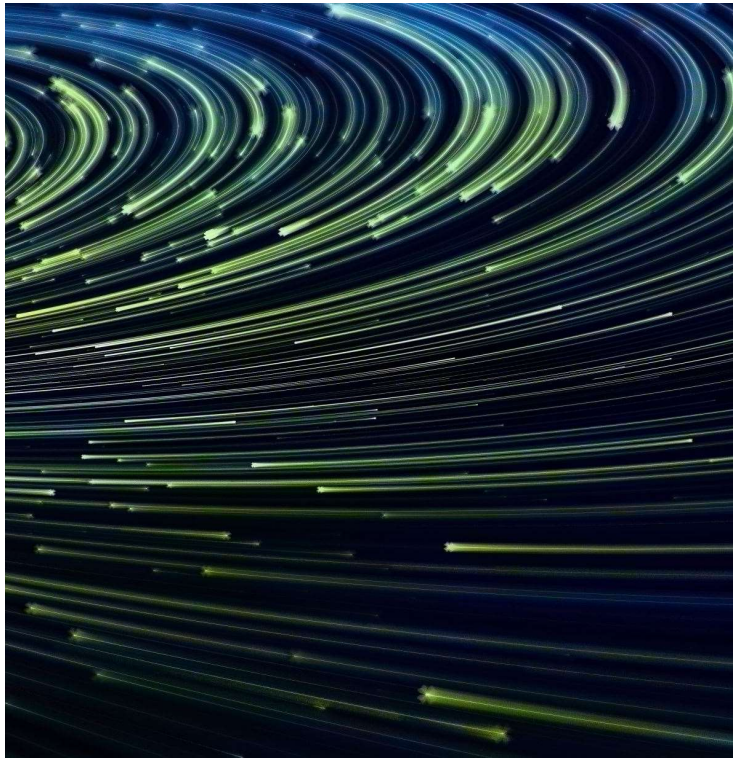
תרגול 9

מערכות הפעלה

על הפרק היום



- מיפוי זיכרון
- שיתוף זיכרון
- זיכרון משותף (בRAM)
- קובץ משותף (בדיסק)
- תקשורת בין תהליכים
- התנסות בסביבת windows

מנגנוני תקשורת בין תהליכים

- במערכות הפעלה מודרניות, ישנם תהליכונים רבים שפועלים בו זמנית.
- אבל, כפי שכבר ראינו – לעיתים יש צורך בתקשורת(בין היתר במובן שיתוף פעולה) בין תהליכונים, מסיבות שונות.
- הכרנו כבר כמה מנגנוני תקשורת – כגון:
  - סמפורים ומנעולים – שהם סוג של מנגנון תקשורת
  - וכן על העברת מסרים ע"י קווי תקשורת
  - וישנן עוד שיטות, חלקן מערבות את מערכת ההפעלה, וחלקן לא.

- במצגת זו, נרחיב על העברת נתונים ותקשורת ע"י שימוש בזיכרון משותף

תקשורת בין תהליכים ע"י זיכרון

- עד עכשיו – ראינו שכל תהליך הוא ישות נפרדת, שיש לה את אזורי הזיכרון שמיועדים לה, ושלאף תהליך אחר אין גישה אליהם.
- אמנם - במקרה של תהליכונים בנים של אותו תהליכון אב – ישנה גישה לנתונים משותפים, וניתן לסנכרן בין התהליכונים, ולהעביר מידע מתהליכון לתהליכון.
- אבל במקרה שמדובר בתהליכונים של תהליכים שונים – הרי שישנה הפרדה מוחלטת, כך שלכל תהליך יש אזור זיכרון שלו, שרק הוא יכול לגשת אליו .

- אם כן, כיצד יוכלו מספר תהליכים לשתף מידע ביניהם במקרה הצורך?

60 שניות של מושג בקצרה – **מיפוי זיכרון**

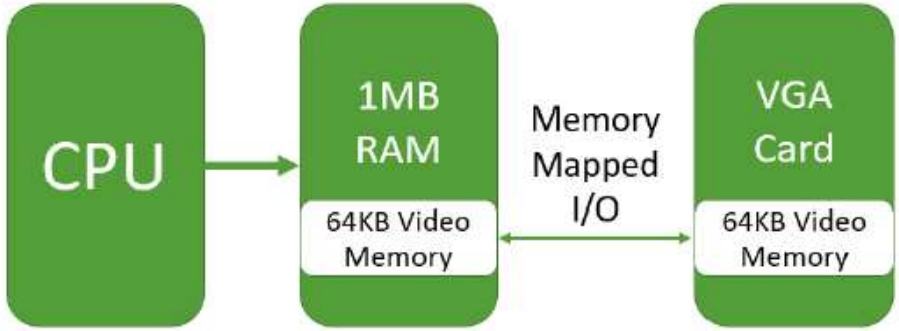
- מיפוי זיכרון הינו תהליך שבו נוצר קישור **בין טווח כתובות** של הזיכרון הראשי (ה RAM) לבין זיכרון משני (של התקן כלשהו).
- לצורך כך:
  - נקבע מקום מוגדר בRAM שמשמש אך ורק עבור התקן זה
  - נוצר עצם המהווה stream להעברת מידע מההתקן לRAM ולהיפך.
- התקן חומרה שבוצע עבורו מיפוי לזיכרון RAM נקרא Memory Mapped I\O (Input\Output = O\I)
- משמעות הקישור היא יצירת מנגנון שבו כל שינוי שמתבצע בזיכרון אחד , יוצר את אותו השינוי בזיכרון האחר.
- שיטה זו מאפשרת למעבד לפנות בקלות להתקני חומרה , ללא צורך בהפעלת פסיקות חומרה (interrups) , וללא צורך בפניה ל-ports ושימוש בפקודות מיוחדות

Memory Mapped I/O – איפה?

- השימוש ב- Memory Mapped I/O קלאסי לכרטיסי גרפיקה, עקב מהירות העדכון הרבה שהם דורשים (לדוגמה: במשחקי מחשב, מיפוי הזיכרון הוא אפשרות מקובלת).
  - במערכת שיש בה מעבד, זיכרון RAM וכרטיס גרפי: כל מה שמוצג למשתמש על המסך חייב להיות בזיכרון של הכרטיס הגרפי.
- כמו כן, ניתן לעשות מיפוי להתקן שמירת קבצים חיצוני (למשל: דיסק הקשיח).
  - הערה: ניתן לבצע מיפוי לקובץ מסוים ולא לכל ההתקן. במקרה כזה נקרא לקובץ: Memory Mapped File

דוגמא למיפוי זיכרון

- במעבד ה-8086 ההיסטורי, בו למעבד הייתה גישה ל-1 MB של זיכרון RAM:
- לצד המעבד היה מותקן כרטיס גרפי VGA שמסוגל להציג 256 צבעים על מסך שגודלו 200X320 פיקסלים.
  - כל פיקסל דרש byte אחד. לכן גודל הזיכרון של הכרטיס היה KB64 (KB64=64000=320X200).
- זיכרון מהכרטיס הגרפי, שהתאים לתצוגת מסך, מופה אל אזור מסוים בזיכרון ה-RAM שהתחיל בכתובת 0xA0000 וגודלו היה 0x10000 (כלומר 64KB)



יתרונות וחסרונות של מיפוי זיכרון

- יתרונות:
  - פשטות ומהירות: כדי לפנות אל זיכרון ההתקן, המעבד פשוט מבצע פקודות העתקה מ/אל אזור הזיכרון של ההתקן שמופה ל- RAM . חוסך את הצורך בפסיקות חומרה איטיות.
- חסרונות:
  - המיפוי גוזל חלק **מטווח הכתובות** מזיכרון ה-RAM של המעבד, שלמעשה הפך להיות בלתי שמיש לכל מטרה שאינה תקשורת עם ההתקן.
- אבל יש עוד יתרונות.... ועל כך בשקופית הבאה!

מה יתרון לאדם בכל עמלו??

- לפעולה של מיפוי עבור קבצים אל ה- RAM יש שני יתרונות:
  - חיסכון במקום ב-RAM(ביחס לטעינת כל הקובץ ל RAM)
    - לצורך כך נמפה בכל פעם רק חלק מהקובץ.
  - אפשרות לתקשורת בין תהלכים
    - ע"י שיתוף אזור זיכרון בין מספר תהליכים.

- תזכורת:
  - בתהליך שנקרא "מיפוי" אנחנו מייצרים קשר בין ה-RAM לקובץ בדיסק הקשיח, כך שכל שינוי באחד -משפיע על השני.

איך נוכל לחסוך בזכרון?

- הרעיון:
- במקום לטעון את **כל** הקובץ מהדיסק הקשיח ל -RAM וממנו אל הזיכרון של ה-Process נבצע בכל פעם טעינה של קטע מידע אחר מהדיסק אל ה– RAM.

  הערה: היה ניתן להגיע לאותה תוצאה ע"י קריאה רגילה של חלק אחר מהקובץ כל פעם,

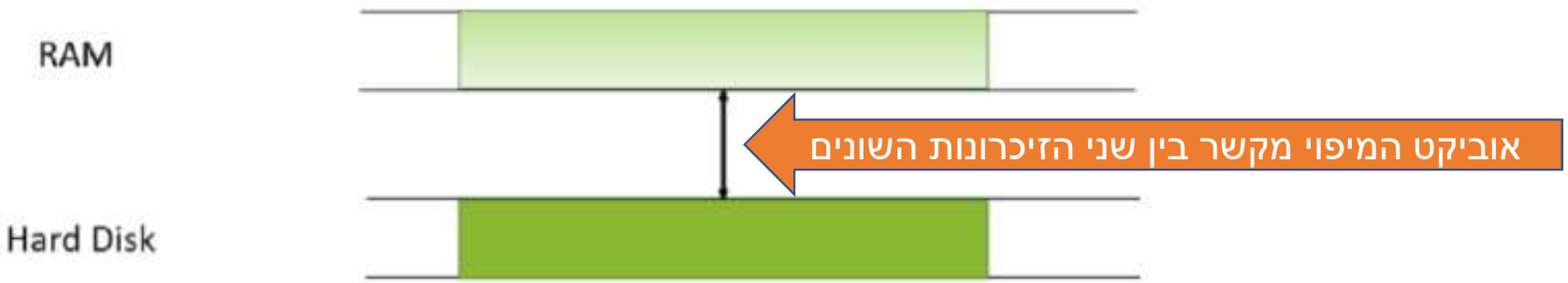
  אך אין לשיטה הזאות את היתרונות האחרים של מיפוי לזיכרון.
- כל עוד המידע מהדיסק יהיה בשימוש על ידי ה -Process, נשמור אותו ב- RAM.
- כאשר לא נזדקק יותר לקטע המידע נזרוק אותו ונשתמש באותו אזור ב-RAM לטובת קטע מידע אחר.

מיפוי קובץ במערכת חלונות

- לצורך מיפוי קובץ במערכת ההפעלה windows, קיימות מספר פונקציות WINAPI:
  - CreateFileMapping
  - OpenFileMapping
  - MapViewOfFile
  - UnMapViewOfFile
  - CloseHandle

CreateFileMapping

- בתהליך המיפוי נוצר אובייקט מיפוי שניתן לגשת אליו.
  - הזיכרון ב-RAM לא באמת מוקצה בשלב המיפוי- אלא, לפי תבנית עיצוב תכנותית -"lazy loading"- המידע נטען מהדיסק הקשיח רק כשצריכים אותו.



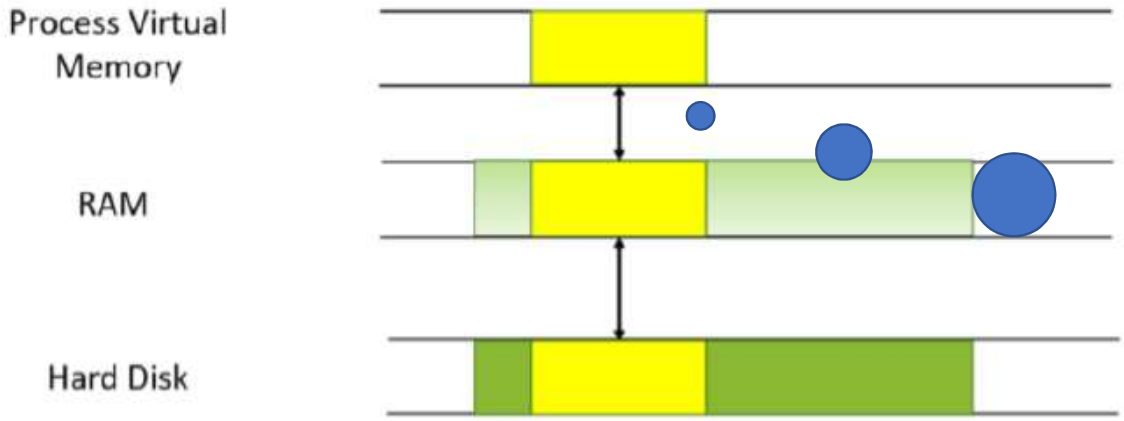
OpenFileMapping

- במקרה שקיים כבר אובייקט מיפוי, הפעולה מאפשרת גישה לאובייקט זה.

- שימו לב – פעולה זו כבר כלולה בפונקציה **CreateFileMapping** כך שאין צורך לבצע open אם התהליך שלנו ביצע create
- בהמשך – נראה שבאמצעות פעולה זו נוכל לגשת לאובייקט המיפוי גם מתהליך נוסף.

MapViewOfFile

- כאשר process מבקש להשתמש בקטע מהקובץ, הוא יבקש למפות (להעביר) אותו מה-RAM לזיכרון הווירטואלי שלו .
  - אם הקובץ לא הועתק עדיין, אובייקט המיפוי יבצע העתקה מהקובץ ל-RAM



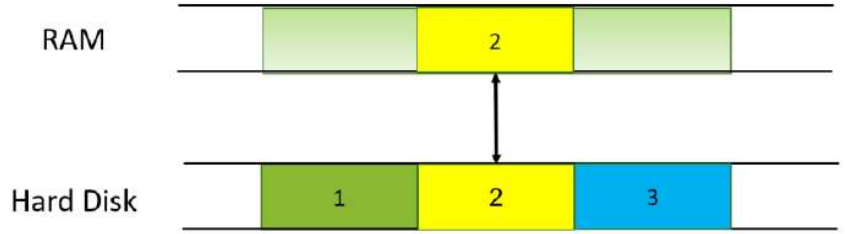
הערה:
עדיין לא למדנו היכן בדיוק ממוקם הזיכרון הווירטואלי של תהליך בזמן ריצה.
בפועל הוא חלק מה RAM, כך שלכאורה מדובר בתהליך פנימי ב RAM

הבחנה דקה, אך חשובה מאד!

- למילה "מיפוי" יש שתי משמעויות דומות, אבל שונות.
- כאשר אנו מדברים על **מיפוי של קובץ** לזיכרון הראשי – הכוונה ליצירת קישור בין שני סוגי הזיכרונות באמצעות אפיק המעביר מידע מאחד לשני.
  - המידע קיים פעמיים – אך נשמר הסנכרון בין שני המיקומים.
- כאשר אנו מדברים על **מיפוי של איזור בזיכרון הראשי** לזיכרון הווירטואלי של תהליך – הכוונה ליצירת קישור בין שני הזיכרונות, כך שהתהליך יכול להשתמש במידע.
  - המידע קיים רק פעם אחת, אבל תהליך אחד או יותר רואים במידע הזה חלק מהזיכרון שלהם.

גודל קטע הקובץ הממופה

- זיכרון ממופה צריך להתחיל מכתובת "עגולה". כלומר: כתובת שמתחלקת בחזקות 2.
- לכן כדאי שגודל אזור הזיכרון הממופה יהיה עגול, וגודל כל החלקים שאנו ממפים אליו יהיה תואם.
  - (בהמשך נקרא לגודל זה "דף" או "מסגרת")



- ההגדרה של גודל במערכת חלונות, מתבצעת ע"י שימוש ב- Alignment Granularity

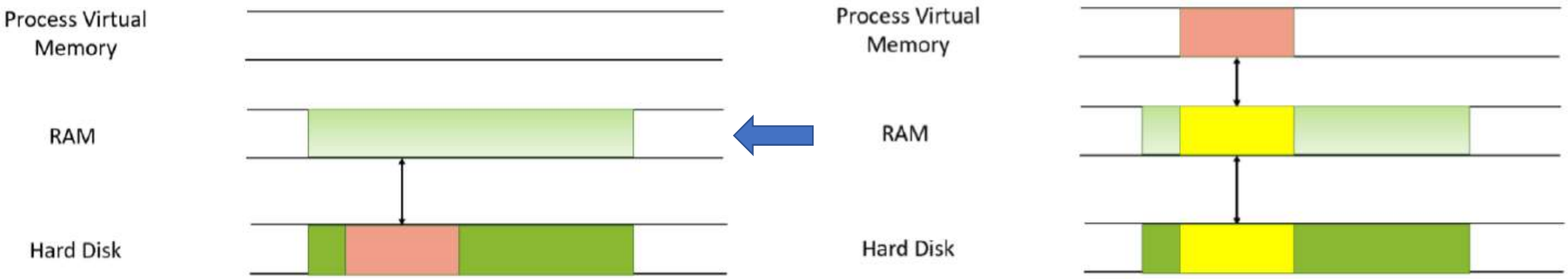
```
// get system memory alignment granularity (usually 65536)
SYSTEM_INFO sys_info;
GetSystemInfo(&sys_info);
int mem_buffer_size = sys_info.dwAllocationGranularity;
```

איך כל זה חוסך לנו זיכרון?

- אפשר לנצל את העובדה שכאשר אנחנו מסיימים לעבור על קטע מהקובץ אנחנו לא צריכים אותו יותר: לאחר שנסיים לעבוד על הקטע שמיפינו, נוכל להסיר את המיפוי שלו מהזיכרון ואז נטען את הקטע הבא בדיוק לאותו מקום בזיכרון.
- בצורה זו, הגודל של הזיכרון שנשתמש בו יהיה די קבוע ודומה לגודלו של הקטע שמיפינו אל הזיכרון.
- גם אם הקובץ שלנו מאד גדול, לא נצטרך להגדיל את כמות הזיכרון, רק להחליף יותר פעמים את הקטע הממופה לזיכרון.

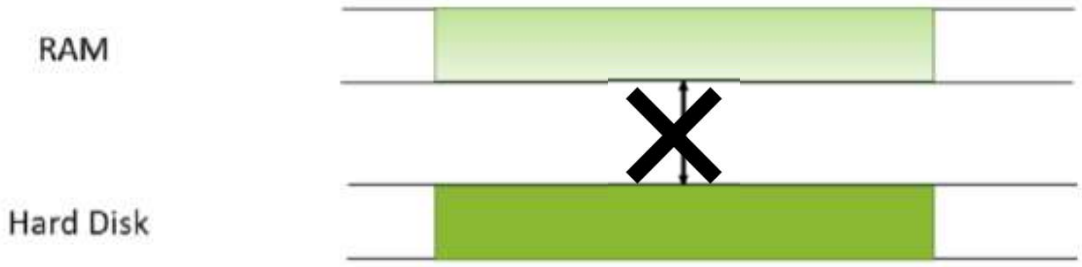
UnMapViewOfFile

- כאשר process מסיים לעבוד על קטע הזיכרון הוא מבקש להסיר את המיפוי של הקטע.
- פעולה זו גורמת לכך שהזיכרון כולל השינויים שנעשו בו נכתבים בחזרה לדיסק הקשיח.



CloseHandle

- עם סיום העבודה על הקובץ הממופה, יש להרוס את עצם המיפוי.
- פעולה זו מבצעת סגירת הגישה לאובייקט המיפוי (הפעולה ההפוכה מ CreateFileMapping)

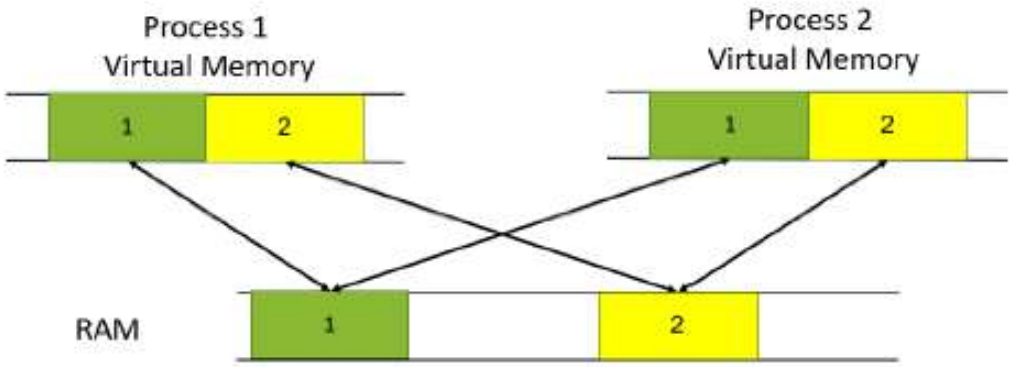


שימוש נוסף למיפוי זיכרון

- מיפוי זיכרון, כאמור – יכול לסייע לנו בחסכון במקום בזיכרון הראשי. אבל הוא יכול לסייע לנו גם לתקשורת בין תהליכים באמצעות זיכרון משותף.
- תקשורת בין תהליכים יכולה להתבצע באמצעות זיכרון – אבל גם באמצעים נוספים.
- גם התקשורת באמצעות הזיכרון יכולה להתבצע בכמה אופנים.
- נבחין בין:
  - קובץ משותף (shared file) – קובץ נמצא בדיסק
  - לזיכרון משותף (shared memory) – הנמצא בזיכרון הראשי (ב RAM).

זיכרון משותף

- מיפוי זיכרון יכול לשמש לשיתוף זיכרון. במצב של שיתוף זיכרון, אותו אזור ב RAM- ממופה לתוך הזיכרון הווירטואלי של שני התהליכים.
- למשל - שני קטעי זיכרון ב- RAM ממופים אל שני תהליכים.
  - כעת, הזיכרונות של שני התהליכים קשורים זה לזה באמצעות קטעי זיכרון ב RAM. אם תהליך אחד ישנה את הזיכרון המשותף, גם התהליך השני "יראה" את השינוי



כיצד נגדיר זיכרון משותף?

- ראשית – יש צורך בקובץ משותף, אותו ממפים – ולכן אחד מהתהליכים צריך ליצור אותו.
- שנית – יש לגשת לזיכרון המשותף.

תהליך 1

1. יצירת אובייקט מיפוי
   - כחלק מתהליך זה – נתינת שם לאובייקט המיפוי = כינוי לאזור בזיכרון הראשי אליו ימופה הקובץ.
2. שימוש במיפוי עבור חלק מהקובץ לפי הצורך
3. כמובן – כאשר ישנה כתיבה לקובץ, מדובר בקטע קריטי, ולכן יש לבצע סינכרון מתאים

תהליך 2

1. פתיחת גישה לאובייקט המיפוי
   - לצורך פתיחת הגישה יש לדעת את שם אובייקט המיפוי.
2. שימוש במיפוי עבור חלק מהקובץ לפי הצורך.
3. כמובן – כאשר ישנה כתיבה לקובץ, מדובר בקטע קריטי, ולכן יש לבצע סינכרון מתאים

שיתוף זיכרון – קוד מערכת ההפעלה

- כפי שלמדנו, חלק ממרחב הזיכרון של כל תהליך מוקדש לקוד של ה Kernel (כלומר - של מערכת ההפעלה), כדי שכל תהליך יכול לפנות לפונקציות של מערכת ההפעלה במקרה הצורך.

- נבצע ייעול: כאשר מבצעים החלפת הקשר בין תהליכים, אין צורך בעצם לבצע החלפה של קטע הזיכרון של מערכת ההפעלה, שהרי גם התהליך החדש זקוק לאותו קוד עבור מערכת ההפעלה במרחב הזיכרון שלו.
  - הרבה יותר פשוט לבצע מיפוי של קוד הזיכרון שכבר מצוי ב RAM לזיכרון הווירטואלי של התהליך!
- ייעול נוסף: הרי בעצם ראינו שניתן לטעון בכל פעם רק חלק מהזיכרון של התהליך, ע"י מיפוי של קוד התהליך = שימוש בקטע זיכרון קטן בלבד ובו נחליף את הקוד לפי הצורך (כפי שראינו קודם). הדבר מאפשר לנו לטעון בו זמנית כמה קטעי קוד – של תהליכים שונים.
  - נוכל לשתף את מיפוי קוד הזיכרון של מערכת ההפעלה עבור כל התהליכים!

- כמובן: כל אזור הזיכרון המשותף המכיל את קוד מערכת ההפעלה – לא יקבל הרשאות כתיבה, ואף תהליך לא יכול לשנות אותו. כך תישמר האבטחה.

שיתוף זיכרון – קבצי DLL

- על בסיס הרעיון שהראנו כעת, נוכל לבצע שיתוף זיכרון גם עבור קטעי קוד נוספים.

- קובץ DLL הינו חבילת קוד מקומפל שאפשר לייבא לתוך הקוד שלנו . אלו בעצם קבצי "ארכיון" המכילים בתוכם מימוש של פקודות שונות לפי שפת התכנות, ועוד.
  - ראינו קבצים כאלו עבור שפת C, למשל: עבור קלט-פלט סטנדרטי.
- במקרה שכמה תהליכים זקוקים לאותו קובץ DLL – נוכל ליישם את רעיון הזיכרון המשותף.
- במקום שכל תהליך יעתיק את חבילת הקוד לתוך הזיכרון הראשי, נמפה את הקוד לתוך הזיכרון הראשי פעם אחת – והזיכרון הראשי הזה ימופה לכל הזיכרונות הווירטואליים של התהליכים שביצעו ייבוא לקוד זה.

תקשורת בין תהליכים

כמו שאמרנו בתחילה – שימוש בקובץ משותף ובמיפוי זיכרון
זו רק דרך אחת להעברת מידע בין תהליכים.
ננצל הזדמנות זאת להזכיר אמצעי תקשורת נוספים
– השימושיים מאד בסביבת Linux

אמצעי תקשורת בין תהליכים

- ישנם אמצעי תקשורת רבים בין תהליכים, השונים זה מזה הן בצורת העבודה, הן ביכולות ובגודל המידע המועבר, והן בשאלה בין איזה סוגי תהליכים ניתן להשתמש בהם.
- נושא זה מוכר כ: IPC- Inter Process Communication
- נכיר מבין כלי ה IPC האפשריים רק את השימוש ב:
  - אות, סיגנל - SIGNAL
  - צינור - PIPE
  - תור - QUEUE

בקצרה...

SIGNAL

שליחה של אות מוסכם דרך מערכת ההפעלה.
ישנו מספר גדול מאד של סיגנלים אפשריים.
זהו כלי פרימיטיבי יחסית.

PIPE

צינור הוא אפיק (stream) מוסכם – בו תהליך אחד כותב לצינור, והתהליך השני קורא ממנו
זהו כלי מקובל מאד בסביבת יוניקס.

QUEUE

בדומה לצינור, תהליך אחד כותב לתור, ותהליך שני קורא ממנו.

שימוש בסיגנל

- כאמור, סיגנל מערב את מערכת ההפעלה כמעבירת המסרים בין התהליכים.
- השימוש בסינגל כולל שני שלבים:
  - קביעת הפונקציה שתקושר לסיגנל – כלומר, הפונקציה שתופעל כאשר הסיגנל ישלח
- **signal**(signalName, funcPtr)
  - הפעלת הסיגנל, האיתות – כלומר, שליחת הודעה לתהליך מסוים להפעלת הפונקציה
- **kill**(pid,signalName)
  - במערכת Linux קיימים שני סיגנלים פנויים לשימוש המשתמש: SIGUSR1, SIGUSR2

דוגמא לשימוש בסיגנל

מדוע חשוב שתהליך האב יבצע המתנה (כלומר – שלא יתבצע לפני הבן)?

- תהליך האב מבצע המתנה, והוא יתעורר ויבצע את הפונקציה כאשר הבן ישלח לו סיגנל

```
#include <signal.h>
void father_funtion();
void main()
{
 pid_t pid;
int s;
signal(SIGUSR1, &father_function);
if (fork() == 0) { // child process
     pid = getppid();
     kill(pid, SIGUSR1);
} else // parent process
     wait(&s);
}
```

קביעת הפונקציה שתקושר לסיגנל

הפעלת איתות ע"י הסיגנל

```
void father_function()
 {
   printf("I heard you my poor son\n");
  exit(1);
}
```

דוגמת הרצה:

```
> gcc -o signal signal.c
> ./signal
I heard you my poor son
```

שימוש בצינור משורת הפקודה

- כאשר משתמשים בצינור, מריצים במקביל שני תהליכים ומסמנים את היחסים ביניהם:

./pipe1 | ./pipe2

- כעת, כאשר pipe1 יוצר קלט – הוא ייכתב לצינור ולא למסך.
- וכאשר pipe2 מבקש קלט – הוא יקבל אותו מהצינור ולא מהמשתמש
- המידע נשלח ומגיע כזרם של בתים, ויש לקחת בחשבון טיפול בזיהוי המידע המגיע ובגודלו.

דוגמא לשימוש בצינור

pipe1.c

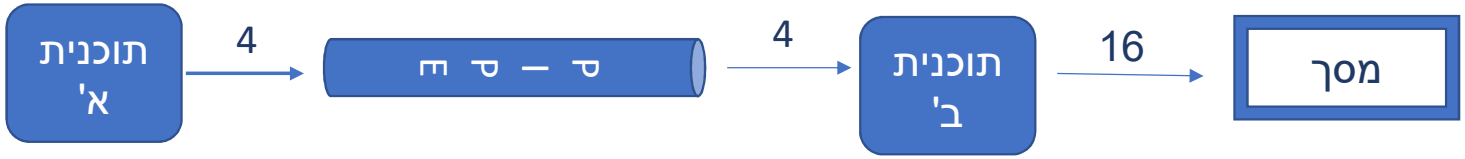
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
 puts("4");
 return EXIT_SUCCESS ;
}
```

pipe2.c

```
#include <stdio.h>
#include <stdlib.h>
int main() {
 int n ;
 scanf(" %d", &n) ;
 printf("%d\n", n*n) ;
 return EXIT_SUCCESS ;
}
```

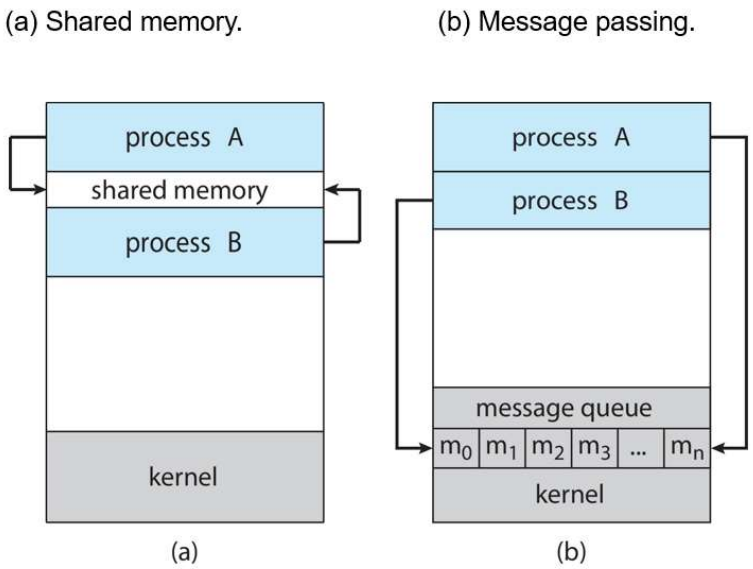
דוגמת הרצה:

```
> gcc -o pipe1 pipe1.c
> gcc -o pipe2 pipe2.c
> ./pipe1 | ./pipe2
16
```



שימוש בתור

- התור שייך לקבוצת כלי הסנכרון – יחד עם סמפורים וזיכרון משותף.
- יש דמיון קל בין צינור לתור, אך בהבדלים הבאים (חלקי מאד...):
  - כל הודעה מגיעה בנפרד
  - להודעה יש תכונת סוג וניתן למיין אותן בהתאם.
  - התור יכול לשמש לשני הכיוונים.
  - לתור יש שם והפתיחה והסגירה שלו נעשות בקוד עצמו
  - קריאה וכתיבה לתור אלו קריאות מערכת מיוחדות
- יש דמיון קל גם בין תור לזיכרון משותף.
  - האיור מדגים את ההבחנה בין שני כלי הסנכרון הללו



- יש כמה סוגים של תורים.
  - הסוג המסורבל והישן יותר נמצא יותר בשימוש
  - הסוג החדש נקרא בשם POSIX והוא קריא ונוח יותר.

מעבדת כיתה ותרגיל בית

- בזמן שנותר ננסה להדגים את מה שהסברנו כעת:
  - הפחתה של גודל הזיכרון במקרה של מיפוי קובץ
  - זיהוי זיכרון משותף עבור תהליך מסוים

- בתרגיל הבית תבצעו שימוש בזיכרון משותף על מנת לשנות קובץ.

בהצלחה רבה!