מבוא

מערכת מחשב

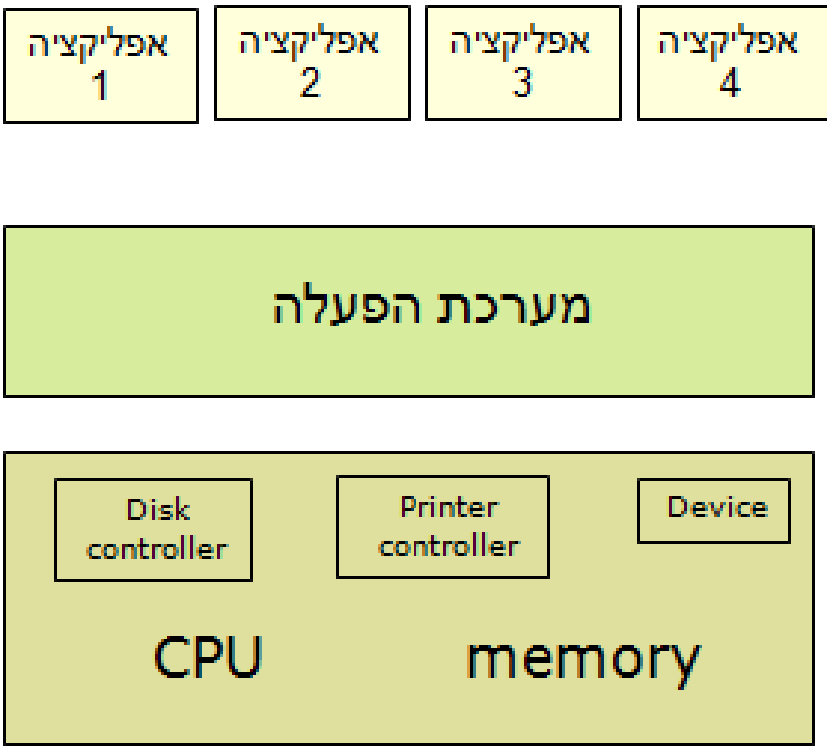
מערכת מחשב מתחלקת לארבעה חלקים :

1. השכבה "הנמוכה" , זו **החומרה** HARDWARE . החמרה מכילה: דיסק קשיח, ספק כוח, מקלדת, עכבר, מסך, מדפסת, מעבד, חיווט, כרטיסי מחשב, זכרון, מודם ועוד .

2. מעל החמרה, נמצאת **מערכת ההפעלה** OPERATING SYSTEM . זו תכנה, גדולה יחסית ומורכבת, המתווכת בין החמרה לתכניות היישום והמשתמשים השונים . היא מאפשרת שימוש מקבילי יעיל של רכיבי המחשב, ע"י ביצוע תאום בין צרכי התכנה, משאבי המחשב והנחיות המשתמשים .

3. מעל מערכת ההפעלה, נמצאת שכבת האפליקציות (**יישומים**) , APPLICATION PROGRAMS . הכוונה לקומפיילרים, מסדי נתונים, משחקים, תכנה לחישוב משכורות, לוח טיסות ועוד.

4. **המשתמשים** USERS, הם האנשים, המכונות, מחשבים אחרים וכד' , המשתמשים במערכת המחשב .



מהי מערכת הפעלה?

בעולם המחשבים אין כיום הגדרה אחת מוסכמת למושג מערכת הפעלה אך בגדול **מערכת הפעלה היא תוכנה המשמשת מתווך בין משתמש במחשב לחומרה של המחשב. או במילים אחרות, שכבת תכנה לניהול והסתרת פרטים של חומרת המחשב.**

יותר פשוט להגדיר מערכת הפעלה לפי מה עליה לבצע מאשר מהי .

שתי מטרות של מערכת ההפעלה שלעיתים מנוגדות זו לזו :

- **נוחות –** שיפור בביצועי המחשב מבחינת נוחות השימוש בו . (נכון למחשבים אישיים) .
- **יעילות –** מערכת ההפעלה משפרת את יעילות המחשב. לשם כך דרושה מערכת הפעלה מורכבת יותר ויקרה . (נכון למערכות גדולות) .

מה מבצעת מערכת ההפעלה ?

- **הקצאת / שיתוף משאבים –** למערכת ההפעלה יש מקורות רבים, חמרה ותכנה, הנדרשים לפתרון בעיה : זמן CPU (מעבד) , מקום בזכרון, מקום לאיחסון קבצים, התקני ק/פ וכד' . אפליקציה רוצה את כל המשאבים ומערכת ההפעלה היא זו שמחליטה אילו משאבים להקצות לאילו דרישות, כך שיהיה יעיל והוגן . מערכת ההפעלה נותנת לכל אפליקציה **אשליה** שכל המערכת היא שלה .
- **הגנה על החמרה –** מערכת ההפעלה משמשת כתכנית בקרה המבקרת את ביצועי תכניות המשתמשים כדי למנוע שגיאות . למשל, מונעת כתיבה למקום אסור בזכרון וע"י כך מונעת הרס מבני נתונים . או למשל דואגת לכך שאם תכנית נכנסת ללולאה אינסופית, תכניות אחרות יוכלו להמשיך לרוץ ללא בעיה ומבלי שייגרם להן נזק . מערכת ההפעלה מגינה על שטחי תכניות, זו מפני זו .
- **הגנה על המעבד -** מבוססת על ההבחנה בין שני מצבי ריצה: **מצב משתמש** – **USER MODE** ו**מצב מיוחס KERNEL MODE** **או PRIVILEGED MODE .** המעבד <u>תמיד</u> נמצא באחד משני המצבים האלה .
  - במצב המיוחס (גרעין) KERNEL – המעבד מבצע כל פקודה בתכנית .
  מערכת ההפעלה עובדת והמשתמש אינו יכול לבצע דבר .
  - במצב משתמש USER – המעבד לא יבצע פקודות מסוימות .

מערכת ההפעלה בוחנת למעשה, כל פעולה של תכנית ומחליטה לפי הנ"ל אם המעבד צריך לבצע או לא .
כמו כן, דואגת שכל התכניות ישתמשו במעבד עפ"י תור מסוים .
מערכת ההפעלה נקראת גם **MONITOR** .

סוגי מחשבים:

PC - מחשב בודד למשתמש בודד ,דגש על נוחות שימוש וביצועים סבירים.

MAINFRAME / MINICOMPUTER – משתמשים רבים על אותה חומרה דרך **מסופים – TERMINALS** . הדגש הוא על ניצול משאבים ואבטחה .

WORKSTATION – מחשב עם חומרה למשתמש אחד המחובר ומתקשר ברשת עם מחשבים נוספים. יש איזון בין ניצול משאבים ונוחות שימוש .

- תפקיד מערכת ההפעלה בהקשר הנ"ל הוא לנהל את השימוש בחומרה באמצעות התוכנות.

התפתחות מערכות הפעלה

- חומרה יקרה ואיטית, כוח-אדם זול
  Batch jobs , ניצול החומרה 24x7: IBM S/360
- חומרה יקרה ומהירה, כוח-אדם זול
  Unix :Interactive time-sharing
- חומרה זולה ואיטית, כוח-אדם יקר

  מחשב אישי לכל משתמש: MS-DOS
- חומרה זולה מאוד, כוח חישוב רב.
  ריבוי משימות: Windows NT, OS/2,
- ריבוי מעבדים וריבוי ליבות (multi-core)
  שיתוף משאבים בסיסי: דיסקים, מדפסות, ...
- רשתות מהירות.
  הרשת היא המחשב: SETI@home, Grid Computing

  הרשת היא אמצעי אחסון: SAN, Web storage

מנגנוני חמרה לתמיכה במערכת ההפעלה

- שעון חמרה .
- פעולות סנכרון אטומיות .
- פסיקות .
- קריאות מערכת הפעלה .
- פעולות בקרת קלט / פלט .
- הגנת זכרון .
- אופן עבודה מוגן .
- פעולות מוגנות .

החיים ללא מערכת הפעלה

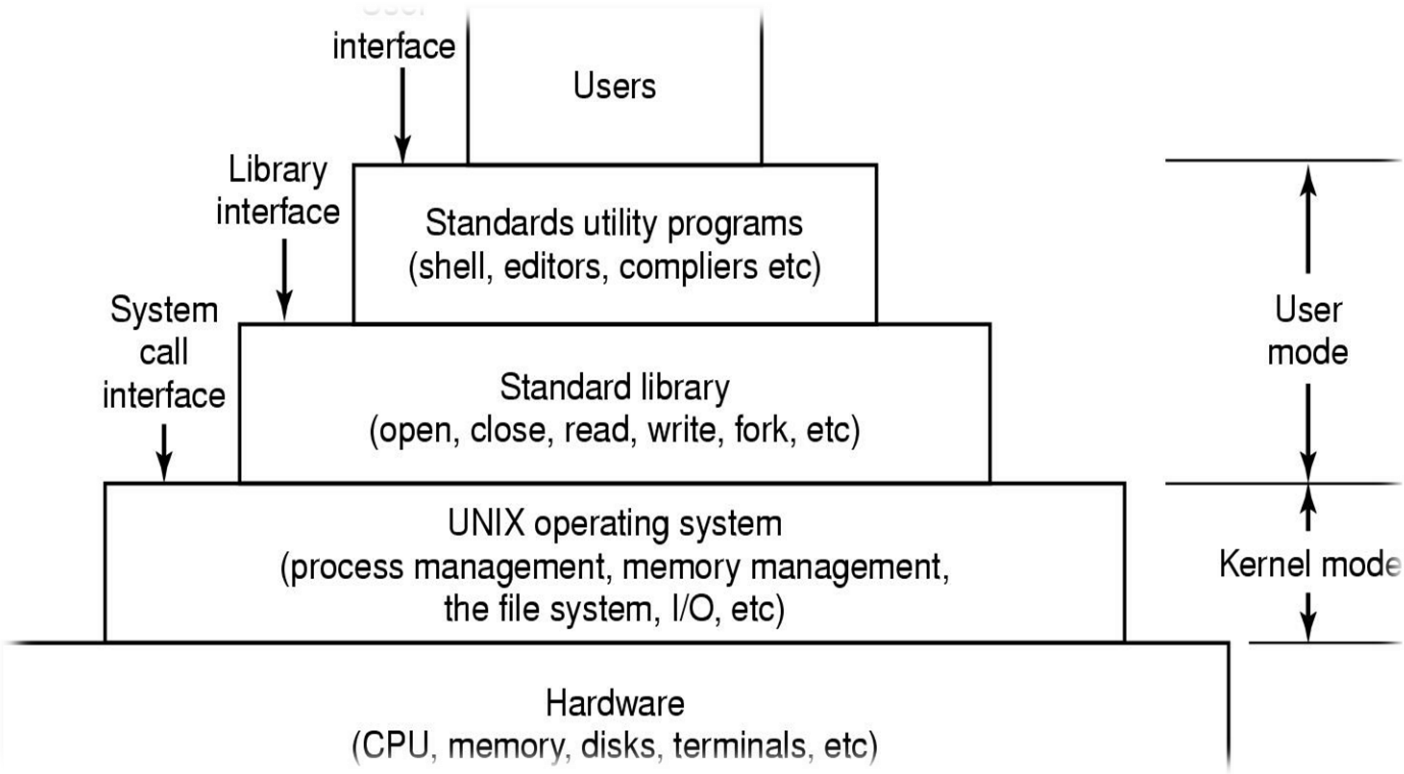
❖ **כל מתכנת היה חייב:**

1. להכיר את החומרה אותה הפעיל לעומק.
2. להיות מסוגל להגיע לחומרה ולנהל אותה פיזית.

❖ **כל תוכנית הייתה:**

1. צריכה להכיל קוד עבור ביצוע כל תהליך.
2. מבצעת טעות אחת או יותר בעת הריצה.

UNIX high-level architecture


interface
Users
Library
interface
Standards utility programs
(shell, editors, compliers etc)
System
call
interface
Standard library
(open, close, read, write, fork, etc)
UNIX operating system
(process management, memory management,
the file system, I/O, etc)
Hardware
(CPU, memory, disks, terminals, etc)
User
mode
Kernel mode

הגדרות

מושגי יסוד

KERNEL (גרעין מערכת) – התכנית הראשית של כל מערכת הפעלה. מכילה רשימות קוד לכל שימושי המערכת. **תמיד** נמצאת בזכרון הראשי .

BATCH FILE (קובץ אצווה) – קובץ המכיל שורות טקסט של פקודות למערכת ההפעלה, לשם הפעלתן של תכניות שונות.

BATCH PROCESSING (עיבוד באצווה) – שיטת עבודה במערכות מחשוב שבמסגרתה מתבצעות עבודות **לא מקוונות** באופן שנקבע מראש, ללא **עבודה אינטראקטיבית** . אופן הפעלת מערכת האצווה נקבע באמצעות **קבצי אצווה** הכוללים פקודות **בשפת תסריט** . אלו מערכות שבד"כ עתירות במשאבים ומשך הרצתן ארוך. הזמן בו מורצים עיבודי האצווה העיקריים נקרא **חלון אצווה** ובדרך כלל הוא בשעות הלילה (למשל גיבויים, עדכון חשבונות בנק וכד') .

SCRIPTING LANGUAGE (שפת תסריט) – זו שפת תכנות לכתיבת תסריטים (SCRIPTS). תסריט זו תכנית מחשב הנכתבת כדי למכן ביצוע משימות. הן נכתבות ומורצות מיד ללא צורך במהדר או מקשר . התכנה המריצה תסריטים נקראת מפרש .

PROGRAM (תכנית) – קובץ ממשי בשפת מכונה הנמצא על רכיב פיזי , למשל בדיסק .
(אפליקציה היא מקרה פרטי של program).

PROCESS או JOB (תהליך) – תכנית בעת ביצוע. כאשר תכנית שנמצאת על הדיסק נכנסת לריצה היא הופכת ל-Process.

רכיבי התהליך

1. **התכנה שרצה .**
2. **הנתונים עליהם רצה התכנית .**
3. **המשאבים שהתהליך צורך (זיכרון, קבצים וכד') .**
4. **מצב התהליך (ריצה, המתנה וכד') .**

TIME SHARING (חלוקת זמן) - כל תהליך מקבל ממערכת ההפעלה הזדמנות לרוץ.

MULTIPROGRAMMING (תכנות מרובב) –מחשב שבו רצות מספר תוכניות על אותה מערכת. התכניות לא רצות במקביל, אלא זמן ה-CPU מתחלק בין כל התכניות.

מאפיינים:

1. **מערכת ההפעלה מספקת טיפול בהתקני ק/פ .**
2. **ניהול זכרון – מערכת ההפעלה צריכה למקם מספר תהליכים בזכרון .**
3. **תזמון ה – CPU (המעבד) –מערכת ההפעלה צריכה לבחור תהליך אחד מבין מספר תהליכים שמוכנים לריצה.**
4. **מיקום התקנים .**

MULTITASKING – מערכות של **Time-sharing.** המחשב יודע לחלק את הזמן בין התהליכים בצורה מהירה מאוד. בנוסף הוא יודע לחלק כל תהליך למשימות. הדבר מאפשר לעבוד עם מספר משתמשים במקביל. זהו מקרה פרטי של **Multiprogramming**.

THREAD (חוט או תהליכון) – Thread זה חלק של תכנית שיכול לרוץ באופן עצמאי מהחלקים האחרים.
תהליך וחוט דומים .
בעיני מערכת ההפעלה , כל אחת מהתכניות הרצות היא **Process** (תהליך).
מערכות הפעלה מודרניות מאפשרות לנהל במסגרת ריצה של תהליך (Process) מספר תהליכונים הרצים במקביל במרחב כתובות אחד.
במערכות אלו כל תהליך חדש מתחיל את ביצועו באמצעות 'תהליכון ראשי' אשר עשוי בהמשך ליצור תהליכונים נוספים (כמו למשל בזמן קריאת קובץ או הדפסה). מנגנון הריצה באמצעות תהליכונים מאפשר לספק למשתמש במערכת ההפעלה מהירות תגובה ורציפות פעולה כאשר התהליך (יישום) מבצע מספר משימות במקביל .

MULTITHREADING (ריבוי חוטים) – מערכת הפעלה בה חלקי התהליכונים (**Threads**) יכולים לרוץ בו זמנית . למשל, תכנה אחת המאפשרת טיפול בריבוי משתמשים בו-זמנית , שרת web ועוד.
אם משתמשים רבים משתמשים בתכנה בו-זמנית, נוצר ומנוהל Thread עבור כל אחד מהמשתמשים . ה – Thread מאפשר לתכנית לדעת איזה משתמש מקבל שירות .

הערה : THREADS יכולים להיות ממומשים גם ברמת המשתמש (**USER**) אבל אז מערכת ההפעלה לא מודעת אליהם . מבחינת מערכת ההפעלה התהליכים הם Single threaded .

מושגים בשיטת עבודה

INTERACTIVE (עבודה אינטראקטיבית) - התכנה מגיבה לקלט מהמשתמש .
למשל, באמצעות הוראות או נתונים. (עבודה לא אינטראקטיבית, שאינה מגיבה לפקודות משתמש, היא למשל מהדר (Compiler) או עבודה באצווה (**Batch**) .

OFFLINE (עיבוד בלתי מקוון) – עיבוד נתונים באמצעות התקן שאינו מחובר אל המעבד המרכזי ישירות, כך שהוא אינו מצוי תחת תכנית בקרה.

ONLINE (עיבוד מקוון) – עיבוד המתבצע ע"י משתמשים העובדים בעבודה **אינטראקטיבית**
מול המערכת. עיבוד תנועות מקוון מאופיין במספר רב של משתשמשים העובדים במקביל.
בניגוד לעיבוד באצווה (**batch**), מספר המשתמשים בזמן נתון וסוג העבודה אינם צפויים מראש.
בעיבוד מקוון, יצירת הקלט מתבצעת במקביל להזנתו. כל תנועה שמתרחשת, עוברת מיד למחשב המרכזי .
דוגמא – מקסימום משיכות ליום בכספומט. עבודה ישירה עם קבצי הנתונים האמיתיים וביצוע בקרות בעת הזנת עדכון או שליפת המידע.

Real Time Systems

לרוב משתמשים במערכות אלו כאשר ישנו זמן מוגבל לביצוע פעולה. כאשר מגיע זרם הנתונים יש לעבד אותו במהירות. למערכת כזאת יש מערכת תזמון מאורגנת.

תהליכים PROCESSES

תהליך

- תהליך הוא תכנית בביצוע.
- תהליך צריך משאבים כגון: זמן CPU , זכרון , קבצים והתקני ק/פ (I/O) .
- המשאבים מוקצים לתהליך כאשר הוא נוצר או בזמן שהוא מתבצע .

מערכת מחשב מכילה אוסף של תהליכים : תהליכי מערכת ההפעלה המכילים קוד של מערכת ההפעלה – **System Code** , ותהליכים המכילים קוד משתמש – **User Code.**

מערכת ההפעלה אחראית על ניהול התהליכים : יצירת תהליך, מחיקת תהליך , ניהול תהליכי משתמש ומערכת , תזמון תהליכים , מנגנון הסנכרון בין התהליכים , תקשורת בין התהליכים ומניעת קפאון (Dead Lock) .

- תהליך הוא ישות אקטיבית. הוא יותר מאשר קוד התכנית . הוא מכיל את הפעילות המיוצגת ע"י **ערכים עכשוויים של מונה התכנית Program Counter** , תוכן הרגיסטרים , תוכן המחסנית , נתונים זמניים (כמו פרמטרים של פרוצדורה וכתובת חזרה) ואזור נתונים המכיל משתנים גלובליים .
- תכנית אינה תהליך ! תכנית היא ישות פסיבית כמו למשל תוכן של קובץ על הדיסק. בתכנית יכולים להיות מספר תהליכים . כמו למשל , משתמש המפעיל את תכנית העורך (editor) . כל שימוש בעורך הוא תהליך נפרד. למרות שאזור הטקסט זהה, הנתונים שונים .

תאור מסלול עבודה של תהליך

- כאשר תהליך מתבצע, הוא מחליף מצבים .
- מצב התהליך מוגדר לפי הפעילות של התהליך באותו רגע .
- כל תהליך , יכול להיות באחד המצבים הבאים :

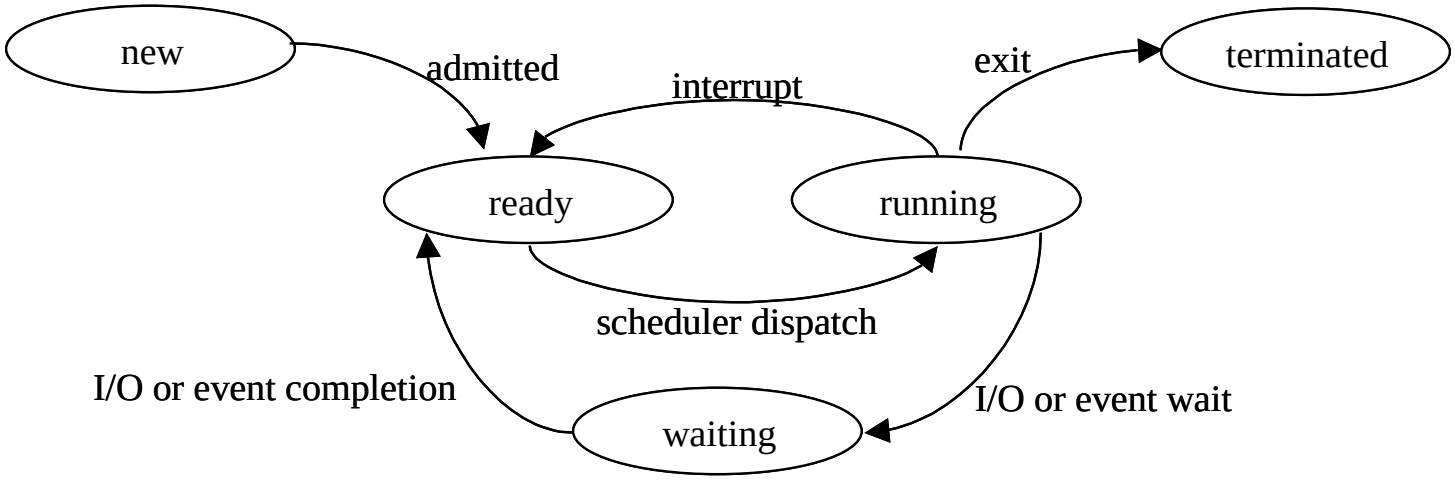
New – התהליך נוצר .

Ready – מצב מוכן

Running – מצב ריצה. ההוראות מבוצעות .

Waiting - התהליך ממתין לאירוע שיקרה . כמו למשל, השלמת פעולת I/O .

Terminated – התהליך מסיים את הביצוע .



- כל תהליך מיוצג במערכת ההפעלה ע"י **PCB** (Process Control Block) , המכיל מידע המשויך לתהליך מסוים .

PCB מכיל :

- **שם התהליך ID .**
- **מצב התהליך (אחד מהנ"ל) .**
- **PC – כתובת הפקודה הבאה לביצוע .**
- **תוכן האוגרים (Regsters) .**
- **מידע עבור תזמון ל – CPU : עדיפות התהליך, מצביעים לתורים (כדי שמערכת ההפעלה תחליט מי ירוץ בכל רגע) .**
- **מידע אודות ניהול הזיכרון: תוכן האוגרים Base ו – Limit , טבלאות דפים וטבלאות סגמנטים .**

- **Account – מידע אודות החשבון : זמן שימוש ב – CPU , מגבלות זמן , מה התהליך קבל בעבר .**
- **מידע אודות מצב I/O – רשימת התקני I/O שהוקצו לתהליך, רשימת קבצים פתוחים, למי שייך כל קובץ לאילו קבצים התהליך רשאי לגשת (יש טבלה המכילה את רשימת הקבצים . ה – PCB מכיל מצביע לטבלה זו . .**
- **USER – משתמש – מי המשתמש שמריץ את התהליך (מגדיר הרשאות גישה לקבצים).**

Process Scheduling - תזמון תהליכים .

המטרה של Multiprogramming היא שכל הזמן ירוצו תהליכים כך שיהיה ניצול מקסימלי של ה – CPU .
המטרה של Timesharing היא להעביר את ה – CPU בין התהליכים בתדירות גבוהה .

במערכת מחשב עם CPU אחד, יכול להיות תהליך אחד בלבד ב – CPU . התהליכים האחרים , ממתינים בתור עד שיקבלו את ה – CPU .

כאשר תהליך נכנס למערכת, הוא נכנס **לתור התהליכים** ה - **Job Queue.** תור זה מכיל את כל התהליכים במערכת. תהליכים שנמצאים בזיכרון הראשי ומוכנים לריצה או ממתינים לריצה נשמרים ברשימה הנקראת **תור המוכנים Ready Queue.** רשימה זו נשמרת לרוב במבנה נתונים של רשימה מקושרת. ראש הרשימה יכיל מצביעים ל-PCB הראשון והאחרון ברשימה (כל PCB יכיל בנוסף מצביע ל-PCB הבא אחריו).

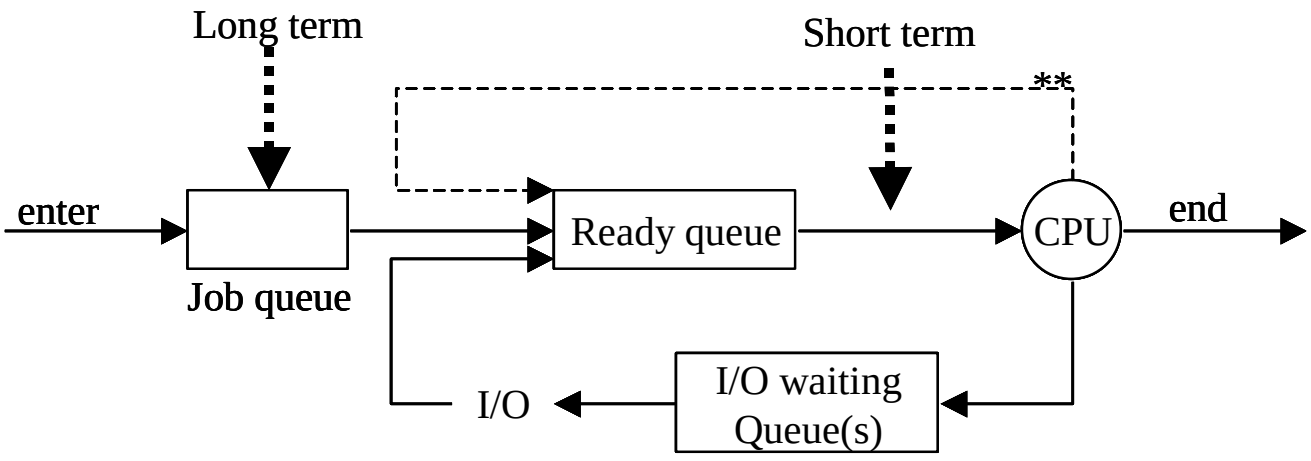
כל זמן שהתהליך קיים, הוא נודד בין מספר תורי-תזמון. מערכת ההפעלה חייבת לבחור תהליכים מתוך התורים, ומשתמשת לצורך כך בשני מתזמנים:

1. Long Term Scheduler – (נקרא גם **job scheduler**) . המתזמן של התור **Job Queue**. מחליט אילו תהליכים יעברו למצב **Ready**. מקור השם הוא בכך שהדגימות בתור זה נעשות בדחיפות נמוכה. גישה בטווחים של שניות או אפילו דקות.

   שולט בכמות התהליכים שמוכנים להריץ במקביל.

2. Short Term Scheduler – (נקרא גם **CPU Scheduler**). המתזמן של התור **Ready Queue**. מחליט מי מהתהליכים יקבל את ה-CPU. הדגימות מאוד מהירות. מידע יוצא ונכנס מהזיכרון בצורה מהירה מאוד.

   ההבדל העיקרי בין שני המתזמנים הוא תדירות הגישה שלהם.



חשוב מאוד לשים לב לצורה בה בוחר ה-**Long Term Scheduler** את התהליכים. באופן כללי, רוב התהליכים מוגדרים כ-**I/O Bound** או **CPU Bound**:

1. **I/O Bound Process** – תהליך שכל הזמן זקוק ל-I/O. תהליך מסוג זה נמצא הרבה במצב **המתנה** **-Waiting** .
2. **CPU Bound Process** – תהליך שצריך כל הזמן פעולות CPU.

- המערכת בעלת הביצועים הטובים ביותר תהיה בעלת מתזמן הדואג לשלב את שני הסוגים בתהליכים אותם הוא בוחר.

Context Switch - החלפת הקשר – החלפת ה – CPU לתהליך אחר . החלפה זו, דורשת שמירת המצב הקיים עבור התהליך הישן (זה שיוצא) וטעינת המצב החדש (של התהליך שנכנס) .
בזמן הזה, המערכת לא עובדת אלא מתעסקת בחילוף . זה נקרא **תקורה** **Overhead** .

Terminated - כאשר תהליך סיים את עבודתו, הוא שולח בקשה למערכת ההפעלה להפסיק אותו , ע"י שימוש ב – Exit System Call . כל משאבי התהליך (קבצים פתוחים, I/O buffers ועוד) משוחררים ע"י מערכת ההפעלה.

<u>תזמון תהליכים PROCESS SCHEDULING</u>

מרכיבי מערכת ההפעלה

תזכורת – מערכת MULTIPROGRAMMING צריכה לבצע את הדברים הבאים:

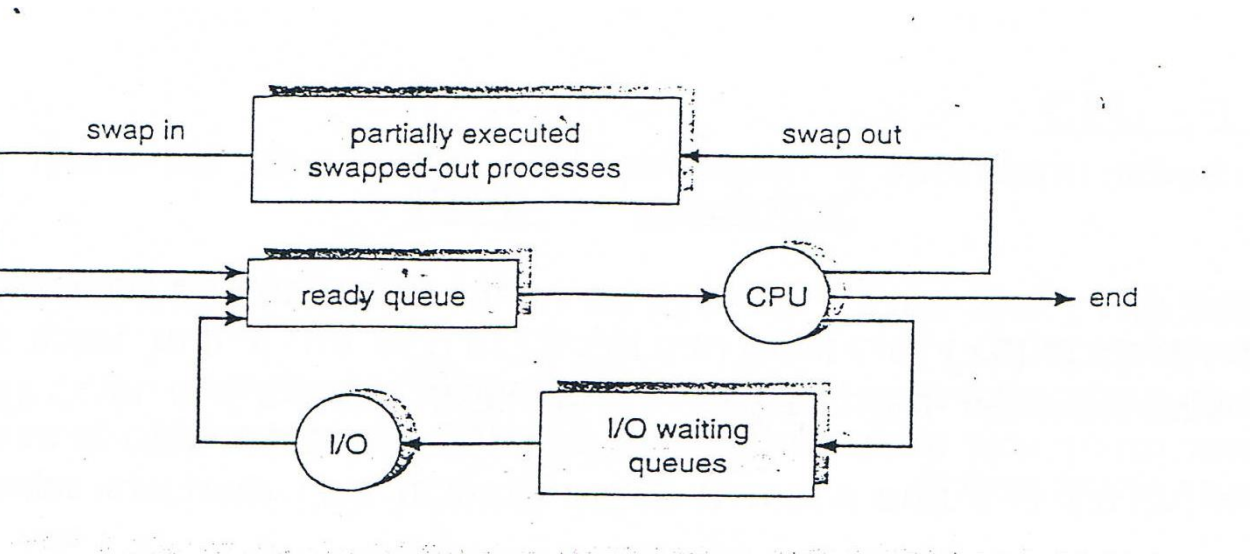
- **ניהול תהליכים .**
- ניהול זיכרון .
- ניהול מערכת הקבצים .
- ניהול פסיקות .
- ניהול התקני קלט / פלט .
- תקשורת בין תהליכים .
- ניהול התקנים כלליים – מניעת קיפאון .
- מערכת הגנה .

<u>ניהול תהליכים</u> – תהליך שרץ במחשב זקוק למספר משאבים: זמן CPU, זיכרון, קבצים והתקני I/O. התהליך מקבל משאבים אלו ומחזיר אותם בסיום למערכת ההפעלה.

מערכת ההפעלה דואגת ל-

- יצירה ומחיקה של תהליכים.
- השעיה או הפעלה מחדש של תהליכים.
- מנגנון תזמון בין תהליכים – סנכרון ותקשורת בין התהליכים.

מסלול העבודה של תהליך -



SWAPPING

הוצאת תהליך החוצה לזכרון או מהזכרון החוצה כי הזכרון מלא ואז מוציאים את כל התהליך בבת אחת לדיסק. רצוי להוציא ל-swap disk (דיסק שמיועד לכך, מהיר יותר וגישה מהירה יותר).

SWAP OUT - פעולת ההוצאה .

SWAP IN – החזרת התהליך לתור המוכנים – READY QUEUE.

<u>הנחות יסוד -</u>

המטרה של MULTIPROGRAMMING היא להחזיק מספר תהליכים רצים במשך כל הזמן, בכדי לנצל את ה-CPU באופן מקסימאלי. במערכות UNIPROCESSOR רק תהליך אחד יכול לרוץ. יתר התהליכים חייבים להמתין עד שה-CPU מתפנה.

- במערכת מחשב פשוטה, כאשר התהליך צריך להשתמש ב I/O, ה-CPU יעבור למצב של **IDLE (סרק)**. כלומר, כל זמן ההמתנה הזה מבוזבז.
- ב MULTIPROGRAMMING רוצים לא לבזבז זמן זה ולהשתמש בו בצורה יעילה. מספר תהליכים נשמרים בזיכרון באותו זמן, וכאשר תהליך נמצא במצב המתנה, מערכת ההפעלה מעבירה את ה-CPU מהתהליך ונותנת אותו לתהליך אחר.

❖ תזמון הינו אחת מהפעולות העיקריות של מערכת ההפעלה. כמעט כל משאבי המחשב מתוזמנים לפני שימוש.

ריצת תהליך

ריצת תהליך מורכבת ממעגל של ריצה ב-CPU והמתנה ל-I/O, כאשר כל הזמן התהליך עובר בין השניים. כמו כן, **התהליך מתחיל ב-CPU ומסתיים בו**.

תוכנית המבוססת בעיקר על I/O סביר שהשימוש ב – CPU יהיה קצר, ואילו תוכנית המבוססת על CPU ישתמש זמן רב ב - CPU.

❖ הצלחת מתזמן ה-CPU תלויה למעשה בריצת התהליך.

CPU Scheduler – מתזמן ה – CPU

ברגע שה-CPU נכנס למצב של סרק, מערכת ההפעלה חייבת לבחור תהליך מתוך תור המוכנים, ה- READY QUEUE , ולתת לו לרוץ על ה-CPU.

החלטות המתזמן עשויות לקרות באחד מארבעת המצבים הבאים :

1. כאשר תהליך עובר ממצב ריצה למצב המתנה.
2. כאשר תהליך עובר ממצב ריצה למצב ready.
3. כאשר תהליך עובר ממצב המתנה למצב ready.
4. כאשר תהליך מסתיים.

במקרים **1,4** אין אפשרות בחירה. התהליך מסתיים מיוזמתו, ותהליך חדש חייב להיבחר מתוך התור. מקרים מסוג זה נקראים **NON-PREEMPTIVE** (**מדיניות ללא-הפקעה**).

NON-PREEMPTIVE (**מדיניות ללא-הפקעה**) - המשמעות היא שברגע שה-CPU הוקצה לתהליך, התהליך שומר על ה-CPU עד שהוא משחרר אותו, ולא ייתכן מצב שמערכת ההפעלה תפריע לו באמצע, או תעצור אותו ותיתן עדיפות לתהליך אחר. זוהי מערכת שאדישה לשינויים.

❖ **תהליך בעל עדיפות גבוהה יותר לא יכול להפריע לתהליך שכבר רץ.**

עבור מקרים **2,3** קיימת אפשרות בחירה. מקרים אלו נקראים **PREEMPTIVE (מדיניות עם הפקעה)**.

PREEMPTIVE (מדיניות עם הפקעה) - קיימת עדיפות לתהליך. מערכת מסוג זה אינה אדישה לשינויים, והיא בודקת כל הזמן את מצבי התהליכים ועשויה להפסיק תהליך אחד עבור תהליך בעל עדיפות גבוהה יותר.

תזמון מסוג זה גורם לעלויות – זמן ה-**CONTEXT SWITCH**. בעיה נוספת במערכת מסוג זה היא הסנכרון.

DISPATCHER

זהו מודול המעביר את הבקרה של ה-CPU לתהליך שנבחר ע"י ה-short-term scheduler.

פעולה זו כוללת:

1. **CONTEXT SWITCHING**.
2. **מעבר ל –USER-MODE** .
3. **קפיצה למקום הנכון בתוכנית המשתמש במטרה להתחיל להריץ אותה.**

❖ **מודול זה צריך להיות מהיר ככל האפשר**, שכן הוא נקרא בכל פעם שמחליפים תהליך.

מדדים להערכת שיטות התזמון

להלן קריטריונים המשווים בין אלגוריתמים של שיטות תזמון. כאשר מחליטים איזה אלגוריתם לבחור, יש להשוות בין מאפייני האלגוריתמים השונים .

- **ניצול CPU** – השאיפה היא שה – CPU יעבוד באופן רציף כל הזמן ויהיה עסוק כל הזמן. ניצול ה-CPU יכול לנוע מ-0 אחוז ל-100 אחוז. במערכת אמיתית הטווח צריך להיות בין 40 אחוז ל-90 אחוז. לכן מערכות עובדות ב – MULTITASKING .

- **זמן סבב (Turnaround)** – זמן שהיית התהליך במערכת . זהו המרווח בין הזמן שהתהליך ביקש לרוץ לבין הזמן שהתהליך הסתיים .

❖ **זמן סבב הוא סכום משך הזמנים שהתהליך בזבז בהמתנה לזיכרון, המתנה בתור ready, ריצה ב-CPU והפעלת I/O.**

- **תפוקה (Throughput)** – אם ה - CPU עסוק בהרצת תהליכים, הרי שהאלגוריתם בצע את עבודתו. אחת הדרכים לבדוק את עבודת האלגוריתם היא מספר התהליכים שסיימו ביחידת זמן אחת .

  התפוקה עולה כאשר נצילות ה – CPU גדלה וזמן השהייה יקטן.

- **זמן המתנה (WAITING TIME)** – האלגוריתם אינו משפיע על הזמן בו תהליך רץ או משתמש ב-I/O, אלא רק על הזמן שתהליך מעביר בהמתנה בתור ה-ready.

❖ **זמן ההמתנה הוא סכום משך הזמנים הכולל שהתהליך העביר בתור ה-ready.**

- **זמן תגובה (RESPONSE TIME)** – במערכת אינטראקטיבית (ON LINE), זמן הסבב אינו יכול להיות קריטריון מתאים, משום שלעיתים קרובות, תהליך יכול לספק חלק מהפלט די מהר, ולהמשיך בחישובים אחרים בזמן שחלק מהפלט כבר מוצג למשתמש. לכן אמת מידה נוספת היא הזמן שלוקח **מרגע הבקשה ועד לתגובה הראשונה**.
  - ❖ **זמן התגובה הינו משך הזמן שלוקח למערכת להתחיל להגיב, אבל לא כולל את זמן התגובה עצמו.**

- **קריטריון חשוב נוסף אשר נלקח בחשבון הוא הרעבה – STARVATION . זו תופעה לא רצויה שבה תהליך ממתין לאירוע אשר לא יכול להתרחש .**

המטרה היא:

- להביא את ניצול ה-CPU ואת התפוקה למצב מקסימלי .
- למזער כמה שיותר את זמן הסבב, זמן ההמתנה וזמן התגובה.

ברוב המקרים מנסים לייעל את הממוצע של כל הקריטריונים, אך קיימים מצבים בהם רצוי לייעל את ערכי המקסימום או המינימום ולא את הממוצע. לדוגמא, כאשר רוצים להבטיח שכל המשתמשים יקבלו שירות טוב, נרצה להקטין את זמן התגובה המקסימלי.

שיטות תזמון Scheduling Algorithms

FCFS - First Come, First Served

ה-CPU מוקצה לתהליך הראשון שביקש אותו. הדרך הפשוטה ביותר לממש אלגוריתם זה היא בעזרת תור FIFO. ברגע שתהליך נכנס ל-ready queue, ה-PCB שלו מקושר לזנב התור. כאשר ה-CPU פנוי, הוא מוקצה לתהליך שנמצא בראש התור.

זמן ההמתנה הממוצע במקרה זה הוא לרוב **ארוך**, והוא תלוי בזמן ההגעה של התהליכים.

דוגמה:

Process	Burst time
P1	24
P2	3
P3	3

- **BURST TIME – (פרץ הזמן) –** משך הזמן הרציף של CPU שתהליך דורש .
  מושפע ממהירות ה – CPU , גודל הזכרון , אורך הקלט, סוג הקלט/פלט) .

אם התהליכים הגיעו על פי הסדר הבא: P1, P2, P3 נקבל את המצב הבא:

זמן המתנה ממוצע (WAITING TIME):

זמן ההמתנה של P1 יהיה 0 (משום שהוא התחיל מיד) .

זמן ההמתנה של P2 יהיה 24.

וזמן ההמתנה של P3 יהיה 27.

זמן ההמתנה הממוצע שמתקבל , הוא 17 = 3 / (0+24+27).

זמן סבב ממוצע (TURNAROUND) :

הזמן ש - P1 היה במערכת הוא 24 .

הזמן ש – P2 היה במערכת הוא 27 .

הזמן ש - P3 היה במערכת הוא 30 .

זמן סבב ממוצע שמתקבל, הוא (24+27+30) / 3 = 27

שאלה: מה היה קורה אם התהליכים היו מגיעים בסדר P2,P3,P1 ?

לסכום:

- **השיטה קלה למימוש (הפרמטר היחיד שמובא בחשבון הוא זמן ההגעה).**
- **האלגוריתם הוא NON-PREEMPTIVE (מרגע שה – CPU הוקצה לתהליך, התהליך מחזיק בו כל זמן שהוא צריך) .**
- **אלגוריתם זה בעיקר בעייתי במערכות time-sharing .**
- **באלגוריתם זה אין הרעבה . תתכן הרעבה רק אם היתה שגיאה והתהליך נכנס ללולאה אינסופית .**
- **האלגוריתם לא טוב לתהליכים בעלי BURST TIME קצר . הם מעוכבים ע"י תהליכים בעלי BURST TIME ארוך .**

SJF - Shortest Job First

אלגוריתם זה מקשר לכל תהליך את ה BURST TIME הבא שלו.

- ברגע שה-CPU נגיש, הוא מוקצה לתהליך בעל BURST TIME הבא **הקצר ביותר**.
- אם לשני תהליכים יש את אותו אורך BURST TIME, יעשה שימוש באלגוריתם FCFS לבחירת התהליך הראשון.

❖ שימו לב !!! **האלגוריתם לא מתייחס לזמן הכולל הנדרש לתהליך על ה-CPU, אלא רק לפרץ ה-CPU הבא.**

זהו אלגוריתם אופטימלי הנותן **זמן המתנה מינימלי.**

דוגמה:

Process	Burst time
P1	6
P2	8
P3	7
P4	3

שימוש באלגוריתם זה יתזמן את התהליכים בסדר הבא: P4, P1, P3, P2.

זמן המתנה ממוצע (WAITING TIME):

זמן ההמתנה של P4 יהיה 0 .

זמן ההמתנה של P1 יהיה 3.

זמן ההמתנה של P3 יהיה 9.

זמן ההמתנה של P2 יהיה 16.

זמן ההמתנה הממוצע יהיה (3+16+9+0) / 4 = 7

<u>שאלה</u>: מה היה זמן ההמתנה הממוצע אם שיטת התזמון היתה FCFS ?

זמן סבב ממוצע (TURNAROUND) :

הזמן ש - P4 היה במערכת הוא 3.

הזמן ש - P1 היה במערכת הוא 9.

הזמן ש – P3 היה במערכת הוא 16 .

הזמן ש - P2 היה במערכת הוא 24 .

זמן סבב ממוצע שמתקבל, הוא (3+9+16+24) / 4 = 13

<u>שאלה</u>: מהו זמן הסבב הממוצע אם שיטת התזמון היא FCFS ?

דוגמה:

יש לחשב זמן סבב ממוצע וזמן המתנה ממוצע במערכת הבאה שבה מבוצע אלגוריתם SJF .

Process	Burst time
P1	24
P2	3
P3	3

<u>לסכום:</u>

- **זו שיטה תיאורטית שאינה ניתנת למימוש כי אי אפשר לחשב מראש את ה – BURST TIME .**
- **תור המוכנים (READY) ממוין לפי BURST TIME בסדר עולה . מהפרץ הקצר לפרץ הארוך .**
- **תתכן הרעבה כי תהליך נכנס לפי תורו ולא נכנס לסוף התור .**
- **לשיטה זו יש שתי גרסאות: PREEMPTIVE ו – NON PREEMPTIVE .**

הדוגמה הראשונה היא NON PREEMPTIVE . תהליך נכנס לתור לפי ה BURST TIME ומקבל את מלוא הזמן. אם מגיע תהליך חדש עם BURST TIME יותר קצר מזה

שב – CPU , הוא ייכנס לתור במקום המיועד לו וימתין שהתהליך ב – CPU יסיים את עבודתו .

באלגוריתם PREEMPTIVE , תהליך כזה, יפקיע את ה – CPU מהתהליך שרץ ויקצה את ה – CPU לתהליך החדש .

אלגוריתם כזה, נקרא גם Shortest remaining time first (SRTF) .

- **בשתי השיטות תתכן הרעבה !!!**

<u>שעורי בית</u>

1. להלן מערכת :

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
P4	3	5

יש לחשב את זמן הסבב הממוצע ואת זמן ההמתנה הממוצע :

א. בשיטת FCFS .

ב. בשיטת SJF , NON PREEMPTIVE .

ג. בשיטת SJF , PREEMPTIVE (SRTF) .

2. בשיטת SJF , אם ניתן היה להודיע על ה – BURST TIME , האם היה כדאי להודיע על זמן זה כקצר יותר מהזמן האמיתי, כארוך יותר או שהיה כדאי לא לרמות את המערכת ? הסבר !

תזמון תהליכים PROCESS SCHEDULING המשך

תזמון לפי עדיפות - PRIORITY SCHEDULING

- באלגוריתם זה, כל תהליך שנוצר, מקבל **עדיפות** . זהו מספר שלם .
- ככל שהמספר קטן יותר, העדיפות של התהליך גדולה יותר .
- בד"כ העדיפות ניתנת לפי ה – USER שיצר אותו או באופן מפורש.
- עדיפות זו, קובעת את מיקום התהליך בתור .
- ה – CPU מוקצה לתהליך בעל העדיפות הגבוהה ביותר. אם לשני תהליכים יש אותה העדיפות, הבחירה תהיה לפי אלגוריתם FCFS .

קיימים שני סוגי עדיפויות:

- **עדיפות סטטית STATIC PRIORITY** - תהליך נשאר עם אותה עדיפות מרגע ההיווצרות ועד אשר הוא מסתיים .
  אחת הבעיות באלגוריתם מסוג זה היא **הרעבה** – מצב שבו תהליך שמוכן לרוץ אינו מקבל את ה-CPU (**מערכת לא מסוגלת לזהות הרעבה**). תהליך שנמצא במצב של המתנה נחשב **לחסום – BLOCKED**.

- **עדיפות דינמית – DYNAMIC PRIORITY –** העדיפות משתנה . משתמשים בשיטת ה – AGING . שיטה שבה מגדילים את העדיפות של תהליך ככל שזמן ההמתנה שלו גדל (הפז"מ) . כל פרק זמן קבוע, מעלים את העדיפות של כל התהליכים בתור ב – 1 או עפ"י נוסחה מסוימת . שינוי זה מבטיח שתורו של כל תהליך יגיע .

עדיפות עם הפקעה – (PREEMPTIVE)

בכל פרק זמן קבוע, בודקים את התהליך שבראש התור . אם עדיפותו גבוהה מהעדיפות של התהליך שרץ ב – CPU , מתבצעת החלפה : והתהליך שרץ, חוזר לתור המוכנים עם אותה עדיפות שהייתה לו והתהליך בעל העדיפות הגבוהה, עובר למצב ריצה .

עדיפות ללא הפקעה – (NON PREEMPTIVE)

תהליך שנבחר לרוץ ב – CPU , ירוץ מבלי שה CPU יילקח ממנו בגלל תהליך בעל עדיפות גבוהה יותר .

דוגמה :

להלן מערכת :

PROCESS	ARRIVAL TIME	PRIORITY	BURST TIME
P1	0	5	2
P2	1	2	4
P3	2	4	3
P4	3	1	2

יש לחשב את זמן הסבב הממוצע ואת זמן ההמתנה הממוצע :

א. בשיטת עדיפות ללא הפקעה .

ב. בשיטת עדיפות עם הפקעה .

ויסות מעגלי - ROUND-ROBIN SCHEDULING (RR)

אלגוריתם זה עוצב במיוחד למערכת TIME-SHARING.
בשיטה זו, כל תהליך מקבל פרק זמן מעבד קצר, יחידת זמן הנקראת **קוואנטום** או **פלח זמן**), בדרך כלל בין 10 ל – 100 מילישניות. ברגע שהזמן עבר, התהליך מעבר לסוף התור והתהליך שבראש התור מקבל את ה – CPU .

- מתייחסים לתור המוכנים (READY) כאל תור מעגלי .
- לכאורה, זהו בעצם תור FCFS עם זמן מוגבל לכל תהליך .
- תהליך חדש, מתווסף **תמיד** לסוף התור .

ייתכנו שני מקרים:

1. התהליך צריך זמן קצר יותר מה time quantum המוקצה לו . במקרה כזה, התהליך משחרר ביוזמתו את ה – CPU והמתזמן בוחר את התהליך הבא בתור.

2. התהליך זקוק ל זמן ארוך יותר מה - time quantum המוקצה לו . במקרה זה, ה – timer יפעיל פסיקה . יופעל שינוי הקשר (context switch) . התהליך יעבור לסוף התור והמתזמן יבחר את התהליך הנמצא בראש התור .

קביעת גודל ה - time quantum

- **בחירת** time quantum **גדול מידי** – השיטה הופכת ל – **FCFS** כי כך כל תהליך מתבצע עד לסיומו , לפי סדר ההגעה . כלומר, זמן התגובה – ארוך .

- **בחירת** time quantum **קטן מידי**, יהיו החלפות רבות (context switch) ונקבל מערכת **לא יעילה** שזמן רב מתבזבז בה על תקורה (בזמן זה, ה – CPU יהיה במצב סרק idle). כמו כן, תהליך יעבוד מעט זמן ואז ימתין לתורו סבב שלם בתור המוכנים . כלומר, מתקבלת תפוקה נמוכה .

- **יש לבחור** time quantum שייתן ביצועים מיטביים לזמן תגובת מערכת קצר ככל שניתן ובו בזמן , תפוקת תהליכים גבוהה ככל שניתן .
הנהוג הוא בחירת time quantum גדול מספיק , כך שלפחות 80% מפרצי ה – CPU הדרושים, יהיו קטנים ממנו (כלומר, כ – 80% מהתהליכים יוכלו להיות במצב ריצה מבלי שה – CPU יופקע מהם בגלל סיום פלח הזמן) .

יתרונות השיטה

1. בשיטה זו **אין הרעבה** .
2. שיטה טובה **לתהליכים קצרים** .
3. שיטה **הוגנת** .
4. שיטה **פשוטה** .

חסרונות השיטה

1. בחירת time quantum מתאים למערכת . לשם כך, יש לבדוק היסטוריה של המערכת
(מונה ב – PCB).
2. השיטה אינה טובה לתהליכים ארוכים אשר ה – CPU מופקע מהם פעמים רבות .
3. השיטה אינה טובה לתהליכים כאשר הם נעצרים לקראת סיום .

סיבות לא לסיים את ה - time quantum

- הסתיים ה – burst time ועוברים להמתנה ל – I/O .
- התהליך הסתיים .
- הזיכרון של התהליך נלקח ממנו ומורידים אותו זמנית לדיסק SWAP OUT כדי להעלות תהליך אחר במקומו (SWAP IN) .

תרגילים:

1. יש לחשב את זמן הסבב הממוצע ואת זמן ההמתנה הממוצע במערכת שהתזמון בה הוא RR , כאשר נתון:

PROCESS	BURST TIME
P1	24
P2	9
P3	3

2. לתהליך נחוץ 10 יחידות של זמן CPU רציף . כמה שינויי הקשר (context switch) יהיו אם ה – quantum הוא :
   א. 12 יחידות .
   ב. 6 יחידות .
   ג. 4 יחידות .
   ד. יחידה אחת .

אם ידוע שרוב התהליכים במערכת הם כמו התהליך הנ"ל, איזה quantum מהנ"ל תבחר? יש לנמק .

תורים מרובי רמות - MULTI LEVEL QUEUES

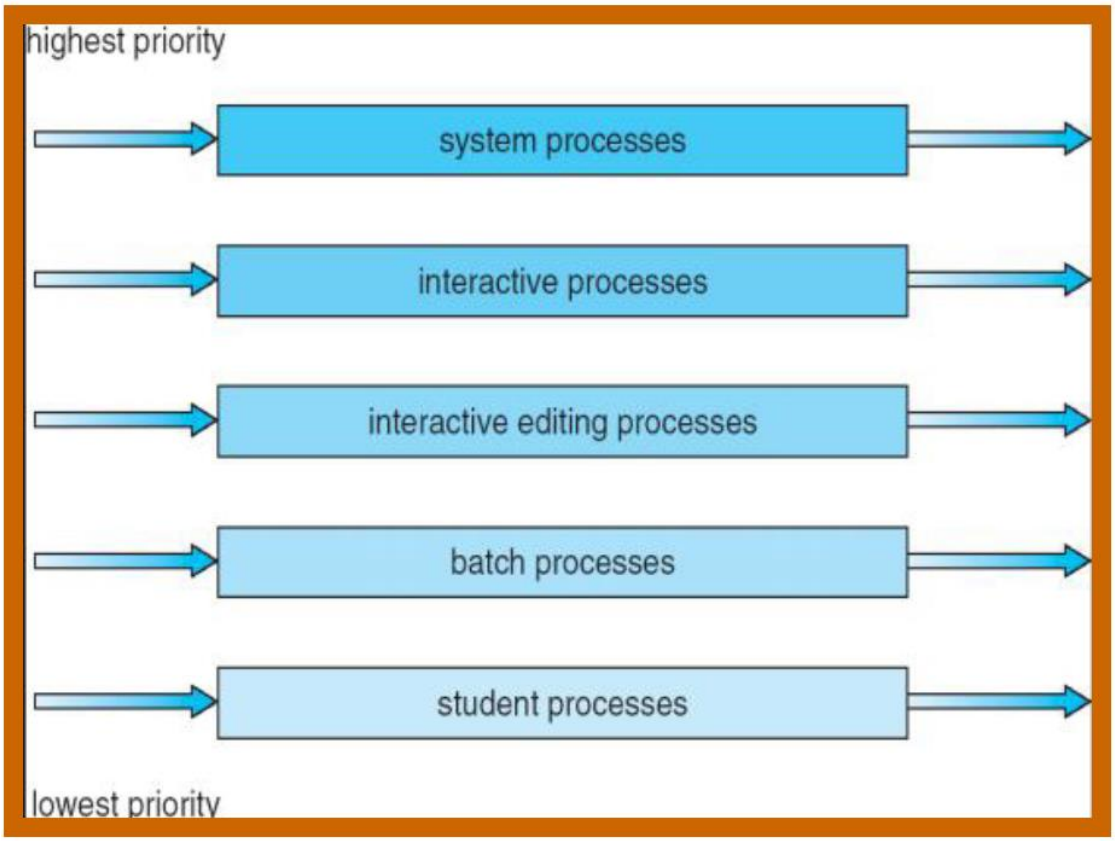
בגישה זו, תור המוכנים (READY) מחולק למספר תורים.

- לכל תור יש אלגוריתם תזמון משלו.
- התהליכים מוקצים באופן קבוע לאחד התורים, בד"כ עפ"י תכונות התהליך (למשל, עדיפות; למשל, סוג תהליך וכו') .
- קיים מתזמן בין התורים .

❖ **מדיניות מתזמן התורים יכולה להיות מבוססת RR או PREEMPTIVE .**

RR – הקצאת זמן מסוים לכל תור כדי שיריץ את התהליכים הנמצאים בו .
PREEMPTIVE – של עדיפויות קבועות . תור בעל עדיפות נמוכה, לא ירוץ לפני תור בעל עדיפות גבוהה . **בשיטה זו, תתכן הרעבה !**

❖ **היתרון בשיטת תורים מרובי רמות הוא ניצול גבוה של ה – CPU .**



תורים רבי רמות עם משוב -FEEDBACK_MULTI LEVEL QUEUES

באלגוריתם הקודם, נוצרה בעיית הרעבה כי תהליכים לא יכלו לעבור בין רמות. לעומת זאת, האלגוריתם של תורים רבי רמות עם משוב, מאפשר לתהליך לעבור בין התורים (הרמות) .הרעיון הוא להפריד תהליכים עפ"י מאפייני פרץ ה – CPU שלהם.

- תהליכים בעלי דרישות לזמן CPU רב, יעברו לתור עם עדיפות נמוכה .
- תהליכים בעלי דרישות I/O גבוהות, ותהליכים אינטראקטיביים, יהיו בתורים בעלי עדיפות גבוהה יותר .
- תהליך שממתין זמן רב מידי בתור בעל עדיפות נמוכה, יעבור לתור בעל עדיפות גבוהה .
- שיטה זו נקראת **aging** והיא מונעת הרעבה .

תור משוב חייב לקיים את כל התנאים הבאים:

- **יש מספר תורים .**
- **חייב להיות אלגוריתם ניהול לכל רמה .**
- **חייב אלגוריתם ניהול כללי. כלומר, איזו רמה תשלח תהליך לביצוע.**
- **חייב אלגוריתם על פיו מעלים תהליך לתור בעל עדיפות יותר גבוהה .**
- **חייב אלגוריתם על פיו מורידים תהליך לתור בעל עדיפות נמוכה גבוהה .**
- **חייב אלגוריתם על פיו קובעים לאיזה תור ייכנס תהליך חדש .**

ניהול תהליכים בסביבה רבת מעבדים

הניהול מעט יותר מסובך . כאן נדרש :

- תזמון בין המעבדים השונים.
- ניהול חלוקת המשאבים והמשימות בין כל המעבדים

מבחינים בין שני סוגי סביבות:

- סביבת מעבדים הומוגנית, בה כל המעבדים זהים בתפקודם .
- סביבה אסימטרית, בה יש מעבד מרכזי עם גישה לזיכרון ולנתונים והוא מנהל ומחלק את העבודה בין תתי המעבדים האחרים .

ניהול זמן אמת – REAL TIME SCHEDULING

מערכת זמן אמת היא מערכת המבצעת פעולות כמעט ללא השהיה, אפילו לא של מילי שניות בודדות.

ישנם שני סוגי מערכות זמן אמת :

- **זמן אמת נוקשה HARD REAL TIME – מבטיח ביצוע משימות קריטיות תוך פרק זמן מיידי .**
- **זמן אמת גמיש SOFT REAL TIME – מבטיח שמשימות קריטיות יבוצעו בעדיפות גבוהה על חשבון שאר המשימות .**

תרגילים להגשה

תרגיל 1 - להלן מספר אמירות . יש לכתוב ליד כל אמירה אם היא נכונה או לא ולהסביר את התשובה .

1. שיטת תזמון **RR** מפלה לרעה תהליכים קצרים .
2. שיטת תזמון **FCFS** טובה עבור תהליכים קצרים .
3. שיטת **SJF** טובה עבור תהליכים קצרים .
4. שיטת תזמון **PRIORITY NON PREEMPTIVE** טובה לתהליכים קצרים .

תרגיל 2 – איזה מהטיעונים הבאים לגבי **ויסות מעגלי (RR)** עם **פלח זמן (TIME SLICE)** קצר ביותר, נכון ?

א. זהו למעשה **FCFS** .
ב. גורם למספר גבוה של **מיתוגי הקשר (CONTEXT SWITCH)** .
ג. גורם לאלגוריתם להיות **בלי לקיחה בכח NON PREEMPTIVE** .
ד. כל התשובות הנ"ל נכונות .
ה. אף תשובה מבין התשובות א – ד אינה נכונה .

שיטות תזמון	FCFS – First Come First Served	SJF – Shortest Job First	PRIORITY	ROUND ROBIN
הסבר	מתייחסת לזמן הגעת התהליכים לתור לפי FIFO	זוהי שיטה תיאורטית בלבד לפי תהליך עם ה-BURST TIME הקצר ביותר	ה- CPU מוקצה לתהליך בעל עדיפות גבוהה יותר	תור ה- Ready הוא תור מעגלי אשר בו כל תהליך לפי תורו מקבל Q . של זמן (אם התהליך לא הספיק להסתיים בQ שהוא קיבל מתבצעת ההלפה)
ניצולת ה- CPU	נמוכה	בינונית	בינונית	גבוהה
TURNAROUND זמן סיבוב	גבוה - תלוי בתהליך	בינוני	גבוה	גבוה יותר מאשר SJF
THROUGHPUT תפוקת המערכת	נמוכה	תפוקה מרבית	נמוכה	בינונית
TIME REASPONSE זמן תגובה	נמוך	בינוני	גבוה - תלוי בעדיפות התהליך	נמוך - טוב יותר מאשר SJF
WAITING TIME זמן המתנה	ארוך ותלוי בזמן ההגעה של התהליכים	מינימאלי	תלוי בעדיפות התהליך	ארוך
STARVATION הרעבה	תיתכן הרעבה ברגע שתהליך נכנס ללולאה אינסופית	גרמת הרעבה	ישנה הרעבה כאשר תהליך בעל עדיפות נמוכה עלול לחכות לנצח	לא תיתכן הרעבה
מתי נשתמש	מכ"מ, קוצב לב	בתהליכים קצרים	חדר מיון(שיטה עם הפקעה)	מערכות time-sharing
דרכים לשיפור	הכנסת PRIORITY	לבחור שיטה שאין בה הרעבה ?	הפתרון להרעבה הוא AGING שבה נגדיל את העדיפות של התהליך ככל שזמן ההמתנה שלו גדל.	?

התזמון ב – UNIX

התזמון הוא לפי תורים רבי רמות עם משוב (FEEDBACK Multi Level Queue) .

- ישנם תורים במערכת , ולכל תור יש ערך . תורי הערכים **השליליים** , שמורים לתהליכים של **המערכת** ותורים עם ערכים **חיוביים** שמורים לתהליכי **משתמש** .
- בכל שלב, בוחרים את התהליך שנמצא בתור עם העדיפות הכי גבוהה .
- בכל תור מריצים **RR** .
- לכל תהליך יש שדה **cpuUsage** . כאשר התהליך רץ, מעלים את ערכו של השדה לאחר כל פעימת שעון . לאחר פרק זמן קבוע (נניח , לאחר כל שנייה) , עושים חישוב מחדש של עדיפויות התהליכים .
- תהליך שהשתמש לא מזמן ב – CPU מקבל עדיפות גרועה .
- ככל שעובר הזמן, עדיפות התהליך משתפרת .
- תהליכים שיוותרו מרצונם על המעבד (עתירי ק / פ) יחזרו אליו מהר יותר מתהליכים שעברו הפקעה (עתירי חישוב)

- **(** החישוב :

$$newPriority(P_i) = type(P_i) + cpuUsage(P_i)/2 + nice$$

$type(P_i)$ – לפי סוג התהליך (לפי ערך התור) .

$cpuUsage(P_i)/2$ - מעודד תהליכים שלא ניצלו את כל זמן ה – CPU . החילוק ב – 2 נועד להשיג את "אפקט השכחה". כלומר, ככל שהזמן עובר, מתייחסים ל – cpuUsage ישן , פחות .

nice – ה USER יכול להוסיף מספר ובכך להוריד בכוונה את העדיפות שלו (לא נפוץ כי לכל אחד יש אינטרס לקדם את התהליך שהוא כתב) . **)**

התזמון ב – WINDOWS

Win NT,Win XP – דומה ל – UNIX .

- יש **32 תורים** : תורים 0 עד 15 ל USER ו – 16 עד 31 ל system.
- לכל תור מוצמד ערך **חיובי** .
- העדיפות לפי הערכים – הפוכה. ככל שהערך יותר גבוה, העדיפות יותר גבוהה. 31 זו העדיפות הגבוהה ו – 0 זו העדיפות הנמוכה .
- בתוך התור – מדיניות RR .
- העדיפות יורדת אם נצרך כל ה-quantum
- העדיפות עולה אם תהליך עובר מ-wait ל-ready
- העדיפות נקבעת על-פי סוג הקלט / פלט :
  קלט מהמקלדת מקנה עדיפות גבוהה יותר מאשר קלט/פלט מדיסק .
-
- **שונה מ – UNIX** – כאשר **אף אחד לא רוצה לרוץ**, ישנו תהליך הנקרא zero page thread שתפקידו לנקות דפים ממערכת הזכרון (מנקה ומפנה משאבים לא נחוצים ← daemon).