מבחן אמצע במערכות הפעלה (8 ביולי 2008)

מרצה: דר׳ דני הנדלר; מתרגלים: דניאל גורדון, אלון גרובשטיין, ליאנה דיזנדרוק; בודק: עדי סוויסה

משך המבחן: שעה וחמישים דקות
חומר עזר: אסור
פתור את כל השאלות: סה״כ 100 נקודות.

1. תהליכים (processes) וחוטים (threads) (33 נקודות)

א. קיראו את קטע הקוד שלהלן.

```
int value=0

int thread_func(int id) {
  int temp;
  temp=value+id;
  printf("Thread%d value: %d", id, temp);
  value=temp;
}

int main() {
  int fork_id, status, i;
  pthread_t tids[3];

  fork_id=fork();
  if (fork_id == 0) {
    for (i=1; i≤3; i++)
      pthread_create(&tids[i-1], NULL, thread_func, i);
    for (i=0; i≤2; i++)
      pthread_join(tids+i, &status);

    printf("Second process value: %d", value);
  }
  else {
    wait(&status);
    printf("First process value: %d", value)
    }
```

העתיקו את הטבלה הבאה למחברת והשלימו את הערכים החסרים: בכל שורה, כתבו את הערכים האפשריים אותם יכול התהליך או החוט המתאים להדפיס בקטע הקוד שלעיל.

First process value	
Second process value	
Thread1 value	
Thread2 value	
Thread3 value	

ב. כאשר תהליך מבצע קריאה ל-fork, ה-scheduler של מערכת ההפעלה יכול לתת לתהליך האב להמשיך ולרוץ. לחילופין, ה- scheduler יכול לבצע preemption לתהליך האב ולהריץ במקומו את התהליך הבן. קיראו את שלוש הטענות החלופיות שלהלן.

ככל שעולה אחוז המקרים בהם התהליך הבן מבצע קריאה לפונקציה exec מייד לאחר היווצרו, כך:

(a) יורדת הישילות של scheduler המריץ את התהליך הבן מייד לאחר ה- fork לעומת scheduler הממשיך להריץ את התהליך האב.

(b) עולה הישילות של scheduler המריץ את ההתליך הבן מייד לאחר ה- fork לעומת scheduler הממשיך להריץ את התהליך האב.

(c) אין שום קשר בין אחוז המקרים הנ"ל לבין הישילות היחסית של שני סוגי ה- schedulers.

סמנו את הטענה הנכונה לדעתכם מבין שלוש הטענות שלמלעה. הסבירו **בקצרה אך באופן מדוייק**את בחירתכם.

2. Deadlocks (33 נקודות)

א. האם בהכרח קיים deadlock במצב לא בטוח (unsafe state)? נמקו תשובתכם בקצרה.

ב. אחד מארבעת התנאים ההכרחיים להיוצרות מצבי deadlock נקרא hold-and-wait. תארו תנאי זה בקצרה. נמקו במדויק מדוע אם תנאי זה אינו מתקיים לא יווצרו מצבי deadlock.

ג. התבוננו בטבלאות הבאות, המתארות את הקצאות ובקשות המשאבים במערכת כלשהיא ברגע נתון. במערכת קיימים חמישה תהליכים וארבעה סוגי משאבים.

Exists (E)			
RA	RB	RC	RD
8	5	9	7

Available (A)			
RA	RB	RC	RD
1	2	2	2

Current Allocation (C)				
	RA	RB	RC	RD
P0	1	0	1	1
P1	0	1	2	1
P2	4	0	0	3
P3	1	2	1	0
P4	1	0	3	0

Additional Requirements (R)				
	RA	RB	RC	RD
P0	2	2	0	3
P1	0	1	3	1
P2	1	1	0	2
P3	1	3	2	0
P4	2	0	0	3

i. הוכיחו כי מצב המערכת המתואר למעלה הינו מצב בטוח (safe state).

ii. האם מותר ל- scheduler לתת משאב מסוג RA לתהליך P0? במילים אחרות, האם הקצאה כזו מביאה את המערכת למצב בטוח? הוכיחו תשובתכם.

3. סינכרוניזיה (34 נקודות)

בשאלה זו עליכם לממש באמצעות semaphores (ומשתנים התומכים בקריאות וכתיבות בלבד) פתרון ל**בעיית השיירה**. בבעייה זו, כל thread מייצג מכונית. בדומה ל- tunnel problem, גם כאן על המכוניות לעבור דרך מנהרה חד כוונית. ההבדל הוא, שכאן מותר רק לקבוצה של 5 מכוניות **בדיוק** (הנוסעות כולן באותו כוון) לעבור את המנהרה. רק לאחר שקבוצה זו סיימה לעבור דרך המנהרה מותר לקבוצה נוספת (הנוסעת לאותו הכוון או לכוון הנגדי) לעבור דרך המנהרה.

מבנה הקוד הינו כלהלן:

Variable Declarations

PrepareToCross(int direction)

CROSS

DoneWithCrossing(int direction)

עליכם לממש את הפרוצדורות *PrepareToCross()* ו- *DoneWithCrossing()* ואת הגדרות המשתנים (כולל ערכי ההתחלה) בהם הן משתמשות. מותר להשתמש במשתני semaphore (גם binary וגם counting) ובמשתנים התומכים בקריאות/כתיבות בלבד. אין לבצע לולאות של busy waiting.

שני כווני המעבר האפשריים מיוצגים ע"י קבועים 0 ו- 1. כאשר thread מבקש לעבור בשיירה, הוא קורא לפרוצדורה *PrepareToCross* ומעביר לה ארגומנט המייצג את כוון המעבר. כאשר thread מסיים את המעבר, הוא קורא לפרוצדורה *DoneWithCrossing* ומעביר לה את אותו הארגומנט.

נאמר כי **thread i עובר את המנהרה** אם *thread* זה יצא מן הקריאה ל- *PrepareToCross* וטרם קרא ל- *DoneWithCrossing* **או** שהוא עדיין ב- *PrepareToCross* אך כבר אינו ממתין על semaphore וכשיקבל זמן ריצה יצא מן הפרוצדורה ללא המתנה.

הדרישות מהמימוש הן אלו:

i. לעולם אין במנהרה שני threads העוברים אותה בשני כוונים שונים.

ii. Threads עוברים את המנהרה בקבוצות של חמישה. כאשר הראשון שבהם יוצא מן הקריאה ל- *PrepareToCross*, ישנם בדיוק ארבעה threads אחרים העוברים את המנהרה באותו כוון.

iii. ישנה גם דרישת התקדמות (progress): כאשר ישנם חמישה (או יותר) threads המבקשים לעבור את המנהרה באותו הכוון, אזי יהיה מעבר נוסף של המנהרה.

בהצלחה!