***מערכות הפעלה סיכום מושגים:***

**מושגים:**

1. יציב – stable : לא קורס.
2. נייד – portable : יכול להריץ קוד עליותר ממכונה אחת.
3. אמין – reliable : תמיד בגיב באותה צורה.
4. מוגן – safe : לא מבצע דברים מסוכנים.
5. התנהגות טובה – well behaved : מתבצע על סביבת עבודה נוחה.

**Multi-programming:**

אפשרות להריץ מספר תהליכונים (threads) במקביל, עושים זאת בעזרת מתג-הקשר (context-switch):

1. אומר למעבד לעצור את המשימה הנוכחיות שלו ולסמן אותה בהשהייה.
2. שומר PC, SP, FP ו-Registers אחרים של המשימה
3. טוען PC, SP, FP ו-Registers אחרים מהמשימה החדשה שהולכת להגיע.
4. מעדכן את המשימה החדשה במתבצעת.
5. מתחיל לבצע תהליך חדש.

**Systems-Calls:**

ממשק בין המשתמש (user application) ושירותים המסופקים מהמערכת הפעלה או הליבה.

System-calls מתחלקים ל-5 קטגוריות עקריות:

1. תהליך שליטה – process control (למשל יצירה/עצירה של תהליך).
2. ניהול קבצים – file management (למשל קריאה/כתיבה).
3. מנהל התקנים – device management.
4. אחזקת מידע – information maintance (למשל הגדרת זמן או תאריך).
5. תקשורת – communication (למשל שליחת הודעות).

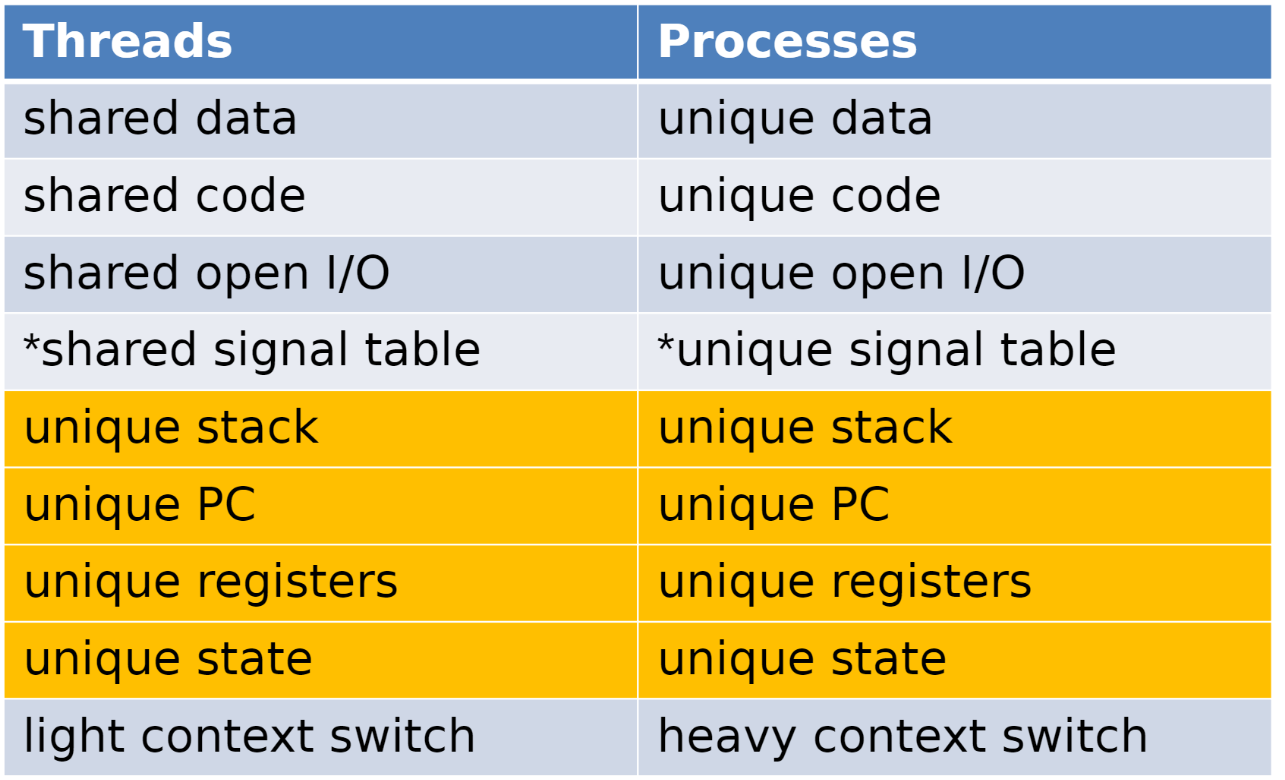
**Threads:**

**יתרונות של threads:**

1. יצירה/העתקה/שינוי של thread יותר מהיר מ-process פי 10-100.
2. מספר threads יכולים לשתף בינהם מבנה נתונים, משתנים גלובאלים, תהליכון בן (child process).
3. Threads יכוים לתקשר אחד עם השני מבלי להשתמש ב-system calls.

**חסרונות של threads:**

1. פחות מאובטח ויציב כאשר מספר threads ניגשים בשיתוף לאותו אובייקט (משתנה גלובאלי, תהליך בן ועוד...).
2. איתות של thread אחד משפיע על כל ה-threads ב-process מסויים.



**User level threads::**

Threads אשר ממומשים (implement) בעזרת user level libraries, מעבר בין user level threads לא דורש קריאה למערכת הפעלה וכך לא יוצר הפרעה, ה-user application schedule הוא מחליט איזה thread ישתמש ב-CPU, מנצל רק ליבה אחת מה-CPU.

**יתרונות של user level threads:**

1. תאימות – compatibility : יכול להתממש (implement) על מערכת הפעלה שלא תומכת ב-threads.
2. הצגה פשוטה – simple representation : כל thread מיוצג בצורה פשוטה עם PC, registers, stack והכל נשמר ב-user process address space.
3. פשוט וניהול מהיר : יצירה, סנכרון, מעבר בין threads, מבוצעים מבלי לערב את הליבה של המערכת ההפעלה והכל עובד במהירות גבוהה.

**חסרונות של user level threads:**

1. איבוד קורדינאטות בין threads לבין הליבה של מערכת ההפעלה.
2. קריאה אחת לליבה של מערכת ההפעלה עלולה לגרום למערכת ההפעלה לחסום את כל התהליך, אפילו אם נשארו עוד runnable threads בתהליך.
3. בעיית סנכרון זמנים בין kernel level threads לבין user level threads.

**Kernel level threads:**

כל ה-threads יכולים לגשת ע"י המעבד, בליבה אחרית על הניהול והזמנים בין המ-threads.

קיים system call כדי ליצור ולנהל threads.

יתרונות של kernel level threads:

1. הליבה אחראית על ניהול וזמנים בין ה-threads בצורה חכמה.
2. תהליכונים בשכבת הליבה טובים במיוחד עבור אפליקציות אשר שחסומות בתדירות גבוהה.
3. במעבד ,multi-processor מספר threads יכולים לעבוד בו זמנים על אותה ליבה.

חסרונות של kernel level threads:

1. יצירה, ניהול ומעבר בין threads הרבה יותר איטי ויקר מאשר user level threads.