**הגנות:**

* רשימת המשימות – מפני תהליכים בתוך הבריכה
* רשימת המשימות – תהליכים מבחוץ שרוצים להוסיף משימה לבריכה (לא רק התהליך היוצר)
* בריכת החוטים – להגן מפני הריסה כפולה

**פסאודו-קוד:**

תסתכלו בקובץ pseodocode.c בנפרד

**סכנות:**

* גם פונקציית הוספת משימה וגם הפונקציה של החוטים צריכה לקרוא את מצב ה-threadPool (כלומר, האם הוא עומד להיהרס) וגם צריכים לנעול את תור המשימות. יש כאן סכנת LIVELOCK איפשהו, אני די בטוח...  
  לדעתי בעצם לא יהיה LIVELOCK או DEADLOCK אם נממש כמו שצריך את הקריאה המרובה והכתיבה היחידה של המצב state, כי אם פונקציית הוספת המשימה עושה acquire קודם היא משחררת מהר ואם הפונקציה של החוט עושה acquire קודם, היא לא צריכה לחכות לפונקציית ההוספה כדי לקרוא את הערך של state כי אנחנו אמורים לאפשר קריאה בו-זמנית
* אולי יש סיכוי ש:
  + חוט יקרא את read\_state ויגלה שהמבנה במצב ALIVE (ויקרא את התור ויגלה שהוא ריק), ולכן יכנס לתוכן לולאת ה-while
  + CONTEXT SWITCH – לחוט הראשי
  + החוט הראשי מחליט להרוס את המבנה
  + בתוך פונקציית ההריסה, יתבצע broadcast. מעכשיו, לא נוספים איברים לתור ולא עושים שוב destroy ולכן משתנה התנאי לא יקבל signal
  + CONTEXT SWITCH – בחזרה לחוט הקודם
  + החוט (שנכנס לתוך ה-while לפני ה-CS) מגיע לפקודת wait ויחכה לנצח

פתרון: זה לא בעיה בעצם! זה כל הקטע. בקטע הראשון של הפונקציה של החוט, מהרגע שהוא נועל את task\_lock (בתחילת ה-while הראשי) ועד שהוא נכנס ל-switch, המצב היחיד שבו אין לו את המנעול task\_lock הוא בתוך פקודת ה-wait בו הוא ממתין לסיגנל או ממתין למנעול.  
מצד שני, בתוך פונקציית ההריסה של המבנה, יש נעילה של המנעול הזה! לכן אין סכנה שחוט יחכה לנצח לסיגנל...

* מקורה קורה אם מנסים להוסיף המון משימות ומיד קוראים ל-destroy עם פרמטר DO\_ALL? איך יודעים שכל המשימות יתבצעו?

הוספת משימה ופעולת הריסה קורים לפי הסדר – בכל פונקציה כנ"ל יש נעילה של תור המשימות.