זיכרון הגישה האקראית

מאת עידן שכטר

הקדמה

זיכרון הינו מושג נפוץ בעולם הטכנולוגי ולו מספר רב של סוגים וצורות, במאמר זה נעסוק בסוג ספציפי של זיכרון, והוא זיכרון הגישה האקראית, או בשמו המוכר - RAM (Random Access Memory)

במאמר נדון בנושאים הבאים:

- כיצד מוקצה הזיכרון
- שיטות שונות לניהול זיכרון
- תופעות שונות המתקשרות לעבודה עם זיכרון

כמו כן, נענה על שאלות מרתקות בנושאים הנ"ל.

לצורך הנאה מיטבית מתוכנו של המאמר, אמליץ לקרוא אותו עם רקע בעולם מערכות ההפעלה, או לפחות ידע בסיסי בתהליכים - אם כי הדבר אינו חובה.

אז למה בכלל צריך RAM?

ייתכן כי השאלה בכותרת של פסקה זו מובנת מאליה, אך במידה ולא - נבין תחילה, למה אנחנו בעצם צריכים RAM.

למעשה, ללא RAM כלל לא נוכל להשתמש במחשב שלנו, מנגנוני ה-BIOS השונים הנמצאים במחשבים מודרניים, כלל לא יטענו את מערכת ההפעלה במידה ולאותו מחשב לא מחובר זיכרון, שכן מערכת ההפעלה עצמה נטענת אל ה-RAM בתהליך העלייה.

לצורך העניין, נניח שהצלחנו לטעון את מערכת ההפעלה ללא RAM (אף על פי שהדבר אינו הגיוני) וכעת נרצה להריץ תוכנה כלשהי, תיאורטית, נרצה שמערכת ההפעלה תיצור תהליך של אותה תוכנה, אך מאחר ותהליך נמצא בזיכרון, ולנו אין זיכרון - לכן לא נוכל לבצע פעולה זו.

כאשר מערכת ההפעלה יוצרת תהליך מסוים, נניח של תוכנית שכתבנו ב-C, היא מספקת לתהליך משאבים הנחוצים להרצתו - זיכרון הוא אחד מהם, אל הזיכרון יטענו חלקי התוכנית השונים, בין היתר הקוד של התוכנית עצמה - בדרך זו המעבד למעשה "מריץ" את התוכנית שכתבנו.

למעשה, מידע הנטען לRAM מיועד לשימוש המעבד, וללא RAM למעבד אין דרך לגשת למידע הנחוץ לו על מנת לבצע את עבודתו - במהירות נוכל להבין כי **RAM** הינו רכיב אינטגרלי במחשב.

סוגים שונים של RAM

נוכל לחלק זיכרונות RAM ל-2 סוגים עיקריים:

1. זיכרון סטטי - SRAM
2. זיכרון דינמי - DRAM

כדי להבין לעומק את ההבדלים בין השניים - נדרש מאמר שלם, אך באופן רחב ההבדל הניכר הינו בביצועים.

זיכרון סטטי (SRAM) יצרוך פחות חשמל והוא נחשב מהיר יותר בעוד שזיכרון דינמי (DRAM) מציע גודל אחסון גדול יותר ועלויות הייצור שלו נמוכות יותר. כיום נעשים שימושים רבים בזיכרון סטטי ודינמי במוצרים ורכיבים טכנולוגיים שונים, אך סוג הזיכרון הרלוונטי לנו הוא זיכרון דינמי - והוא הזיכרון המשמש את המחשבים שלנו כיום.

סוג זיכרון זה נקרא DDR SDRAM ולו מספר דורות שפותחו לאורך השנים, בנוסף להיותו זיכרון ראשי, הוא משמש כרטיסי מסך מודרניים בגרסה מותאמת הנקראת GDDR. ההתקדמות הטכנולוגית בין דור ודור מתבטאת בנפח זיכרון גדול יותר, ביצועים טובים יותר וצריכת חשמל פחותה.

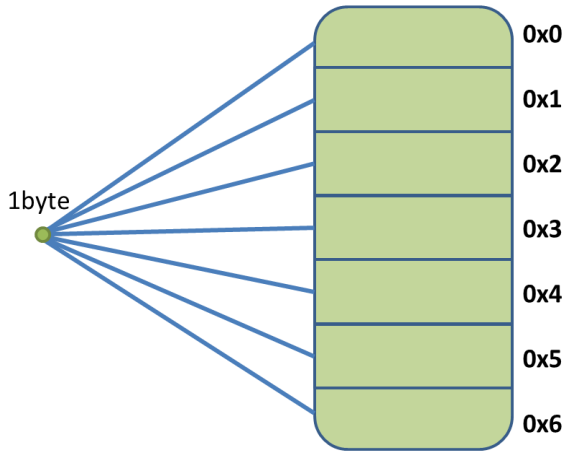
השוואה בין הדורות השונים נוכל לראות בטבלה הבאה:

Standard (Approximate Year Introduced)	Operating Voltage	Amount of Data Transferred (Words per Clock Cycle)	Associated RAM Clock Rates	Approximate Transfer Rates
SDRAM (1993)	3.3 V	1	66 – 133 MHz	100 – 166 MT/s
DDR SDRAM (2000)	2.6 V, 2.5 V	2	100 – 200 MHz	200 – 400 MT/s
DDR2 SDRAM (2003)	1.8 V, 1.55 V	4	200 – 400 MHz	400 – 1066 MT/s
DDR3 SDRAM (2007)	1.5 V, 1.35 V	8	400 MHz – 1066 MHz	800 – 2133 MT/s
DDR4 SDRAM (2014)	1.2 V	8	1066 – 1600 MHz	1600 – 3200 MT/s

[מקור: https://www.microcontrollertips.com/understanding-ddr-sdram-faq]

איך המחשב "תופס" זיכרון?

כאשר נדבר על זיכרון RAM, נוכל להתייחס אליו כאל מערך גדול, כך שכל תא במערך מיוצג על ידי כתובת הקסדצימלית ייחודית, השם "זיכרון גישה אקראית" (Random Access Memory) מגיע מהעובדה שמעבד יכול לגשת לכל תא בזיכרון בסיבוכיות O(1), או במילים אחרות - בהינתן כתובת נקבל תא, גודל כל תא הוא בייט אחד.



ההבדל בין ארכיטקטורת 32bit ל-64bit

כאשר מדברים על ארכיטקטורות שונות, מתכוונים לגודל האוגרים של מעבד.

למעבד בארכיטקטורת 32bit יהיו אוגרים בגודל 32bit, כלומר - כל אוגר יכול להכיל 2^{32} (4294967296) ערכים שונים, בהקשר של זיכרון, אוגר יכול להחזיק 4,294,967,296 כתובות שונות, מאחר וגודל כתובת הינו בייט אחד, מדובר בסדר גודל של 4GB.

מכאן עולה כי מעבד בארכיטקטורת 32bit יכול לגשת למקסימום של 4GB זיכרון גישה אקראי. לכן, במידה ומחברים למחשב בו מעבד 32bit זיכרון של 8GB, הוא יוכל להשתמש רק ב-4GB מתוך כל ה-8, ולכן נוצר מצב של Overkill.

כמו כן, כמות התהליכים שיכולים לרוץ במקביל מוגבלת - ככל שכמות הזיכרון המנוצל על ידי המעבד תתקרב ל-4GB, הביצועים של המחשב ילכו וידעכו, בנוסף - לא יהיה ניתן להריץ תהליך שדורש יותר מ-4GB של זיכרון.

מהר מאד נבין שעבודה עם ארכיטקטורת 32bit עלולה ליצור "תקרה ביצועית" כאשר הזיכרון הנדרש ברגע נתון מתקרב לגודל הזיכרון המוקצה למעבד.

הפתרון? להגדיל את האוגרים

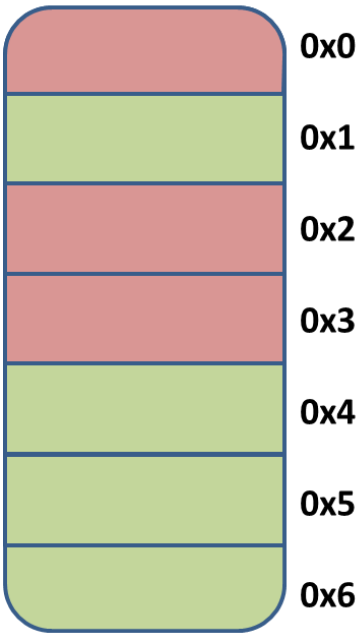
מעבד בארכיטקטורת 64bit עובד בצורה דומה לארכיטקטוריה של 32bit, רק שבו גודל האוגרים בגודל 64bit, ולכן הוא יכול לגשת ל-2^{64} כתובות - סדר גודל של 18.4 הקסהבייט (אם טרהבייט הם 1,000 ג'יגהבייט, ופטהבייט הם 1,000 טרהבייט, אז הקסהבייט הם 1,000 פטהבייט, או בקיצור - המון) כעת, בעבודה עם ארכיטקטורת 64bit המעבד יוכל לגשת למרחב זיכרון גדול משמעותית וה"תקרה הביצועית" נהיית גבוהה הרבה יותר. (כמו כן, חשוב לזכור שגם בעבודה עם ארכיטקטורת 64bit תקרה זו קיימת)

ניהול זיכרון

לאחר שלמדנו שניתן להתייחס לזיכרון במחשב כאל מערך גדול, בואו נבין איך מערכת ההפעלה מנהלת את הזיכרון של המחשב לצרכיו השונים.

מאחר ומערכת ההפעלה היא תוכנה בעצמה, היא נטענת אל הזיכרון בתהליך המכונה Boot, מערכת ההפעלה נטענת אל האזור הנמוך בזיכרון (הכתובות הנמוכות, 0 והלאה) המכונה Low Memory, שאר הזיכרון המכונה High Memory מיועד לתהליכי משתמש.

נשאל את עצמנו את השאלה הבאה - באיזה צורה טוענת מערכת ההפעלה מידע לזיכרון? נניח ונרצה לטעון לזיכרון מידע כלשהו, בגודל של 4bytes, האם נטען את הבתים אחד אחרי השני? (כלומר בכתובות עוקבות) אולי נטען אותם במקומות נפרדים בזיכרון אך נקבע חוקיות היוצרת ביניהם סדר? (רשימה מקושרת). נניח ונרצה לטעון את המידע בצורה רציפה, לשם כך נצטרך 4 כתובות רציפות בזיכרון בהן לא נעשה שימוש. בשרטוט שלפנינו נוכל לראות מערך בעל 7 תאים, כך שתא אדום מסמן תא שנמצא בשימוש ותא ירוק מסמן תא הפנוי לשימוש.



נניח שמערך זה הוא זיכרון של מחשב כלשהו, וברצוננו לטעון לזיכרון זה מידע בגודל 4bit באופן רציף. נוכל לשים לב שישנים 3 תאים אדומים ו-4 תאים ירוקים. מאחר והמידע אותו נרצה לטעון דורש 4 תאים ירוקים, במחשבה ראשונה נראה שנוכל לטעון את המידע לזיכרון בלי בעיה, אך נזכר ונבין כי על מנת לטעון את המידע לזיכרון, יידרשו 4 תאים רציפים.

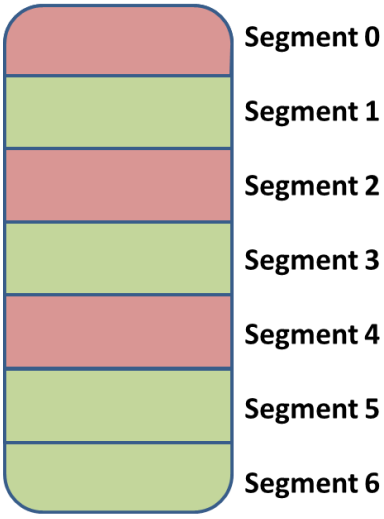
כלומר, הגודל של הזיכרון הדרוש לטעינת המידע זמין, אך לא בצורה רציפה. התופעה אליה נחשפנו זה עתה נקראת External fragmentation גודל הזיכרון הדרוש זמין, אך לא בצורה רציפה - לכן לא נוכל לנצל אותו.

כאשר נמשיל מצב זה למציאות, נוכל לדמיין מצב בו נרצה לטעון לזיכרון תהליך בגודל 4GB, כאשר גודל הזיכרון זמין אך לא בצורה רציפה. טעינתו של תהליך לזיכרון מתעכבת ואף נמנעת, דבר הפוגע משמעותית בחוויית המשתמש.

פתרון יצירתי לבעיה יהיה לחלק את הזיכרון לחלקים שווים הנקראים סגמנטים, כך שכאשר תהליך מסוים יטען לזיכרון, הזיכרון הדרוש יחולק על פני סגמנטים שונים בזיכרון, כך שהם אינם צריכים להיות רציפים וכלל לא תלויים במיקומם, הם יהיו מקושרים זה לזה בצורה כרונולוגיות המיוחסת לתהליך לו הם שייכים, לכן נוכל להסתכל עליהם כעל רשימה מקושרת.

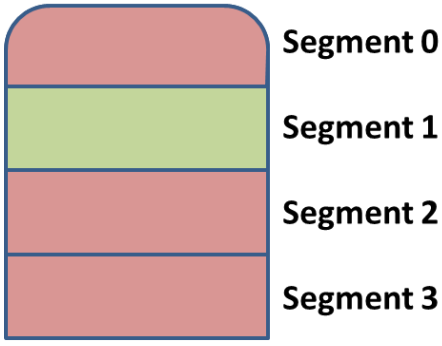
כעת לא נהיה זקוקים למקטע רציף בזיכרון על מנת לטעון את התהליך שלנו - כל עוד הזיכרון זמין נוכל לנצלו. נניח שאנחנו עובדים עם זיכרון בגודל 8GB המחולק לסגמנטים של 4MB כל אחד, כעת נרצה לטעון לזיכרון תהליך בגודל 16MB, מאחר וגודל כל סגמנט הוא 4MB, לטעינת תהליך זה יידרשו 4 סגמנטים, כלומר כל עוד קיימים 16MB זמינים בזיכרון, נוכל לטעון את התהליך לזיכרון, ללא תלות במיקום הסגמנטים. נמחיש את הכתוב התמונה הבאה:

בדומה לדוגמא הקודמת, המערך הבא מייצג זיכרון של מחשב, רק שהפעם כל תא מייצג סגמנט בגודל 4MB (כמובן שהדבר אינו מדויק כי סגמנט מכיל מספר כתובות זיכרון, אך הדבר הוא לצורך המחשה) מאחר וסגמנט אדום מייצג סגמנט הנמצא תחת שימוש וסמגנט ירוק הינו חופשי, נוכל להסיק כי קיימים 16MB זמינים בזיכרון. אף על פי שהסגמנטים אינם "שכנים", נוכל לעשות בהם שימוש.



עושה רושם שפתרנו את הבעיה, אך כעת נבין שגם לשימוש בפרגמנטים יש תופעות לוואי. תארו לכם מצב בו נעבוד עם זיכרון בגודל 8MB המחולק ל-4 סגמנטים שגודלם 2MB. נניח שכל הזיכרון זמין לשימוש, מה יקרה במצב בו נרצה לטעון לזיכרון תהליך הדורש 5MB, 8MB של זיכרון זמינים לשימוש?

מאחר וגודל כל סגמנט הוא 2MB, תהליך בגודל 5MB יתחלק סה"כ ל-3 סגמנטים שונים.



לאחר טעינת התהליך נותרנו עם סגמנט פנוי אחד, או במילים אחרות - עם 2MB זמינים. רגע, אמרנו שאנחנו עובדים עם זיכרון בגודל 8MB והתהליך דורש 5MB, אז איך יצא מצב שנותרנו עם 2MB? הרי 8 פחות 5 הם 3, מה קרה פה בעצם? שרפנו 1MB?

בת'כלס? - כן.

מערכת ההפעלה אכן סיפקה לתהליך 3 סגמנטים, אך בפועל התהליך ניצל 2.5 מהם - 2 סגמנטים נוצלו במלואם, ועוד שרק חצי מהסגמנט הנותר נוצל (1MB) ולמעשה התהליך קיבל יותר זיכרון ממה שהוא זקוק לו, ובכך הוא מבטל את זמינות זיכרון זה (אך בואו לא נמהר להאשים את התהליך, מערכת ההפעלה היא האחראית). תופעה זו מכונה Internal Fragmentation.

אמנם בדוגמא שראינו נוצר ביזבוז של 1MB, ואין ספק שבעבודה עם 16GB גודל זה נשמע מזערי, אך תארו לכם מצב בו גודל של סגמנט היה 64MB, תהליך בגודל 65MB ידרוש 2 סגמנטים, את אחד מהם ינצל במלואו, אך ינצל רק 1MB מהסגמנט השני, כלומר נוצר בזבוז של 64MB! שזו כמות הרבה יותר משמעותית, נכפיל את בזבוז זה במספר גדול של תהליכים ונקבל כמות גדולה של זיכרון שהולכת לפח...

הפתרון? שימוש בסגמנטים קטנים ככל הניתן.

זיכרון וירטואלי

במחשב הביתי שלי יש 16GB של זיכרון, המשחק Call Of Duty: Modern Warfare שוקל 175GB. במחשבה ראשונית ופזיזה ניתן להסיק שבצורה כלשהי מערכת ההפעלה טוענת משחק במשקל 175GB לזיכרון בגודל 16GB, דבר שנשמע לא כל כך אפשרי, בהתחשב בעובדה שהמשחק שוקל פי 10 מכמות הזיכרון הזמינה במחשב.

מהר מאד ניתן להבין שניתן להסתכל על המשקל של המשחק כ"כל מה שיש למשחק להציע", מודלים תלת ממדיים של מפות, כלי נשק, רכבים, פס קול, וכמובן מנוע המשחק. מאחר ולא כל המידע הנ"ל רלוונטי בכל רגע נתון במשחק (כאשר נתחיל משחק, רק המפה בה נשחק רלוונטית, לכן אין בכלל צורך בשאר המפות בזיכרון), לכן, בעת ריצת המשחק נטען רק את מה שרלוונטי לרגע הנתון, מה שבפועל ינוע בין 3GB - 5GB של זיכרון - נראה שפתרנו את הבעיה די בקלות. (כמובן שיש לציין שקיימות שיטות רבות שבאות לידי ביטוי על מנת להקטין את כמות הזיכרון הנדרשת להרצת המשחק, אך שיטות אלו אינן רלוונטיות למאמר זה). מה לגבי סרט? נניח ועל המחשב שלי גרסת BluRay של סרט עדכני ששוקל 25GB - איך ייתכן שנוכל להריץ אותו כאשר הזיכרון הזמין לנו הוא 16GB?

ובכן בפועל, אומנם הסרט שוקל 25GB, אך המידע הרלוונטי לנו כצופים הוא אך ורק המידע או "הקטע" בו אנחנו צופים באותו רגע, יתר הסרט אומנם רלוונטי להמשיך הצפייה או להרצה קדימה \ אחורה של

הסרט, אבל לא לאותו רגע - לכן אין צורך לטעון אותו לזכרון. או במילים אחרות, התוכנה בה נשתמש על מנת לצפות בסרט, לא צריכה לקרוא את כל הסרט בכל רגע נתון, אלא רק את החלק הרלוונטי לנו כצופים.

אז איך זה קורה?

מערכות הפעלה מודרניות משתמשות בשיטת ניהול זיכרון הנקראת זיכרון וירטואלי, בשיטה זו מערכת הפעלה מנצלת את הדיסק הקשיח של המחשב (שבו בדרך כלל זיכרון גדול משמעותית מזיכרון ה-RAM) על מנת להריץ תהליכים בצורה יעילה.

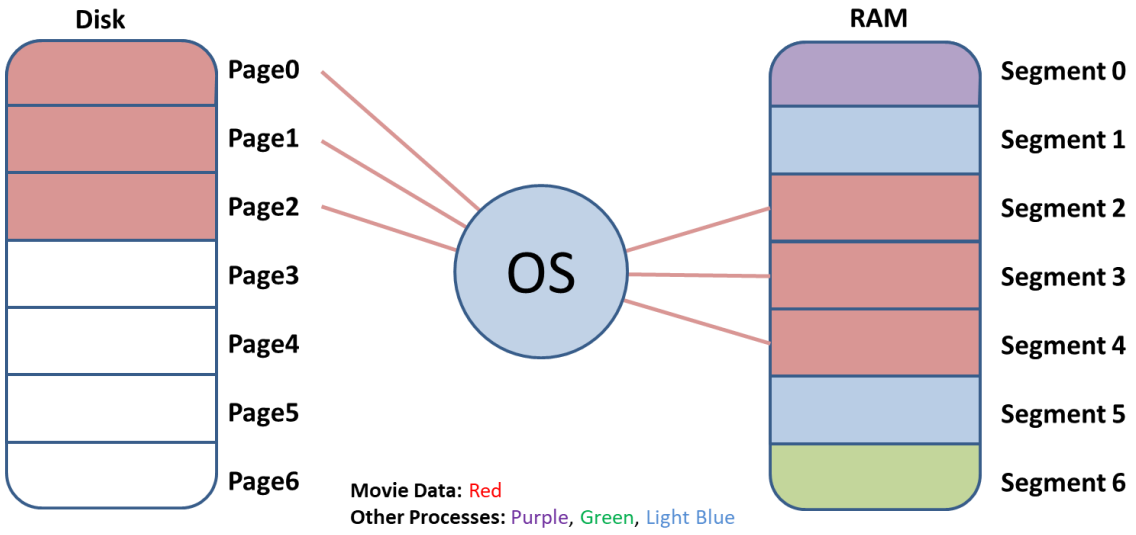
במקרה שלנו, "סרט" אינו תהליך, אלא קובץ בפורמט וידאו שעובר עיבוד על ידי תוכנת Media Player כלשהי, אך כמו שדנו קודם לכן, אותה תוכנה לא יכולה לקרוא את כל הסרט במלואו בו זמנית, מאחר ולמחשב לא קיים מספיק זיכרון לכך.

בחלק הקודם במאמר למדנו על שיטת ניהול זיכרון מודרנית בה הזיכרון מחולק לסגמנטים בגודל קבוע, בעוד שהזיכרון מחולק לחלקים המכונים סגמנטים, ה"סרט" יחולק בדיסק הקשיח לחלקים שווים הנקראים דפים (Pages), בחלוקה זו גודל דף זהה לגודלו של סגמנט, מטרתם של הדפים היא "להכין" את חלקי הסרט לקראת טעינתם לזיכרון.

לאחר שהסרט מחולק לדפים שונים בדיסק הקשיח, כאשר נרצה לצפות בסרט ותוכנת ה-Media Player תתחיל לפרש את הסרט מנקודת ההתחלה שלו, יחידת ניהול הזיכרון של המחשב (MMU, רכיב חומרתי שבין היתר אחראי על ניהול זיכרון וירטואלי) תטען את ה-Page הרלוונטי, כלומר את ה-Page המכיל את המידע של השניות הראשונות בסרט.

בצורה מתמשכת יחידת ניהול הזיכרון תטען את הדף הרלוונטי לצפייה, בעוד שתפסיק את השימוש בדפים שאינם רלוונטיים לרגע הצפייה.

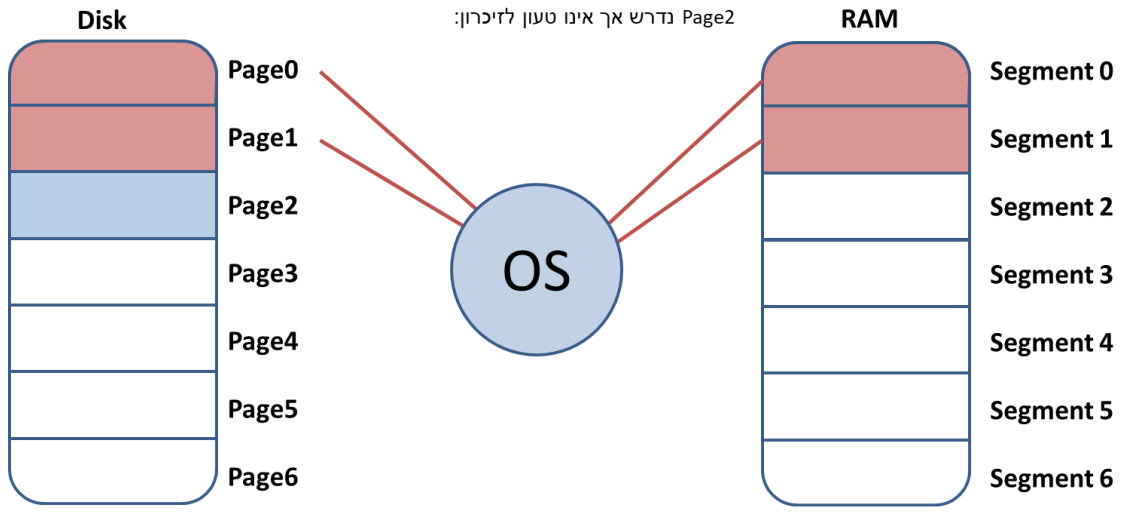
בדרך זו מערכת ההפעלה חוסכת בכמות הזיכרון הנדרשת להרצת הסרט בכל רגע נתון.

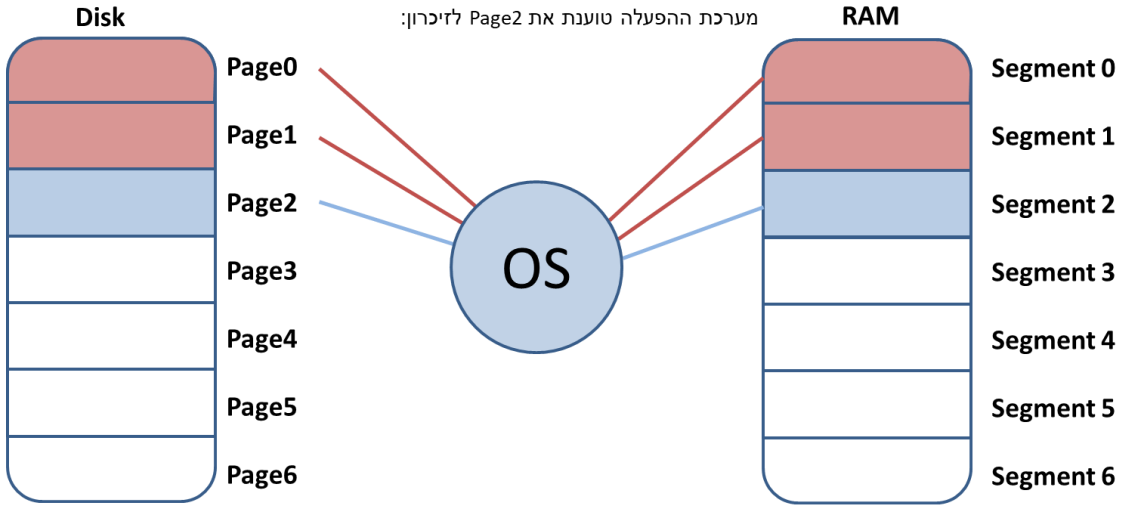


בפועל, כאשר תוכנת ה-Media Player מפרשת את קובץ הסרט, יחידת ניהול הזיכרון תבדוק האם המידע הרלוונטי לתוכנת ה-Media Player נמצא כבר בזיכרון, או שיש לשלוף את ה-Page הרלוונטי מהדיסק הקשיח - פעולה זאת נעשית בעזרת טבלת דפים (Page table) שתפקידה הוא בין היתר למפות את הדפים הנמצאים בזיכרון על ידי יצירת רשומה (Entry) ייחודית עבור כל דף. במצב בו המידע הרלוונטי לא מתועד בטבלת הדפים מערכת ההפעלה תטען לזיכרון את ה-Page הרלוונטי, מצב זה נקרא Page Fault והוא מתחלק ל-3 סוגים.

1. Minor Page Fault - מצב בו ה-Page הרלוונטי כבר נטען לזיכרון אך יחידת ניהול הזיכרון לא "מודעת" לכך, במצב זה תיווצר רשומה רלוונטית בטבלת הדפים על מנת לתעד את השימוש באותו דף.
2. Major Page Fault - מצב בו ה-Page הדרוש לא נטען לזיכרון ועל מערכת ההפעלה למצוא לו מקום מתאים על מנת שיוכל להיעשות בו שימוש.
3. Invalid Page Fault - מצב בו לא ניתן לטעון את ה-Page הרלוונטי לזיכרון (חוסר זיכרון זמין, Page שהתוכן שלו עובר שינוי על ידי תוכנה אחרת, וכו')

מצב של Major Page Fault ניתן להמחיש בצורה הבאה:





לצורך המחשת הרעיון שזה עתה למדנו, בתמונה הבאה נוכל לראות את הזיכרון שנגן המדיה VLC צורך כאשר הוא אינו מנגן סרט:

Name	Status	4% CPU	31% Memory	0% Disk	0% Network	4% GPU
Apps (5)						
Google Chrome (27)		0.6%	1,105.6 MB	0.1 MB/s	0.1 Mbps	0%
Spotify (5)		0.1%	134.6 MB	0.1 MB/s	0 Mbps	0%
Task Manager		0.4%	23.1 MB	0 MB/s	0 Mbps	0%
VLC media player		0%	15.8 MB	0 MB/s	0 Mbps	0%
Windows Explorer (2)		1.3%	45.0 MB	0 MB/s	0 Mbps	0%

במידה ונגן ב-VLC את הסרט הקלאסי Fight Club בגרסת 1080p, משקל הסרט הוא 12.8GB ואורכו 2:19:08:

Name	Status	2% CPU	33% Memory	0% Disk	0% Network	5% GPU
Apps (6)						
Google Chrome (27)		0%	1,097.2 MB	0 MB/s	0 Mbps	0%
Paint		0%	66.6 MB	0 MB/s	0 Mbps	0%
Spotify (5)		0.1%	138.5 MB	0.1 MB/s	0 Mbps	0%
Task Manager		0.2%	23.2 MB	0 MB/s	0 Mbps	0%
VLC media player		0.5%	98.1 MB	2.8 MB/s	0 Mbps	4.7%
Windows Explorer (3)		0%	48.7 MB	0 MB/s	0 Mbps	0%

ניתן לראות בבירור שכמות הזיכרון ש-VLC צורך עלתה באופן משמעותי, וזאת משום שכעת קיים יותר מידע "שרלוונטי" ל-VLC ברגע נתון. דוגמא נוספת היא ניגון הסרט Days Of Future Past מסדרת סרטי X-Men בגרסת UHD ב-VLC - משקל הסרט הוא 21.2GB וארכו 2:11:35:

Name	Status	2% CPU	34% Memory	0% Disk	0% Network	9% GPU
Apps (7)						
Google Chrome (27)		0.2%	1,097.9 MB	0.1 MB/s	0.1 Mbps	0%
Paint		0%	132.1 MB	0 MB/s	0 Mbps	0%
Spotify (5)		0.1%	137.3 MB	0.1 MB/s	0 Mbps	0%
Task Manager		0.3%	23.2 MB	0 MB/s	0 Mbps	0%
VLC media player		0.2%	253.6 MB	0 MB/s	0 Mbps	8.6%
Windows Explorer (3)		0.2%	48.7 MB	0 MB/s	0 Mbps	0%

פעם נוספת ניתן לראות עלייה משמעותית בכמות הזיכרון ש-VLC צורך, על פי העקרונות שלמדנו נוכל להסיק שמשום שהסרט כבד יותר באופן משמעותי (מה שנובע מאיכות הפורמט) המידע הדרוש ל-VLC בכל "רגע של ניגון" גדול יותר.

סיכום

לאורך המאמר נחשפנו לעקרונות טכניים שונים שעזרו לנו להבין בצורה עמוקה את חשיבותו של זיכרון כמרכיב חיוני במחשב שלנו, למדנו על סוגים שונים של זיכרון ועל שיטות שונות בהן מערכת ההפעלה מנהלת זיכרון תוך כדי הקפדה על יעילות מרבית והבטחת חווית המשתמש.

מאמר זה מראה לנו שלזיכרון השלכות רבות על תחומים טכנולוגיים שונים, חומרתיים ותוכנתיים כאחד, ולכן הבנה טובה יותר של זיכרון מקנה ידע רלוונטי שיסייע לנו להבין עקרונות טכנולוגיים שונים. לסיום, מקווה שנהניתם מקריאת המאמר, וכמו כן אשמח לקבל תגובות בנושא.

תודה רבה, עידן.