תכנות מרובה ליבות - תרגול 1#

הקדמה:

- מתרגל: גיא קורלנד.
- כל בעיה בתרגילי הבית לברר מול הבודקת.
- ניתן להירשם לעדכוני RSS (רצוי), ויש פורום.
- יהיה שימוש נרחב בספר. באתר יש הפניה לתיקון טעויות בספר.
- תרגילים: יהיו 5 תרגילים, צריך להגיש 4 מתוכם (ניתן להגיש את כולם, ה-4 הטובים ילקחו). 5% לכל תרגיל, 80% למבחן.

תרגילים:

- נעבוד על שרת aries, ניתן להגיע אליו דרך nova, זהו שרת Multicore עם מ"ה solaris.
- התרגילים יהיו ב-Java.
- רצוי לעבוד (אך לא חייבים) לעבוד עם ant, פירוט באתר.
- נעבוד עם JDK6.0, ניתן גם ב-JDK5.0.

יצירת חוטים ב-Java:

```
class MyThread extends Thread{
        @Override
        Public void run() {…}
…
```

הסבר מפורט במצגת intro1, שקופית 12.

- Start() – מתחיל thread.
- Join() – מתודה דינמית של חוט, מתודה זו מחכה עד ש-run() של ה-thread ישתחרר. יכול לשמש כמעיין מנעול (הריצה אחרי קטע קוד זה לא תמשיך עד שהחוט יסיים את פעולתו). מתודה זו זורקת InterruptException ולכן נדרשת עטיפת try-catch block.

ה-exception הזה משמש כדי לאפשר לעצור את ההמתנה לחוט – ע"י הפעלת מתודת חוט interrupt. אם הופעל על חוט interrupt ייזרק החריג הזה. כך, החוט ימשיך לרוץ ברקע, פשוט לא ימתינו לסיומו.

במצגת בשקופית 13 יש לינקים ל-API על locks ו-cocurrent – דברים שחשוב להכיר לשימוש בתרגילי הבית.

הערה: לשים לב ששרת ה-aris חסום למי שאינו תואר שני החל משעה מסויימת (Login יכשל...)

מבוא:

<u>מעבדים, ליבות</u>: processors, cores; יש הבדל ביניהם – מעבד יתייחס לרכיב על חתיכת סיליקון אחת, וכמה ליבות על חתיכת סיליקון אחת. אנחנו נתייחס לליבה כמעבד, יחידת עיבוד אטומית.

על כל מעבד רץ חוט – thread, יחידת חישוב עצמאית עם stack משלו (לא נכון באמת בפועל, אך לא רלוונטי עבורינו) ורגיסטרים משלו, ו-thread local - איזור בזיכרון פרטי עבור החוט הזה.

<u>הבעיה</u>: שימוש יעיל ב-single shared memory.

אובייקטים חיים בזיכרון (על ה-heap) ולא מניחים שום דבר על מערכת ההפעלה ואופן חלוקת המעבדים לחוטים (יש הנחות נכונות בפועל, אך נשתדל לא להשתמש בהן).

<u>בעית הפילוסופים הסועדים</u>:

5 פילוסופים סביב שולחן עגול נזקקים אחד לצ'ופסטיק של האחר כדי לאכול (מצגת 2, שקופית 5). המטרה: שכולם יאכלו, כלומר שהתוכנית תסתיים.

בעיה: ניתן להגיע ל-dead-lock – אם כולם מרימים את הצ'ופסטיק שמימינו, וכולם יחכו שהשמאלי ישתחרר – כולם ימתינו לעד במעגל.

פתרונות מוצעים:

- פתרון סיריאלי: תיאום סדר בין הסועדים. בעיה: צריך שעון שיסנכרן את כולם, וכאן אם אחד מתעכב, כולם מתעכבים.

פורמליזציה:

- Safety: רק אחד יכול להחזיק כל מקל
- Liveness: שלא יהיה deadlock כמתואר לעיל.

עוד פתרונות:

- כל סועד מרים מקל אחד ואחריו מקל שני, ואם לא מצליח (תפוס), מניח וממתין כמה דקות. בעיה: עדיין יכול להיות deadlock.
- מסדרים את המושבים לפי עדיפויות, ואם יש קונפליקט על מקל, ניתנת עדיפות לאחד מהשניים. בעיה: starvation.
- פתרון המלצר: אחד יחיד (אחר) שנותן אישור לאכול. הפתרון שומר על safety, liveness ומונע starvation.

נוסחת חישוב המהירות:

$$max - speed \leq \frac{f + (1-f)}{f + \frac{1-f}{s}} = \frac{s}{1 + f * (s-1)}$$

במקסימום אם אין הגבלות נוספות (למשל הגבלות מהירות של מערכת ההפעלה או משהו אחר), זהו חוק Amdahl:

$$speedup = \frac{1}{1 - p + \frac{p}{n}}$$

כאשר p הוא החלק שמיקבלנו ו-$1 - p$ הוא החלק הסיריאלי, n מספר **המעבדים** (לא חוטים, המיקבול האפשרי בפועל הוא המשנה).

מחוק זה נובע שהחלק הסיריאלי משפיע מאוד על חסם ה-speedup על מיקבול תוכנה.

<u>שאלה לדוגמא (היתה באחד המבחנים, שאלה שסגנונה חוזר במבחנים)</u>:

נניח נתונים שני שרתים, אחד עם מעבד יחיד ואחד עם 16 שרתים. זה עם מעבד אחד מריץ 5 זיליון פעולות בשניה, וכל מעבד במכונה השניה מריץ רק זיליון פעולות בשניה. מה עדיף מביניהם?

<u>תשובה</u>:

תלוי בתוכנית, תלוי בחלק המקבילי והחלק הסיריאלי. אם התוכנית מאוד מקבילית, עדיף זה עם ה-16. אחרת, נעדיף את השני. אם נסתכל על הגרף (בשקופית 21), נראה שנעדיף את המכונה עם ה-16 מעבדים אם החלק המקבילי 90% ומעלה.