



תרגיל חישוב מבוזר

בתרגיל זה נבנה מערכת לחישוב מבוזר ב-Python ונמצא בעזרתה מה המחרוזת שיצאה את ה-MD5 הבא:

EC9C0F7EDCC18A98B1F31853B1813301 (הערה לשם הקלה, המחרוזת היא מספר באורך 10 ספרות)

התוכנה תהיה בצורת Client-Server, כאשר יהיה שרת אחד שיחלק את המשמות ומספר Client-ים שיבצעו את החישובים.

ה-Server יהיה אחראי לחלק את העבודה ל-Client-ים השונים. הוא ידע אילו אפשרויות נוסו עד עכשיו, ויחלק לכל Client שיבקש ממנו בלוק מספרים שאותו ה-Client צריך לנסות.

כל Client יהיה אחראי על המחשב שעליו הוא רץ. אחריות זו תבוא לידי ביטוי על ידי ניצול מירבי של משאבי המחשב שעליו הוא רץ (כך ש-Client שרץ על מחשב עם 4 ליבות יבקש פי 4 יותר עבודה מ-Client הרץ על מחשב בעל ליבה אחת). בנוסף, מרגע קבלת העבודה, ה-Client ידאג שמספר העבודות שברשותו יתאמו את מספר המעבדים במחשב (כך שהוא באמת ינצל באופן מירבי את יכולות המחשב).

השתמשו במודול ה-threading על מנת לנצל את מירב המשאבים של כל מחשב.

לאחר שמצאתם את המחרוזת שייצרה את ה-MD5 הנ"ל הדפיסו אותו מהשרת (וניתן להפסיק לשלוח עבודה לכל ה-Client-ים).

את הפיתוח והבדיקות בצעו על המחשב שלכם (כשהוא גם ה-Client וגם ה-Server) ורק לאחר מכן נשתמש במספר מחשבים על מנת להשיג יותר כוח חישוב.

נקודות מחשבה להמשך (בנוסף למימוש) – איך הייתם מממשים מצב בו Client-ים חדשים מצטרפים באופן דינאמי לרשת שלכם, ו-Client-ים קיימים נעלמים בפתאומיות? שאלה זו היא אחת מהבעיות המרכזיות שעומדות בלב החישוב המבוזר הגדול באמת (אצל גוגל, אמזון ומיקרוסופט).

הערה – אומנם מודול ה-threading של פייתון תומך בהרצת מספר thread-ים באופן תיאורטי, בגלל צורת המימוש של Python הסטנדרטי (CPython) ה-Thread-ים שמוצרים בפייתון אינם "אמיתיים" כמו במערכת ההפעלה ואינם רצים במקביל. למרות זאת, אם נריץ את הקוד אותו תכתבו על מימוש שונה של Python שתומך ב-Thread-ים אמיתיים (נגיד IronPython או Jython) אז הקוד יעבוד בדיוק כפי שמצופה. ניתן להשתמש במודול ה-multiprocessing על מנת להשיג מקביליות אמיתית ב-CPython, אך זה ישאר כתרגיל הרחבה.