

תרגיל בית #4 - רטוב-2

מטרת התרגיל היא:
תכנות בעזרת חוטים.





שימו לב: מקוריות הקוד תיבדק, להזכירכם העתקת שיעורי בית הינה עבירה משמעת בטכניון לכל המשתמע מכך.

הקדמה:

בתרגיל בית זה נממש בנק בו מוחזקים חשבונות ומספר משתנה של כספומטים דרכם ניתן לגשת לחשבונות אלו. על הפתרון להיות מקבילי ככל האפשר.
מטרת התרגיל הינה שימוש חוטים בעזרת ספריית pthreads.

תיאור התוכנית:

עליכם לממש:

1. כספומטים מקומיים (ATMs) – ישנם N ATMs בבנק. לכל ATM יש מספר סידורי.
2. בנק – בבנק שמורים כל החשבונות.
3. חשבון - לכל חשבון המאפיינים הבאים:
 - מספר חשבון
 - סיסמא
 - יתרה

תהליך עבודת התוכנית:

התוכנית מתחילה באתחול מספר כספומטים (המתקבל כקלט מהמשתמש).

פעולה – קיימות מספר פעולות: פתיחת חשבון, משיכה, בירור יתרה, הפקד והעברה בין חשבונות. כל פעולה לוקחת שניה (גם אם הפעולה נכשלה), כלומר בעת ביצוע כל פעולה יש לבצע `sleep(1)` (אם יש צורך במנעול, ה `sleep(1)` ימצא בתוכו ז"א לא משחררים מנעול שדרוש לפעולה למשך שנייה) ורק אז לסיים ולשחרר את החשבון.

חשבון – יש לשמור על נכונות לדוגמא: עדכון החשבון מ-2 כספומטים שונים בו זמנית זה לא טוב. יש לממש קוראים כותבים בתרגיל זה על החשבונות! **מימוש זה ייבדק!**

כספומט - מתעורר פעם ב-100 מילישניות ומבצע פעולה בודדת ואז שוב נרדם לעוד 100 מילישניות. הפעולות שעל הכספומט לבצע ניתנות לו בקובץ `txt`, שם הקובץ מתקבל כקלט מהמשתמש. הכספומט מסיים לאחר ביצוע כלל הפקודות בקובץ.

הבנק - כל 3 שניות מחליט הבנק לקחת עמלות מכל החשבונות, העמלה היא מספר אחוזים שלם **רנדומלי** בין 2%-4% (יש להגריל פעם אחת כל 3 שניות עבור כלל החשבונות), את הכסף שהבנק הרוויח מהעמלות הוא מעביר לחשבון הפרטי שלו (יש לבצע עיגול לint הקרוב ביותר ולחסר את int המעוגל מחשבון הבנק שממנו נגבתה העמלה).

מומלץ לממש את הבנק בעזרת חוטים שאחד מהם תפקידו לגבות עמלות.

מימוש:

עליכם לכתוב תוכנית המממשת את הבנק ואת החשבונות תוך שימוש בספריית pthreads של לינוקס.
הנחיות למימוש:

1. על התוכנית לשמור לקובץ log.txt הודעה על כל אירוע שמתרחש בבנק.
2. כל כספומט מקומי מקבל קובץ קלט משלו לדוגמה עבור כספומט N - ATM_n_input_file.txt, שם הקובץ מתקבל כקלט בהרצת התוכנית.
3. טיפ: נא לממש את המנגנון המטפל בבקשות של הכספומטים בעזרת פונקציה המקבלת שורת טקסט המכילה את אחת הפקודות שרשומות בסעיף 4 למטה.
4. הפעולות הנתמכות ע"י הכספומטים:
 - פתיחת חשבון:


```
0 <account> <password> <initial_amount>
```

 אם קיים חשבון בעל אותו המספר תחזור ללוג הודעת שגיאה:


```
Error <ATM ID>: Your transaction failed - account with the same id exists
```

 אם החשבון נפתח בהצלחה תחזור ללוג ההודעה:


```
<ATM ID>: New account id is <id> with password <pass> and initial balance <bal>
```
 - הפקדה:


```
D <account> <password> <amount>
```

 אם הסיסמא שגויה תחזור ללוג הודעת שגיאה:


```
Error <ATM ID>: Your transaction failed - password for account id <id> is incorrect
```

 אם ההפקדה התבצעה בהצלחה תחזור ללוג ההודעה:


```
<ATM ID>: Account <id> new balance is <bal> after <amount> $ was deposited
```
 - משיכה:


```
W <account> <password> <amount>
```

 אם הסיסמא שגויה תחזור ללוג הודעת שגיאה:


```
Error <ATM ID>: Your transaction failed - password for account id <id> is incorrect
```

 אם ערך המשיכה גדול מהיתרה תחזור ללוג הודעת שגיאה:


```
Error <ATM ID>: Your transaction failed - account id <id> balance is lower than <amount>
```

 אם המשיכה התבצעה בהצלחה תחזור ללוג ההודעה:


```
<ATM ID>: Account <id> new balance is <bal> after <amount> $ was withdrew
```
 - ביורר יתרה:


```
B <account> <password>
```

 אם הסיסמא שגויה תחזור ללוג הודעת שגיאה:


```
Error <ATM ID>: Your transaction failed - password for account id <id> is incorrect
```

 אם הביורר התבצע בהצלחה תחזור ללוג ההודעה:


```
<ATM ID>: Account <id> balance is <bal>
```
 - סגירת חשבון:


```
Q <account> <password>
```

 אם הסיסמא שגויה תחזור ללוג הודעת שגיאה:


```
Error <ATM ID>: Your transaction failed - password for account id <id> is incorrect
```

 אחרת החשבון ייסגר (יימחק) ויודפס ללוג:


```
<ATM ID>: Account <id> is now closed. Balance was <bal>
```
 - העברה מחשבון לחשבון:


```
T <account> <password> <target_account> <amount>
```

 לאחר ביצוע ההעברה יתרת חשבון היעד תגדל בamount ויתרת חשבון המקור תקטן בamount.
 אם הסיסמא שגויה תחזור ללוג הודעת שגיאה:


```
Error <ATM ID>: Your transaction failed - password for account id <id> is incorrect
```

 אם ערך ההעברה גדול מיתרת חשבון המקור תחזור ללוג הודעת שגיאה:


```
Error <ATM ID>: Your transaction failed - account id <id> balance is lower than <amount>
```

 אם ההעברה התבצעה בהצלחה תחזור ללוג ההודעה:


```
<ATM ID>: Transfer <amount> from account <account> to account <target_account> new account balance is <account_bal> new target account balance is <target_bal>
```

בכל הפעולות אם יש ניסיון גישה לחשבון שלא קיים (או שנסגר) תודפס הודעת שגיאה ללוג:
 Error <ATM ID>: Your transaction failed - account id <id> does not exist
 במקרה של העברה מחשבון לחשבון יש לבדוק ראשית את החשבון המקור ולאחר מכן את חשבון
 היעד ולהדפיס רק הודעת שגיאה אחת (כלומר אם גם המקור וגם היעד לא קיימים יש להדפיס
 שגיאה רק עבור המקור).

5. הפעולות המתבצעות ע"י הבנק:

- גביית עמלות:

תודפס ללוג ההודעה עבור כל חשבון:

Bank: commissions of <#> % were charged, the bank gained <#> \$ from account <acc id>

6. על התוכנית להדפיס את מצב כל החשבונות (בסדר עולה עפ"י מספר החשבון), כל חצי שנה במקביל לעבודה השוטפת בבנק:

Current Bank Status
 Account 123: Balance - 12 \$, Account Password - 1234
 Account 124: Balance - 100 \$, Account Password - 1111
 Account 133: Balance - 10 \$, Account Password - 5555
 .
 .
 The Bank has 1500 \$

- שימו לב שמצב הבנק צריך להיות מודפס תמיד החל מהפינה השמאלית העליונה של המסך. הדפסת מצב חדש תמחק את המצב הקודם כך שתיווצר מעין אנימציה. לשימושכם: printf("\033[2J") – מוחק את המסך. printf("\033[1;1H") – מזיז את הסמן לפינה השמאלית העליונה של המסך.
7. על ההדפסה לייצג את המצב המדויק של החשבונות (snapshot). לפעמים החשבונות יהיו נעולים וזה בסדר אם ההדפסה תאחר בזמן שלוקח לשחרר חשבון.
8. הבנק פועל כל עוד יש פקודות בקבצי הקלט שלא התבצעו.
9. הרעיון העיקרי בתרגיל הינו יצירת חוטים וסינכרון ביניהם כך שתתקבל מקביליות. לכן, בכל פעולה שאתם מממשים חישובו אם דרוש סינכרון, ואם כן כיצד לבצע אותו בצורה הטובה ביותר.
- ניקוד מלא יינתן לתרגיל בו תהיה מקבילות מירבית – חובה לממש מנגנון קוראים כותבים עבור החשבונות של הבנק (כולל מנגנון קוראים/כותבים על המבנה של מאגר הנתונים אם צריך נעילות גם שם).
- כל ישות בתוכנית (בנק, כספומטים...) צריכה להיות עצמאית (רמז: thread לכל אחד).
 - על כל ישות (כספומט/בנק) להפריע כמה שפחות לישויות האחרות, כלומר, בכל פעולה שנעשית תבצעו את ההגנה המינימלית הדרושה על מבני הנתונים. לדוגמא, כאשר כספומט A מבצע משיכה מחשבון n לכספומט B אסור לבצע פעולות באותו החשבון. הפתרון הטרויאלי יהיה ל"נעול" את כל החשבונות כאשר מתבצעת פעולה, כך שאף פעולה אחרת לא תוכל להתבצע. הגנה כזו כמובן אינה מינימלית ואינה מביאה למקביליות מירבית.
10. נניח כי אי אפשר להיכנס למינוס, כלומר היתרה בכל החשבונות תהיה חיובית, אם יש ניסיון לבצע משיכה גדולה מהיתרה המשיכה תיכשל.
11. מספר חשבון והיתרה הינם בגבולות int, הסיסמא הינה מספר בן ארבע ספרות.
12. ניתן להניח כי הקלט הנקלט מהקבצים הינו קלט חוקי ("חוקי הפורמט") אך הפעולות לא בהכרח, לדוגמא ניסיון משיכה עם סיסמא שגויה – הפעולה תיכשל.
13. הניחו כי מספר הכספומטים המקומיים N ושמות קבצי הקלט ATM_N_input.txt נקלטים כארגומנטים לתוכנית (ראו בהמשך).
14. על הלוג לתאר את סדר הפעולות בבנק כפי שהתבצעו ע"י הכספומטים.

דוגמא לקובץ קלט:

O 12345 1234 100
 W 12345 1234 50
 D 12345 1234 12
 O 12346 1234 45
 W 12345 1234 35
 .

טיפ: במהלך כתיבת התרגיל מומלץ להריץ את התוכנית בהתחלה עם כספומט אחד ולבדוק את קובץ ה-log.txt.

התוכנית של הבנק תורץ בצורה הבאה :

./Bank <Number of ATMs – N> <ATM_1_input_file> <ATM_2_input_file>...<ATM_N_input_file>

עבור ניסיון הרצה שגוי התוכנית תדפיס הודעת שגיאה ("illegal arguments") ותצא במקרים של שגיאות בקריאות system calls יש לצאת מהתוכנית ולהדפיס ל- stderr את תיאור השגיאה.

הנחיות לביצוע

- יש לממש את התרגיל ב-C או C++ בלבד.
- ניתן להשתמש בספריות STL של C++ בחופשיות, ולכן מומלץ לכתוב את התרגיל ב-C++ על מנת להמנע מכתיבת מבני נתונים ב-C.
- אין להשתמש ב-std::thread או pthread_rwlock_t.

הידור קישור ובדיקה

יש לוודא שהקוד שלכם מתקמפל ע"י הפקודה הבאה :
אם כתבתם ב-C++ :

```
> g++ -std=c++11 -Wall -Werror -pedantic-errors -DNDEBUG -pthread *.cpp -o Bank
```

אם כתבתם ב-C :

```
> gcc -std=c99 -Wall -Werror -pedantic-errors -DNDEBUG -pthread *.c -o Bank
```

יש לוודא שנוצר קובץ הרצה ללא שגיאות או warnings.

עליכם לספק Makefile עבור בניית הקוד. הכללים המינימליים שצריכים להופיע ב-Makefile הינם :

- כלל Bank שיבנה את התוכנית Bank.
- כלל עבור כל קובץ נפרד שקיים בפרויקט.
- כלל clean אשר מוחק את כל תוצרי הקימפול.
- יש לוודא שהתוכנית נבנית ע"י הפקודה make.
- יש לקמפל ע"י הדגלים המופיעים בחלק "הידור קישור ובדיקה" לעיל.

יש לוודא שאין דליפות זיכרון בתוכנית באמצעות הכלי valgrind. ניתן להתקין את הכלי במכונה הוירטואלית באמצעות הפקודה :

```
> sudo apt install valgrind
```

הפעלת הכלי מתבצעת באמצעות הפקודה :

```
> valgrind --leak-check=full ./Bank ...
```

ניתן להתעלם מבלוקים המדווחים כ-"still reachable". יש לקמפל באמצעות הדגל '-g' על מנת לקבל הפניה לשורות הרלוונטיות בקוד.

לתרגיל זה מצורף סקריפט check_submission.py (בצורה חלקית) את תקינות ההגשה. הסקריפט מצורף לנוחיותכם, ובנוסף לבדיקה באמצעות הסקריפט, עליכם לוודא את תקינות ההגשה.

הסקריפט מצפה ל-2 פרמטרים : נתיב ל-zip ושם קובץ ההרצה. לדוגמא :

```
> ./check_submission.py 123456789_987654321.zip Bank
```

הגשה:

הנחיות כלליות על אופן הגשת תרגילי הבית הרטובים ניתן למצוא באתר הקורס תחת הכותרת "עבודות בית – מידע ונהלים".

**בבקשה, בדקו שהתוכניות שלכם עוברות קומפילציה.
תוכנית שלא תעבור קומפילציה לא תבדק!**

Useful Man Pages:

pthread_create(2), pthread_join(2), pthread_exit(2), pthread_mutex_init(2),
pthread_mutex_lock(2), pthread_mutex_unlock(2), pthread_mutex_destroy(2),
rand(3),usleep(3)

בהצלחה!!!