ארגון המחשב ומערכות הפעלה

אביב תשפ"ד

תרגול 11 – תתי תהליכים וסנכרון

תתי תהליכים - Threads

- **תהליך** (process) מופרד מתהליכים אחרים (כולל זיכרון פרטי).
  - לתהליכים אין זיכרון משותף!

- **תת תהליך** (Thread, חוט) הוא יחידת ביצוע עצמאית **בתוך תהליך**:
  - תתי תהליכים משתפים ביניהם את משאבי התהליך כמו מרחב הזיכרון.
  - אבל, לכל תת תהליך מחסנית משלו (רק בהחלפת הקשר).

תהליך 1

תהליך 2

תהליך 3

תהליך 4

תהליך 5

תהליך 1

תהליך 2

תהליך 3

תת תהליך 1

תת תהליך 2

תת תהליך 3

תת תהליך 4

תת תהליך 5

תת תהליך 6

תהליך 4

תהליך 5

סנכרון

למה צריך לסנכרן?

נניח משתנה a משותף לשני **תתי תהליכים**.

מה יודפס אם תתי התהליכים הבאים ירוצו במקביל?

```
int b = a;
print(b);
If (a>c)
  a++;
print(a);
```

```
int b = a;
print(b);
If (a>c)
  a--;
print(a);
```

סדר הרצה אפשרי (1)

נניח ש a=1 ו c=0.

```
int b = a;   (5)
print(b);    (6)
If (a>c)     (7)
  a--;       (9)
print(a);    (10)
```

```
int b = a;   (1)
print(b);    (2)
If (a>c)     (3)
  a++;       (4)
print(a);    (8)
```

(2) **print 1**
(3) If (1>0)
(4) a=2
(6) **print 2**
(7) If (2>0)
(8) **print 2**
(9) a=1
(10) **print 1**

סדר הרצה אפשרי (2)

נניח ש a=1 ו c=0.

```
int b = a;  (3)
print(b);   (4)
If (a>c)    (5)
  a--;      (6)
print(a);   (9)
```

```
int b = a;  (1)
print(b);   (2)
If (a>c)    (7)
  a++;      --
print(a);   (8)
```

(2) **print 1**
(4) **print 1**
(5) If (1>0)
(6) a=0
(7) If (0>0)
(8) **print 0**
(9) **print 0**

סמפורים - Semaphores

- בעזרת סמפורים נוכל לסנכרן את ההרצה של כמה תתי-הליכים.
- מנגנון זה מאפשר שבכל רגע נתון, רק תת-תהליך אחד יריץ את הקטע הקריטי של הקוד.
- כל סמפור מתפקד במנעול כניסה ויש לו מונה, תור.
- תור הממתינים מנוהל בשיטת FIFO.

סמפורים - Semaphores

2 פעולות בסיסיות על הסמפור:

1. כאשר תת תהליך מבצע **wait**:
   מקטינים את המונה ב-1. אם מונה הסמפור אי שלילי (לאחר ההורדה) אז התת תהליך מקבל את המשאב. אחרת, התת תהליך נכנס להמתנה בתור.
2. כאשר תת תהליך מבצע **signal**:
   מגדילים את המונה ב-1. אם תור הממתינים לא ריק, אז התת תהליך הראשון בתור מתעורר.

דוגמא

קוד ללא סמפור (2 תתי-תהליכים):

```
#define MAX_VAL   10


void T1()
{
      for (int i = 0; i < MAX_VAL; i++)
      {
            printf(“c%d ", i)
       }
}

void T2()
{
      for (int i = 0; i < MAX_VAL; i++)
      {
      printf("p%d ", i);
       }
}
```

תוצאת הרצה:

c0 c1 c2 c3 c4 c5 c6 c7 c8 c9

p0 p1 p2 p3 p4 p5 p6 p7 p8 p9

או

c0 p0 c1 c2 p1 c3 c4 p2 p3 p4

c5 c6 c7 c8 c9 p5 p6 p7 p8 p9

או כל ערבוב אפשרי אחר

דוגמא

נאתחל Sem sem = s_create(1)

```
void T1()
{
    wait(sem);
    for (int i = 0; i < MAX_VAL; i++)
    {
        printf(“c%d ", i);
    }
    signal(sem);
}
```

```
void T2()
{
    wait(sem);
    for (int i = 0; i < MAX_VAL; i++)
    {
        printf("p%d ", i);

    }
    signal(sem);
}
```

תוצאת הרצה:

c0 c1 c2 c3 c4 c5 c6 c7 c8 c9
p0 p1 p2 p3 p4 p5 p6 p7 p8 p9

או

p0 p1 p2 p3 p4 p5 p6 p7 p8 p9
c0 c1 c2 c3 c4 c5 c6 c7 c8 c9

דוגמא

בעת אנו רוצים את הפלט של T2 ורק אחר כך את הפלט של T1.

נאתחל Sem sem = s_create(0)

```
void T1()
{
    wait(sem);
    for (int i = 0; i < MAX_VAL; i++)
    {
        printf(“c%d ", i);
    }
    signal(sem);
}
```

```
void T2()
{
    /* No wait */
    for (int i = 0; i < MAX_VAL; i++)
    {
        printf("p%d ", i);
    }
    signal(sem);
}
```

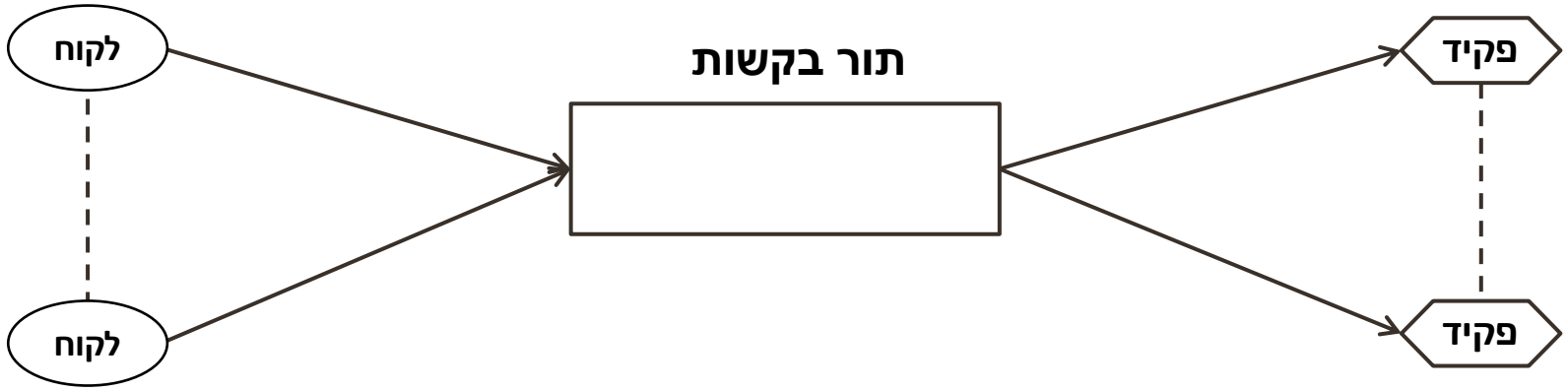
תוצאת הרצה:

p0 p1 p2 p3 p4 p5 p6 p7 p8 p9
c0 c1 c2 c3 c4 c5 c6 c7 c8 c9

תרגיל מסכם

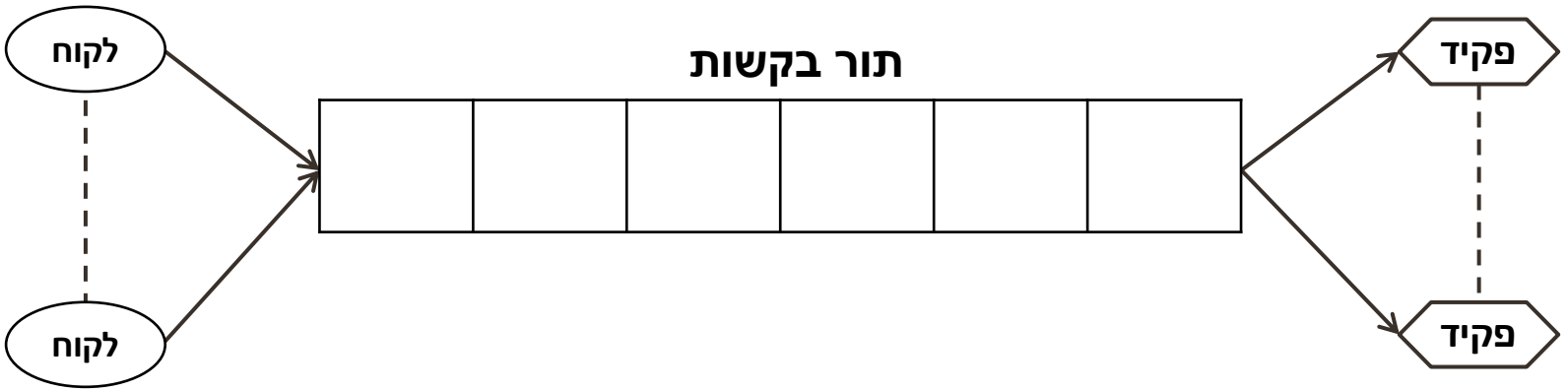
נגדיר מערכת שירות לקוחות שמורכבת מ:

1. לקוחות שיש להם בקשות שירות
2. מערך של פקידים שאמורים לטפל בבקשות
3. תור בקשות מסוננרן. הלקוחות מכניסים אליו בקשות והפקידים שולפים ממנו אותן.



תרגיל מסכם

- **המטרה:** לסנכרן בין הגעות של בקשות לטיפולם.
- **פתרון:** נגדיר סמפור
  - ערך שלילי של סמפור יציין את מספר הלקוחות בתור שעדיין לא טופלו.
  - ערך חיובי של סמפור יציין את מספר הפקידים הפנויים לקבל בקשה לטיפול.



פתרון – מימוש עם 2 סמפורים

אתחול (נניח שיש פקיד אחד במערכת, לשם פשטות):

- Sem sem1 = s_create(0)
- Sem sem2 = s_create(1) (אתחול sem2 בערך 1 משמעותו שקיים פקיד אחד במערכת)

- **לקוח:**

```
wait(sem2);  /*sem2.n=0 */
q.insert_in_fifo(request);
signal(sem1);  /*sem1.n=1*/
```

- **פקיד:**

```
wait(sem1);  /*sem1.n= 0*/
request=q.pop_from_fifo();
handle(request);
signal(sem2);   /*sem2.n=1*/
```

הסבר נכונות הקוד

נראה שסדר הפעולות חייב לעבוד במחזוריות מסוימת שמכתיבה התנהגות טובה.

לפני ה signal הראשון שנקרא במערכת:

- הפקיד מחכה לבקשת הלקוח (בגלל האתחול sem1.n=0)
- כל הלקוחות, פרט לאחד, לא יכולים לעבור את ה wait שלהם או שאינם פועלים ברגע זה

הסבר נכונות הקוד

- בשלב מסוים תהליך (לקוח) רץ ועובר את ה wait שלו.

  בגלל האתחול sem2.n=1 רק אחד יכול לעבור.

- עד שהלקוח מריץ את ה signal שלו, אף תהליך אחר לא יכול להתקדם לאיזור הקריטי (אם יש או אם אין תהליכים אחרים במערכת).

- הלקוח יוצא מהאזור הקריטי שלו ומבצע signal, בסופו של דבר.

הסבר נכונות הקוד

- מתישהו תהליך בקשת לקוח מקבל signal ומתעורר. בנקודה זו כל שאר התהליכים, אם יש כאלו, לא יכולים להתקדם מעבר ל wait שלהם.
- רק כשהתהליך הפקיד מסיים את הקטע הקריטי הוא קורא ל signal בסופו של דבר.
- ברגע זה, הוא יאפשר ריצה של לקוח אחד (ברגע שהוא יגיע).
- זה זהה למצב ההתחלתי והראינו שאחרי המצב ההתחלתי מובטח ששוב נחזור לאותו המצב ההתחלתי. ניתן להראות באינדוקציה שנחזור לאותו המצב גם אחרי שנבצע את הפעולה מספר כלשהו של פעמים ובנוסף, שלעולם לא נגיע למצב "רע" לפיו ניגשים לאזור הקריטי מספר תהליכים במקביל.