מערכות הפעלה תרגול 10

אַתְּת Semaphores

מתרגל-יורם סגל
yoramse@colman.ac.il

Interprocess Communication Mechanisms (IPC)

Generate process

Process

- Fork(),
- Exec(),
- Wait(),
- Exit()

Messages between process

Signals

- Signal(),
- Kill(),
- Pause(),
- Alarm()

Data between process

Process Communication

- Files
- Pipe()
- Fifo()

Shared memory between process

Shared Memory

- shm**get**() יצור
- shm**at**() חיבור
- shm**dt**() ניתוק
- shm**ctl**() בקרה
- ftok() מפתח יחודי

Sockets
Message Queues
Shared Memory
Semaphores

T9 - Contents

- Semaphores
- Critical section
- Semaphores types
- Semaphores Management (system calls):
  semget () – Create Semaphores
  semop () - Ask, Wait, Do, End
  semctl () - Control no. of shared proccess

Contents

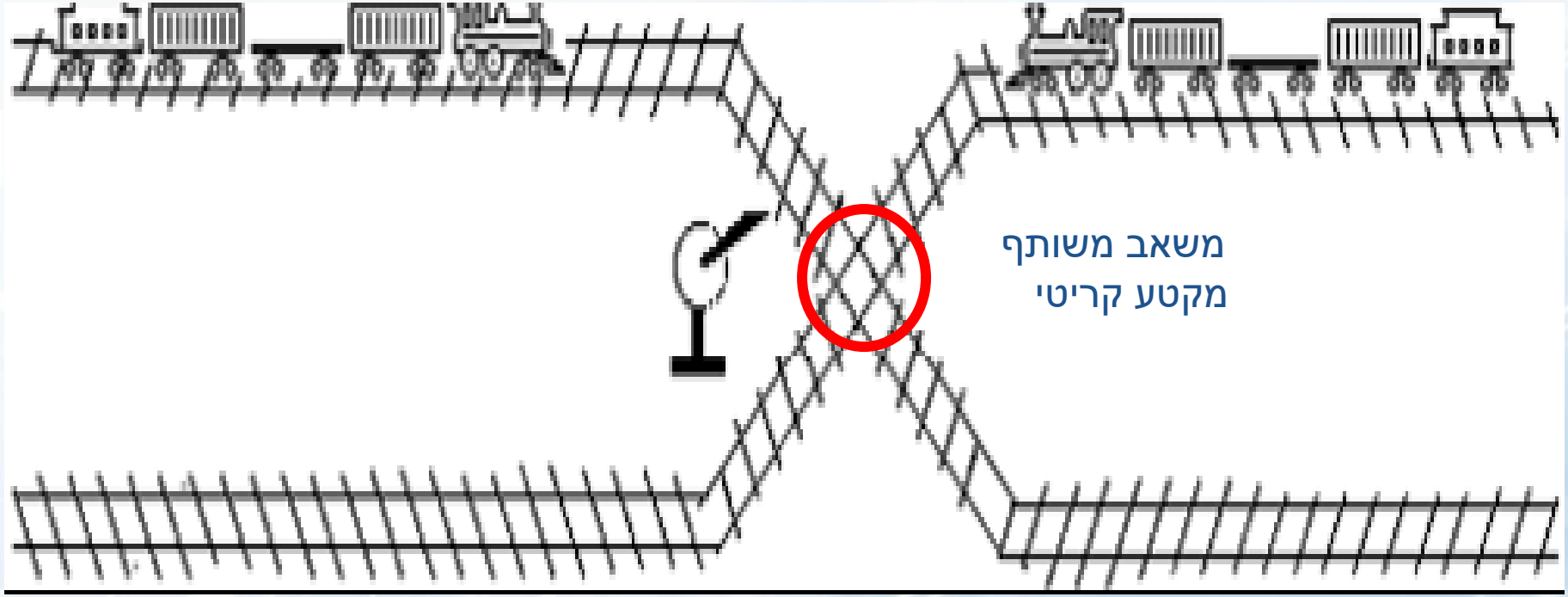
1 semaphores

ויקיפדיה:
סמפור Semaphore (או לפי האקדמיה ללשון העברית אַתֶּת) הוא מנגנון לסנכרון מספר תהליכים הפועלים במחשב במקביל, ומטרתו לפתור את בעיית המניעה ההדדית - Mutual exclusion.

סמפור: טיפול במצב בו יש מספר משאבים משותפים מוגבל,
ויתכן מצב בו יש יותר צרכנים ממשאבים
לדוגמה: יש 3 עמדות קבלת קהל(3 משאבים משותפים) ובתור עומדים 10 אנשים (10 צרכנים)

במקרה שלנו,
משאבים משותפים הם בדר"כ אזורי זכרון משותפים למספר תהליכים, והצרכנים הם התהליכים.

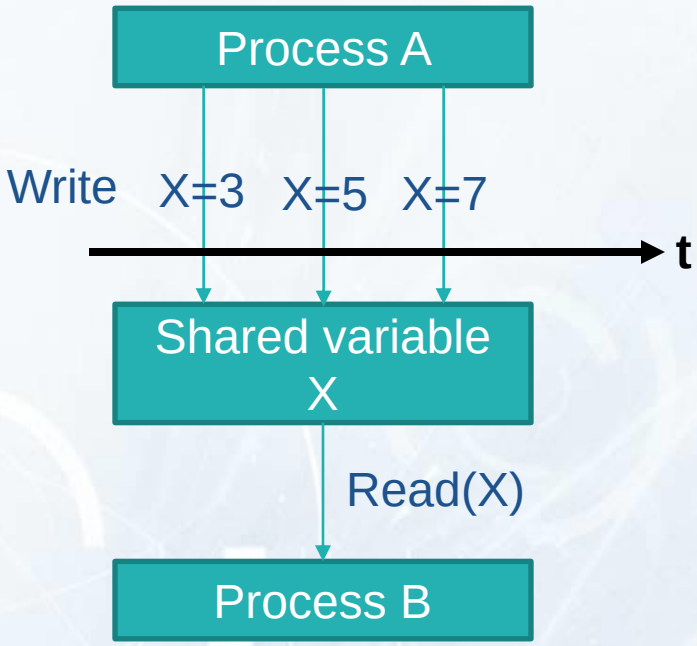
Critical section



נדגיש: שני תהליכים לא יכולים לכתוב בו זמנית לאותו המשאב.
מבחינת התכנת, הנושא הזה שקוף. שכן מערכת ההפעלה מטפלת בבעיה זו.
הנקודה היא שהתכנת אינו יודע, כיצד המערכת תטפל בבעיה ולפיכך לא ברורה התוצאה.

תאור הבעיה

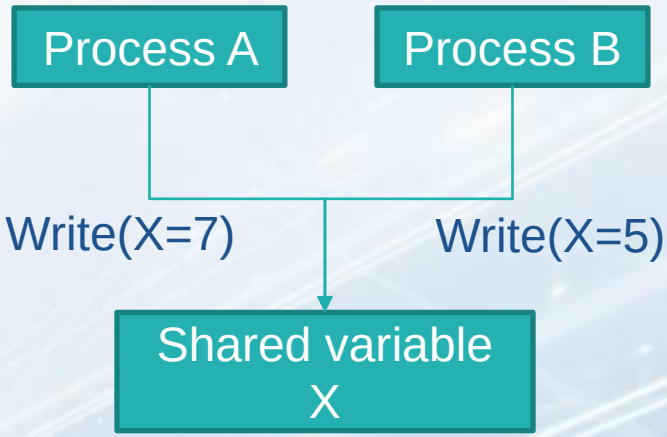
דוגמה ב



X=?

הערך שB יקרא
תלוי מתי הוא
יגש למשתנה X

דוגמא א



X=?

הערך בX
תלוי מי התהליך
האחרון ששינה אותו

תזכורת: בעבוד מקבילי של תהליכים, אין לדעת את סדר פעולת התהליכים
(מערכת ההפעלה מחליטה וזה לא בשליטת התכנת)

קיימים שני סוגים של סמפורים

- **בינארי:** שולט בכניסה למשאב אחד משותף ע"י שינוי ערך הסמפור מאפס (משאב תפוס) לאחד (משאב פנוי) וההפך.
  - דוגמה: נניח שקיים משאב אחד בלבד, נניח משאבת דלק בתחנת דלק. לכן כל פעם רק רכב אחד יכול לתדלק
- **סמפורי מונים:** שולט בכניסה למספר משאבים זהים משותפים
  - דוגמה: נניח שיש מספר משאבים משותפים, אך יכול להיות מצב בו יש יותר צרכנים ממשאבים. נניח שיש 6 משאבות דלק אך יתכן ויגיעו לתחנה יותר מ6 מכוניות בו-זמנית

דוגמה: שני תהליכים שקוראים וכותבים רשומות לקובץ נתונים משותף

- ניתן להשתמש בסמפור עם הערך הראשוני של 1
- סביב קוד ההפעלה של הקובץ, נציב שתי פעולות סמפור:
  - הראשונה: בחינת הערך של הסמפור והקטנתו באחד (ספירה לאחור)
  - השניה: בחינת הערך של הסמפור והגדלתו באחד (ספירה קדימה)
- התהליך הראשון שניגש לקובץ ינסה להקטין את ערך הסמפור והוא יצליח.
- הערך של הסמפור הוא כעת 0
- כעת התהליך יכול להשתמש בקובץ הנתונים
- אם תהליך השני מבקש להשתמש בקובץ, הוא מנסה כעת גם להוריד את ערך של הסמפור, הוא ייכשל מכיוון שהתוצאה תהיה 1-.
- תהליך זה תהליך מספר 2) יושעה עד שהתהליך הראשון יסיים להשתמש בקובץ הנתונים.
- לאחר סיום השימוש בקובץ ע"י התהליך הראשון, התהליך הראשון, יגדיל את הערך של הסמפור ויעשה אותו שוב 1.
- כעת ניתן להעיר את תהליך השני, שנמצא במצב ההמתנה, והפעם הניסיון שלו להגדיל את הסמפור יצליח. והוא יצליח לעבוד עם הקובץ.

איך להגן על ה Critical Section(CS)?

- נבצע הגנה על CS בעזרת semaphore:
  - נייצר מבנה (structure) סמפור, המשותף למספר תהליכים (Process).
  - כאשר תהליך נכנס לסמפור הוא צריך להוריד באחד את ערכו של מונה מסויים שיש בתוך מבנה הסמפור
  - **כדי שהסמפור יעבוד, פעולות הורדת המונה ומידע על ערך המונה צריכות להיות אטומיות**

איך סמפור עובד

- על התכנת לקבוע כי אזור מסויים בקוד הוא קריטי כלומר: Critical Section (CS)
- יוצרים סמפור להגן עליו
- התהליך הראשון שנכנס ל CS נועל את הסמפור
- כל שאר התהליכים שרוצים להכנס ל CS ממתינים ברשימת הממתינים של הסמפור (לינוקס מכניס אותם למצב המתנה).
- כאשר התהליך הראשון יוצא מה CS הוא מאותת לשאר התהליכים שהוא סיים
- הסמפור (בעזרת מערכת ההפעלה) מעיר את אחד התהליכים שממתין ונותן לו גישה ל CS
- כל מנגנון פעולות ההמתנה והאיתות מתבצעות אוטומטית ע"י מערכת ההפעלה
- **לכן מצב המתנה <u>עקב סמאפור</u> לא גוזל משאבי CPU**

Critical Regions/ Critical Section

A critical code that only one process at a time should be executing

תאור איכותי

```
S=1; //Init. Semaphores
Write_to_file()
{
        S=S-1; //Ask for Semaphores-                    Start Critical Regions
        While(1)
                If S==0
                        //The Semaphores section is ready to be use
                        write()                         Critical Regions
                        S=S+1;//free the Semaphores-End of Critical Regions
                        return;
                Else
                wait()
}
```

Semaphores - System V IPC vs POSIX

'System V IPC' (SysV)

Unix System V, commonly abbreviated SysV written—as "System Five" .
With SysV, AT&T introduced three new forms of IPC facilities (message queues, semaphores, and shared memory)

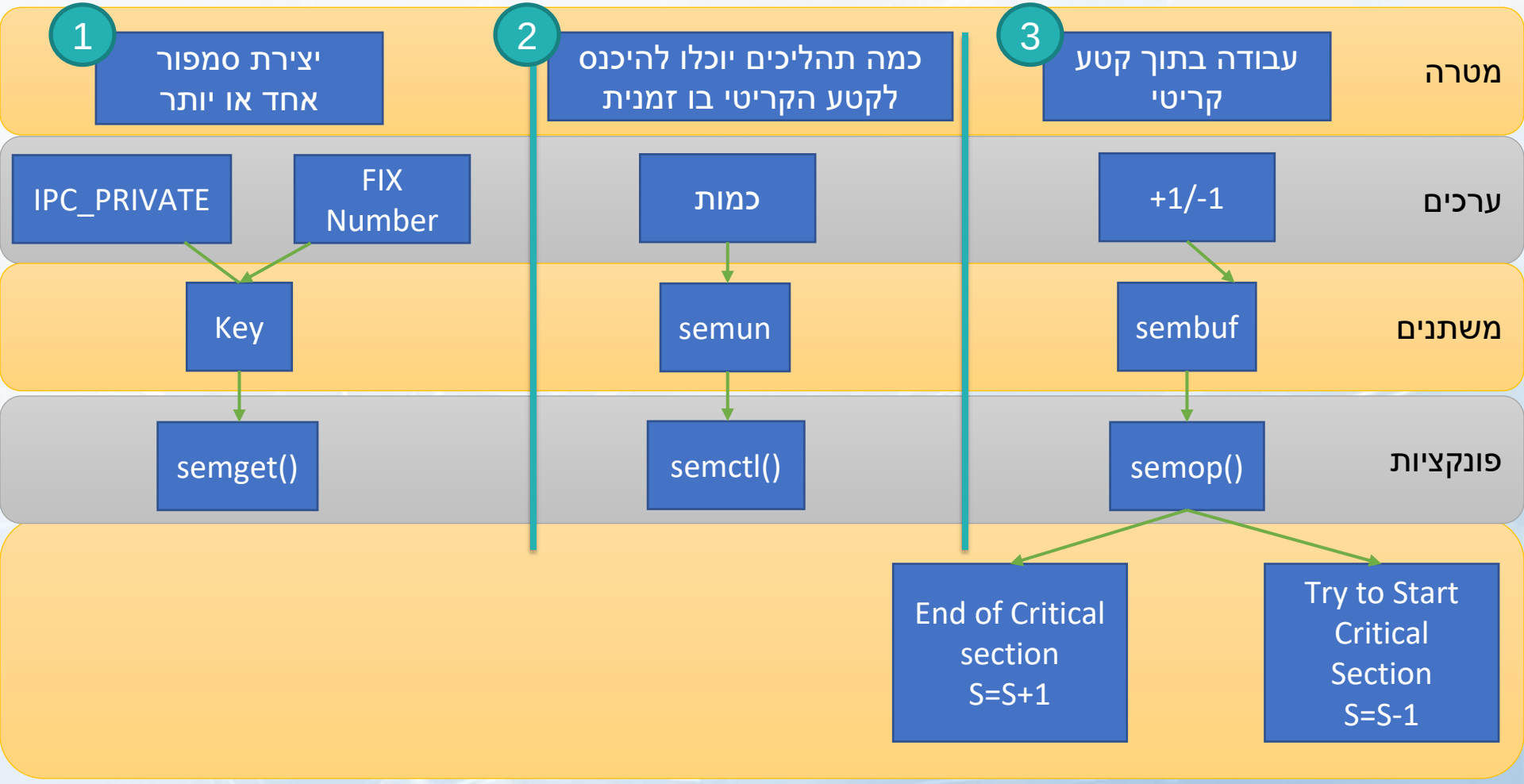
semctl()
semop()
semget()

POSIX

The Portable Operating System Interface (POSIX)[1] is a family of standards specified by the IEEE for maintaining compatibility between OS. POSIX defines the application programming interface (API), along with command line shells and utility interfaces, for software compatibility with variants of Unix and other OS

sem_close(), sem_destroy(), sem_getvalue(), sem_init(),	**sem_open(), sem_post(), sem_trywait(), sem_unlink().**

System V IPC semaphores functions



Semaphores Example

הגדרות (הכנות מקדימות)

This section describes the System V IPC semaphores.

```
#include <sys/sem.h>
#define KEY 0x1111 ;                    //define a key in both the parent as well as the child to refer to the same IPC structure
union semun {                           // Union for the purpose of semaphore operations.
                int val; //The number of processes that can simultaneously access this critical section
                struct semid_ds *buf;
                unsigned short *array;                  };
```

Define your try (semwait) and raise (semsignal) structures.

```
struct sembuf p = {0, -1, SEM_UNDO}; # semwait
struct sembuf v = {0, +1, SEM_UNDO}; # semsignal
```

יצירת סמפור
אחד או יותר

1

Getting the id for our IPC semaphore from operating system.

```
int id;
// 2nd argument is number of semaphores
// 3rd argument is the mode (IPC_CREAT creates the semaphore set if needed)
if ((id = semget(KEY, 1, 0666 | IPC_CREAT) < 0) {
   /* error handling code */
}
```

כמה תהליכים יוכלו להיכנס
לקטע הקריטי בו זמנית

2

In the parent, initialize the semaphore to have a counter of 1.

```
union semun u;
u.val = 1; //The number of processes that can simultaneously access this critical section
if (semctl(id, 0, SETVAL, u) < 0) { // SETVAL is a macro to specify that you're setting the value of the semaphore to that specified by the union u
/* error handling code */ }
```

3

At the start of your critical section, decrement the counter using the semop() function:

```
if (semop(id, &p, 1) < 0)
     { /* error handling code */ }
```

To increment the semaphore, you use &v instead of &p:

```
if (semop(id, &v, 1) < 0)
     { /* error handling code */ }
```

חסימה
עבודה בתוך קטע
קריטי
שחרור

semget

יצירת סמפור

1

```
int id;
// 2nd argument is number of semaphores
// 3rd argument is the mode (IPC_CREAT creates the semaphore set if needed)
if ((id = semget(KEY, 1, 0666 | IPC_CREAT) < 0) {
   /* error handling code */
}
```

- יצירת קבוצת סמאפורים לבקרה

```
#include <sys/sem.h>
Int semget(key_t key, int nsems, int semflg);
```

- פרמטרים:
  - *key* – מפתח זיהוי או הערך IPC_PRIVATE
  - *nsems* – מספר תתי הסמפורים הפנימיים (אפשר לתת את הערך 0 אם משתמשים בסמפור קיים)
  - *semflg* – (אחד או יותר מופרדים ע"י |)
    - IPC_CREAT – ימצא שטח חדש (אם הסמפור עבור המפתח כבר קיים – המזהה של הסמפור חוזר)
    - IPC_EXCEL – אקסקלוסיבי – מבטיח כישלון אם הוגדר לפני כן CREAT
    - mode_flags הרשאות הגישה.
    - עם היצירה, 9 הסיביות הפחות משמעותיות של ה- semflg מגדירות את ההרשאות (לבעלים, לקבוצה ואחרים) עבור סט סמפור זה.
- ערך מוחזר
  - ערך מספרי המזהה באופן ייחודי את הסמאפור (semid)
  - 1- בכישלון

```
id = semget(KEY, 1, 0666 | IPC_CREAT)
```

Semaphores Example

הגדרות

This section describes the System V IPC semaphores.

```
#include <sys/sem.h>
#define KEY 0x1111 ;                //define a key in both the parent as well as the child to refer to the same IPC structure
union semun {                       // Union for the purpose of semaphore operations.
        int val; //The number of processes that can simultaneously access this critical section
        struct semid_ds *buf;
        unsigned short *array;          };
```

Define your try (semwait) and raise (semsignal) structures.

```
struct sembuf p = { 0, -1, SEM_UNDO}; # semwait
struct sembuf v = { 0, +1, SEM_UNDO}; # semsignal
```

יצירת סמפור

1

Getting the id for our IPC semaphore from operating system.

```
int id;
// 2nd argument is number of semaphores
// 3rd argument is the mode (IPC_CREAT creates the semaphore set if needed)
if ((id = semget(KEY, 1, 0666 | IPC_CREAT) < 0) {
   /* error handling code */
}
```

כמה תהליכים יוכלו להיכנס לקטע הקריטי בו זמנית

2

In the parent, initialize the semaphore to have a counter of 1.

```
union semun u;
u.val = 1;  //The number of processes that can simultaneously access this critical section
if (semctl(id, 0, SETVAL, u) < 0) { // SETVAL is a macro to specify that you're setting the value of the semaphore to that specified by the union u
/* error handling code */ }
```

3

At the start of your critical section, decrement the counter using the semop() function:

```
if (semop(id, &p, 1) < 0)
     { /* error handling code */ }
```

עבודה בתוך קטע קריטי

To increment the semaphore, you use &v instead of &p:

```
if (semop(id, &v, 1) < 0)
     { /* error handling code */ }
```

semctl

כמה תהליכים יוכלו להיכנס לקטע הקריטי בו זמנית

2

```
union semun u;
u.val = 1;  //The number of processes that can simultaneously access this critical section
if (semctl(id, 0, SETVAL, u) < 0) { // SETVAL is a macro to specify that you're setting the value of the semaphore to that specified by the union u
/* error handling code */ }
```

- שינוי פרמטר של האובייקט המשותף – גם לסגירה

```
#include <sys/sem.h>
int semctl(int semid, int semnum, int cmd, semun arg);
```

- פעולה:
  - מאפשר שליפה ועדכון של פרמטרים של הסמפור המשותף, מאפשר לסגירה
- פרמטרים:
  - *semid* – מזהה האובייקט מהשקף get id = semget(KEY, 1, 0666 | IPC_CREAT)
  - *semnum* – מזהה הסמפור הספציפי בתוך ה- set (מתחיל ב-0, כמו מערך)
  - *cmd* – (אופרטור יחיד!)
    - **IPC_STAT** – מאפשר שליפה של הסטאטוס של הסמפור לתוך הפרמטר האחרון: *arg* **semun** *(u.buf)*
      (מתעלמים מהפרמטר שלפניו: **int** semnum)
    - **IPC_SET** – מאפשר קביעה של חלק מפרמטרי הסמפור מתוך arg.buf
      ישנה את הפרמטרים: sem_perm.mode sem_perm.uid sem_perm.gid
      (מתעלמים מהפרמטר שלפניו: semnum)
    - **IPC_RMID** – מורה על השמדה (מתעלמים מהפרמטר semnum)
    - **ראו שקף נוסף על פקודות נוספות (לאחר הגדרת ה-arg)**
  - *arg* – מבנה הנתונים ייחודי שמותאם לשימוש עם cmd (ראו שקף הבא)

GETNCNT
GETPID
GETZCNT
GETVAL
SETVAL
GETALL
SETALL

```
struct ipc_perm {
    key_t  key;
    ushort uid;        /* owner uid and gid */
    ushort gid;
     .
     .
     .
    ushort mode;
    ...
};
```

```
struct semid_ds{
        struct ipc_perm sem_perm; (ראה שקף הבא )
        u_short sem_nsems;
        time_t sem_otime;        //last time of semop
        time_t sem_ctime;        //last time of semctl
};
```

A **union** is a special data type available in C that allows to store different data types in the same memory location.

Special semctl() commands

- **GETNCNT** – יחזיר מהפונק' את מספר התהליכים שממתינים שמשאב ישתחרר (עבור הסמפור ה semnum-th)
- **GETPID** – יחזיר מהפונק' את ה-PID של מי שביצע semop() לאחרונה (עבור הסמפור ה semnum-th)
- **GETZCNT** – יחזיר מהפונק' את מספר התהליכים שממתינים שמשאב יתפס (עבור הסמפור ה semnum-th)
- **GETVAL** – יחזיר מהפונק' את הערך של הסמפור מתוך ה-set (עבור הסמפור ה semnum-th)
- **SETVAL** – יקבע את הערך עבור הסמפור ה semnum-th מתוך ה-set, לפי השדה val מתוך ה-union
- **GETALL** – יחזיר לתוך השדה array את ערכי הסמפורים ב-set (מתעלמים מהפרמטר semnum)
- **SETALL** – יקבע עבור כל הסמפורים ב-set את ערכם ע"פ המידע שבשדה array (מתעלמים מהפרמטר semnum)

- **הבהרות**:
  - הערך של ה-semaphore הספציפי בתוך ה-set = 0 כאשר המשאב שהסמאפור מתאר הינו בניצול מלא, אחרת ה-val > 0 וניתן לנצל שוב את אותו semaphore
  - השימוש ב-buf הינו כמו בכל pointer
    עליכם ליצור אובייקט sem מקומי "למילוי"

```
semctl(semid , 0, SETVAL, semarg)
```

Semaphores Example

הגדרות

This section describes the System V IPC semaphores.

```
#include <sys/sem.h>
#define KEY 0x1111 ;                //define a key in both the parent as well as the child to refer to the same IPC structure
union semun {                       // Union for the purpose of semaphore operations.
        int val; //The number of processes that can simultaneously access this critical section
        struct semid_ds *buf;
        unsigned short *array;          };
```

Define your try (semwait) and raise (semsignal) structures.

```
struct sembuf p = { 0, -1, SEM_UNDO}; # semwait
struct sembuf v = { 0, +1, SEM_UNDO}; # semsignal
```

יצירת סמפור

1

Getting the id for our IPC semaphore from operating system.

```
int id;
// 2nd argument is number of semaphores
// 3rd argument is the mode (IPC_CREAT creates the semaphore set if needed)
if ((id = semget(KEY, 1, 0666 | IPC_CREAT) < 0) {
   /* error handling code */
}
```

כמה תהליכים יוכלו להיכנס לקטע הקריטי בו זמנית

2

In the parent, initialize the semaphore to have a counter of 1.

```
union semun u;
u.val = 1;  //The number of processes that can simultaneously access this critical section
if (semctl(id, 0, SETVAL, u) < 0) { // SETVAL is a macro to specify that you're setting the value of the semaphore to that specified by the union u
/* error handling code */ }
```

3

At the start of your critical section, decrement the counter using the semop() function:

```
if (semop(id, &p, 1) < 0)
     { /* error handling code */ }
```

עבודה בתוך קטע קריטי

To increment the semaphore, you use &v instead of &p:

```
if (semop(id, &v, 1) < 0)
     { /* error handling code */ }
```

semop

```
if (semop(id, &p, 1) < 0)
    { /* error handling code */ }
if (semop(id, &v, 1) < 0)
    { /* error handling code */ }
```

```
{ 0, -1, SEM_UNDO};
```

- פעולה על sem מסויים (שימוש) להמתנה ותפיסה של קטע קוד קריטי

```
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, unsigned nsops);
```

- פרמטרים:
  - *semid* – מזהה האובייקט מהשקף get
  - *sops* – מערך של אובייקטים, כאשר כל אובייקט מציין פעולה על הסמפור, ומכיל את השדות הבאים (סה"כ המערך מציין סדרה של פעולות על הסמפור):
    - sem_num – מספר ה-semaphore הספציפי ב-set עליו לבצע את הפעולה
    - sem_op – סוג הפעולה
      - חיובי = הוספת מספר זה לסמפור, מקביל לשחרור משאב
      - שלילי = המתן עד שהערך של הסמפור גדול שווה לערך המוחלט של sem_op ואז תחסר את הערך שיש ב sem_op, קביל לנעילת הסמפור. במילים אחרות אנו מחכים שיתפנו מספיק משאבים בסמפור.
      - 0 = המתן ל-val שיהיה אפס (כלומר ניצול מלא – ורק אז תמשיך הלאה)
    - semp_flg – (מופרד ע"י |)
      - IPC_NOWAIT – יגרום לפקודה להשתחרר מיד, אם המשאב לא פנוי (יוחזר ערך 1- על כישלון)
      - SEM_UNDO – המתן עד שקטע קריטי יהיה פנוי. אם תוך כדי המתנה אתה מבצע exit אז הפעולה שביצעת על הסמפור נמחקת (הסמפור נשאר).
  - *nsop* – מספר האובייקטים(פעולות) במערך sops

דוגמאות לסמפורים

- T9_1.c דוגמה לסמפור בין אבא לבן
- T9_2.c דוגמה לסמפור בין 2 תהליכים שונים (יש להריץ את אותו הקוד פעמיים)
- T9_3.c דוגמא לשינוי ערכי סמפור בעזרת ה "פקודות הנוספות"
- T9_4.c דוגמא לשימוש ב SEM_UNDO