תרגיל בית 5

מטרת התרגיל: לבנות יישום מרובה חוטים ולסנכרן ביניהם.

כשלמדנו על סינכרוניזציה וסמפורים בתהליכים וחוטים, הזכרנו את הפונקציות `sem_lock()` ו-`sem_unlock()`. אבל יש בחבילה זו פונקציה נוספת: `sem_trywait()`. פונקציה זו פועלת כמו `sem_lock()` אם הסמפור גדול מאפס, אך בשונה ממנה, כאשר הסמפור שווה אפס, במקום לגרום לתהליך הקורא לעצור ולהכנס למצב `blocked`, היא חוזרת מיד, אך מחזירה את הערך `-1` (ציון של שגיאה), ומכניסה ב-`errno` את הערך `EAGAIN`. (גם במנגנוני הסינכרוניזציה של חוטים יש פונקציה מקבילה: `pthread_mutex_trylock()` שעושה את אותו הדבר, בהבדל שכדרך פונקציות `pthreads`, מחזירה 0 אם היא מצליחה ו-`EAGAIN` אם היא נכשלת.)

כך, תהליך יכול לנסות להשיג משאב כלשהוא, ולקבל אותו באופן אטומי אם הוא פנוי, אך לא להתקע בציפיה שישתחרר אם הוא תפוס. במקום זאת, התהליך יכול לנסות להשיג משאב אחר, אולי אקויולנטי לראשון. בתרגיל זה נבנה מערכת שעושה שימוש ביכולת זו של סמפורים.

נניח קבוצה של מתאמני כושר שמוטל עליה לעבור מסלול מכשולים. במסלול יש כמה סוגי מכשולים, כגון קיר (שצריך לקפוץ מעליו), חבל (שיש לטפס עליו) או מחילה (צרה שיש לזחול דרכה). על כל מתאמן לעבור את כל סוגי המכשולים **בסדר כלשהו** (אין הכרח שכולם יעברו את המכשולים באותו הסדר).

על כל מכשול אפשר שיהיה **מתאמן אחד בלבד** בכל רגע. מתאמן יכול לגשת לכל מכשול ולנסות לעבור אותו, אם הוא פנוי. כיון ששיש מכשולים שדורשים יותר זמן, יש במסלול כמה מכשולים מאותו הסוג, כדי לאפשר ליותר ממתאמן אחד להתמודד עם הסוג הזה (אחרת, המכשול הזה היה הופך צוואר בקבוק והיה מצטבר מאחוריו תור.) כדי להשלים את המסלול, על מתאמן לעבור מכשול אחד **מכל סוג**.

מימוש באמצעות `mutex` מתאים למצב בו יש משאב יחיד שכולם מחכים לו. כאן יש כמה: אם מתאמן צריך עדיין לעבור חבל ומחילה כדי להשלים את המסלול, הוא יכול לגשת לכל אחד ממתקני החבל או המחילות שאינם תפוסים. אולם רק כדי לברר אם מכשול תפוס עליו ליטול את המנעול שלו, ואז, אם הוא אכן תפוס, ההמתנה בלתי נמנעת, למרות שאולי יש מכשול אחר פנוי שאתו אפשר היה דווקא להתקדם. המחשב, שלא כמו מתאמן אנושי, לא יכול להעיף מבט מסביב ולראות איזה מכשול פנוי: עליו לעבור עליהם אחד אחד ולשאול כל אחד אם הוא פנוי. במקרה שהוא אינו פנוי, התהליך נתקע על המכשול המסוים שעליו שאל.

כאן מגיעה שעתה היפה של `sem_trywait()`: באמצעות פונקציה זו מתאמן יכול **לנסות** לתפוס מכשול מסוג שעליו עדיין לעבור, ואם הוא במקרה תפוס, לעבור מיד למכשול אחר מאותו הסוג, או אולי אפילו למכשול מסוג אחר, שגם אותו עליו עדיין לעבור.

ומה אם כל המכשולים תפוסים? במקרה זה אין לו ברירה ועליו לחכות. (אפשר, כמובן, להמשיך ולנסות את כל המתקנים בסבב בלי הפסקה, אבל זה שקול ל-`busy wait` המקולל: בזבוז משאבי מחשב יקרים בחוסר יעילות משווע.)

אבל **למי** לחכות? נניח שמתאמן מסויים צריך עדיין לעבור מכשול חבל, ושיש שלושה מתקני חבל במסלול, אך שלושתם תפוסים כרגע. אם הוא יבחר לחכות על אחד מהם, הוא עלול להפסיד הזדמנות לתפוס את אחד ממתקני החבל האחרים, שאפשר שיתפנה קודם. מה עליו לעשות?

ההצעה כאן היא לנהל רשימה של מתאמנים מחכים. כל מתאמן שניסה את כל המכשולים הרלוונטיים לו ומצא את כולם תפוסים, יוסיף את עצמו לרשימה זו ויצא ל-"נבצרות" – יחכה שמתאמן אחר יעיר אותו. מצד שני, כל מתאמן המסיים להשתמש במכשול ומשחרר את הנעילה שעליו, יעבור גם על רשימת המתאמנים המחכים ויעיר גם אותם – את כולם. כתוצאה, כולם יחזרו לבדוק את מצב המכשולים: יתכן שאחד מהם יוכל, אולי, לנצל את המכשול שהתפנה. האחרים, אם אינם מוצאים דבר מה לעשות, יכניסו את עצמם חזרה לרשימת הממתינים וישובו ל-"נבצרות".

כדי למנוע עדכונים בו זמניים לרשימת המחכים, עליה להיות מוגנת בידי סמפור, כמובן. סמפור זה משותף לכל המתאמנים, ועל כן שייך למסלול בכללותו. עקב כך, בכל פעם רק מתאמן אחד יכול להוסיף את עצמו לרשימה או להסיר את עצמו ממנה.

כאמור, כל מתאמן יוצא ל-"נבצרות" אחרי שהוסיף עצמו לרשימה. זאת ניתן להשיג על ידי המתנה לסמפור נוסף. כיון שכל מתאמן מחכה באופן נפרד, על סמפור זה להיות מוקדש למתאמן.

הבה נבצע ספירת מלאי של הסמפורים בתרגיל זה:

1. לכל מכשול יש סמפור מסוג `mutex` שמבטיח שלכל היותר רק מתאמן אחד משתמש בו בכל רגע נתון.
2. לכל מתאמן יש סמפור המאותחל ל-0, עליו הוא מחכה בשעה שהוא ב-"נבצרות" ונמצא ברשימת הממתינים.
3. `mutex` נוסף מבטיח אטומיות של הגישות לרשימת הממתינים עצמה.

מהלך התכנית

- בניית המסלול על המכשולים שבו:
  בחרו מבני נתונים מתאימים שיאפשרו טיפול נוח במתקני המסלול. פרטי המסלול מתוארים בקובץ המצורף לתרגיל (ראו להלן). יש להצמיד לכל מתקן מנעול ששומר שרק מתאמן אחד משתמש בו בכל רגע. דרושה גם רשימה של כל **סוגי** המכשולים, שכל מתאמן יוכל לסמן בה את המכשולים שכבר עבר ולדעת אילו עליו עדיין לנסות.

- בניית קבוצת המתאמנים:
  גם את המתאמנים כדאי לקלוט לתוך מבנה נתונים מתאים. פרטי כל המתאמנים מצויים בקובץ נוסף (ראו להלן). יש להצמיד מנעול גם לכל מתאמן, כזה שמלחתחילה הוא במצב נעול, עליו המתאמן יוכל לחכות כאשר הוא ממתין בשעה שכל המתקנים תפוסים.

- בניית רשימת הממתינים:
  כדאי לבנות מנגנון נוח וקל להוספה והסרה של מתאמנים מהרשימה – אילו פעולות שעשויות להתבצע פעמים רבות. גם פעולות אילו חייבות להיות מאובטחות בידי מנעול, כיון שכל מתאמן מוסיף או מסיר את עצמו מהרשימה בעצמו, וסכנת הקונפליקט עם מתאמן אחר גדולה.

- הרצת האימון:
  נראה לי טבעי להקצות רצף הוראות נפרד לכל מתאמן. אני ממליץ על שימוש בחוטים, למרות שניתן לבצע את התרגיל גם עם תהליכים, אלא שאז התקשורת מסובכת בצורה ניכרת.

- איסוף המידע:
  על כל מתאמן לאסוף מידע על פעולותיו לצורך דווח בסוף. מידע זה כולל:
  - השעה בה התחיל האימון עבורו
  - השעה בה ניגש לכל אחד מהמכשולים
  - השעה בה סיים כל אחד מהם – ההפרש בין השניים אומר כמה זמן נדרש לו לעבור את המכשול
  - השעה בה סיים את המכשול האחרון
  - זמן הביצוע הכולל שלו
  - זמן ההמתנה, בו בילה ב-"נבצרות" כיון שלא נמצא מתקן פנוי אחד שהיה עליו לעבור

אני ממליץ מאד לשמור את המידע באופן מקומי ולדווח עליו עם סיום ריצת המתאמן (דרך הערך המוחזר של פונקצית ה-`run()` של החוט). אין להדפיס מידע החוצה ישירות מתוך החוט: גם מפני שאז המידע מחוטים שונים מתערבב, וגם משום שהדפסות כרוכות בסינכרוניזציה משל עצמן, מה שיכול לעכב את הריצה.

הערות

1. כדי לברר את השעה אפשר להשתמש בפונקציה הבנויה `gettimeofday()`. בררו את דרך פעולתה בתיעוד (man או Google). היא עושה שימוש במבנה בעל שני שדות: אחד מונה שניות והשני מיקרושניות.
מיקרושניות מהוות דיוק גדול מדי, ולשניות דיוק לא מספיק. אני מציע להשתמש במילישניות: הראשון כפול 1000 ועוד השני לחלק ל-1000. (אלה המספרים המופיעים בדוגמא לעיל.) זכרו שהתוצאה גדולה מ-`int`, יש להשתמש ב-`long`.

2. התרגיל ביסודו הוא סימולציה של תהליך שאפשר שיתרחש במציאות. אבל סימולציות מחשב יכולות להאיץ או להאט את המערכות אותן הן מדמות (יחסית לאופן שבו הן פועלות במציאות) מבלי לאבד נאמנות למקור. אם נניח שכל שניה במציאות היא מילישניה בסימולציה, ונקפיד על המרה זו בכל הפנים של המימוש, התוצאה שתתקבל תהיה זהה לזו שהיתה מצופה במציאות, אך זמן הריצה יהיה קצר פי 1000. דוגמת הפלט שלהלן מדמה מרוץ שנמשך למעלה משעתיים, בשעה שהתכנית הפיקה את הפלט תוך שניות ספורות.

3. הדמית מעבר מכשול נעשית באמצעות עיכוב החוט (המדמה מתאמן) למשך הזמן שנדרש לו לעשות זאת (ביחידות סימולציה, שעשויות להיות קצרות בהרבה, לפי (2) לעיל). זמן זה נקבע על פי שלושה גורמים:
   a. הזמן הממוצע הנדרש למעבר המכשול (כפי שצויין בקובץ הגדרת סוגי המכשולים)
   b. המיומנות של המתאמן, שיכולה להאריך או לקצר זמן זה ב-עד 25% (לפי המידע שבקובץ הגדרת המתאמנים)
   c. רכיב אקראי, שיכול להאריך או לקצר את הזמן ב-10% נוספים. לצורך רכיב זה אפשר להשתמש בפונקציה הבנויה `random()` (שגם את דרך פעולתה שלה אפשר לברר מהתעוד. זהירות: היא שונה מאד מ-`Math.random()` של `Java`.)

4. השתמשו ב-`malloc()` ו-`free()` כדי להגדיר משתנים שגודלם ידוע רק בזמן הריצה (כגון מספר המכשולים, או מספר המתאמנים). הטכניקה של הגדרת מקום גדול מראש היא גם בזבזנית וגם לא מבטיחה טיפול נכון בקלט גדול אפילו יותר.

5. הקפידו לבדוק את הערך המוחזר מכל פונקציה של `Unix` בה אתם משתמשים, ולהוציא הודעת שגיאה מתאימה אם הערך הזה מורה על שגיאה. השתמשו ב-`perror()` בכל מקום אפשרי.
(זכרו שפונקציות של `pthread`, בניגוד לקריאות המערכת האחרות, מחזירות 0 במקרה של הצלחה, ובמקרה של שגיאה, את הערך שכרגיל מגיע ל-`errno`.)

6. זכרו לחכות לחוטים שיסתיימו בסוף הריצה, ולקבל מהם את המידע שאספו במהלכה, ולהדפיסו.
כמו כן, אל תשכחו לשחרר את כל הזכרון הדינאמי שהוקצה (עם `malloc()`) במהלך התכנית.

7. אחרי שסיימתם לקודד, בדקו את התכנית עם מתאמן יחיד (פשוט שנו את המספר שבראש הקובץ `team_desc.txt` ל-1) וודאו שהוא עובר את כל המכשולים, ומדווח בסוף על זמנים סבירים.
אחר כך נסו עם שניים, וראו שגם הם מסתדרים ביניהם. אחר כך נסו עם 16 (שזה יותר מתאמנים מאשר מתקנים במסלול, מה שמבטיח שמישהו מהם יצטרך לחכות), וודאו שגם הם מסתדרים. רק אז נסו עם הקבוצה המלאה.

הגדרת המסלול והמתאמנים

לתרגיל זה מצורפים שני קבצים: course_desc.txt ו-team_desc.txt. הראשון מתאר את המסלול והמכשולים שבו, והשני את המתאמנים.

1. course_desc.txt:
מבנה קובץ הוא כדלקמן:
בשורה הראשונה יש מספר המציין את מספר השורות שיש בהמשך הקובץ.
כל שורה אחריה מתארת סוג מכשול אחד:
   a. שם המכשול
   b. כמה מכשולים מסוג זה יש במסלול
   c. הזמן הממוצע הדרוש לעבור את המכשול

לדוגמה, הנה תוכן של קובץ כזה:

2. team_desc.txt:
מבנה קובץ הוא כדלקמן:
בשורה הראשונה יש מספר המציין את מספר השורות שיש בהמשך הקובץ.
כל שורה אחריה מתארת מתאמן אחד:
   a. שם המתאמן (יחידאי – אין שני מתאמנים בעלי אותו השם)
   b. מספר (בין 25- ל-25+) המציין בכמה אחוזים המתאמן טוב (או גרוע) מהממוצע

לדוגמה, הנה תוכן של קטע מקובץ כזה:

כל השדות בקבצים מופרדים על ידי התו tab ('\t'); שורה מסתיימת ב-new-line ('\n').

הפלט הנדרש

להלן דוגמת פלט מפתרון של תרגיל זה:

```
Avraham results:
     enter: 1608424311216
     0.    wall obs: 1 start: 1608424311216 finish: 1608424311465 duration: 249
     1.    rope obs: 2 start: 1608424312704 finish: 1608424313162 duration: 458
     2.   crawl obs: 3 start: 1608424318176 finish: 1608424318841 duration: 665
     3. balance obs: 1 start: 1608424319648 finish: 1608424320049 duration: 401
     4.  bridge obs: 1 start: 1608424316344 finish: 1608424316604 duration: 260
     exit: 1608424320049   elapsed: 8833   active: 2033   wait: 6800

Boaz results:
     enter: 1608424311216
     0.    wall obs: 0 start: 1608424313633 finish: 1608424313865 duration: 232
     1.    rope obs: 1 start: 1608424311216 finish: 1608424311732 duration: 516
     2.   crawl obs: 4 start: 1608424311821 finish: 1608424312408 duration: 587
     3. balance obs: 2 start: 1608424318767 finish: 1608424319166 duration: 399
     4.  bridge obs: 0 start: 1608424316443 finish: 1608424316660 duration: 217
     exit: 1608424319166   elapsed: 7950   active: 1951   wait: 5999

Bilha results:
     enter: 1608424311216
     0.    wall obs: 1 start: 1608424315860 finish: 1608424316209 duration: 349
     1.    rope obs: 2 start: 1608424311216 finish: 1608424312004 duration: 788
     2.   crawl obs: 0 start: 1608424317871 finish: 1608424318846 duration: 975
     3. balance obs: 0 start: 1608424314733 finish: 1608424315310 duration: 577
     4.  bridge obs: 0 start: 1608424312637 finish: 1608424312981 duration: 344
     exit: 1608424318846   elapsed: 7630   active: 3033   wait: 4597

Bader results:
     enter: 1608424311216
     0.    wall obs: 1 start: 1608424315041 finish: 1608424315299 duration: 258
     1.    rope obs: 1 start: 1608424313508 finish: 1608424313976 duration: 468
     2.   crawl obs: 4 start: 1608424311216 finish: 1608424311821 duration: 605
     3. balance obs: 2 start: 1608424316115 finish: 1608424316442 duration: 327
     4.  bridge obs: 1 start: 1608424313066 finish: 1608424313281 duration: 215
     exit: 1608424316442   elapsed: 5226   active: 1873   wait: 3353
```

כיון שזו הדמיה עם מרכיבים אקראיים, הפלט שלכם יכול להיות שונה, כמובן.

את התרגיל יש לבצע בזוגות בבית על מחשב אישי או על מחשב במעבדה במכללה. יש לקבץ את קוד התכנית ואת הפלט שלה באמצעות ZIP, RAR או תוכנה דומה. שם הקובץ שתיצרו צריך להיות בפורמט הבא:

Israel_Israeli_123456789_Ploni_Almoni_987654321_Ex5.zip

כלומר: השם באנגלית ומספר הזהות של כל אחד מהמגישים, והמחרוזת "Ex5", מופרדים באמצעות קו תחתון ביניהם. את הקובץ (היחיד) יש להעלות באמצעות moodle לאתר הקורס. נא ציינו, בנוסף, את שמותיכם (בעברית) ואת מספרי הזהות שלכם בגוף ההגשה בקובץ ReadMe.txt. את התרגיל יש להגיש עד יום ראשון, 8 ביוני 2025.

בהצלחה!