digital whisper
Technologic Papers

תורת הכאוס של הזיכרון

מאת ניקול פולינקובסקי

הקדמה

פעם עצרתם לחשוב איך הזיכרון במחשב שלכם באמת עובד? איך ייתכן שמערכת ההפעלה מצליחה לתמרן בין אינספור תהליכים שרצים בו-זמנית?

ניהול זיכרון הוא אחד המרכיבים הקריטיים בתכנון ותפעול מערכות מחשוב, והוא משפיע באופן ישיר על הביצועים, היציבות והאבטחה של מערכת ההפעלה והתוכנות הפועלות בה. כך למשל, המערכת חייבת לנהל את המשאבים בצורה שתספק גישה מהירה לקוד שאנחנו משתמשים בו בתדירות גבוהה, ובמקביל לאפשר את השמירה של קוד שצורכים אותו פחות בזיכרון איטי יותר. הגישה הזו מאפשרת לוודא שזיכרון מהיר ינוצל למשימות תכופות יותר, בעוד שזיכרון איטי ישמור מידע פחות קריטי, אך הכנסתו אליו עשויה לקחת יותר זמן ולהשפיע על הביצועים.

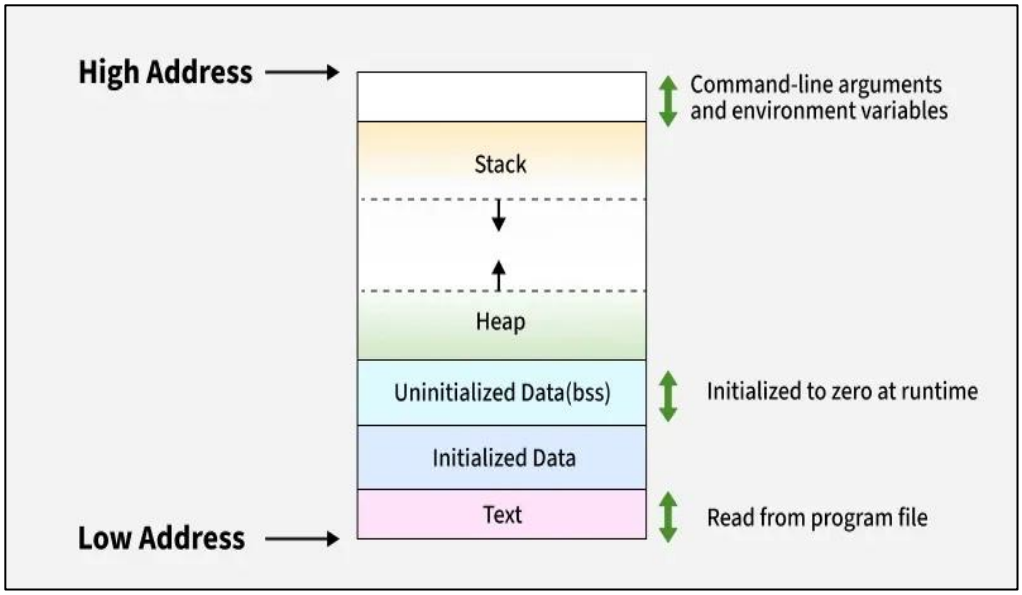
החלטתי לצלול לעומק ניהול הזיכרון של Windows ולבחון את המנגנונים שמאפשרים לו לתפקד בצורה אופטימלית.

במאמר נדבר על הנושאים הבאים:

- מבנה הזיכרון: איך הזיכרון מאורגן ומהם המרכיבים השונים שמרכיבים אותו?
- מנגנוני ניהול הזיכרון: כיצד מערכת ההפעלה מחליטה כמה זיכרון להקצות לכל תהליך ומהם הגורמים המובילים להחלטות אלו?
- הקצאות זיכרון: הבדל בין הקצאה דינאמית לסטטית, וההשלכות של כל אחת על ביצועים ויציבות.
- זיכרון פיזי מול זיכרון וירטואלי: איך המערכת משתמשת בזיכרון וירטואלי להרחיב את הזיכרון מעבר למגבלות הפיזיות?
- מנגנוני נעילה: איך מערכת ההפעלה שומרת על סדר ויעילות במצבים של ריבוי תהליכים, כדי למנוע פקקי תנועה וזליגת זיכרון.

מבנה הזיכרון במערכת ההפעלה

במערכות מחשוב, זיכרון ה-RAM מחולק למקטעים שונים, כאשר כל אחד מהם מיועד לשימוש ייחודי בתהליך הריצה של התוכנית. חלוקה זו מאפשרת לתהליכים להתנהל בצורה מאורגנת, משפרת את הביצועים ומונעת התנגשויות שעלולות לגרום לשגיאות קריטיות, כמו קריסת תוכניות או חשיפת מידע רגיש.



אזורי הזיכרון העיקריים בתהליך הם:

אזור ה-Text - מקטע הטקסט (המכונה גם מקטע הקוד) הוא החלק בזיכרון שבו מאוחסן הקוד הבינארי של התוכנית. הוא מכיל את קוד ה-Machine של הפונקציות וההוראות שהורכבו מהקוד המקורי. מקטע זה מוגדר לרוב לקריאה בלבד ונמצא בחלקים הנמוכים של הזיכרון, במטרה למנוע שינויים לא מכוונים בקוד בזמן ריצת התוכנית.

אזור ה-Data - מקטע הנתונים (המכונה לעיתים data.), הוא חלק מקובץ האובייקט או ממרחב הכתובות של התוכנית, המכיל משתנים סטטיים מאותחלים. כלומר, הוא כולל משתנים גלובליים ומשתנים מקומיים סטטיים שהוגדרו מראש בקוד. גודלו של מקטע זה נקבע בהתאם לערכים שהוגדרו בתוכנית, ואינו משתנה במהלך זמן הריצה.

בנוסף, קיימת קטגוריה נוספת הנקראת אזור ה- block started by symbol, אזור זה מאחסן משתנים סטטיים לא מאותחלים. כלומר, משתנים שהוקצו בזיכרון אך לא קיבלו ערך התחלתי. כאשר התוכנית רצה, אזור ה-BSS מאותחל לערכי ברירת מחדל (למשל, אפס) על ידי מערכת ההפעלה. גם אזור זה נחשב לחלק ממקטע הנתונים בתוכנית, אך הוא נבדל מהמקטע data. בכך שהוא לא מכיל ערכים מאותחלים מראש.

אזור ה-Heap - מקטע ה-Heap הוא אזור בזיכרון שבו מתבצעת הקצאת זיכרון דינמית. הוא מתחיל בסוף מקטע ה-BSS וצומח כלפי כתובות גבוהות יותר.

ניהולו מתבצע באמצעות פונקציות כמו malloc(), realloc() ו-free(), אשר עשויות להפעיל קריאות מערכת כגון brk ו-sbrk, על מנת להגדיל או להקטין את גודלו בהתאם לצורכי התוכנית.

אזור ה-Stack - מקטע ה-Stack משמש לאחסון משתנים מקומיים ולניהול קריאות לפונקציות. בכל קריאה לפונקציה, נוצרת מסגרת מחסנית (stack frame) המכילה את המשתנים המקומיים של הפונקציה, את הפרמטרים שלה ואת כתובת החזרה. מסגרת זו נשמרת בתוך מקטע ה-Stack. מקטע ה-Stack ממוקם לרוב בכתובות הגבוהות של הזיכרון וגדל כלפי מטה, בניגוד למקטע ה-Heap, הגדל כלפי מעלה. בגלל הסידור הזה, כאשר מצביעי ה-Stack וה-Heap נפגשים, המשמעות היא שהתוכנית מיצתה את הזיכרון הפנוי העומד לרשותה, דבר שעלול להוביל לקריסה (Stack Overflow או Heap Exhaustion).

סוגי זיכרון

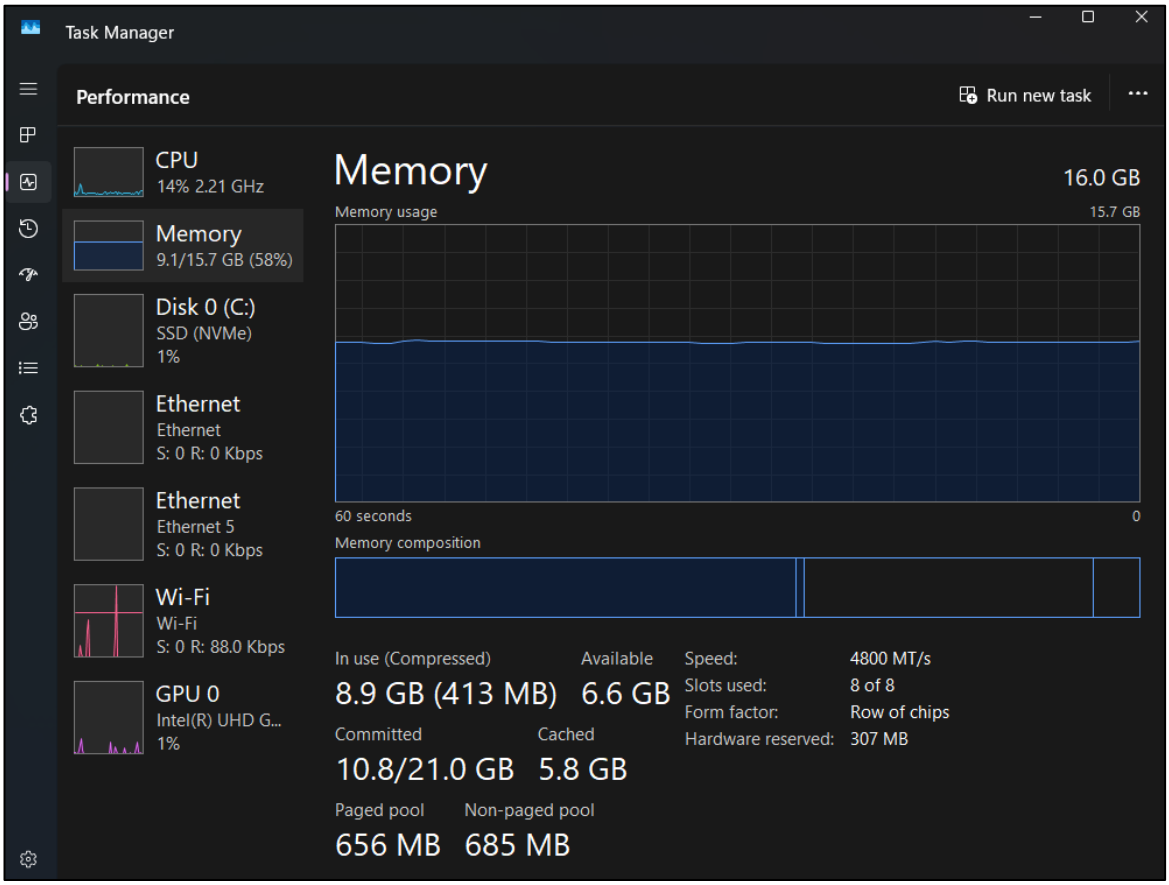
זיכרון מתייחס לרכיבים הפיזיים במערכת המחשב המשמשים לאחסון נתונים ותוכנות, בין אם באופן זמני או קבוע. כל סוג של זיכרון ממלא תפקיד חשוב במערכת המחשב, ותורם לניהול היעיל של המשאבים והביצועים.

קיימים מספר סוגי זיכרון:

1. זיכרון RAM (Random Access Memory):

זיכרון RAM הוא זיכרון נדיף (Volatile Memory), כלומר, הנתונים מאוחסנים בו נמחקים כאשר המחשב נכבה. תפקידו העיקרי הוא לאחסן מידע וקוד מכונה שהתוכנה משתמשת בהם בזמן אמת. כמות גדולה יותר של זיכרון RAM מאפשרת למחשב לעבד יותר נתונים בו-זמנית ולספק ביצועים מהירים יותר. אם אנחנו רוצים לבדוק את כמות ה-RAM הזמינה, ניתן לעשות זאת דרך מערכת ההפעלה:

- ב-Windows: ניתן ללחוץ על Ctrl + Shift + Esc כדי לפתוח את Task Manager, ואז לבחור בלשונית "Performance" כדי לראות את כמות ה-RAM הזמינה.



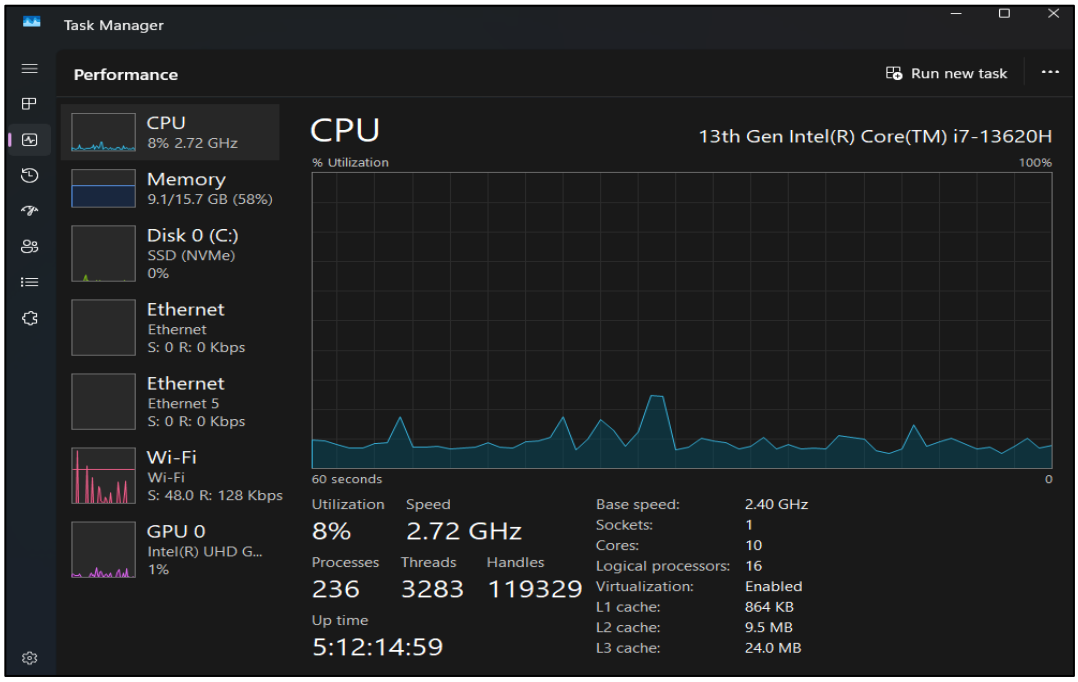
2. זיכרון ROM (Read-Only Memory)

בניגוד ל-RAM, זיכרון ROM הוא לא נדיף (Non-Volatile Memory), כלומר, הנתונים בו נשמרים גם לאחר כיבוי המחשב. זיכרון זה משמש לאחסון קושחה (Firmware) - תוכנה שפועלת בחומרה ואינה דורשת עדכונים תכופים. דוגמה לכך היא ה-BIOS או ה-UEFI, המנהלים את פעולתו הבסיסית של המחשב בזמן אתחול.

3. זיכרון מטמון (Cache Memory)

זיכרון מטמון הוא זיכרון קטן במיוחד אך מהיר מאוד, שמשמש כמתווך בין המעבד (CPU) לבין ה-RAM. מטרתו היא לאחסן נתונים והוראות שהמעבד ניגש אליהם באופן תדיר, ובכך להאיץ את תהליך העיבוד. זיכרון המטמון מחולק לשלושה סוגים עיקריים:

- **L1 Cache** - הקטן ביותר והמהיר ביותר, נמצא בתוך הליבה של המעבד.
- **L2 Cache** - גדול יותר מ-1L אך מעט איטי יותר.
- **L3 Cache** - המאגר הגדול ביותר, נמצא מחוץ לליבות המעבד ומשותף לכל הליבות.



מנגנוני ניהול זיכרון

ניהול זיכרון הוא תהליך קריטי במערכות הפעלה, אשר אחראי על האופן שבו המערכת מנהלת את הזיכרון הזמין ומחלקת אותו בין התהליכים השונים. ישנם מספר מנגנונים שונים המאפשרים למערכת להשתמש בזיכרון בצורה היעילה ביותר, והם כוללים את מנגנונים הבאים:

Paging (הקצאת זיכרון בדפים)

טכניקת ניהול זיכרון שמבטלת את הצורך בהקצאת זיכרון רציף. כל תהליך מקבל "דפים" (Pages) של זיכרון, שהם יחידות קטנות שיכולות להיות ממוקמות בזיכרון בכל מקום. טכניקה זו משולבת לעיתים עם הקצאה ושחרור של מסגרות דפים ואחסון דפים על מדיה משנית, כמו דיסקים, לצורך הרחבת מרחב הזיכרון של המחשב מעבר לזיכרון הפיזי.

כיצד זה עובד בפועל? כל תהליך יכול לגשת לדפים שהוא זקוק להם, אפילו אם הדף לא נמצא כרגע בזיכרון הפיזי (RAM). במצב כזה, המערכת טוענת את הדף הנדרש מזיכרון משני (כמו דיסק קשיח או SSD) ומחזירה אותו לזיכרון הפיזי.

פעולה זו מתבצעת באמצעות מנגנון שנקרא Page Fault. כאשר הדף נמצא על הדיסק, המערכת חייבת להעביר דף אחר אל הדיסק כדי לפנות מקום לדף החדש.

תהליך זה נקרא Swap. המערכת משתמשת ב-Pagefile, או Swap File שנמצא על הדיסק הקשיח, על מנת לאחסן את הדפים שלא נמצאים בזיכרון הפיזי ברגע נתון. כאשר דף נדרש שוב, המערכת מבצעת את הפעולה ההפוכה ומחזירה אותו לזיכרון הפיזי.

Segmentation (סגמנטציה)

בטכניקת ניהול זיכרון זו, הזיכרון מחולק לסגמנטים, שהם חלקים לוגיים של התוכנית, כמו פונקציות או מערכים. בניגוד ל-Paging, שבו הנתונים מחולקים לדפים בעלי גודל קבוע, בסגמנטציה הגודל של כל סגמנט עשוי להשתנות בהתאם למבנה של התוכנית. ההפניות לזיכרון בסגמנטציה כוללות את הסגמנט שבו נתון מאוחסן ואת האופסט בתוך הסגמנט, כלומר המיקום המדויק של הנתון בתוך הסגמנט. לדוגמה, תוכנה יכולה להחזיק סגמנט לקוד, סגמנט לנתונים, וסגמנט עבור מערכים.

Virtual Memory (זיכרון וירטואלי)

זיכרון וירטואלי הוא טכניקת ניהול זיכרון שבה הזיכרון המשני (למשל דיסק קשיח) משמש כמשאב נוסף לזיכרון הראשי (RAM), כך שנראה לתוכניות שהן משתמשות בזיכרון "אין סופי".

טכניקה זו מאפשרת למערכת הפעלה להפעיל תהליכים שמשתמשים ביותר זיכרון ממה שיש במחשב בפועל (RAM). המערכת שומרת את הנתונים המיותרים על אחסון משני ומבצעת Page Swapping בין הזיכרון הפיזי לבין הזיכרון הווירטואלי.

היתרון של זיכרון וירטואלי הוא שהוא מספק תחושה שהזיכרון גדול יותר ממה שיש בפועל במחשב. היישום יכול לעבוד כאילו יש לו גישה ליותר זיכרון ממה שיש במחשב, כשהוא משתמש בזיכרון המשני כאשר הוא זקוק לכך. כאשר יישום מפעיל נתונים, הם מאוחסנים בכתובת פיזית במערכת (RAM). אולם אם יש צורך בזיכרון עבור משימה אחרת, מערכת ההפעלה יכולה להעביר נתונים לזיכרון וירטואלי (כמו דיסק קשיח או SSD).

בזיכרון וירטואלי, הכתובת הפיזית שאליה תהליך פונה לא בהכרח תואמת את הכתובת הווירטואלית שמוקצת לו. כל פעולה של כתיבה או קריאה לזיכרון דורשת תרגום כתובת וירטואלית לכתובת פיזית. ה-MMU (יחידת ניהול זיכרון) אחראית לתרגום אוטומטי של כתובת זיכרון לכתובת פיזית במערכת. כשיש צורך להחזיר נתונים לזיכרון, ה-MMU מבצע את פעולת Context Switch כדי להחזיר את הביצוע לתקנו.

כאשר המערכת לא מצליחה למצוא את הדף בזיכרון הפיזי, היא מבצעת Page Fault, שהיא פעולה שמביאה לטעינה מחדש של הדף מהזיכרון המשני. מערכת ההפעלה עושה שימוש ב-Pagefile, קובץ בו מערכת ההפעלה שומרת דפים מתוך הזיכרון, כאשר לא מספיק זיכרון RAM פנוי לפעולה. כדי לשמור נתונים שמפנים מקום בזיכרון הפיזי. (אפשר להקביל את זה לשימוש במדף וארון, כאשר המדף מייצג את הזיכרון הפיזי והארון את הזיכרון המשני).

הקצאה סטטית לעומת הקצאה דינאמית

הקצאת זיכרון היא התהליך שבו תוכנית מחשב או תהליך מקבלים כתובת פיזית וכתובת וירטואלית בזיכרון. ההקצאה יכולה להתבצע בשני שלבים: לפני ריצת התוכנית או במהלך הריצה שלה.

ישנם שני סוגים עיקריים של הקצאות זיכרון:

- הקצאה דינאמית
- הקצאה סטטית

הקצאה דינאמית

הקצאה דינאמית היא הקצאת זיכרון שמתבצעת בזמן ריצת התוכנית, כלומר כאשר התוכנית כבר פועלת והמשאבים נדרשים על ידי הפונקציות השונות שלה. טכניקה זו גמישה יותר, משום שהיא מאפשרת לתוכנית לשנות את גודל הזיכרון שלה במהלך הריצה.

הטכניקה הזו נפוצה במצבים בהם גודל הזיכרון הדרוש משתנה בזמן הריצה, למשל במערכות עם כמות נתונים משתנה כגון חיפוש במאגרי נתונים. ההקצאה הדינאמית מתבצעת באמצעות פונקציות כמו malloc ו־calloc בשפת C. פונקציות אלה מקצות זיכרון בתהליך ריצה ומחזירות כתובת שמצביעה לאותו זיכרון. כשזיכרון זה כבר לא בשימוש, על המתכנת לשחררו באמצעות הפונקציה free, אחרת עלול להתרחש "דליפת זיכרון".

דוגמת קוד של הקצאה דינאמית:

```c
int *arr = malloc(10 * sizeof(int));
if (arr == NULL) {
    printf("Memory allocation failed");
    return 1;
}

for (int i = 0; i < 10; i++) {
    arr[i] = i * i;
}

free(arr);
```

כאן, ההקצאה מתבצעת בזמן ריצה, כך שהתוכנית יכולה לשנות את כמות הזיכרון הנדרשת ולהתאים את הזיכרון על פי הצורך במהלך הפעולה שלה. בסיום השימוש בזיכרון, יש לשחררו, כדי למנוע בזבוז משאבים (דליפת זיכרון).

הקצאה סטטית

הקצאה סטטית, בניגוד לדינאמית, מתבצעת בזמן ה-Compile Time. כלומר, ברגע שהקוד מתורגם לקוד מכונה לפני שהתוכנית מתחילה לרוץ. ההקצאה נעשית על ידי הקומפיילר, והזיכרון שמוקצה קבוע בגודלו במשך כל זמן הריצה של התוכנית. המשמעות היא שלא ניתן לשנות את גודל הזיכרון אחרי שהוקצה, והוא לא יכול להיות שונה במהלך פעולתה של התוכנית. לרוב, ההקצאה הסטטית נפוצה במצבים בהם ידוע מראש גודל הזיכרון הדרוש, כמו מערכים קבועים או משתנים גלובליים.

דוגמת קוד של הקצאה סטטית:

```c
#include <stdio.h>

int arr[10];

int main() {
    for (int i = 0; i < 10; i++) arr[i] = i * 2;

    for (int i = 0; i < 10; i++) printf("%d ", arr[i]);

    return 0;
}
```

כאן, המערך arr מוקצה מראש עם גודל קבוע של 10 אלמנטים, ולא ניתן לשנות את גודלו בזמן הריצה של התוכנית. הקצאה כזו מתבצעת לפני הריצה, ואם התוכנית לא משתמשת בכל הזיכרון שהוקצה, זה עשוי להוביל לבזבוז זיכרון.

השוואה בין ההקצאות:

- הקצאה דינאמית: מאפשרת גמישות רבה יותר, והזיכרון מוקצה במהלך הריצה. ניתן להקצות ולשחרר זיכרון על פי הצורך. אבל היא יכולה להוביל לבעיות כגון Memory Leaks, מכיוון שמי שאחראי לניהול הזיכרון הוא המתכנת.
- הקצאה סטטית: זיכרון מוקצה מראש, בזמן הידור. פחות גמישה, מכיוון שהגודל נשאר קבוע במשך כל זמן הריצה של התוכנית.

זיכרון פיזי VS. זיכרון וירטואלי

זיכרון פיזי

זיכרון פיזי הוא הזיכרון שנמצא בפועל במערכת המחשב, כלומר הוא החומרה הממשית שבאמצעותה המחשב שומר נתונים בצורה של 1-ים ו-0-ים. מדובר במתקנים פיזיים כמו דיסק קשיח ,כונני SSD, זיכרון RAM ודומיהם. כל רכיב כזה תופס מקום פיזי במחשב ויכול לאחסן כמות נתונים מוגבלת.

כשהמחשב זקוק ליותר מקום בזיכרון, הוא לא יכול "להמציא" זיכרון, ולכן נדרשת טכניקת ניהול שמבוססת על מנגנונים נוספים (כגון זיכרון וירטואלי), שמאפשרים עבודה עם כמות גדולה יותר של נתונים ממה שהזיכרון הפיזי יכול להחזיק.

זיכרון וירטואלי הוא אחד מהמנגנונים שמסייעים בפתרון בעיה זו, ומאפשר למחשב להיראות כאילו יש לו כמות אינסופית של זיכרון.

זיכרון וירטואלי

זיכרון וירטואלי הוא מנגנון שמיועד לאפשר למחשב "להרחיב" את הזיכרון הפיזי על ידי שימוש בזיכרון משני, כמו דיסק קשיח או SSD. הרעיון הוא שלכל תהליך במערכת הפעלה יש מרחב זיכרון משלו, כך שהתהליך מרגיש כאילו יש לו זיכרון אינסופי, גם אם בפועל הזיכרון הפיזי מוגבל.

למרות שהזיכרון הפיזי מוגבל, תהליך מחשב לא תמיד זקוק לגישה לכל הנתונים בו זמנית. במילים אחרות, מערכת ההפעלה מנצלת את העובדה שתהליכים משתמשים רק בחלק מהנתונים במקביל. כך שחלק מהנתונים נשמרים בזיכרון משני (כגון דיסק קשיח או SSD), ורק כאשר נדרש גישה אליהם, הם מועתקים לזיכרון הפיזי.

תהליך מחשב יכול לפנות למרחב כתובת וירטואלי מבלי לדעת אם הנתונים נמצאים בזיכרון הפיזי או אם הם הועתקו לדיסק. המערכת מתפקדת כך שכתובת וירטואלית של תהליך לא מתאימה תמיד לכתובת פיזית בזיכרון, אלא ישנה המרה בין כתובת וירטואלית לפיזית. כל קריאה לכתובת וירטואלית מתורגמת על ידי מערכת ההפעלה לכתובת פיזית בזיכרון הפיזי באמצעות מנגנון הנקרא MMU (Memory Management Unit), אשר אחראי על המרת כתובת וירטואלית. (אפשר להקביל את זה לארון קיץ שבו נשמרים בגדים בעונה החמה, וזיכרון וירטואלי כארון חורף שבו שומרים בגדים לחורף, ומעבירים בין הארונות לפי הצורך.)

ההמרה מכתובת וירטואלית לפיזית

ההמרה בין כתובת וירטואלית לכתובת פיזית מתבצעת על ידי רכיב במעבד שנקרא **MMU** (Memory Management Unit). המטרה של ה-MMU היא לנהל את המרה בין הכתובת הווירטואלית, שמשמשת את התהליך, לבין הכתובת הפיזית שבאמצעותה המעבד גורס את הנתונים במערכת הזיכרון.

כשהמעבד ניגש לזיכרון, הוא שולח כתובת וירטואלית ל-MMU. ה-MMU בודק אם יש תרגום מהיר (למשל, אם הכתובת נמצאת ב-TLB). אם הכתובת לא נמצאת ב-TLB, המערכת מחפשת את המידע בטבלת העמודים.

שלבי המרת כתובת וירטואלית לפיזית

המרת הכתובת הווירטואלית לפיזית נעשית בשלבים:

בדיקה ב-(Translation Lookaside Buffer) TLB - TLB הוא אזור בזיכרון מעבד המשמש כאחסון ביניים (cache) לתרגום כתובות וירטואליות לפיזיות. כאשר המעבד שולח כתובת וירטואלית, ה-MMU קודם כל בודק אם התרגום כבר קיים ב-TLB. אם כן, הוא מבצע את התרגום במהירות (התגובה היא מהירה מאוד).

חיפוש בטבלת העמודים - אם הכתובת אינה נמצאת ב-TLB, ה-MMU יפנה לחיפוש בטבלת העמודים, שהיא בעצם מבנה נתונים שמכיל את המיפוי של כל הכתובות הווירטואליות לכתובות פיזיות. במילים פשוטות, זוהי "מפת הכתובות" שמראה איך כל כתובת וירטואלית תואמת לכתובת פיזית במערכת.

בדיקת הרשאות גישה - ה-MMU לא רק מתרגם כתובת, אלא גם בודק אם יש גישה לרשומות בזיכרון (האם יש הרשאה לקריאה, כתיבה או הרצה). אם הכתובת לא עומדת בדרישות הרשאה, נוצר חריג גישה (Access Violation).

Page Fault - אם הכתובת לא נמצאת בזיכרון הפיזי (למשל, אם היא נמצאת בדיסק או בזיכרון משני), תיווצר חריגת Page Fault. במקרה כזה, מערכת ההפעלה תטען את הדף החסר מהדיסק או מהזיכרון המשני אל הזיכרון הפיזי.

TLB - Translation Lookaside Buffer

ה-TLB הוא אזור בזיכרון המשמש כזיכרון מטמון (cache) לתרגום כתובות וירטואליות לפיזיות. הרעיון הוא לחסוך זמן על ידי כך שנשמרים בתוכו תרגומים שנעשו לאחרונה של כתובות. המעבד בודק קודם כל אם הכתובת נמצאת ב-TLB, ואם כן, הוא עושה את התרגום במהירות רבה. אם לא, הוא בודק בטבלת העמודים, שזה תהליך איטי יותר.

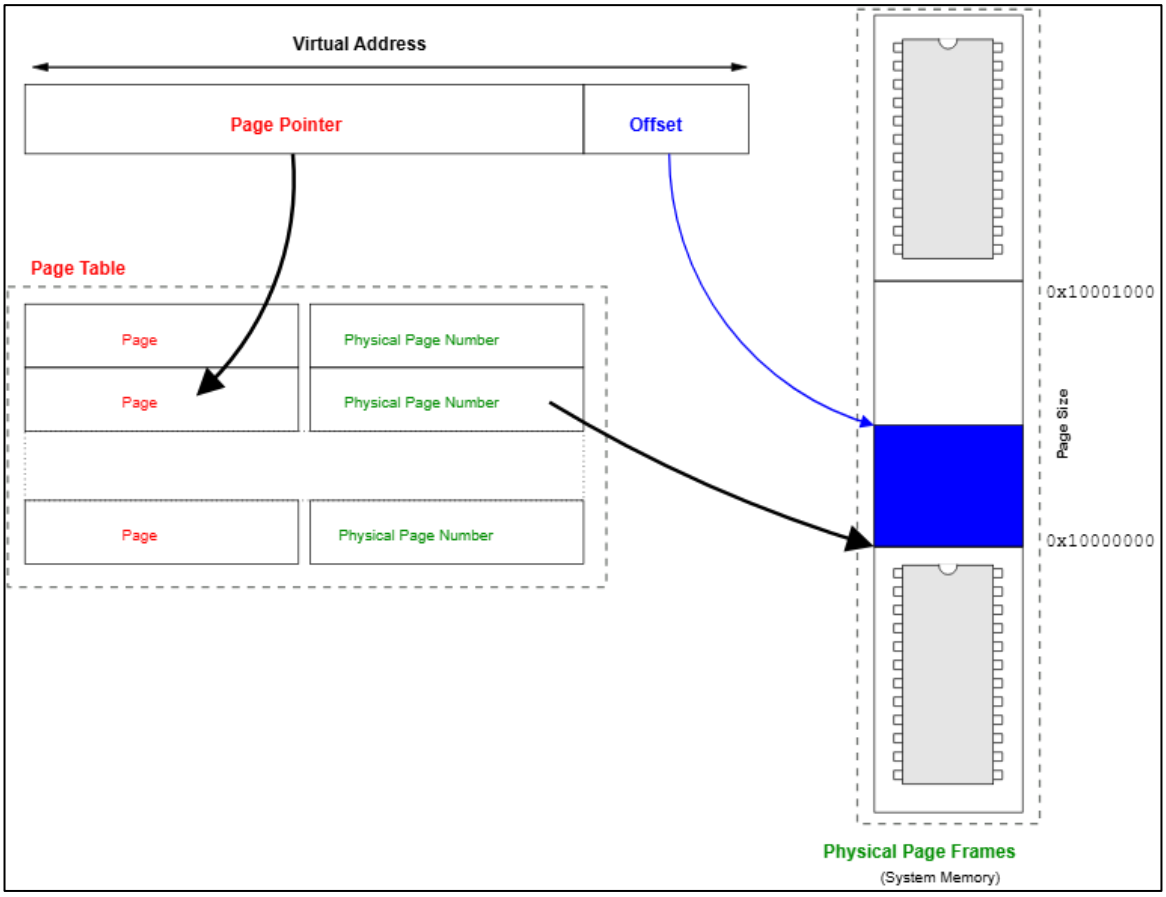
ה-TLB דומה למעין סימניות של ספר. כשאנחנו קוראים ספר, במקום לחפש כל הזמן את הפרק או העמוד שצריכים, אנחנו שמים סימנייה בעמוד האחרון שקראנו, כך שנוכל לפתוח ישירות לאותו עמוד מבלי להתחיל לחפש אותו מחדש. כך גם ה-TLB - הוא שומר את "הסימניות" של הכתובות שנמצאו לאחרונה ומפנה אליהם בקלות ובמהירות.

איך המערכת יודעת לאן למפות את הדפים?

כאשר מעבד מבצע גישה לכתובת וירטואלית, היא מתחלקת לשני חלקים:

- מספר דף (Page Number) - מציין את הדף בזיכרון הווירטואלי.
- היסט (Offset) - מציין את המיקום בתוך הדף בזיכרון הפיזי.

ה-MMU יבדוק את מספר הדף בטבלת העמודים וימיר אותו למסגרת פיזית בזיכרון. אם הדף נמצא ב-TLB, הוא יוכל להחזיר את המיפוי במהירות. אם לא, המערכת תעשה חיפוש בטבלת העמודים. במידה והדף נמצא בזיכרון הפיזי, התהליך יוכל להמשיך. אם לא, ייווצר Page Fault ומערכת ההפעלה תטפל בה:



החומרה והמנגנון של הזיכרון הווירטואלי

במהלך פעולתה, מערכת ההפעלה מחלקת את הזיכרון הפיזי לדפים בגודל קבוע. כל דף מאחסן את הנתונים של תהליך מסוים. כאשר אין מספיק מקום בזיכרון הפיזי עבור כל הדפים, המערכת שומרת את הדפים שלא בשימוש באזור דיסק ייעודי שנקרא Swap Space. כאשר דף שהיה לא פעיל נדרש מחדש, הוא מועבר בחזרה לזיכרון הפיזי, ותהליך זה נקרא "דפדוף" או "Swapping".

אם תהליך מנסה לגשת לדף שנמצא בזיכרון משני (כמו דיסק קשיח או SSD), מערכת ההפעלה מעוררת Page Fault - המתרחש כאשר הדף לא נמצא בזיכרון הפיזי. במקרה כזה, המערכת תטפל בזה על ידי העמסת הדף החסר חזרה לזיכרון הפיזי, ותעדכן את המיפוי בין הכתובת הווירטואלית לפיזית.

מנגנוני נעילה בזיכרון

מנגנוני נעילה הם קריטיים לניהול זיכרון במערכות הפעלה, שכן הם מונעים גישה לא מבוקרת למשאבי זיכרון משותפים, דבר שעלול להוביל לשגיאות או בעיות ביצועים. במערכות של היום, הזיכרון מחולק בין Processes ו-Threads. כדי למנוע מהם לגשת לאותם נתונים בו-זמנית, יש צורך במנגנוני סנכרון. לשם כך, מערכת ההפעלה משתמשת במנגנוני נעילה (Locks) שמונעים גישה לא מבוקרת למשאבים בזיכרון.

Mutex - Mutual Exclusion

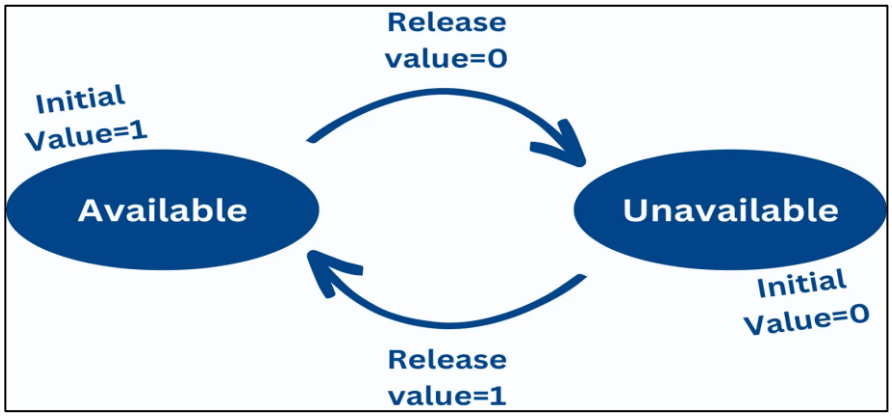
מנגנון נעילה בסיסי שמבטיח בלעדיות לגישה למשאב. כאשר תהליך אחד נועל את המשאב, כל תהליך אחר שמנסה לגשת אליו מושהה עד לשחרור הנעילה. כלומר, רק תהליך אחד בלבד יכול להחזיק ב-Mutex בכל רגע נתון. אם ה-Mutex תפוס, התהליך יעבור למצב המתנה עד לשחרורו. בסיום השימוש במנעול, התהליך אמור לשחרר אותו, אך אם הוא לא עושה זאת (למשל, אם התהליך קרס), עשויה להתרחש בעיית Deadlock.

דוגמה: נדמיין מספר אנשים שחולקים משקולת אחת בחדר כושר. רק מתאמן אחד יכול להחזיק במשקולת בכל רגע נתון. אם מישהו כבר משתמש בה, כל שאר המתאמנים צריכים להמתין עד שהוא יסיים ו"ישחרר" אותה.

Semaphore

מנגנון המהווה הרחבה של ה-Mutex ומאפשר לתהליכים שונים לגשת למשאב, אך עד גבול מסוים. Semaphore כולל מונה (Counter) שמייצג את מספר הגישות המותרות. כאשר תהליך רוצה לגשת למשאב, הוא מקטין את המונה. אם המונה מגיע ל-0, הגישה נחסמת.

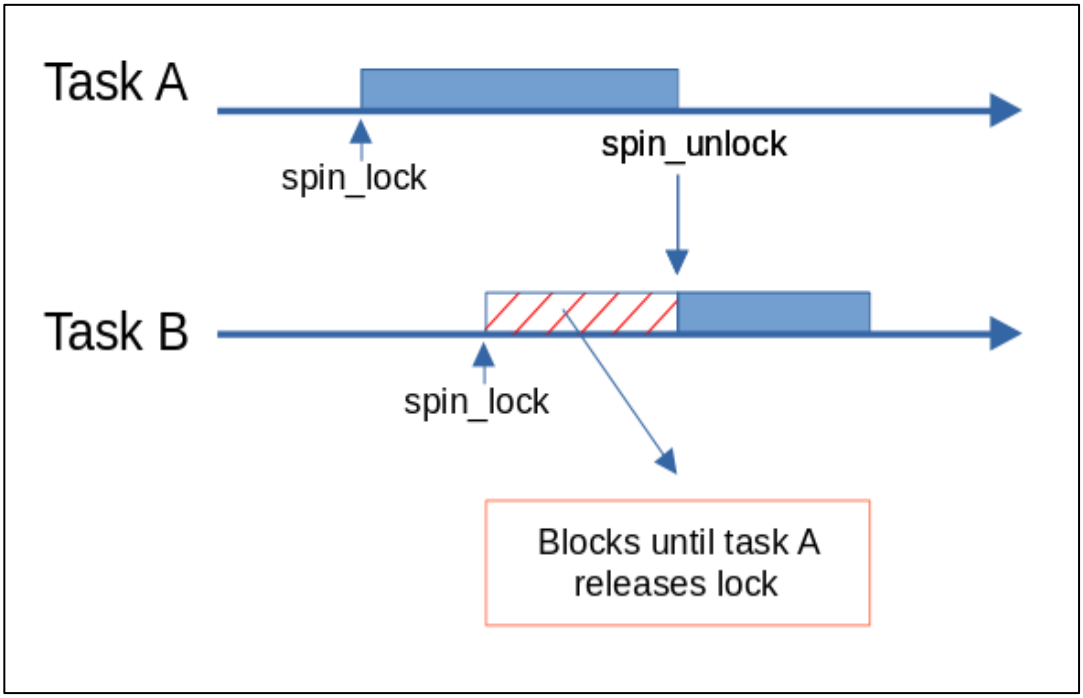
כאשר תהליך מסיים את השימוש במשאב, הוא מגדיל את המונה ומאפשר לתהליכים אחרים לגשת אליו:



SpinLock

SpinLock הוא מנגנון נעילה שבו תהליך המנסה לנעול משאב אינו עובר להמתנה, אלא ממשיך לבדוק בלולאה עד שהמשאב יתפנה. נשתמש במנעול מסוג זה כאשר זמן ההמתנה למשאב קצר מאוד, ואין טעם להעביר את התהליך למצב המתנה.

כמו כן, SpinLock נפוץ במקרים שבהם הנעילה מתבצעת בתוך לולאות קריטיות ב-Kernel Mode, שבהן המעבר למצב המתנה עלול לגרום לעיכובים מיותרים ולפגוע בביצועים.



למרות שהמנעולים אמורים לעזור לנהל את הזיכרון, כמו בכל דבר בחיים, עלולות להיווצר בעיות בניהול הנעילות, מה שעלול להוביל למספר תרחישים בעייתיים:

Deadlock

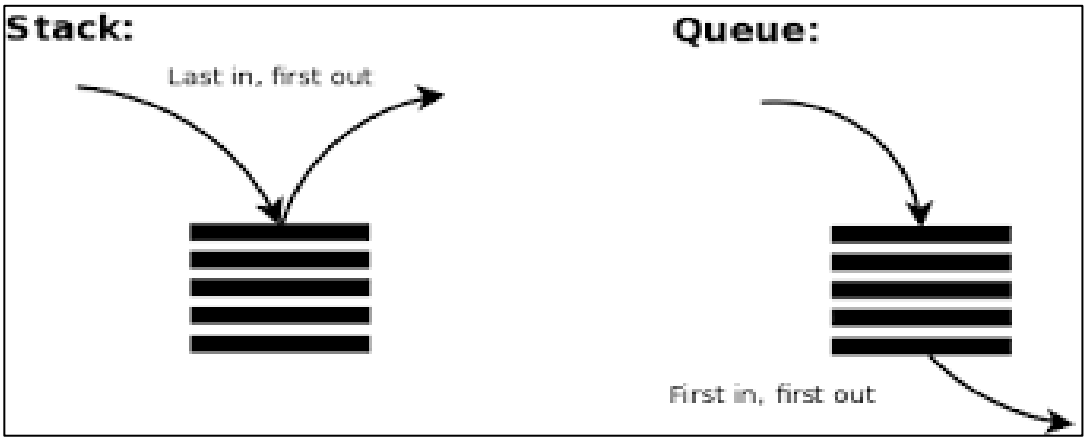
מצב שבו שני תהליכים מחכים לנעילה אחד של השני, ללא אפשרות להתקדם. זה יוצר לולאה אין-סופית שבה שני התהליכים נתקעים ואף אחד מהם לא יכול להמשיך בפעולה.

דוגמה: ניתן להקביל את זה לשני מכוניות שמנסים לעבור דרך כביש צר בו זמנית, אך שניהם מסרבים לזוז אחורה ולפנות את הדרך. התוצאה היא תקיעה מוחלטת, שבה אף אחד מהם לא מתקדם.

Starvation

מצב שבו תהליך נמנע מגישה למשאב כי תהליכים אחרים מקבלים כל הזמן קדימות על פניו. זה עלול להוביל למצב שבו תהליך מסוים לעולם לא יקבל גישה למשאב שהוא זקוק לו.

פתרון אפשרי: שימוש באלגוריתם FIFO (First In, First Out), שמבטיח שתהליך שממתין זמן רב יקבל עדיפות גבוהה יותר:



דוגמה: אפשר להקביל את זה למצב של אדם שמחפש מקום חניה ברחוב, וכל פעם שהוא מוצא מקום פנוי, הוא לא מצליח להחנות בו כי הוא לא נמצא בדיוק במקום הנכון. הוא נאלץ להמשיך לחפש בזמן שמקומות פנויים נוספים נתפסים על ידי אחרים.

זהו בדיוק המצב של Starvation - בניגוד ל-Deadlock, שבו אף אחד לא מתקדם, כאן חלק מהתהליכים כן מקבלים גישה, אך אחרים נותרים מאחור ללא סיכוי להתקדם.

סיכום

צללנו לעולם המורכב של ניהול הזיכרון במערכת ההפעלה. גילינו כי מערכת ההפעלה עושה קסמים עם זיכרון פיזי מוגבל על ידי שימוש בזיכרון וירטואלי, שמרחיב את המרחב כאילו היו לנו אין סוף מדפים בארון, הזיכרון הפיזי והווירטואלי עובדים יחד כדי שהתהליכים ימשיכו לרוץ בלי הפרעות, ואם אחד מהם תופס יותר מדי מקום - יש מנגנונים שמוודאים שלא יקרה פה אסון.

וכדי למנוע תוהו ובוהו מוחלט, מערכת ההפעלה משתמשת במנגנוני נעילה שלא נותנים לשני תהליכים להיתקע אחד מול השני כמו טיטינסקי ומאריסה בעונה 5 פרק 18 של קופה ראשית.

בסוף, כל הסנכרון הזה מאפשר למחשב לתפקד בלי דרמות, לפחות עד שהוא מחליט לקרוס בדיוק לפני הגשת ממ"ן.

מי אני?

יועצת אבטחת מידע ב-White Hat, שאוהבת את התחום ההתקפי. בין עבודה, אפיית בראוניז (התשובה להכל היא בראוניז ולא 42) אני אוהבת לחקור בזמן הפנוי על מתקפות, ענן ומערכות הפעלה.

מקורות מידע

- ספר מערכות הפעלה של מרכז החינוך לסייבר: https://data.cyber.org.il/os/os_book.pdf
- סרטון הסבר על זיכרון וירטואלי: https://youtube.com/watch?v=p9yZNLeOj4s
- סרטון הסבר על הקצאה דינאמית: https://www.youtube.com/watch?v=9uhSYDY4sxA
- מקור על RAM: https://en.wikipedia.org/wiki/Random-access_memory
- מקורות על Semaphore:

https://learnloner.com/operating-systems/semaphores-in-operating-systems/

https://www.youtube.com/watch?v=ukM_zzrIeXs

- סרטון הסבר על Mutex: https://www.youtube.com/watch?v=1tZhmTnk-vc
- הסבר על מבנה הזיכרון: https://www.geeksforgeeks.org/memory-layout-of-c-program/
- מקורות על זיכרון וירטואלי:

https://www.youtube.com/watch?v=A9WLYbE0p-I

https://en.wikipedia.org/wiki/Virtual_memory

- הסבר על Spinlock: https://en.wikipedia.org/wiki/Spinlock