**מבוא למערכות הפעלה, תרגיל מספר 2**

**סנכרון Threads**

כללי :

מועד ההגשה: ה-3 במאי 2019, 23:59 בלילה.

אופן הגשה: מערכת ההגשה האוטומטית של המחלקה.

שאלות הקשורות לעבודה יש לשאול **אך ורק** בפורום המתאים.

רקע:

בתרגיל זה עליכם להתמודד עם מערכת בה פעילים מספר גדול של Threads הניגשים בו זמנית למשאבים. התרגיל מתמקד בסנכרון ה-threads בכדי למנוע מירוצים.

לתרגיל מצורף פרויקט C# ובו מספר מחלקות כאשר בחלקן חסר מימוש (מסומן על ידי "your code here") אותו עליכם להשלים. התרגיל מתמקד בסינכרון נכון של threads כלומר בהתמודדות עם מירוצים. איתור ותיקון המירוצים הוא החלק המרכזי בתרגיל. עליכם להתמודד עם בעיה זו בעצמכם. נסו להתעלם מהמירוצים ולפתור את התרגיל באופן הפשוט ביותר, ואז תתמודדו עם הבאגים כאשר אלה יופיעו.

בתרגיל נבצע סימולציה של העברת מידע ברשת תקשורת. רשת תקשורת מורכבת ממספר מחשבים המקושרים ביניהם על ידי חיבורים פיזיים (כבלים). בתרגיל זה נדמה חיבור בין מחשבים על ידי תיבת דואר – תור של הודעות ממתינות (המחלקה MailBOx). מחשב יכול לרשום הודעות לתיבות דואר של שכניו המידיים, או לקרוא את ההודעה הראשונה מתיבת הדואר שלו. מחשב לא יכתוב הודעות לעצמו.

כמובן, הרשת אינה מלאה. כלומר, קיימים זוגות מחשבים ביניהם אין קשר ישיר. במידה וישנה הודעה הנשלחת ממחשב אחד למחשב אחר שאינו מחובר אליו ישירות, על ההודעה לעבור דרך סדרה של מחשבים המחוברים ביניהם בכדי להגיע אל מחשב היעד. בפועל, עוברות ברשת הודעות רבות בו זמנית. כל מחשב, בכל רגע נתון, בודק האם ישנה הודעה הממתינה לטיפולו. הודעה זו יכולה להיות הודעה המיועדת אליו, או הודעה המיועדת למחשב אחר. אם ההודעה מיועדת למחשב אחר, על המחשב להעביר אותה בכיוון הנכון. פעולה זו נקראת ניתוב (routing). למחשבים אין תמונה כוללת של הרשת, והם מכירים רק את המחשבים המחוברים אליהם ישירות. המחשבים בונים, עם זאת, טבלת ניתוב המפרטת לכל מחשב ברשת את השכן הקרוב אליו ביותר.

3

2

5

1

7

8

6

4

למשל, ברשת המתוארת להלן, תראה טבלת הניתוב של מחשב מספר 1 כך:

|  |  |  |
| --- | --- | --- |
| מחשב יעד | נתב | מרחק |
| 2 | 5 | 3 |
| 3 | 5 | 2 |
| 4 | 4 | 1 |
| 5 | 5 | 1 |
| 6 | 5 | 4 |
| 7 | 5 | 3 |
| 8 | 4 | 2 |

וטבלת הניתוב של מחשב מספר 3 תראה כך:

|  |  |  |
| --- | --- | --- |
| מחשב יעד | נתב | מרחק |
| 1 | 5 | 2 |
| 2 | 2 | 1 |
| 4 | 8 | 2 |
| 5 | 5 | 1 |
| 6 | 7 | 2 |
| 7 | 7 | 1 |
| 8 | 8 | 1 |

כאשר מחשב מקבל הודעה המיועדת למחשב אחר, עליו למצוא את הנתב (השכן הקרוב ביותר ליעד) בטבלת הניתוב ולהעביר אליו את ההודעה.

**בניית טבלאות ניתוב** – כאשר המחשב מתחיל לפעול, הוא אינו יודע את טבלת הניתוב, ועליו לבנות אותן. האלגוריתם הבא (Bellman-Ford) אותו למדתם בקורס אלגוריתמים מאפשר בנייה של הטבלה, בהינתן מספר הזיהוי ID של המחשב:

T.distance(ID) = 0

Send T to all neighbors

while(true)

receive table T' from neighbor i

foreach computer j in T'

if( j is not in T or T'.distance(j) + 1 < T.distance(j))

T.distance(j) = T'.distance(j) + 1

T.router(j) = i

if( T has changed )

send T to all neighbors

**שלבי העבודה:**

1. בנו סימולציה של רשת מחשבים על ידי threads, כאשר כל thread מדמה מחשב(Node).
   1. ממשו תיבת דואר מסונכרנת. נתון לכם מימוש ראשוני של תיבת דואר על ידי מערך של הודעות. המימוש אינו פותר בעיות של מירוצים. צרו מחלקה SynchronizedMailBox המרחיבה (יורשת) את MailBox כך שהגישה למערך מסונכרנת ואין איבוד הודעות. לצורך חלק זה השתמשו במחלקה Mutex של C#.
   2. כאשר תיבת הדואר ריקה, נרצה שה-thread הקורא ימתין להודעות. ממשו המתנה זו על ידי המחלקה Semaphore של C#.
   3. שימו לב – אין לשנות את המחלקה MailBox. למעשה, בבדיקת התרגיל נשתמש במחלקהMailBox המקורית, כך שכל שינוי עלול לגרום לטעות קומפילציה.
   4. כמו כן, עליכם להשתמש לשמירת הנתונים במערך שב-MailBox המקורית, ולא ליצור מבנה נתונים חדש. שימו לב – לשם הפשטות השתמשנו במערך בעל גודל סופי להחזקת ההודעות. ברור כי אם המערך מתמלא, לא ניתן יהיה לכתוב הודעות נוספות ויהיה איבוד מידע. מותר להניח כי המערך גדול מספיק כך שלא תקרה תופעה זו, ולא להתמודד איתה.
2. ממשו את אלגוריתם Bellman-Ford על ידי שליחת הודעות מטיפוס RoutingMessage.
   1. ממשו במחלקה Node את השיטה Start המייצרת thread חדש ומפעילה אותו.
   2. ממשו במחלקה Node את השיטה HandleRoutingMessage כך שתממש את האלגוריתם.
   3. על מנת לעבור על טבלת הניתוב המוכלת בהודעה(rmsg), יש להשתמש בקוד הבא:
      1. foreach(int iNode in rmsg.GetNodes())
      2. iDistance = rmsg.GetDistance(iNode);
   4. הערה - במחלקה Node ישנו משתנה בשם m\_bDone. מטרתו של משתנה זה היא לסמל ל-thread לסיים את הלולאה המרכזית בשיטה Run וכך לסיים את ריצת ה-thread . כאשר מתקבלת הודעה מטיפוס KillMessage **הקוד כבר ניתן לכם ואין צורך למממש חלק זה.**
3. כעת, משטבלת הניתוב הושלמה, נרצה לשלוח הודעות ממחשב למחשב. המשתמש יכול לבקש ממחשב לשלוח הודעה (מחרוזת) למחשב אחר על ידי השיטה SendMessage אותה עליכם לממש. ברשתות אמיתיות וגם בתרגיל, איננו יכולים לשלוח מחרוזת שלמה בהודעה, ועלינו לחלק אותה לחבילות (packets) כאשר בכל חבילה תו אחד בלבד ופרטים על מיקום התו במחרוזת הסופית.
   1. ממשו את השיטה HandlePacketMessage המטפלת בהודעה מטיפוס PacketMessage. שימו לב שההודעה עלולה להיות עבור המחשב הנוכחי או עבור מחשב אחר.
      1. אם ההודעה עבור מחשב אחר, יש לנתב אותה לשכן הקרוב ביותר למחשב היעד (על פי טבלת הניתוב).
      2. אם ההודעה היא עבור המחשב הנוכחי, יש להוסיף את התו אל רשימת ההודעות. ברגע שההודעה מלאה (כלומר, כל התווים בהודעה הגיעו), יש לרשום את ההודעה למסך.
      3. חלק מהשדות במחלקה PacketMessage הינם:
         1. MessageID - המזהה (identifier) של ההודעה הספציפית. לכל הודעה ינתן קוד זיהוי שונה. כך, אם מחשב מקבל שתי הודעות בו זמנית, הוא יוכל לשייך כל packet להודעה ספציפית.
         2. Location - מאחר ואנו שולחים char אחד כל פעם, ולא ניתן להבטיח שכולם יגיעו על פי הסדר, עלינו להודיע על מיקומו של ה-char בהודעה.
   2. בשלב זה, צפוי שיתגלו בעיות הסנכרון, אם ישנן כאלה. לדוגמה, אם חבילה מסוימת "נאבדה", כלומר, כתוצאה ממרוץ נמחקה מתיבת הדואר, ההודעה לא תושלם ולא תוצג על המסך. ניתן לגרום לבעיות אלו להתגלות על ידי השיטות הבאות
      1. הוסיפו במחלקה MailBox, בשיטה Read, מתחת לפקודה m\_cMessages-, את השורה Thread.Sleep(10).
      2. הוסיפו "רעש" לאלגוריתם הניתוב כך שמדי פעם ישלח את החבילה לשכן הלא נכון.
      3. כמובן, אמצעים אלו נועדו לגילוי באגים ויש להסירם מהתוכנית הסופית.
4. מימוש ובדיקת התוכנית – לנוחיותכם, סיפקנו לכם בקובץ Program מספר פונקציות על מנת לבדוק את התוכנית שלכם.
   1. אתם רשאים, כמובן, לשנות את הקוד ב-Program.cs על מנת ליצור רשתות שונות בגדלים שונים למטרות debug. בכל אופן, הפרויקט יבדק עם main שלנו, כך שאין חשיבות לקובץ Program.cs שתגישו.
   2. כאשר אתם מבצעים הדפסות של הודעות שהתקבלו, עליכם להשתמש ב-Debug.WriteLine ולא ב-Console.WriteLine על מנת לאפשר הדפסה למסך ולקובץ בו זמנית.
   3. פונקציות ליצירת רשתות
      1. CreateRandomGraph – יוצרת גרף כללי עם מספר מחשבים cNodes וצפיפות קשתות dDensity.
      2. CreateBinaryTree – יוצרת רשת תקשורת עם מבנה של עץ בינארי.
   4. פונקציות בדיקה
      1. TestRouting – בודקת את יצירת ה-routing מבלי לשלוח הודעות
      2. TestMessagePassing – לאחר שלב ה-routing שולחת cMessages הודעות מקריות.
      3. בשני המקרים יודפסו לאחר הריצה טבלאות הניתוב של כל מחשב ורשימת ההודעות שלו, כאשר אם חבילה לא הגיעה ליעדה, יודפס סימן שאלה במקום התו שבחבילה.
   5. ניתן לפתור את הפרויקט ללא שינוי מלבד במחלקות Node ו-SynchronizedMailBox. במחלקות אלו אתם רשאים לבצע שינויים כרצונכם. ניתן להוסיף פונקציות חדשות, מבני נתונים, וכן הלאה.

הגינות – אל תעתיקו! אין להעתיק קטעי קוד מתלמידים אחרים, מהאינטרנט, או מכל מקור אחר. להזכירכם, אסור לשלוח קטעי קוד לתלמידים אחרים.

בהצלחה!