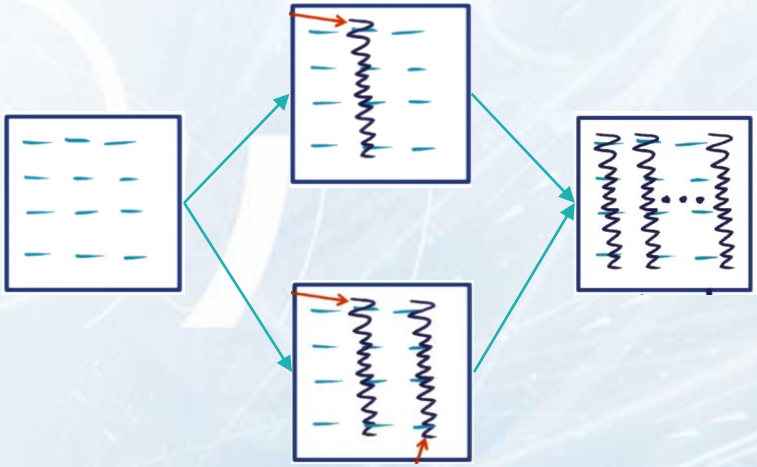
מערכות הפעלה תרגול 11

חוט Threads

מתרגל-יורם סגל
yoramse@colman.ac.il

Contents

Program, Process and Thread

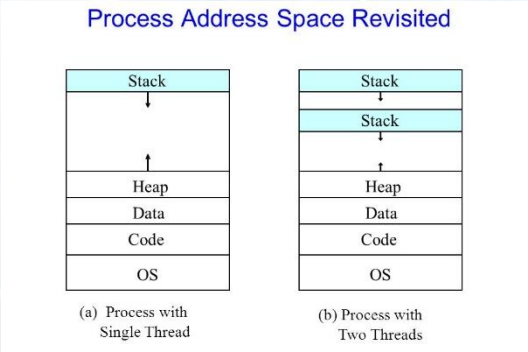


Process=Program + State of all threads executing in the program

הקדמה לחוטים

- חוט הוא יחידת ביצוע עצמאית בתוך תהליך.

- תהליך ב-Linux יכול לכלול מספר חוטים המשתפים ביניהם את כל משאבי התהליך :
  - מרחב הזיכרון.
  - גישה לקבצים והתקני חומרה.
  - מנגנונים שונים של מערכת ההפעלה.



- כל חוט בתהליך מהווה הקשר ביצוע נפרד – לכל חוט מחסנית ורגיסטרים משלו.

הקדמה לחוטים - יתרונות

- החוטים נועדו לאפשר ביצוע בלתי תלוי של חלקים מהמשימה של אותו תהליך.
- מספר חוטים של אותו תהליך יכולים לרוץ במקביל על **מעבדים שונים**.
- ניתן לשפר את ביצוע תהליך באמצעות שימוש בריבוי חוטים גם על מעבד יחיד

הקדמה לחוטים

- תהליך נוצר לראשונה עם חוט יחיד, החוט הראשי (primary thread).
- התקשורת בין חוטים של אותו תהליך היא פשוטה ביותר:
- הוספת חוט לביצוע תכנית זולה בהרבה מהוספת תהליך לאותה מטרה,

process descriptor

- לכל תהליך ב-Linux קיים בגרעין מתאר תהליך (process descriptor), שהוא רשומה המכילה:
  - מצב התהליך
  - עדיפות התהליך
  - מזהה התהליך (pid)
  - מצביע לטבלת איזורי הזיכרון של התהליך
  - מצביע לטבלת הקבצים הפתוחים של התהליך
  - ועוד..

הקדמה לחוטים

- התמיכה בחוטים ב-Linux שואפת להתאים לתקן הכללי של מימוש חוטים במערכות Unix הקרוי POSIX Threads (שבו כל החוטים של אותו תהליך מאוגדים ביחד).

- בלינוקס:
  - ההתייחסות לחוט הינה כאל תהליך רגיל ולכן לכל חוט אמור להיות **מתאר** משלו ו-**PID** משלו.
  - מצד שני, המתכנת, בהתאם לתקן POSIX, מצפה שלכל החוטים השייכים לאותו תהליך ניתן יהיה להתייחס דרך PID יחיד – של התהליך המכיל אותם.
    - ולכן הוא מצפה ש:
      - פעולות על ה-PID של התהליך ישפיעו על כל החוטים בתהליך.
      - פעולת ()getpid בכל חוט תחזיר את אותו ה PID (של התהליך המכיל את החוט).

קבוצת חוטים (thread group)

- הפתרון:

thread group

- כל החוטים השייכים לאותו תהליך נמצאים בקבוצה אחת.
- קבוצת החוטים מוגדרת בגרעין של Linux (מגרסת 2.4.X ומעלה) כדי לאפשר התייחסות משותפת לכל החוטים באותו תהליך.

קבוצת חוטים (thread group)

- מתארי התהליכים של כל החוטים באותה קבוצה מקושרים באמצעות שדה **thread_group** במתאר התהליך (שהוא הprimary thread).

- שדה **tgid** במתאר התהליך מכיל את ה-PID המשותף לכל החוטים באותה קבוצה. למעשה, זהו ערך ה-PID של החוט הראשון של התהליך (הprimary thread).

קבוצת חוטים (thread group)

- ביצוע פעולת getpid() מחזיר את current->tgid.
- פעולות על ה-PID המשותף מתורגמות לפעולה על קבוצת החוטים המתאימה ל-PID.
- אם לחוט כלשהו יש בנים (תהליכים), הם הופכים להיות בנים של חוט אחר בקבוצת האב לאחר מותו.

pthread_create

- יצירת חוט: pthread_create()
- תחביר:
- **#include <pthread.h>**
- int pthread_create (pthread_t *thread, pthread_attr_t *attr, void *(* start_func) (void*),void *arg)
- פעולה: יוצרת חוט חדש המתבצע במקביל לחוט הקורא בתוך אותו תהליך. החוט החדש מתחיל לבצע את הפונקציה המופיעה בפרמטר start_func ונהרג בסיום ביצוע הפונקציה

int pthread_create (pthread_t *thread, pthread_attr_t *attr, void *(* start_func)(void*), void *arg)

pthread_create

- פרמטרים:
  - thread – מצביע למקום בו יאוחסן מזהה (לכל חוט יש מזהה אישי) החוט החדש במקרה של סיום הפונקציה בהצלחה
  - attr – מאפיינים המתארים את תכונות החוט החדש, כגון האם החוט הוא חוט מערכת (PTHREAD_SCOPE_SYSTEM) או חוט משתמש (PTHREAD_SCOPE_PROCESS), האם ניתן לבצע לו join, כלומר להמתין לסיומו, וכו'. בד"כ נספק ערך NULL המציין חוט משתמש שניתן להמתין לסיומו.
  - start_func – מצביע לפונקציה שתהווה את קוד החוט. הערך המוחזר מפונקציה זו במקרה של סיומה הטבעי הינו ערך הסיום של החוט.
  - arg – פרמטר שיסופק לפונקציה עם הפעלתה, שימו לב יש לשלוח את הפרמטרים עם casting לסוג **void *** ולבצע casting מתאים בתוך הפונקציה המקבלת.

Table 3-1 Default Attribute Values for *attr*

Attribute	Value	Result
scope	PTHREAD_SCOPE_PROCESS	New thread is unbound - not permanently attached to LWP.
detachstate	PTHREAD_CREATE_JOINABLE	Exit status and thread are preserved after the thread terminates.
stackaddr	NULL	New thread has system-allocated stack address.
stacksize	1 megabyte	New thread has system-defined stack size.
priority		New thread inherits parent thread priority.
inheritsched	PTHREAD_INHERIT_SCHED	New thread inherits parent thread scheduling priority.
schedpolicy	SCHED_OTHER	New thread uses Solaris-defined fixed priority scheduling; threads run until preempted by a higher-priority thread or until they block or yield.

- ערך מוחזר:
  - 0 במקרה של הצלחה, (המזהה של החוט החדש נשמר במצביע thread)
  - קוד שגיאה אחר במקרה של כישלון

pthread_self

- קבלת מזהה החוט – pthread_self()
  - תחביר:

```
pthread_t pthread_self();
```

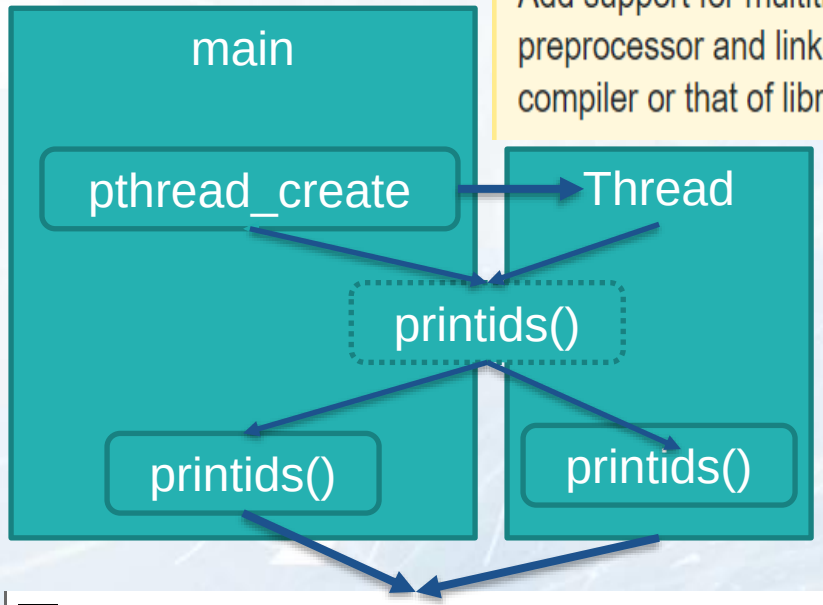
  - פעולה: החוט הקורא מקבל את המזהה של עצמו. מזהה זה הוא פנימי לספרייה Linux Threads ואינו קשור במישרין ל-PID של החוט
  - פרמטרים: אין
  - ערך מוחזר: מזהה החוט

t9_1.c

❖דוגמא ליצירת חוט והדפסת המזהה שלו

-pthread **To compile: gcc -pthread t9_1.c**

Add support for multithreading using the POSIX threads library. This option sets flags for both the preprocessor and linker. It does not affect the thread safety of object code produced by the compiler or that of libraries supplied with it. These are HP-UX specific flags.




```
u2 89-231 41 : gcc -l pthread t10_1.c
u2 89-231 42 : a.out
main thread: pid 17975 tid 2224412416
new thread:  pid 17975 tid 2224404224
u2 89-231 43 :
```

pthread_join

- המתנה לסיום חוט: pthread_join()
  - תחביר:

```
int pthread_join(pthread_t th, void **thread_return);
```

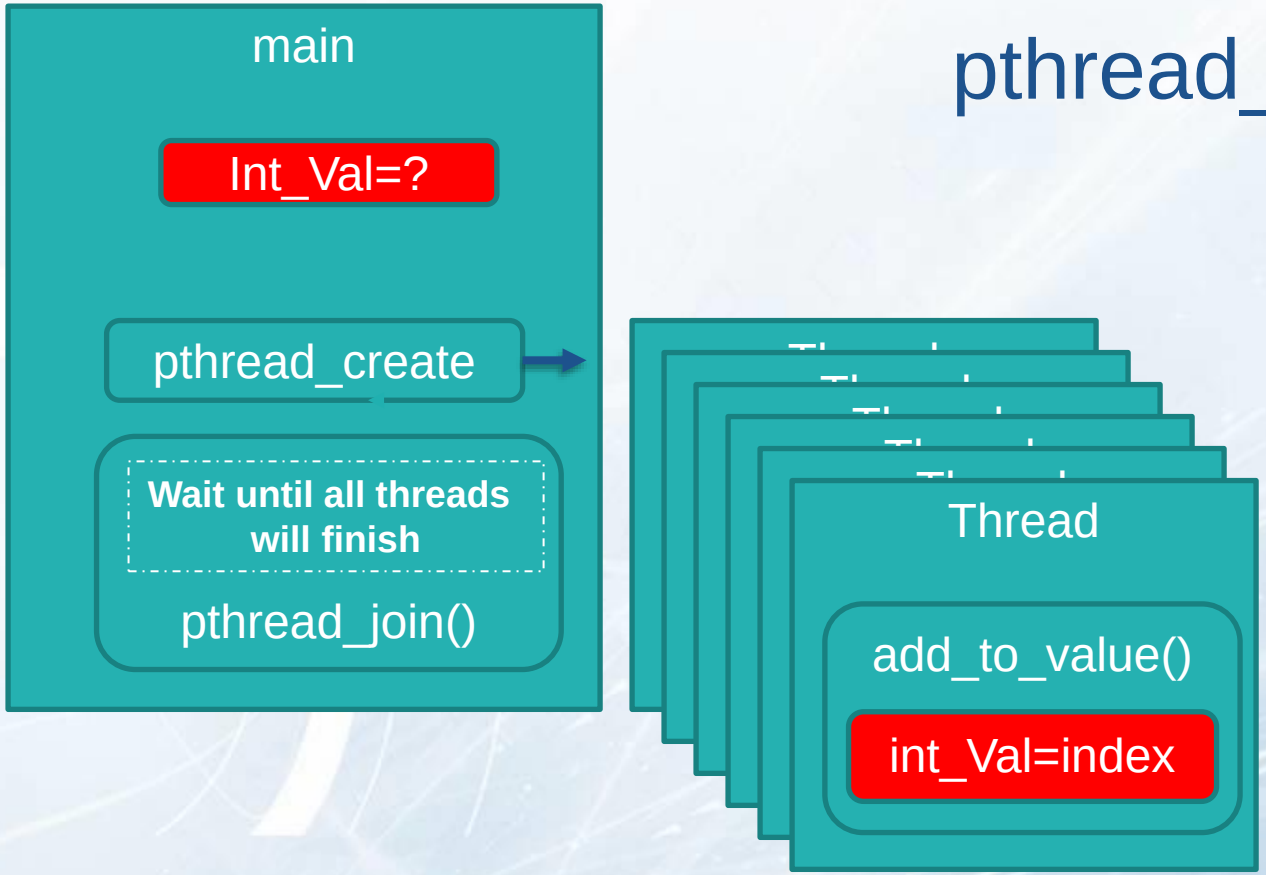
  - פעולה: החוט הקורא ממתין לסיום החוט המזוהה ע"י **th**
    - **ניתן להמתין לסיום אותו חוט פעם אחת לכל היותר.**
      - ביצוע pthread_join על אותו חוט יותר מפעם אחת ייכשל.
    - כל חוט יכול להמתין לסיום כל חוט אחר באותו תהליך.
    - ההמתנה על סיום החוט משחררת את מידע הניהול של החוט ברמת Linux Threads וברמת הגרעין.
    - אם מספר חוטים שונים באותו תהליך מנסים להשתמש ב pthread_join() עבור אותו חוט התוצאה היא לא מוגדרת (תלוית מעבד).

pthread_join

- פרמטרים:
  - th – מזהה החוט שממתינים לסיומו
    - לא ניתן להמתין ל"סיום חוט כלשהו" בדומה ל-wait()
  - thread_return – מצביע לכתובת שבה יאוחסן ערך הסיום של החוט עבורו ממתינים
    - ניתן לציין NULL כדי להתעלם מערך הסיום
- ערך מוחזר:
  - 0 במקרה של הצלחה,
    כמו כן, ערך יציאה ב- thread_return (אם אינו NULL)
  - אחר במקרה של כישלון

t9_2.c

❖דוגמא ל pthread_join



pthread_cancel

- הריגת חוט ממקום חיצוני: pthread_cancel()
  - תחביר:

```
int pthread_cancel(pthread_t thread);
```

  - פעולה: סיום ביצוע החוט המזוהה ע"י thread
  - פרמטרים:
    - thread – מזהה החוט המיועד לסיום
  - ערך מוחזר:
    - 0 במקרה של הצלחה
    - אחר במקרה כישלון

pthread_exit

- סיום חוט מתוך החוט: pthread_exit()
  - תחביר:

```
void pthread_exit(void *retval);
```

  - פעולה: החוט הקורא מסיים את פעולתו (סיום עצמי). ערך הסיום יוחזר לחוט שימתין לסיום חוט זה
    - סיום פעולת החוט הראשי ע"י pthread_exit() **אינו** מסיים את כל החוטים בתהליך
  - פרמטרים:
    - retval – ערך סיום (בדומה לזה של exit())
  - ערך מוחזר: אין

סיום ביצוע תהליכים

- אם חוט כלשהו מתוך תהליך קורא ל-exit() מתבצע סיום ביצוע **התהליך כולו**
  - כל החוטים בקבוצה מופסקים
- לאחר סיום ביצוע קוד החוט הראשי מתבצעת קריאה אוטומטית –לexit() הגורמת לסיום ביצוע התהליך
- אם כל החוטים בתהליך מסיימים באמצעות pthread_exit(), אזי סיום החוט האחרון הוא סיום התהליך

סיום חוט

- חוט יכול להסתיים כתוצאה ממספר אפשרויות שונות:
  - חזרה מהפונקציה הראשית של החוט.
  - קריאה ל-pthread_exit() בתוך קוד החוט.
  - קריאה ל-exit() ע"י חוט כלשהו בקבוצה של החוט המדובר.
  - סיום "טבעי" של החוט הראשי.
  - הריגת החוט ע"י קריאה ל-pthread_cancel() מחוט אחר כלשהו ביישום.

t9_3.c

❖דוגמא ל pthread_exit מהחוט הראשי.

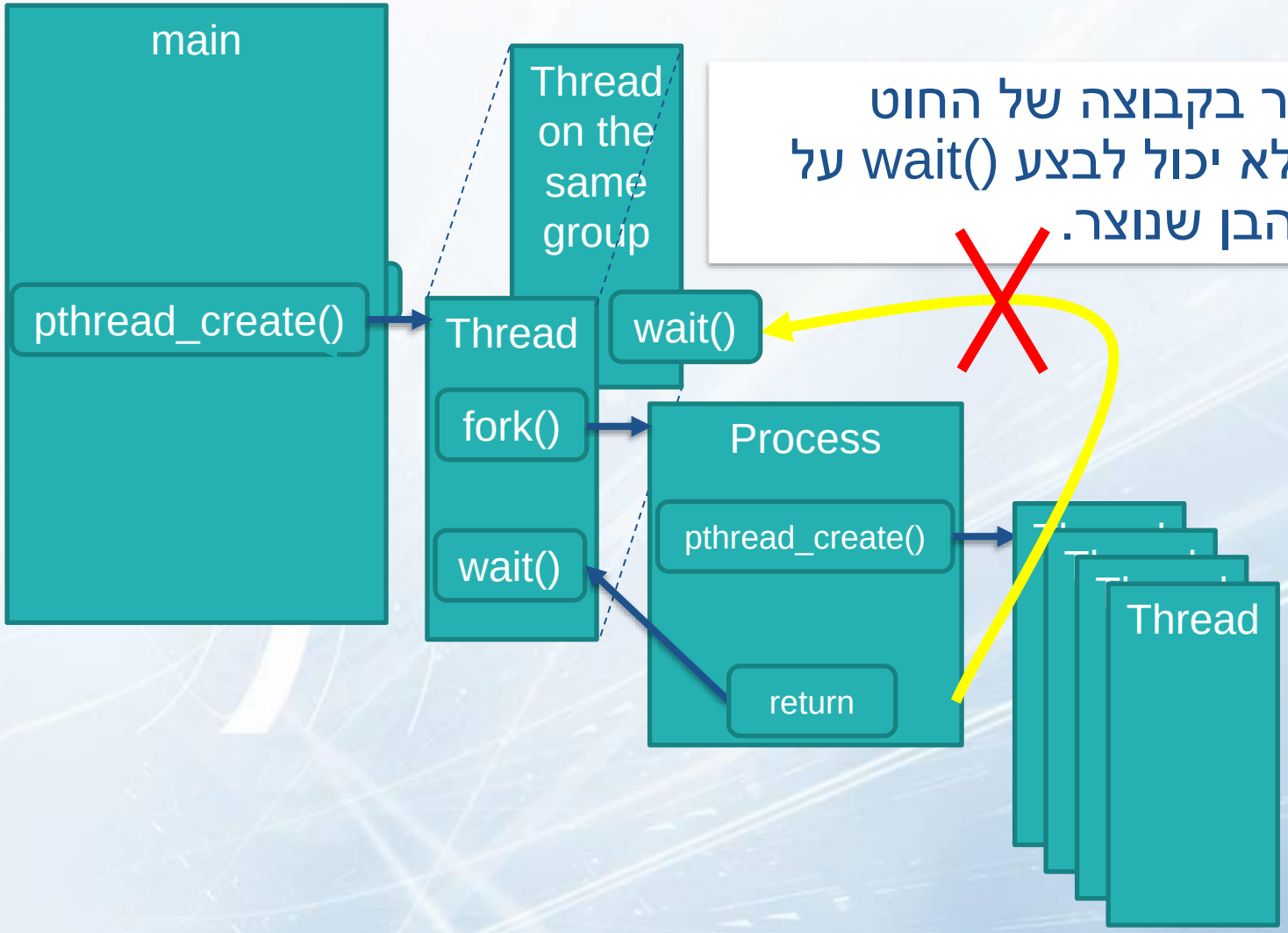
יש לבטל ב main את //sleep(1);

```
u2 89-231 53 : gcc -l pthread t10_3.c
u2 89-231 54 : a.out
my tid is 4107863808, my pid is: 8101, the loop value is: 0
my tid is 4118353664, my pid is: 8101, the loop value is: 0
u2 89-231 55 :
```

יש להוסיף את pthread_exit((void*)0); בתוך main

```
u2 89-231 56 : gcc -l pthread t10_3.c
u2 89-231 57 : a.out
my tid is 3661342464, my pid is: 28638, the loop value is: 0
my tid is 3671832320, my pid is: 28638, the loop value is: 0
my tid is 3661342464, my pid is: 28638, the loop value is: 1
my tid is 3671832320, my pid is: 28638, the loop value is: 1
my tid is 3661342464, my pid is: 28638, the loop value is: 2
my tid is 3671832320, my pid is: 28638, the loop value is: 2
u2 89-231 58 :
```

Fork in Thread



חוט אחר בקבוצה של החוט
הקורא לא יכול לבצע wait() על
תהליך הבן שנוצר.

fork בתוך חוט

- כאשר חוט קורא ל-()fork, נוצר תהליך חדש שהוא הבן של החוט הקורא בלבד
  - חוט אחר בקבוצה של החוט הקורא לא יכול לבצע ()wait על תהליך הבן שנוצר.

- לתהליך הבן החדש יש חוטים משלו. בהתחלה, חוט יחיד – החוט הראשי.
- חוטים נוספים יכולים להיווצר בהמשך בתהליך הבן.
- גם אם תהליך הבן מכיל יותר מחוט אחד, חוט האב יכול לבצע ()wait על תהליך הבן פעם אחת בלבד – להמתין לסיום **תהליך** הבן.

t9_4.c

- דוגמא לחוט ו fork

```
u2 89-231 59 : gcc -l pthread t10_4.c
u2 89-231 60 : a.out
my tid is 3895510784, my pid is: 30095, the loop value is: 0
my tid is 3895510784, my pid is: 30095, the loop value is: 1
my tid is 3895510784, my pid is: 30095, the loop value is: 2
returned
u2 89-231 61 : gcc -l pthread t10_4New.c
u2 89-231 62 : a.out
returned
u2 89-231 63 : my tid is 4222732032, my pid is: 27531, the loop value is: 0
my tid is 4222732032, my pid is: 27531, the loop value is: 1
my tid is 4222732032, my pid is: 27531, the loop value is: 2
```

exec בתוך חוט

- אם קריאה ל-()execv מצליחה, החוט הקורא מתחיל מחדש בתור חוט ראשי בקבוצה חדשה של תהליך חדש
  - כולל הקצאת משאבים מחדש: זיכרון וכו'.
  - כל החוטים האחרים מופסקים.

t9_5.c

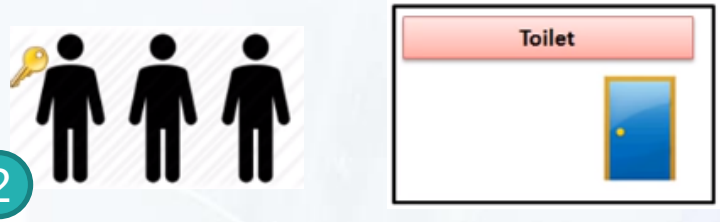
- דוגמא ל exec וחוטים.

```
u2 89-231 64 : gcc -l pthread t10_5.c
u2 89-231 65 : a.out
my tid is 3935069952, my pid is: 13339, the loop value is: 0
my tid is 3945559808, my pid is: 13339, the loop value is: 0
01   a.out  data       lastdates.save  shani.txt  t10_2.c  t10_4.c     t10_5.c  t5_5.c  test.c        usr1.txt
WWW  core   lastdates  shani.c         t10_1.c    t10_3.c  t10_4New.c  t5_4.c   t8_2.c  testfile.txt  usr2.txt
u2 89-231 66 :
```

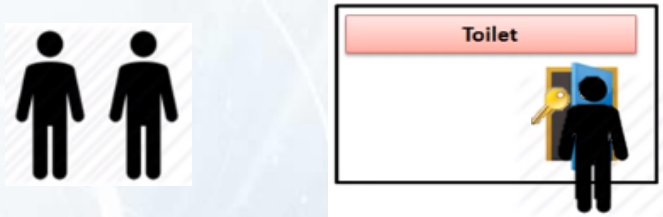
Semaphore Vs Mutex

Mutex

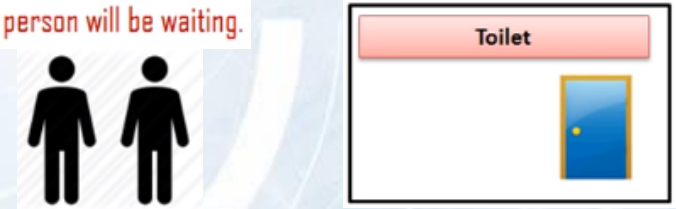
1 s assume toilet has a key and person who is having key can enter into the toilet.



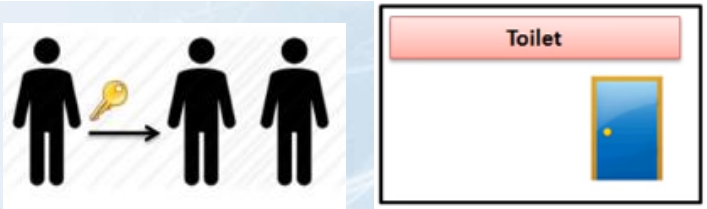
2 Person will unlock the door and then enter into toilet and and close it.



3 Other person will be waiting.



4 Now person will give it's key to next person and next person will enter into toilet.



Semaphore

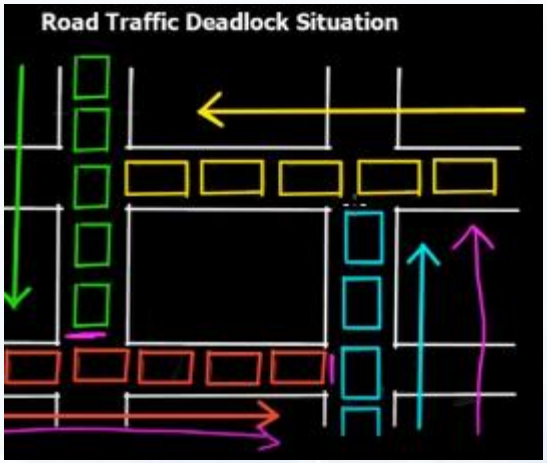
S=2

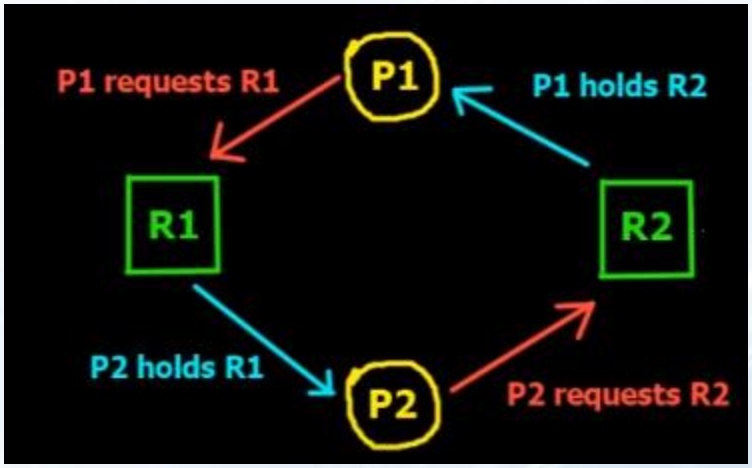
S=1

S=-1
Wait!!!

Yoram Segal

Deadlock





Methods for handling deadlock -

There are three ways to handle deadlock :

1) Deadlock prevention or avoidance -
   >> The idea is to not let the system into deadlock state.

2) Deadlock detection and recovery -
   >> Let deadlock occur, then do preemption to handle it once occurred.

3) Ignore the problem all together -
   >> If deadlock is very rare, then let it happen and reboot the system.
   >> Ignore the problem and pretend that deadlocks never occur in the system

3 Strategies to handle deadlocks :

1) Preemption -
   >> We can take a resource from one process and give it to other.
   >> This will resolve the deadlock situation, but sometimes it does causes problems.

2) Rollback -
   >> In situations where deadlock is a real possibility, the system can periodically make a record of the state of each process and when deadlock occurs, roll everything back to the last checkpoint, and restart, but allocating resources differently so that deadlock does not occur.

3) Kill one or more processes -
   >> This is the simplest way, but it works.

Mutex

❖מוטיבציה: t9_6.c

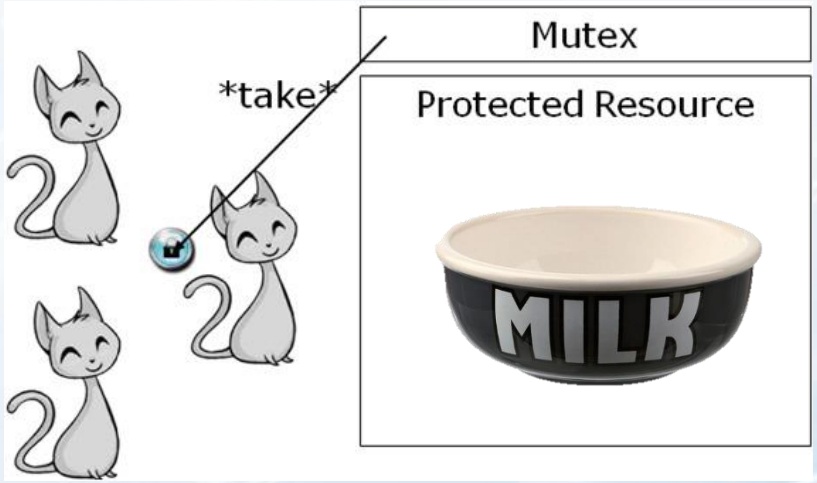
```
u2 89-231 69 : gcc -l pthread t10_6.c
u2 89-231 70 : a.out
counter value is: 3218187
u2 89-231 71 : a.out
counter value is: 3527220
u2 89-231 72 : a.out
counter value is: 6575929
u2 89-231 73 : a.out
counter value is: 6089293
u2 89-231 74 : a.out
counter value is: 4617099
u2 89-231 75 :
```

Mutex

- מנעול mutex מאפשר לחוט אחד בדיוק להחזיק בו (לנעול אותו).
  - כל חוט אחר המבקש להחזיק במנעול ייחסם עד אשר המנעול ישוחרר.
  - רק החוט המחזיק במנעול אמור לשחרר אותו (בעלות על המנעול).
- מנעולי mutex משמשים בדרך-כלל להגנה על גישה לנתונים משותפים, בתוך קטע קוד קריטי, ע"י נעילת המנעול בכניסה לקטע הקריטי ושחרורו בסופו.

Mutex


Mutex
Protected Resource
MILK


Mutex
take
Protected Resource
MILK


Mutex
Protected Resource
MILK


Mutex
Protected Resource
MILK
release

אתחול mutex

#include <pthread.h>

int pthread_mutex_init(pthread_mutex_t **mutex*, const pthread_mutexattr_t **attr*);

mutex- כתובת של אובייקט מסוג pthread_mutex_t

attr:

- **PTHREAD_MUTEX_NORMAL** - for ‘‘fast’’ mutexes
- **PTHREAD_MUTEX_RECURSIVE**
- **PTHREAD_MUTEX_ERRORCHECK**
- **PTHREAD_MUTEX_DEFAULT (or NULL)**
  - אתחול מנעול שכבר מאותחל – יגרור תופעה לא מוגדרת(תלוי ארכיטקטורת מעבד).
  - מומלץ לעבוד עם mutex מסוג "בודק שגיאות", כדי למנוע מצבים בעייתיים כגון אלו המסומנים באדום בשקף הבא

סוגי Mutex

סוג ה-mutex	נעילה חוזרת ע"י החוט המחזיק במנעול	שחרור מנעול ע"י חוט שאינו מחזיק במנעול	שחרור מנעול שכבר משוחרר
mutex מהיר	DEADLOCK	תוצאה לא מוגדרת	תוצאה לא מוגדרת
mutex רקורסיבי	הצלחה, מגדיל מונה נעילה עצמית ב-1	כשלון	כשלון
mutex בודק שגיאות	כשלון	כשלון	כשלון
mutex ברירת מחדל	לא מוגדר	לא מוגדר	לא מוגדר

הריסת mutex

- int pthread_mutex_destroy(pthread_mutex_t *mutex);

- הריסת מנעול גורמת לכך שלא יהיה אפשר להשתמש בו.
- כדי להשתמש שוב במנעול אפשר להפעיל עליו את הפונקציה pthread_mutex_init
- הריסת מנעול שנמצא במצב נעול או לא מאותחל תגרור תופעה לא מוגדרת.

נעילה, נסיון נעילה ושחרור

- נעילת mutex:

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

  - הפעולה חוסמת עד שה-mutex מתפנה ואז נועלת אותו

- נסיון לנעילת mutex:

```
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```

  - הפעולה נכשלת אם ה-mutex כבר נעול, אחרת נועלת אותו.

- שחרור mutex נעול:

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

דוגמה: מנעולי mutex

```
pthread_mutex_t m;
int count;

void update_count() {
   pthread_mutex_lock(&m);
   count = count * 5 ;
   count++;
   pthread_mutex_unlock(&m);
}
```

```
int get_count() {
   int c;
   pthread_mutex_lock(&m);
   c = count;
   pthread_mutex_unlock(&m);
   return c;
}
```

1. מדוע צריך להגן על הגישה ל-count בתוך update_count()?
   כדי למנוע שיבוש ערך count בעדכונים מחוטים שונים.
2. מדוע צריך להגן על הגישה ל-count בתוך get_count()?
   כדי למנוע קבלת תוצאות חלקיות הנוצרות במהלך העדכון

שימו לב! גם אם ביטוי ההגדלה היה count++, לא מובטח שהקוד הנפרש באסמבלר הינו אטומי, ולכן יש להפעיל מנגנון סנכרון לפי הצורך.

נעילה, נסיון נעילה ושחרור

- t9_7.c

```
u2 89-231 82 : gcc -l pthread t10_7.c
u2 89-231 83 : a.out
counter value is: 10000000
u2 89-231 84 :
```

מקורות

- https://www.youtube.com/watch?v=O3EyzlZxx3g
- https://www.youtube.com/watch?v=DvF3AsTglUU
- https://www.youtube.com/watch?v=UVo9mGARkhQ