיקרוסופט הציגו את WSL אף מעולם לא טענה שהיא עומדת על הדרישות של POSIX כי, שוב, אף אחד לא היה אכפת.

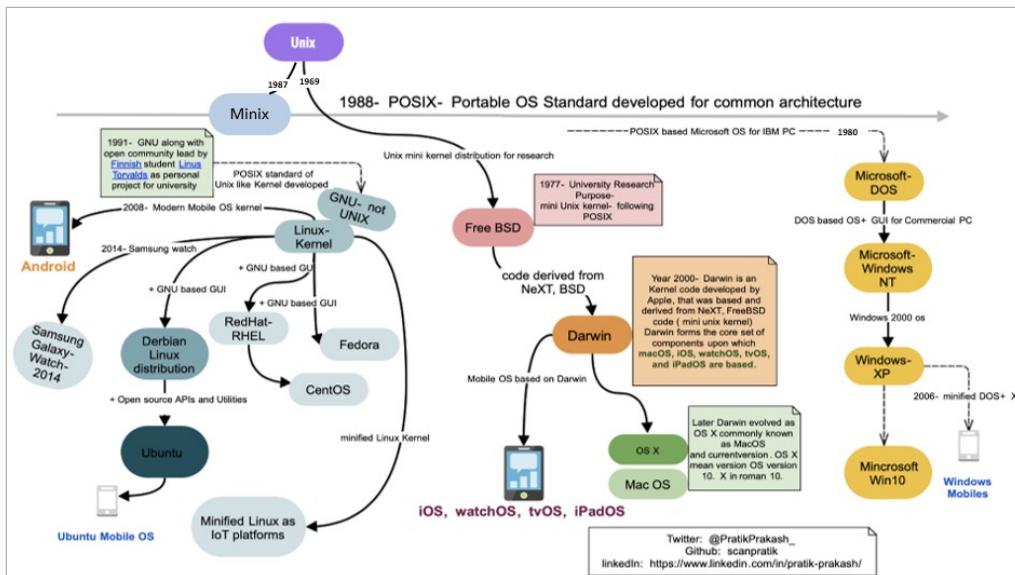
[מקורות](#)

Evolution of Operating Systems



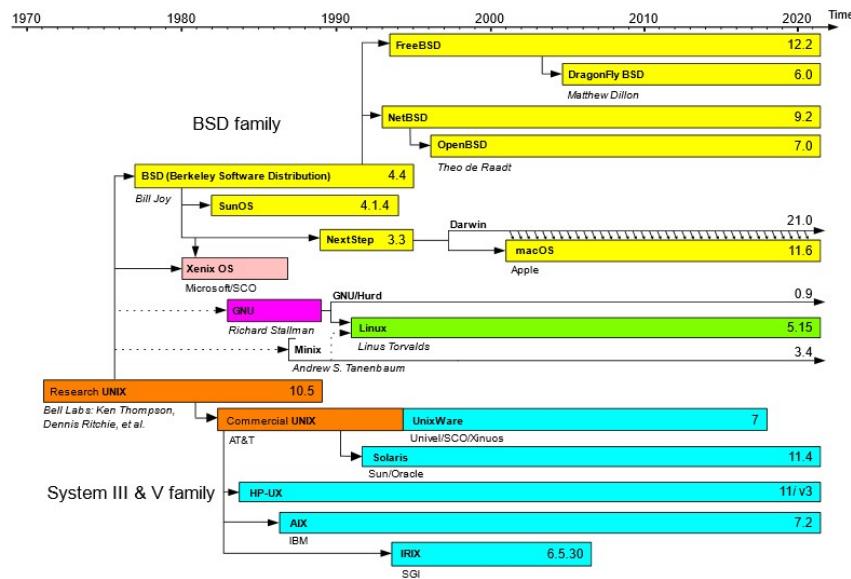
ב-1991 הגיע לינוס טרובלדס והמציא את לינוקס. הוא התחיל אותה כפרויקט אישי.
הסמל של הkernel עוצב בעזרת התוכנה gimp
במקום לבלב במוח אני מצרף ראיון שערך מנכ"ל אינטל עם לינוקס

Evolution of Operating Systems



זאת התמונה המלאה של מערכות הפעלה בימנו

Evolution of Operating Systems



Evolution of Operating Systems – Unix

self-read

In the late eighties, Sun Microsystems and AT&T teamed up to corner the market on the "future" of the Unix. The result was SVR4 (System V Release 4). There were some implementations outside of Sun (e.g. SCO, Motorola, DG), but Sun is the only popular rendition of it today and it CAN be downloaded: [Solaris Operating System - Get Solaris 10](#)

The other major commercial Unix providers, AIX, HP-UX are based off of SVR2 with their own additions. The other, not so popular, commercial Unix is Tru64 and is yet another variant that was part of an organization called OSF that was designed to counter the Sun + ATT juggernaut. All of that is history... but Tru64 (formerly DEC OSF/1 and later Digital Unix) still lives on. Versions of OSF/1 produced outside of DEC's version are considered collector's items.

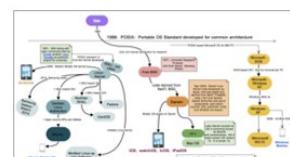
Outside of commercial System V altogether is the other branch of Unix called BSD. There are many variants of it out there. The most popular being things like FreeBSD, OpenBSD, NetBSD, Dragonfly BSD... you can google for links to their freely downloadable distros. Just for completeness, before Sun came up with Solaris (their modified SVR4), they used a BSD variant they called SunOS (1).... technically speaking

SunOS always refers to the Sun kernel, but kernels less than version 2 were BSD based and so was the distribution.

Linux is very Unix like... it has most of what Unix has and has many things that are NOT found in commercial Unix. There are some cases where the implementation in Unix is considered better, but the pros for Linux far outweigh the cons.

Most Unix (non Linux) will have limited platform support (well..less than Linux)... so you have been warned. From experience, using Solaris, OpenSolaris and/or a BSD variant from inside of a virtual machine seems to work well and might be a good option if attempting to install directly

- So IMHO, viable options:
 - Solaris from Oracle
 - OpenSolaris (though work has stopped on this)
 - FreeBSD
 - OpenBSD
 - NetBSD
 - DragonflyBSD



There are some OpenSolaris replacement projects out there... you can google for those (e.g. Illumos, OpenIndiana, etc). Just be warned that NOT all pieces of OpenSolaris were truly open, so sometimes the derivatives might be missing some key pieces.

Introduction: outline

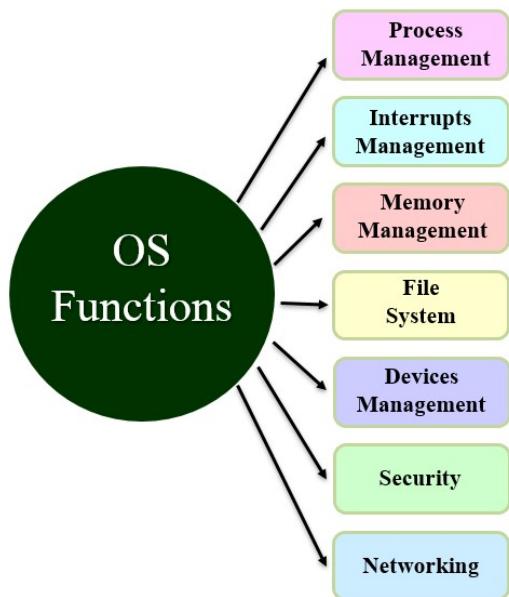
❑ What is an operating system?

❑ Some history

❑ OS concepts

רעיון מרכזים במערכות הפעלה

OS – Key Functions

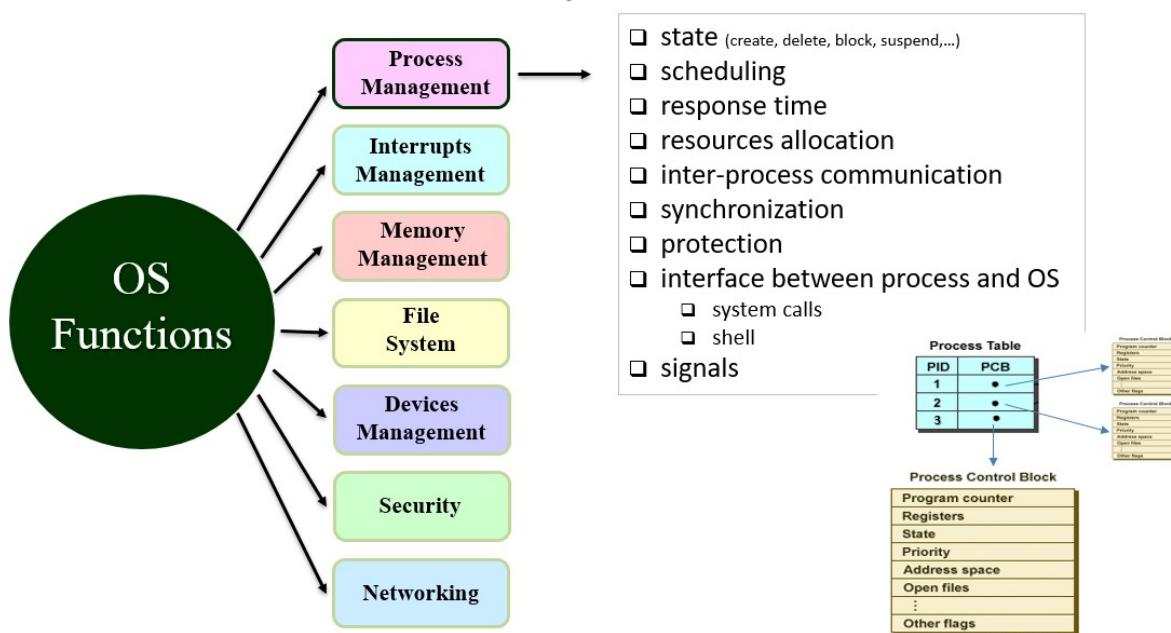


נתמקד רק בנקודות החשובות. השאר נלמד בתרגולים.

ניתן לאפיין את הפונקציונליות של מערכת הפעלה בעזרת 7 הרכיבים הבאים:

ניהול תהליכיים, ניהול פסיקות, ניהול זיכרון, מערכת קבצים, ניהול התקנים, אבטחה ותקשורת.

OS – Key Functions



ניהול תהליכיים

ראשית נבהיר כי תוכנית היא בעצם סדרת הוראות להרצתה (כמו מתקון לעוגה). התהיליך הוא ייצוג הרעיון של הרצת תוכנית. ככל תהיליך יש מרחב כתובות מסויל, רשיימה של מיקומי זיכרון מהם התהיליך יכול לקרוא ולכתוב. מרחב הכתובות כולל את התוכנית עצמה, הנתונים של התוכנית והמחסנית שלה.

על מנת לנהל תהליכיים, מערכת הפעלה מחזיקה טבלה של אלמנטים המאפיינים תהליכיים, כך שלכל תהיליך יש שורה בטבלה.

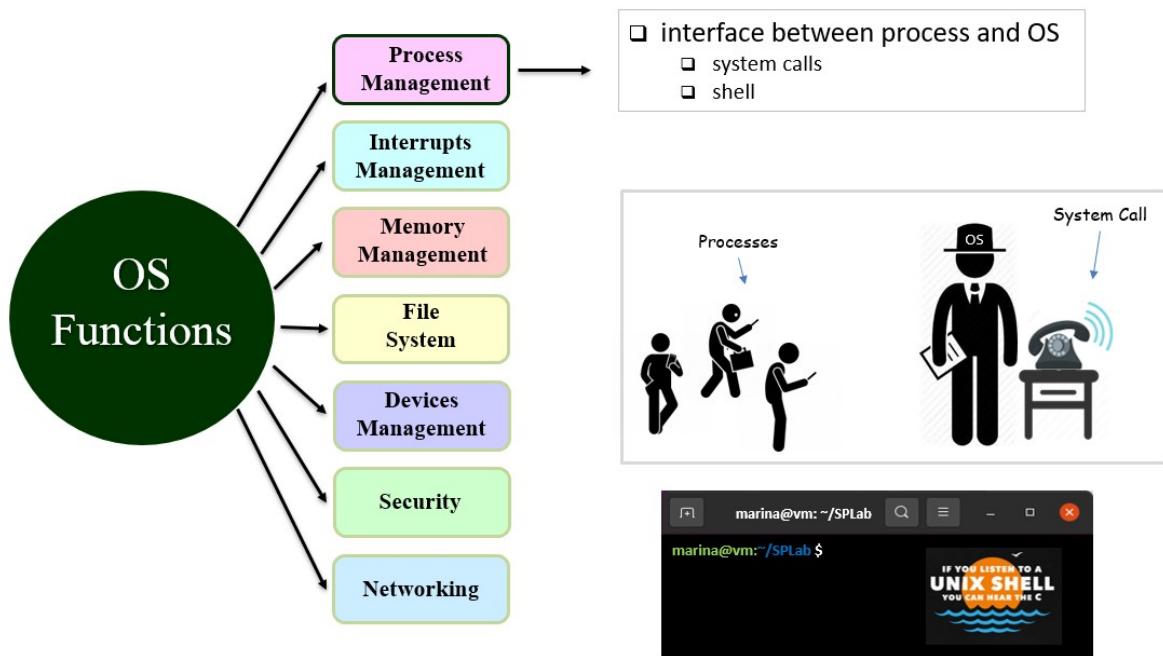
שורות הטבלה מייצגות מבנה נתונים המתארת את קבוצת המשאים של התהיליך - כולל רגיסטרים, רשיימה של קבצים פתוחים, תהליכיים קשורים ומידע נוסף הנחוץ לריצת התוכנית.

במילים אחרות התהיליך הוא בעצם קונטינגר שמחזיק את כל המידע הדרוש להרצת תוכניות.

בהמשך אנו נראה כי במערכות התומכות בריבוי תהליכיים, מספר תהליכיים יכולים להיות פעילים בו זמן נתן. לשם כך מערכת הפעלה מחליטה מיד פיים לעזר ריצת תהיליך אחד על מנת לאפשר לתהיליך אחר לרווח.

בנוסף, ראיינו בקורסים קודמים כי תהיליך יכול ליצור תהליך בן, דבר היוצר עץ תהליכיים. בכך, תהליכיים אשר קשורים אחד לשני יכולים להיות בקשר ולנסגר את הפעולות שלהם.

OS – Key Functions



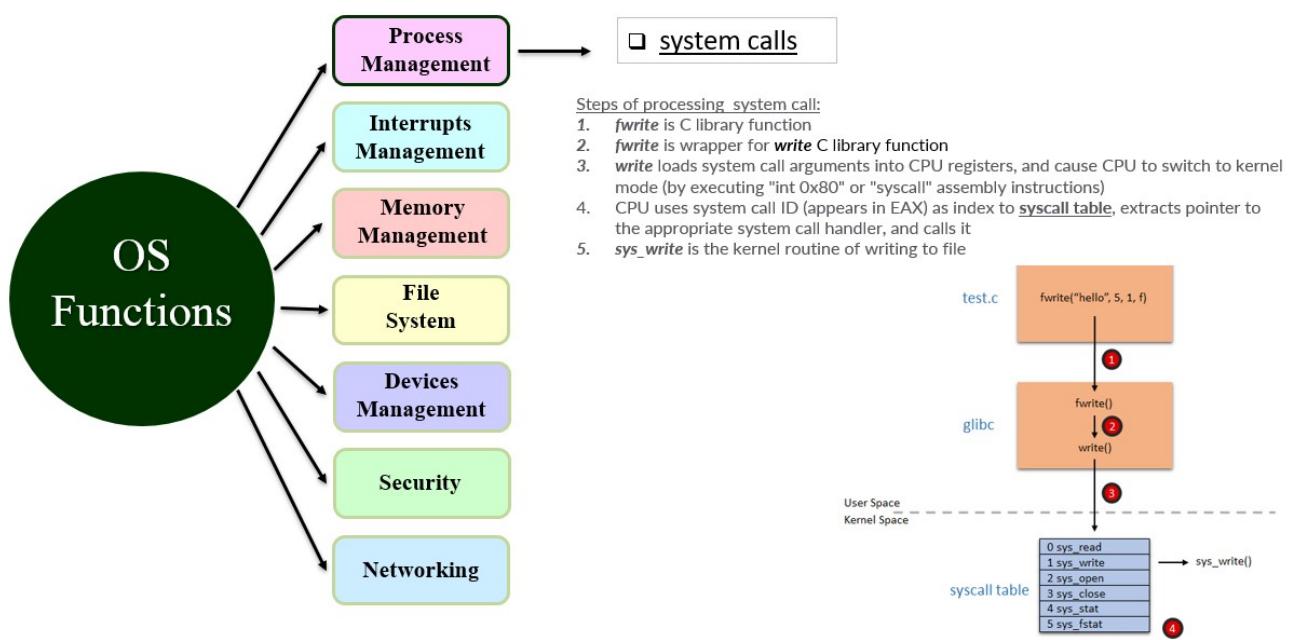
shell

בעוד שהוא לא חלק ממערכת הפעלה, הוא משחק תפקיד חשוב כմמשק בין המשתמש למערכת הפעלה. הוא מנצח הרבה מהתוכנות של מערכת הפעלה ומשמש כדוגמה ראשית לאופן שבו קריאות מערכות עובדות.

זהו הממשק הראשי בין המשתמש למערכת הפעלה, במיוחד כאשר המשתמש לא משתמש בINTERFACE GRAPHIQUE (GUI).

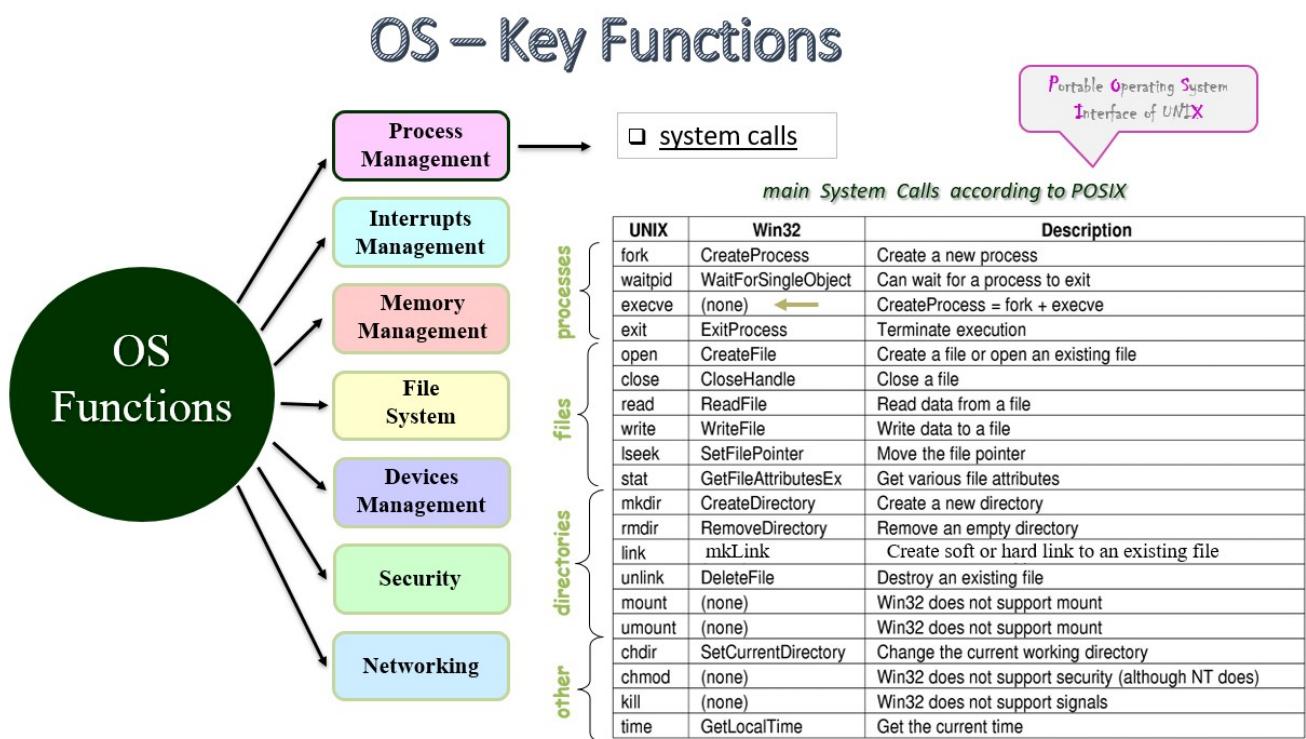
כאשר משתמש מתחבר למחשב, ה-shell מאותח על המסוף קלט ופלט סטנדרטיים (stdin, stdout).

OS – Key Functions



איך קריית מערכת עובדת:
נניח ויש בקוד שלנו `fwrite` - זאת לא קריית מערכת אלא פונקציית C סטנדרטית שמחביהה בתוכה קריית מערכת.

ברגע שמתבצעת קריית מערכת יש מעבר למרחב המשתמש למרחב הkernel. במרחב הkernel יש טבלה של קריאות מערכת כאשר כל אינדקס (מס' קרייה) מהוות פוינטר לפונקציה המתאימה אשר מבצעת את הוראות קריית המערכת.



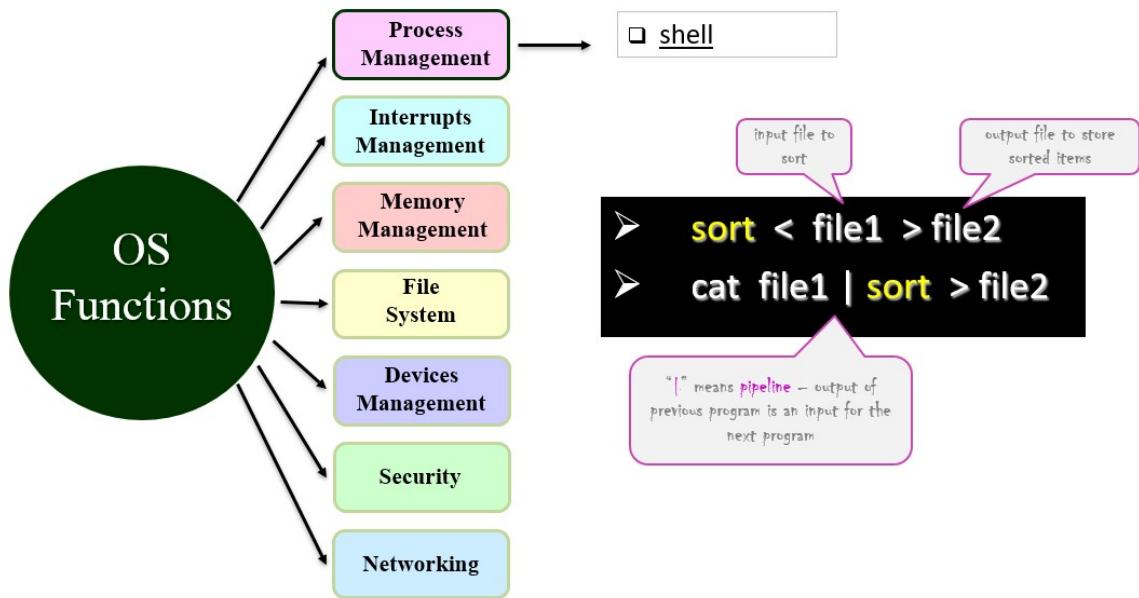
אליהם קריאות המערכת הראשיות שכל מערכת הפעלה שתואמת ל-POSIX חייבות למעשה. בוינדוס אין מקבילה ל-execve אבל כן יש את createProcess שגם עשו fork וגם execve. השאלה סטייל מבחן: האם החלטות שקולות? החלטות לא שקולות.

יוניקס מאפשרת יותר שליטה. אפשר לקנפה את התהיליך בן כדי להתאים. ווינדוס לא מאפשרת, כי אז יש פחות context switch (כי יש פחות קריאות מערכת).

החלטה של לינוקס: עשינו fork ולא עשינו execve. יש לנו 2 IC שונים. כאשר עושים fork לא באמת משכפלים שכפול עמוק של תהיליך. הם למעשה פונים לאותם כתובות בזיכרון אלא אם הטענה שינה.

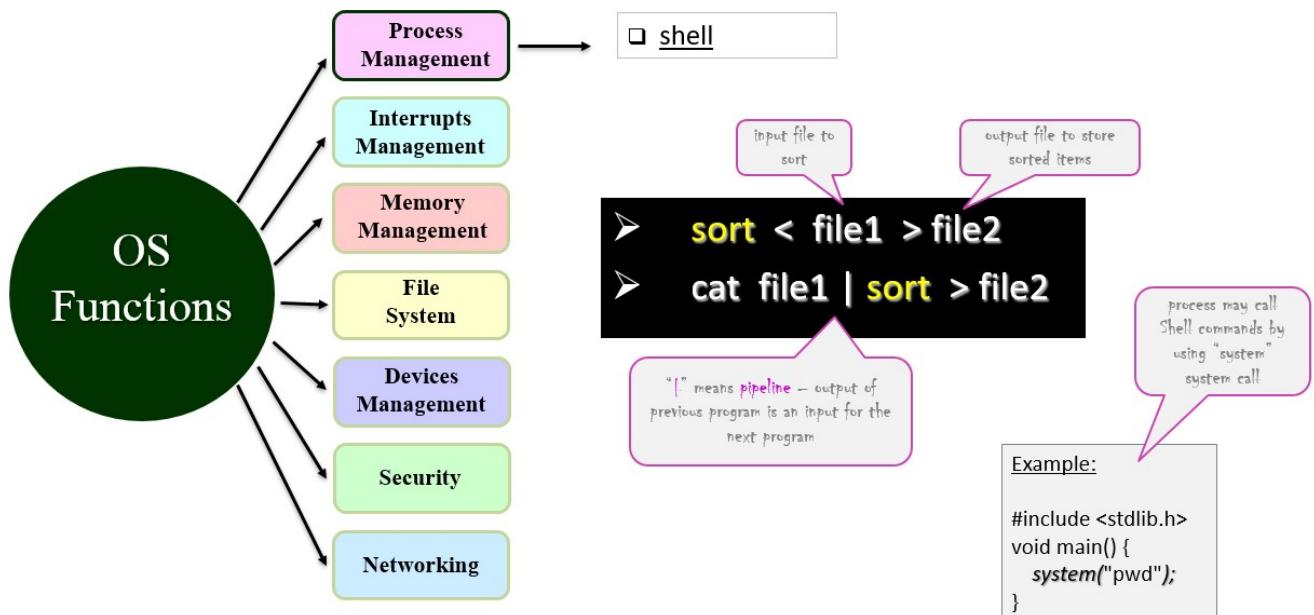
נניח ושלחנו הוראה `6, x mov` למעבד. מערכת הפעלה לא יודעת לזה איז השינוי יבוצע אצל שנייהם ולכן היא מסמנת את כל הסקיישנים בזיכרון readOnly-icamente וברגע שמעבד ינסה לבצע את הפקודה, הוא ישלח אותן למערכת הפעלה להרוג תהיליך על השגיאה. מערכת הפעלה יודעת שהטהיליך ניסה לשנות משהו כי זה תוצאה של fork. מה שמערכת הפעלה תעשה זה לשכפל את אותם泰安ים בזיכרון.

OS – Key Functions

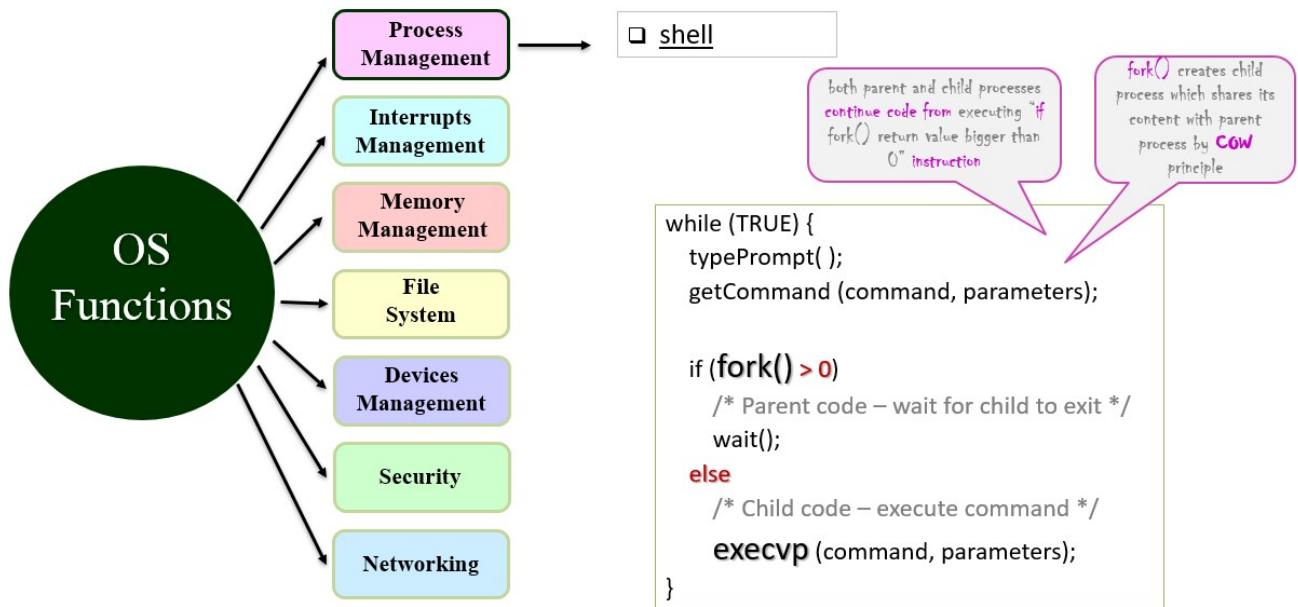


. התוכנית מאפשרת למשתמשים לחוות מחדש את הקלט והפלט הסטנדרטיים לקבצים (input-output redirect) . בנוסף, היא מאפשרת לפולט של תוכנית אחת להיות הקלט של תוכנית אחרת, על ידי שימוש ב-pipe.

OS – Key Functions

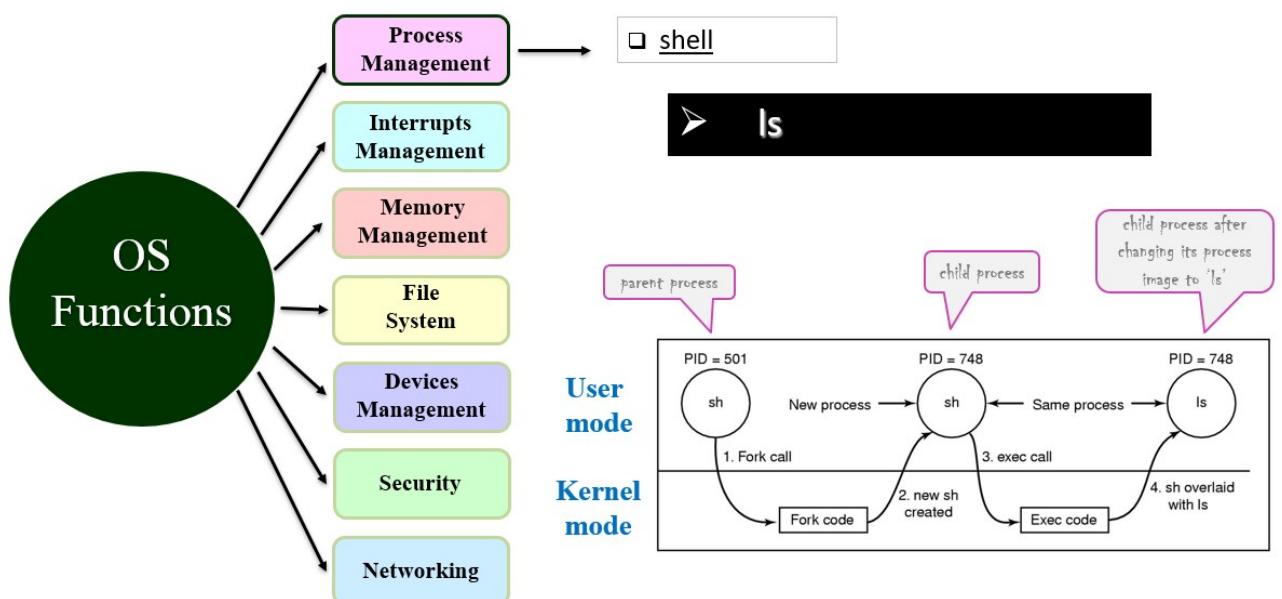


OS – Key Functions



כאשר משתמש מקליד פקודה, ה-shell מייצר תהליך בן ומריץ את הפקודה, שהוא בעצם תוכנית, בטור הבן. כל עוד תהליך הבן רץ, תהליך הבן ממතין עד שיסים. בעת הסיום, ה-shell מאפשר לקבל את ההוראה הבאה.

OS – Key Functions



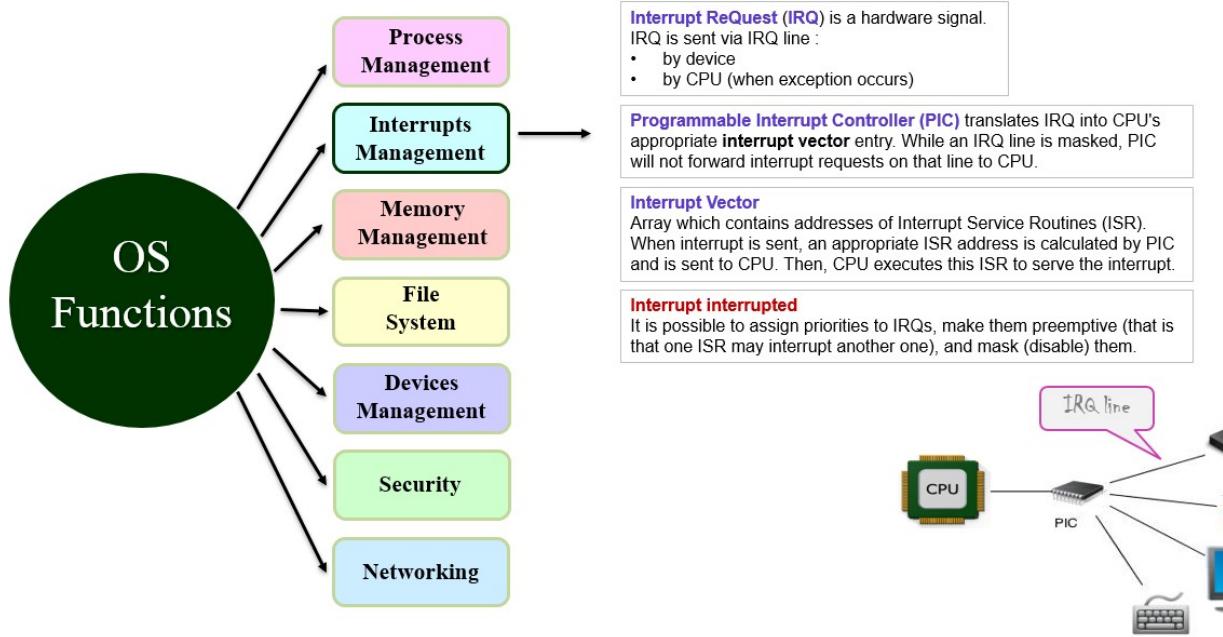
הערה:

בעוד שרוב המחשבים האישיים משתמשים במשק משתמש אגרפי, חשוב לזכור כי הממשק האגרפי הוא רק תוכנית אשר ריצה על גבי מערכת הפעלה, דומה ל-shell.

במערכות לינוקס, משתמשים יכולים לבחור מתוך מספר ממשקים אגרפיים (סביבות עבודה) כמו gnome, kde, Ai

אפשר מוביל להשתמש בסביבה ארכיטקטורתית ולהשתמש רק במסוף.

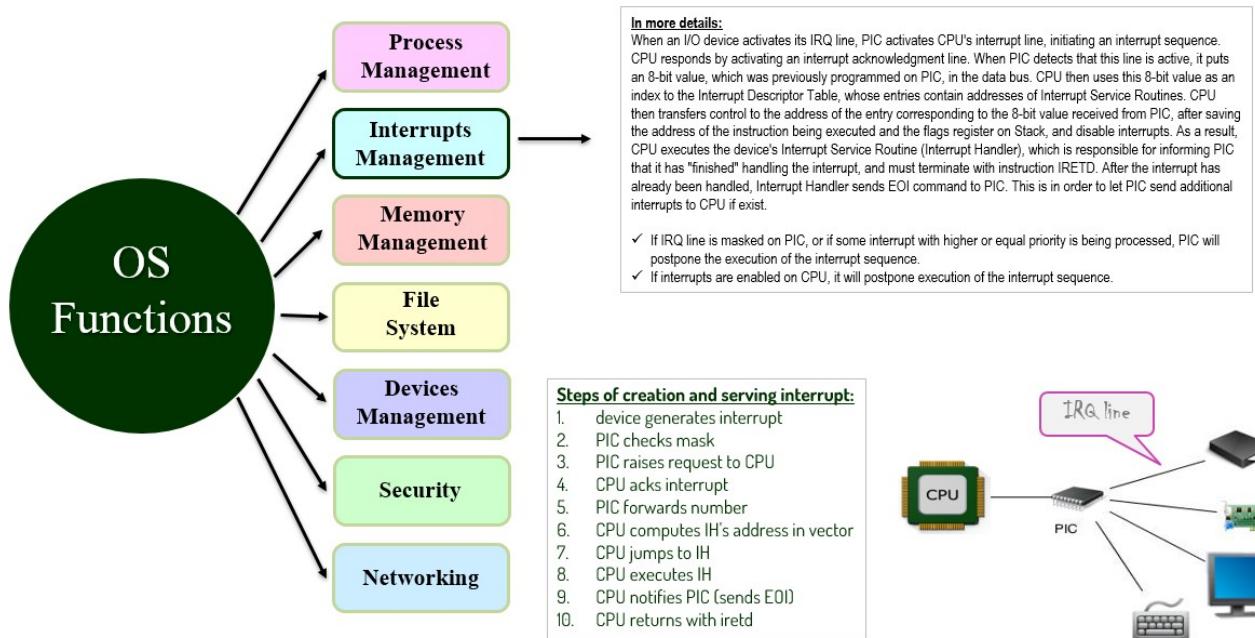
OS – Key Functions



ניהול פסיקות

הבעיה העיקרית עם IO הוא שהמעבד שולח הוראות רבות להתקן על מנת לבדוק אם הוא מוכן לביצוע פעולה חדשה. לשם כך ישנו קו בקרה מיוחד אשר נקרא **IRQ**. בנוסף ישנו בקר מיוחד הנקרא **PIC** וכן קטגוריה נוספת אשר יתדי ממערכת הפעלה יכולה לנוהל פסיקות.

OS – Key Functions



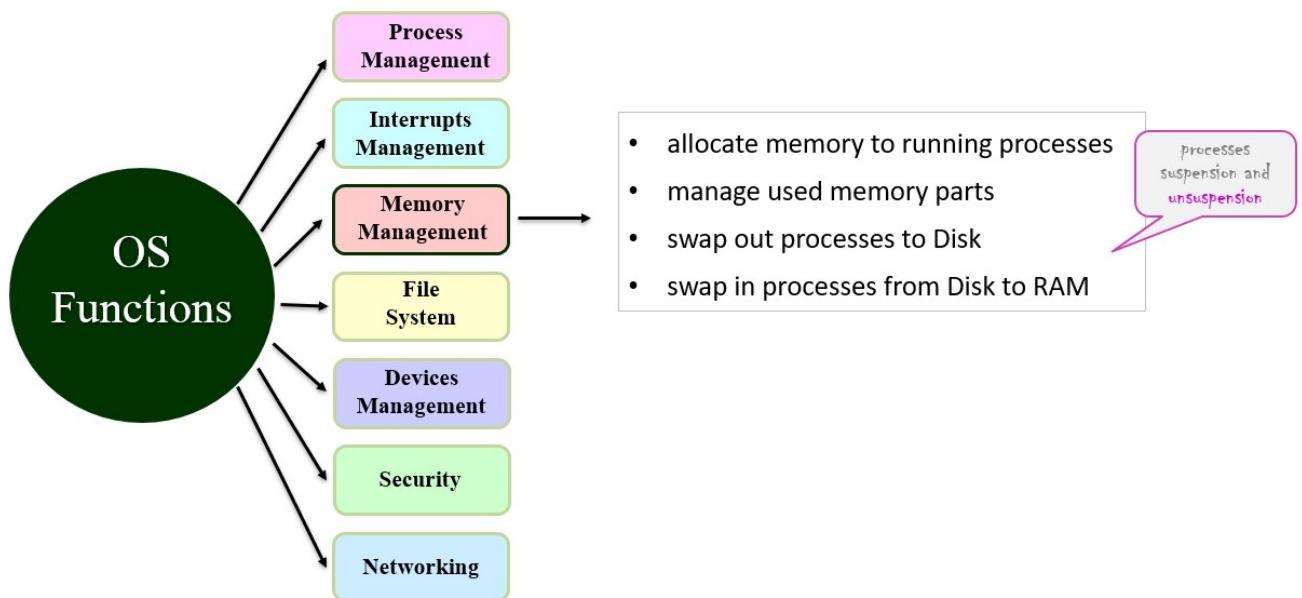
כאשר התקן קלט-פלט שולח אותו דרך קו הבקרה IRQ, ה-CPU מקבל זאת ושולח אותה לمعالד דרך קו הפסיקות של המمعالד. ככלומר בזמן שה-IO מבצע פעולות דרך התקן המمعالד מבצע מטלה כלשהי ורק לאחר מכן יסייע את ביצוע ההוראה הנוכחית הוא יודיע שהוא מוכן לקבל נתונים מהתקן. ההודעה מתבצעת באמצעות אות הנקרא

כארה ה-CIC מזיהה שהקו פעיל, הוא שם ערך בגודל 8 ביטים. המעבד משתמש ב-8 ביטים אלו כדי לאינדקס לטבלת זיהוי פסיקות שכל אחד מערכיו כולל כתובות של רוטינת הטיפול בפסיקה.

המעבד לאחר מכן מכין מעביר את השליטה לכתובה לערך המתאים לאותם 8 ביטים לאחר שהוא מגבה את המצביע הקודם (טרם הטיפול בפסיקה) ואני לא יכול פסיקות נוספות במהלך הטיפול בפסיקה.

כתוצאה לכך המעבד מרים את רוטינת הטיפול בפסיקה אשר אחראית לידעו ה-CIC כי המעבד סיים לטפל בפסיקה. לאחר סיום הטיפול בפסיקה ה-EOI שלוח אותה ל-CIC כדי שהוא יוכל לשולח פסיקות נוספות למעבד, במידה ויש כאלה.

OS – Key Functions



ניהול זיכרון

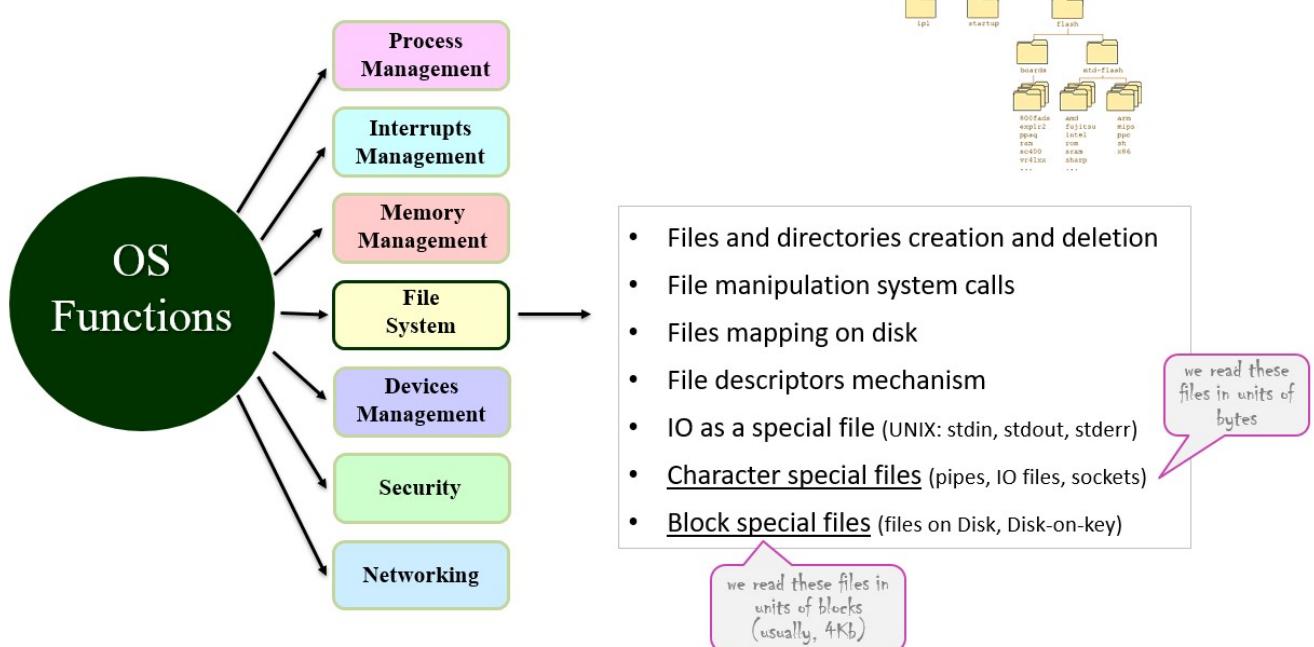
כל מחשב יש זיכרון מרכזי לטובת תחזוק תוכניות רצונות. במערכות הפעלה פשוטות, רק תוכנית אחת יכולה להיות על הזיכרון הראשי. על מנת להריץ עוד תוכניות, התוכנית שזו עתה רצתה נמחקת והבאה בתור מועתקת לזיכרון - תהליך אשר נקרא swapping.

מערכות הפעלה מורכבות יותר מאפשרות במספר תהליכי להיות בזיכרון בו זמן נתן. על מנת למנוע מתוכנית אחת להפריע לתוכנית אחרת ולמערכת הפעלה, יש צורך במנגנון הגנה. בעוד שמנגנון ההגנה מספקת, מערכת הפעלה היא זו ששלטת על זה.

כל תהליך יש קבוצה של כתובות אשר הוא יכול להשתמש. במקרה פשוט, מרחב הכתובות פיזית מוגבל ל- 2^{32} ביטים או 2^{48} ביטים ויתכן יותר מהזיכרון הפיזי. עם זאת במחשבים רבים, מרחב הכתובות פיזית מוגבל ל- 2^{48} ביטים או 2^{64} ביטים ויתכן מצב בו מרחב הכתובות של תהליך גדול יותר מממה שיש למחשב פיזית להציג. לשם כך אנו נראה את הטכניקה אשר נקראת זיכרון וירטואלי, בה מערכת הפעלה משתמשת באחסון (דיסק קשיח או SSD) על מנת שרק חלק מהזיכרון של התהליך יהיה בפועל בראם וחלק אחר שלו יהיה בדיסק.

למעשה בפרק של ניהול זיכרון, אנו נדונם בהקצת זיכרון לתהליכי רצים, ניהול חלק זיכרון אשר יש בהם שימוש, הטעינה של תהליכי בדיסק והחלפה בין תהליכי מהדיסק לראמ (לצורך הש��ה suspension)

OS – Key Functions



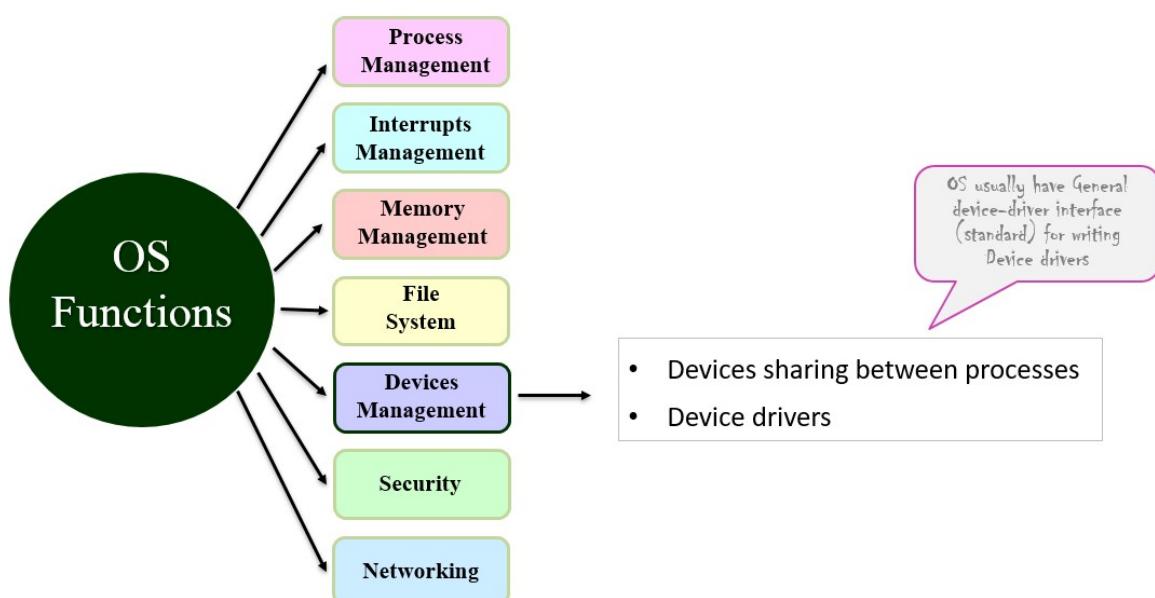
מערכת קבצים

קובץ הוא הדך הראשית בה מערכת הפעלה מפשיטה את האופן בו דיסקים, SSD, והתקני IO אחרים עובדים מעוני המתכנים. לשם כך מערכת הפעלה חייבה לספק ממשק עבודה נוח, דרך קריאות מערכת, איתן גם המתכנים וגם המשתמש יהיו מסוגלים ליצור, למחוק, לקרוא ולכתוב קבצים. כמו כן, רוב מערכות הפעלה משתמשות ברעיון של **תיקיה על מנת לאגד קבצים ביחד**.

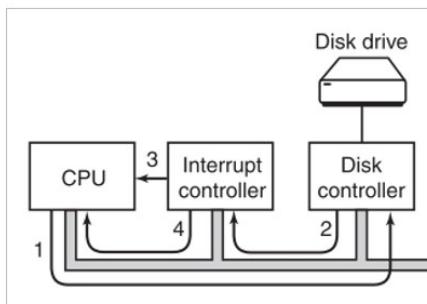
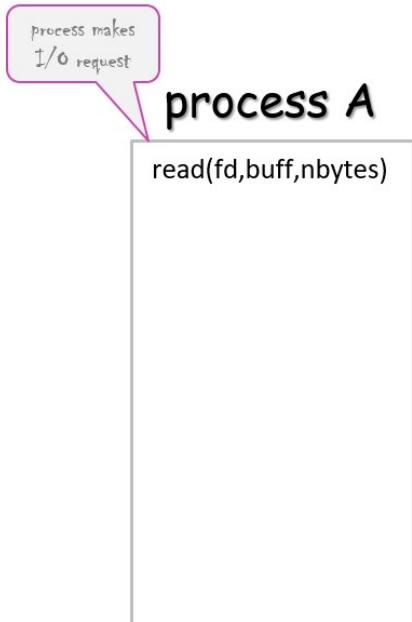
על כן מערכת הפעלה אחראית על ייצור של קבצים ותיקיות, על קריאות מערכות למניפולציות על קבצים, מיפוי קבצים בדיסק, מכניםם ל-fd, ותמייה בקבצים מיוחדים (כמתואר במצגת)

החלק במערכת הפעלה אשר אחראי לזה, נקרא **מערכת הקבצים**.

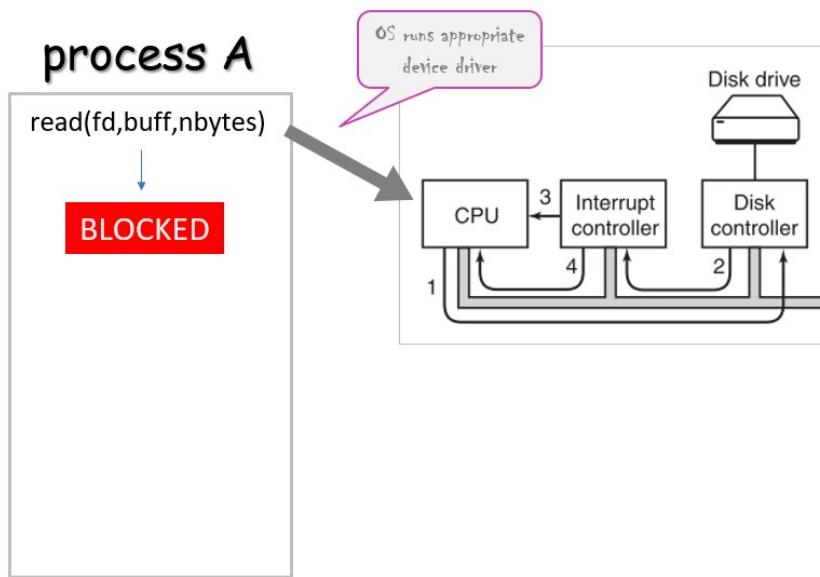
OS – Key Functions



Steps in performing I/O



Steps in performing I/O



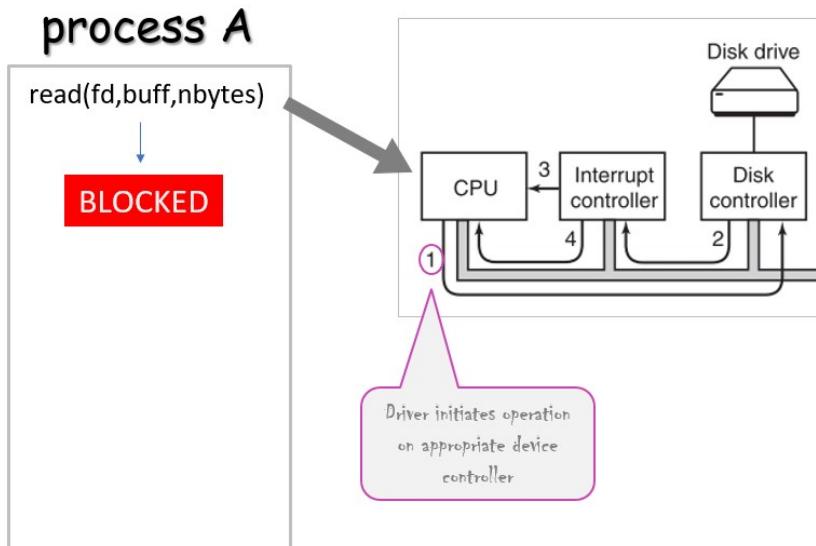
ניהול התקנים

ישנם התקני OS רבים אשר יוצרים תקשורת עם מערכת הפעלה. ראיינו כבר כי שליטה על התקן היא פעולה מורכבת, لكن לכל התקן יש בקר אשר מספק ממשק פשוט יותר למערכת הפעלה. כל בקר דוחש תוכנה שונה - זאת שידעת לתקשר עם הבקר. לפיסת תוכנה זו קוראים מנהל התקן. כל יצרן חייב לספק דרייבר לכל מערכת הפעלה שהתקן תומך בו.

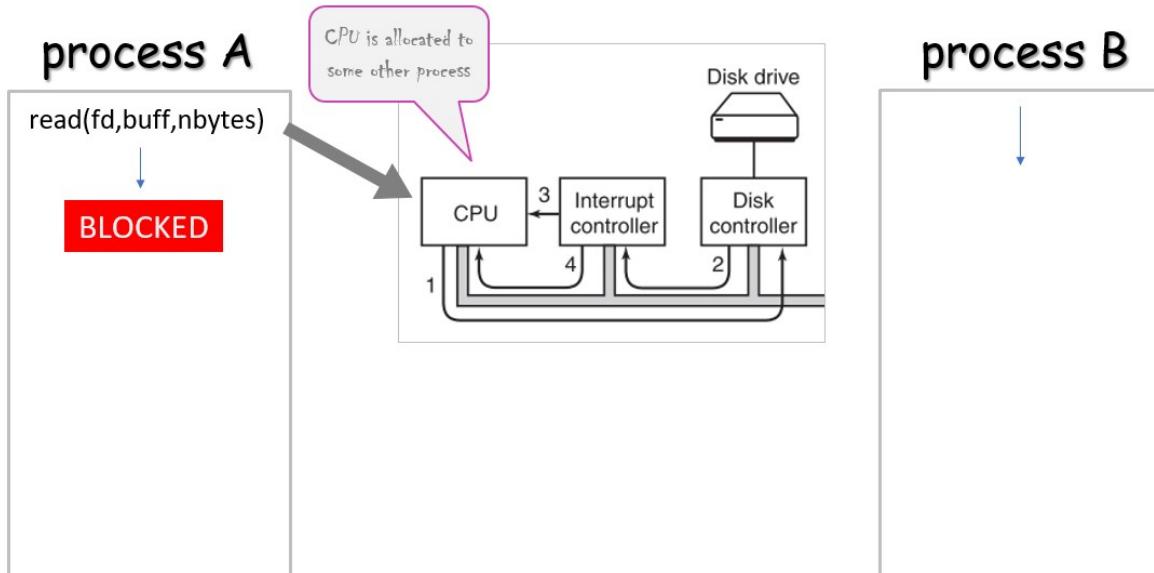
מכאן, שהדריבר חייב להיות חלק מערכת הפעלה על מנת שייעברו למרחב ה kernell. אין זו חובה ומערכות הפעלה מודרניות מאפשרות דרייברים למרחב המשתמש.

לכל בקר יש רגיסטרים אשר נועדו לתקשורת עם הhardware. לדוגמה, בברק לדייסק קשיה חייב להחזיק ברגיסטרים אשר מציינים את כתובות הדיסק, כתובת הזיכרון, מספר הסקטור והכיוון (קריאה או כתיבה).

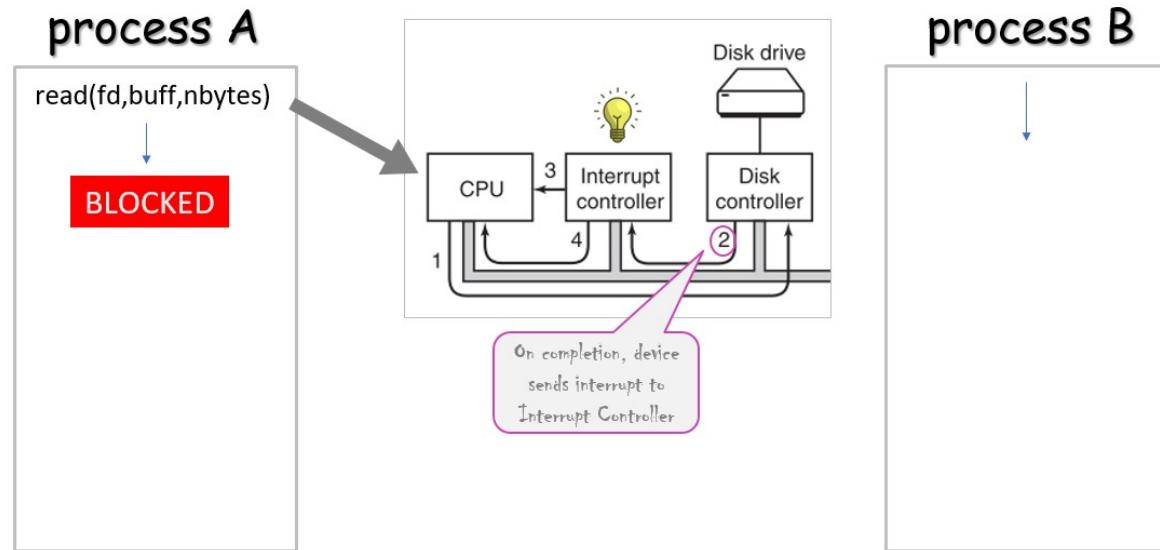
Steps in performing I/O



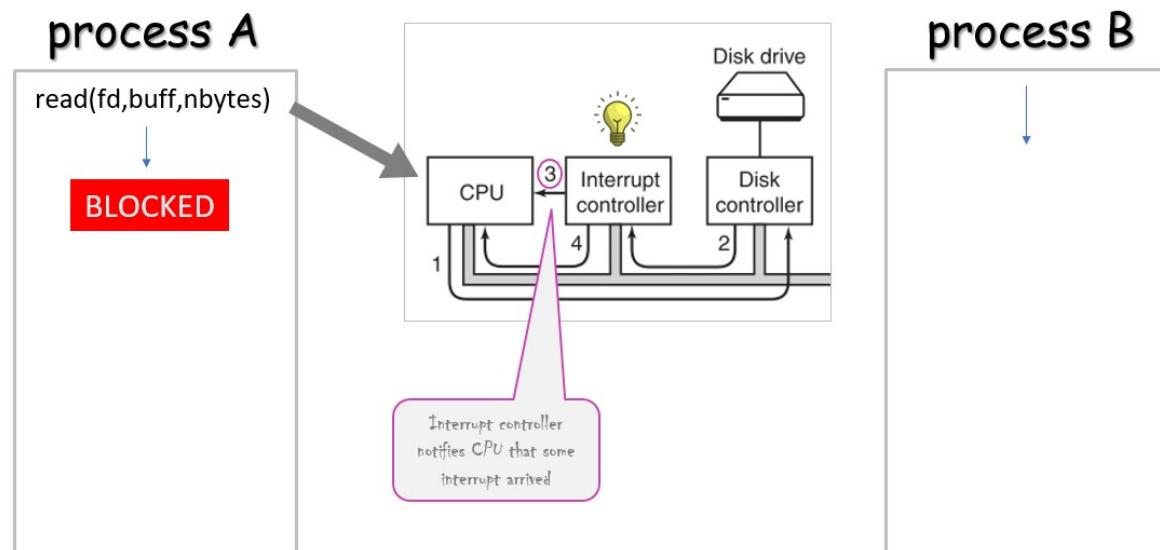
Steps in performing I/O



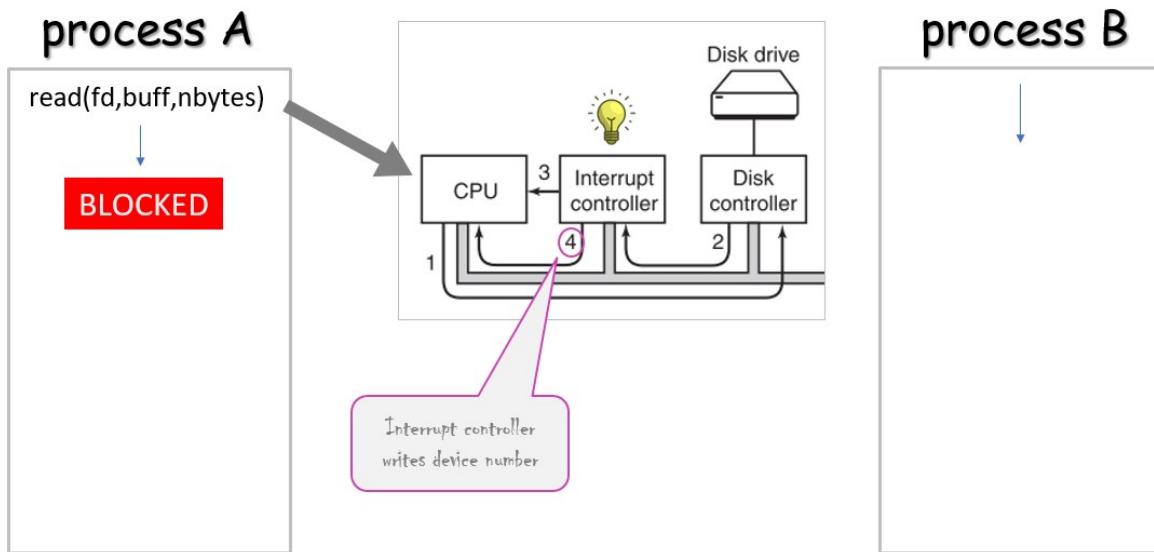
Steps in performing I/O



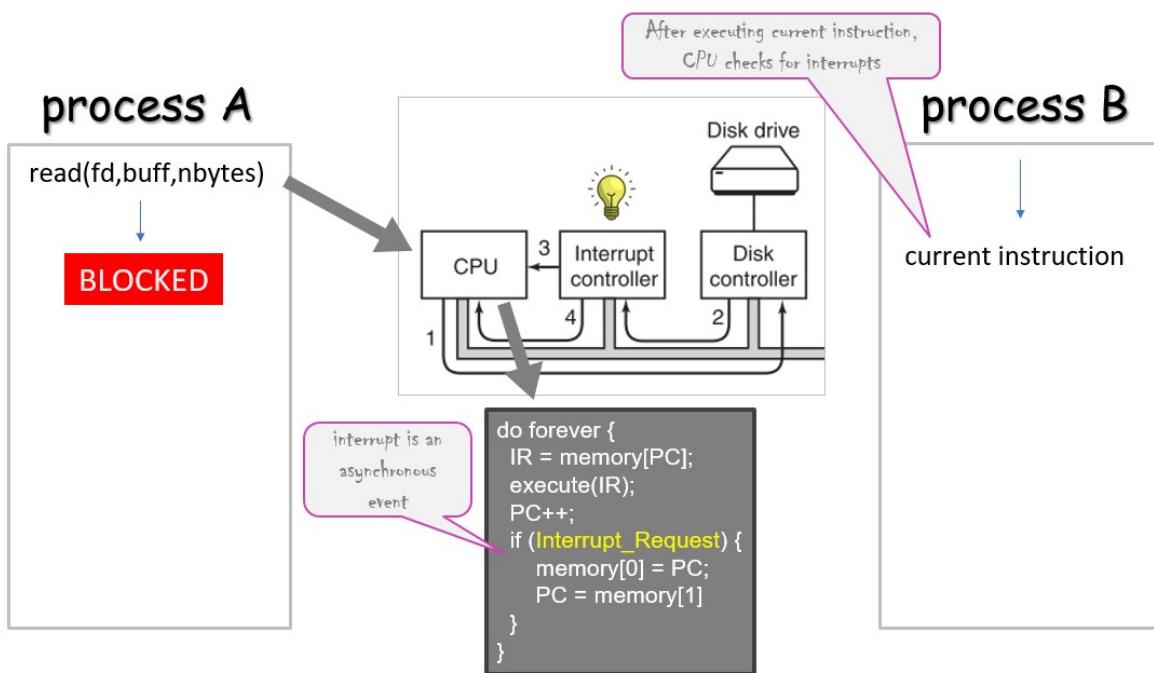
Steps in performing I/O



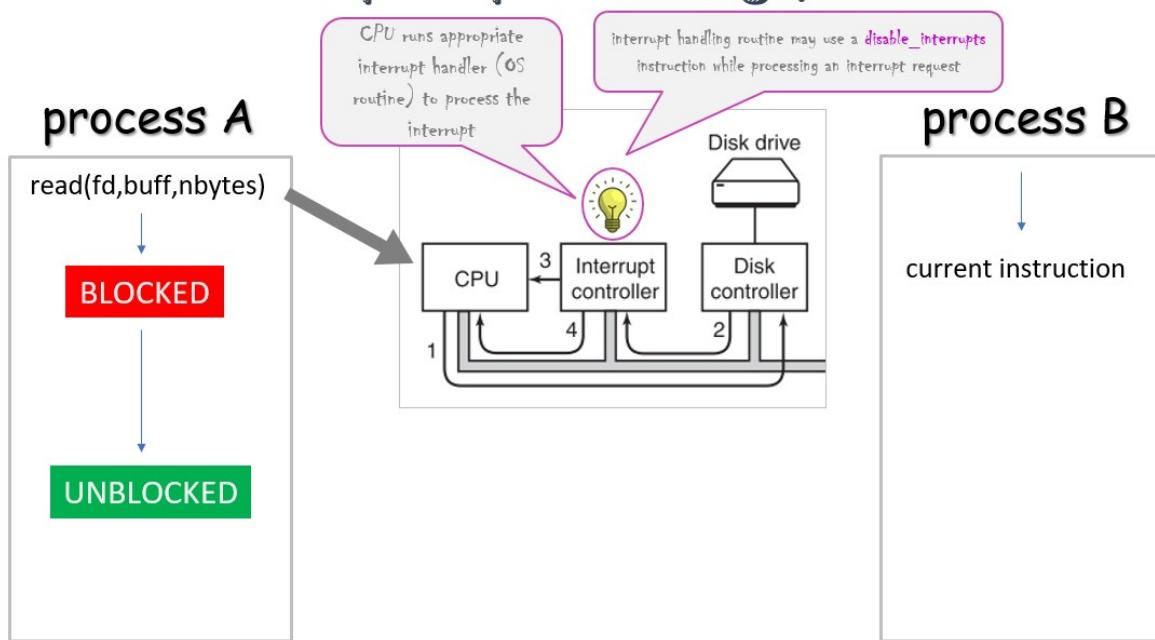
Steps in performing I/O



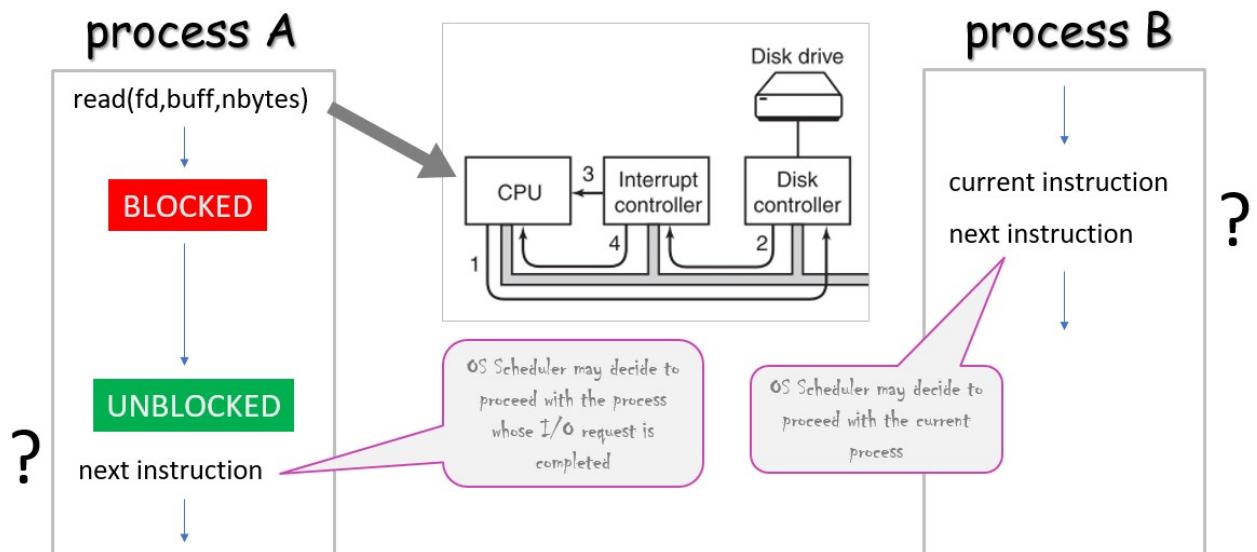
Steps in performing I/O



Steps in performing I/O



Steps in performing I/O



קריאה וכתיבה יכולות להתבצע במספר דרכים, אך הבסיסית ביותר היא דרך קריית מערכת. נדגים את התהיליך דרך תהיליך שורצה לקרוא בתים מתוך קובץ. קריית המערכת ניאשת למעבד שמעבד את הפניה ושולח בקשה דרך הדרייבר לבקר. כתוצאה מכך התהיליך נכנס למצב של `block`. לאחר מכן מנהל התקן מתחילה לבצע את הפעולה על הAKER המתאים, תוך כדי שהמעבד עובר לטפל בתהיליך אחר.

כאשר התקן סיים את הפעולה הוא שולח פסיקה לבקר הפסיקות (PIC) אשר שולח הטראה למעבד שהתרחשה פסיקה ורושמת את מספר התקן למעבד. לאחר ביצוע ההוראה הנוכחית המעבד בודק אחר פסיקות.

המעבד מרים את ה-`interrupt handler` המתאים (רוטינה של מערכת הפעלה) כדי לעבוד את הפסיקה. ניהול פסיקות יכול להשתמש בהוראה שאינה מאפשרת פסיקות נוספות.

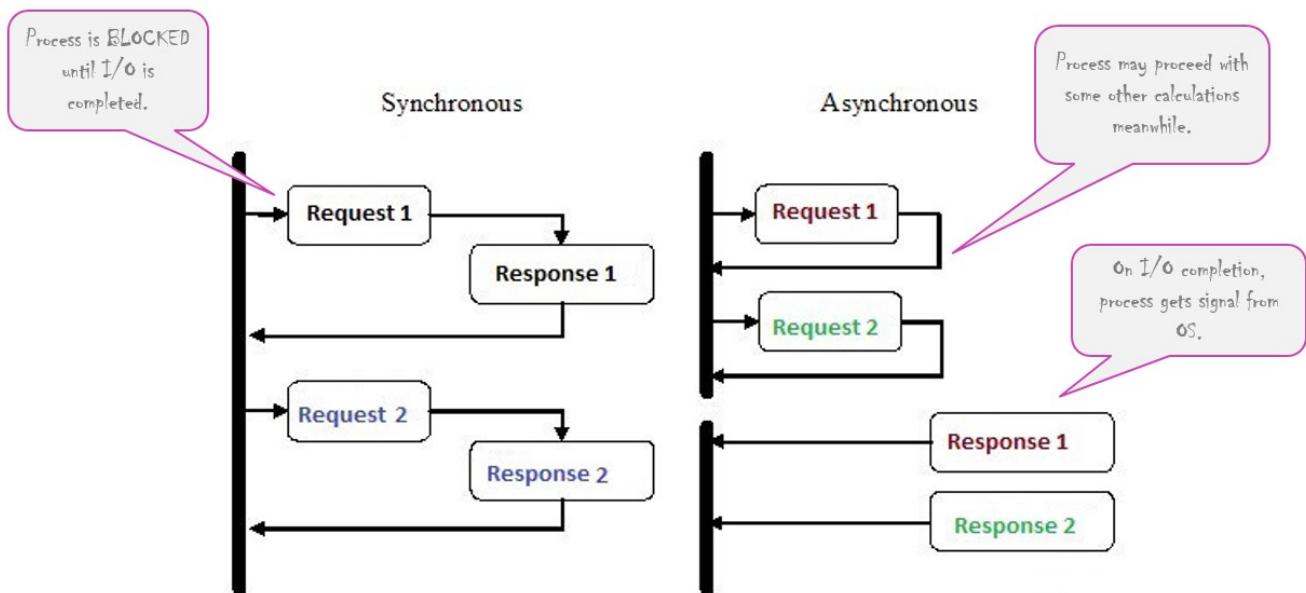
שאלה: האם התהיליך B ימשיך לרוץ או שמערכת הפעלה תעניק עדיפות לתהיליך A?

בפסודו-קוד שהתבוננו בו, המערכת בודקת האם קיימת פסיקה - כלומר אם הביטים דלוקים. זה משמש להתרבות מידית מצד מערכת הפעלה בדומה ל'interrupt'-ב-CPU כאשר ישנה פעולה לא חוקית כמו חלקה באפס. במקרה זה, ההתרבות יכולה להשפיע גם על תזמון התהיליכים.

עדיפות בין התהיליכים משתנה על פי מגבלות שונות, למשל תדריות קריאות IO. אנחנו נעמיק בכך בשלב מאוחר יותר - אך מערכת הפעלה יכולה לתת עדיפות לתהיליכים אשר מסמתקים יותר על התקני קלט-פלט. יתר על כן, מערכת הפעלה יכולה לתת זמן CPU לתהיליכים. אף על פי כן, כמתכנת אפשר להשפיע על התזמון דרך קריאות מערכת. לדוגמה, אם ברצוני לתת עדיפות לתוכנה שמחשבת את מספרי פיבונacci, ניתן להוסיף קריאות מערכת לא רלוונטיות כמו הדפסות, אם כי זה יגרום להארכת זמן הריצה.

למעשה, ישן מערכות הפעלה המשתמשות באלגוריתמי AI כדי ללמד את התהיליך בכל פעם שהוא רץ. לדוגמה, אם רצה ברקע גם אפליקציית WhatsApp, המערכת הייתה מעניקה לה עדיפות גבוהה יותר מאשר התהיליך שוחשב את מספרי פיבונacci. הסיבה היא ש-WhatsApp מבצעת יותר קריאות IO, ולכן המשתמש מתקשר אליה יותר. המערכת לומדת את ההתנהלות הזו במהלך ריצת האפליקציה אך מאבדת את המידע כאשר האפליקציה נסגרת.

Synchronous vs. Asynchronous I/O



קלט-פלט סינכרוני ואסינכרוני

סינכרוני (Synchronous):

במהלך פעולת קלט-פלט סינכרונית, התהיליך נתקע (BLOCKED) עד שהפעולה מושלמת.

אסינכרוני (Asynchronous):

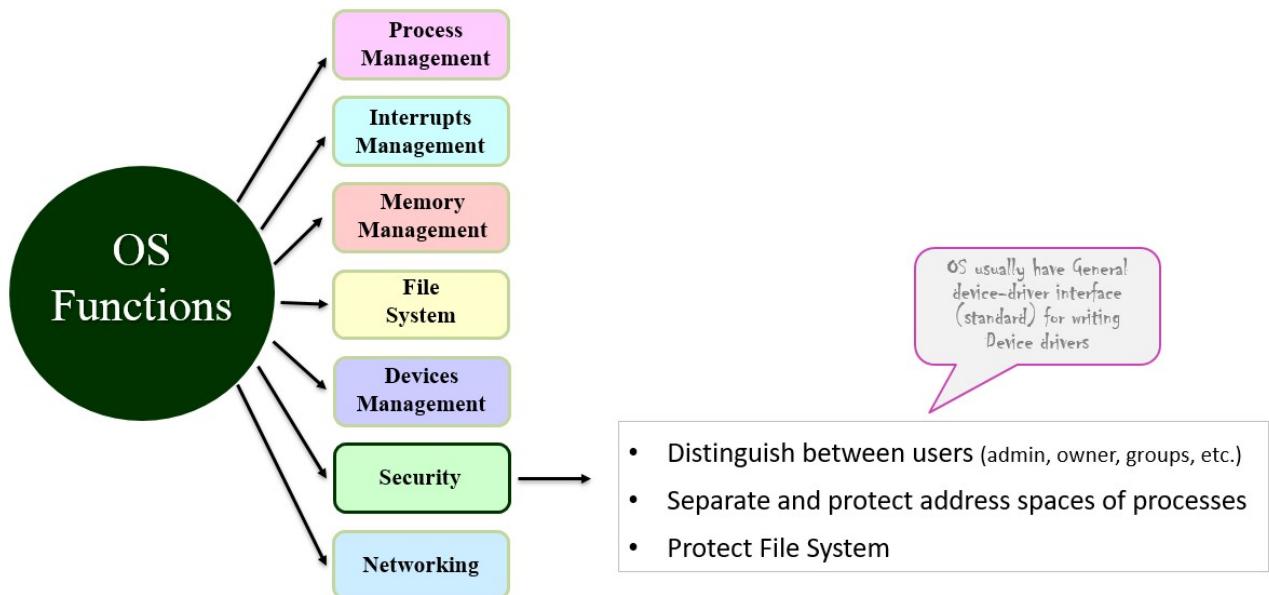
התהיליך יכול להמשיך ולבצע חישובים אחרים במהלך פעולת הקלט-פלט. כאשר הפעולה מושלמת, התהיליך מקבל סיגナル ממכלול הפעלה. זה אומר, למשל, שאם שלחנו בקשה לתקן, אנו יכולים להמשיך בפעולות אחרות

וכאשר ההתקן מגיב, מערכת ההפעלה מזירהיקה לנו סייגל. במקרה של wait suspending, אנו פונים למערכת ההפעלה באופן תדר ושאלים אם התרחש שינוי או התקבלה תגובה.

למעשה, דיברנו על כך בהקשר של קריאה וכתיבה לסקוט במסגרת הקורס SPL: הכוונה היא לתהיליך בו אנו שולחים בקשה, ממשיכים בתהליכיים אחרים שלנו וכאשר הבקשת מתוצאה, מערכת ההפעלה שולחת אלינו סייגל.

הבחירה איך להשתמש - באופן סינכרוני או אסינכרוני - תלויות בצריכים ובבחירה המתכנת.

OS – Key Functions



אנו בקושי נדוז בתקשות וbabtcha בקורס.

