**חוטים Threads**

* **טכניקת תכנות שבו יישום אחד ממומש ע"י מספר תוכניות עצמאיות נקרא Multi Programming.**
* **יש מספר סיבות לממש יישום ככה:**
1. **מידור**
2. **לפעמים נוח לתכנת ככה (סוג של שבירת המטלה).**
3. **היום גם יכול להיות שפעולות מסוימות יתבצעו במקביל הודות לליבות כפולות וריבוי מעבדים.**
4. **לפעמים (בתכנות מדעי) ניתן להפעיל מספר גישות לפתרון לבעיה קשה.**
5. **קל לממש בצורה כזאת תוכניות שעוברות באופן פתאומי מנקודה אחת ביישום לנקודה אחרת.**
6. **לפעמים יישום מבצע מספר גדול של פעולות קטנות ולפעמים נוח לתכנת את הפעולות הללו כתוכניות עצמאיות.**

**יישומי ,multi-programming בעצם מסתמכים על שיתוף זמנים:**

**כל תוכנית מקבלת את המעבד למכסת זמן שעם סיום מכסת הזמן המעבד עובר לתוכנית אחרת.**

**בעיקרו של דבר של דבר, multi-programming ניתן לממש ע"י 2 אפשרויות:**

**ריבוי תהליכים process.**

**ריבוי חוטים threads.**

**תהליך הוא אמצעי של הרצת תוכניות עצמאיות ברמת מערכת ההפעלה.**

**Thread הוא אמצעי שבו תהליך בודד של מערכת ההפעלה מתחלק לתתי תוכניות עצמאיות.**

**ממימוש Thread –ים בא בשני רמות:**

* **System Thread – התמיכה בריבוי תוכניות נתמך ברמת מערכת ההפעלה.**
* **User End Threads – הניהול של תתי התוכניות נעשה ביישום עצמו בדרך כלל בסיוע מערכת ההפעלה בכל הקשור למדידות זמן וכו'.**

**שאלה: למה יש Thread-ים?**

**אנחנו היום עובדים במערכות הפעלה מוגנות, שבין השאר מגינות על תהליכים זה מזה.**

**למה לא לעבוד ברמת התהליכים?**

**תשובה: תהליכים הם משאב של המערכת, הכמות עשויה להיות מוגבלת,**

**המידור בין תהליכים הוא גבוה, ותקשורת בין תהליכים הוא דבר יקר.**

**תהליך הוא משאב יחסית יקר בזמן CPU.**

**מערכת Thread-ים נחשב בעיני המערכת לתהליך אחד, ולכן אין מידור בין ה-Thread-ים: מרחב הכתובות של כולם משותף.**

**לדוגמא, ב-Java השדות הסטטיים של התוכנית משותפים לכל ה-Thread-ים.**

**יצירת Thread מחייבת פחות פעולות מיצירת תהליך.**

**לפעמים יש פחות הגבלות על יצירת Thread-ים.**

**יצירת תהליך הוא מהלך די תלוי במערכת ההפעלה.**

**תמיכה ב-Thread-ים ב-Java מאפשר מימוש קוד בלי תלוי במערכת ההפעלה.**

**העברת מידע בין Thread –ים הוא דרך שטחי זיכרון מה שיוותר מורכב זה הסנכרון ביניהם.**

**מהלך חייו של Thread ב-Java:**

* **נולד במצב new (קפוא).**
* **כאשר עושים start עובר ל-runnable (רוצה לרוץ)**
* **אחרי שעובר ל-runnable Thread עשוי לעבור**

**1. למצב המתנה wait ביוזמתו לזמן לא מוגבל**

1. **למצב המתנה ביוזמתו לזמן מוגבל timed wait**
2. **למצב חסימה אם ניסה לגשת למשאב נעול blocked**
3. **למצב סיום Terminated.**

[**http://math.haifa.ac.il/ronn/realtime/docs/**](http://math.haifa.ac.il/ronn/realtime/docs/)

**הקבצים הרלוונטיים הם**

**Part7.doc**

**עד Part7c.doc**

**דרך פשוטה יחסית ליצירת Thread שמספיקה לתרגיל הבית הוא כלהלן:**

**בכדי ליצור ב-Java "תוכנית עצמאית" הוא צריך להיות יורש של מחלקה מסוימות או ממשקים מסוימים, הכי פשוט להגדיר אותו יורש של המחלקה המובנית Thread או ממש של הממשק המובנה Runnable.**

**תחת**

**import java.lang.Thread;**

**השורה**

**class PrintThread extends Thread**

**גורם לכך מופע של PrintThread יכול להיות Thread למעשה תוכנית עצמאית.**

**זה אומר שאם אני אעשה משהו כמו**

**PrintThread task1 = new PrintThread( "task1" );**

**task1.start();**

**אני מריץ את התוכנית העצמאית.**

**הקצאת המופע של מחלקה לא גורמת לו לרוץ.**

**הוא רץ כאשר על המופע מבצעים מתודה start היא מתחילה לרוץ.**

**הקוד הביצועי של ה- Thread ב-Java הוא המתודה run() void.**

**להריץ Thread כממש של Runnable:**

**public class PrintTask implements Runnable**

**Thread task1 = new Thread(new PrintTask( "task1" ));**

 **task1.start(); // start task1**

**סיכום:**

**יצירת Thread אפשר לעשות ע"י:**

1. **הגדרת מחלקה חדשה כמחלקת בן של Thread.**
2. **הגדרת הקוד הביצועי של ה-Thread ע"י מימוש מתודה**

**public void run()**

1. **בתוכנית הראשית להקצות מופע של המחלקה ולבצע על המופע start().**

**יישום מרובה Thread-ים הוא בעייתי מכמה סיבות.**

**אחד מהם הוא הצורך בסנכרון.**

[**http://math.haifa.ac.il/ronn/java/wordfiles/**](http://math.haifa.ac.il/ronn/java/wordfiles/)

[**http://math.haifa.ac.il/ronn/realtime/docs/**](http://math.haifa.ac.il/ronn/realtime/docs/)

**אפשרויות ליצירת Thread –ים**

**שיטה 1:**

* **המינימום הוא לממש מחלקה שמממשת את הממשק המובנה בשפה Runnable.**
* **חייבים לדרוס את המתודה run לספק תוכנית ביצועית.**

**שיטה 2:**

 **לממש מחלקה שיורשת מ-Thread תוך דריסת run**

**שיטה 3**

 **PrintTask task1 =**

 **new PrintTask( "task1" );**

 **ExecutorService threadExecutor = Executors.newCachedThreadPool();**

 **threadExecutor.execute( task1 );**

* **לממש מחלקה שמממשת את Runnable**
* **להקצות TheadPool**
* **לעשות execute למופע של Runnable**
* **להגדיר מחלקה שמממשת את Runnable**
* **יעיל יחסת כאשר מקצים הרבה thread –ים לפעולות קטנות.**
* **בשיטה הזו נשמרים המשאבים של thread שסיים להקצאה חדשה בהמשך.**
* **מאפשר מימוש מה שנקרא מאגר thread –ים thread pool.**
* **מקצים מאגר של thread-ים, ע"י פקודה**

**ExecuteService *משתנה* = Exceutors.newCachedThreadPool();**

* **הקוד של ה-thread-ים עדיין כמו קודם, מיישמים של Runnable**
* **עדיין מקצים את המופעים ע"י new**
* **מפעלים את ה-thread-ים ע"י execute (במקום start) של מאגר ה-thread-ים.**

***משתנה*.execute(מצביע לחוט)**

**במקום**

**מצביע לחוט.start();**

* **בתוכנית מרובת thread-ים, הקוד של ה-main נחשב בעצמו ל-thread.**

**העברת מידע בין Thread-ים היא עניין פשוט ע"י שימוש או**

* **במשתנים סטטיים,**
* **הקצאה דינמית של שטח המשמש לתקשורת בין Thread-ים והעברת כתובת השטח לידי ה-Thread-ים המעורבים.**

**עדיין יש בעיה של סנכרון או תאום Thread-ים.**

**מה זה סנכרון?**

* **לפעמים צריך לדאוג שתוכניות עצמאיות שמשתפות פעולה יתבצעו בסדר מסוים.**

**לפעמים חלק מה-Thread-ים לא יכולים להמשיך (אין להם מה לעשות) עד שאחד ה-Thread –ים יעשה פעולה מסוימת.**

**דוגמא מיוחדת של הרעיון הזה הוא מניעה הדדית Mutual Exclusion.**

**מניעה הדדית פירושו ש-Thread ניגש למשאב משותף רק כאשר הוא תקין קרי: לפני שינוי או לחלוטין לאחר השלמת שינוי.**

**הבעיה נובעת מכך שבמסגרת מכסות זמן, Thread יכול להיעצר בכל נקודה בכל רגע.**

**מקור הבעיה היא בעצירת thread תוך כדי שינוי משאב גלובלי.**

**פתרון אפשרי הוא מנגנון נעילות:**

 **thread נועל משאב משותף לפני השינוי, וכל thread המנסה לגשת למשאב הנעול – נעצר.**

**כאשר ה-thread הנועל משחרר את המשאב, הראשון מבין הממתינים משתחרר וחוזר חלילה.**

**מנגנון נעילות הוא מנגנון רצוני (וולנטרי) שאינו מונע עצירה של thread באמצע.**

**אפשר לנעול מופע של מחלקה כלשהיא משום שמנגנון נעילות ממומש במחלקה Object.**

**אפשר לנעול ע"י מתודה שהיא synchronized – אם thread ראשון קורא למתודה על מופע מסוים – הוא נועל את המתודה.**

**התודה משתחררת רק כאשר הוא יוצא מהמתודה.**

**אם ה-thread נעצר באמצע – המתודה על המופע נשארת נעולה.**

**אם בזמן שהמתודה על המופע נשארת נעולה היא נקראת ע"י thread אחר ה-thread האחר ייעצר.**

**נעילה על מופע (בניגוד לנעילה על מתודה):**

**class A**

**{**

**}**

**A a = new A();**

**synchronized (a)**

**{**

**בזמן שנמצאים פה, המופע מצביע עליו נעול. a ש-**

**}**

**בכדי לא לאבד את היתרונות של multi-threading צריך לדאוג שהקוד המסונכרן יהיה מינימלי.**