3 הצגת ממשק ה-Sockets API

במערכת יוניקס (Unix), ממשק ה-Sockets API מספק תהליך למערכות תקשורת זו עם זו. הוא תומך במגוון שיטות תקשורת, ואחת מהן היא תקשורת דרך האינטרנט.

וזו השיטה שמעניינת אותנו כרגע.

בשפת C ובמערכת יוניקס, ממשק ה-Sockets API הוא שילוב של קריאות ספרייה וקריאות מערכת (פונקציות המפעילות את מערכת ההפעלה ישירות).

ב-Python, ה-Sockets API הוא ספרייה המפעילה את ממשק ה-Sockets ברמת ה-C. לפחות בסביבות יוניקס. בפלטפורמות אחרות, היא תשתמש בממשק שהמערכת מספקת לתקשורת רשת.

אנחנו הולכים להשתמש בזה כדי לכתוב תוכניות שמתקשרות דרך האינטרנט!

3.1 תהליך חיבור לקוח

הדבר המבלבל ביותר בשימוש ב-Sockets הוא שיש מספר שלבים שצריך לבצע כדי להתחבר למחשב אחר, והם לא תמיד ברורים.

אך הנה הם:

1. **בקש ממערכת ההפעלה Socket.**
   ב-C, מדובר בתיאור קובץ (File Descriptor) - מספר שלם שמייצג את חיבור הרשת מעתה והלאה.
   ב-Python, הפונקציה תחזיר אובייקט שמייצג את ה-Socket. ממשקי שפות אחרות עשויים להחזיר מבנים שונים.
   החלק החשוב הוא שהשלב הזה מספק דרך להתייחס ל-Socket לצורך העברת נתונים בהמשך.
   שים לב: בשלב זה ה-Socket עדיין לא מחובר לשום דבר.
2. **בצע שאילתת DNS כדי להמיר שם קריא-אדם (כמו example.com) לכתובת IP (למשל 198.51.100.12).**
   DNS הוא מסד נתונים מבוזר המחזיק את המיפוי הזה, ואנו שואלים אותו כדי לקבל את כתובת ה-IP.
   נזדקק לכתובת ה-IP כדי לדעת לאיזו מכונה להתחבר.
   טיפ ל-Python:
   בעוד שניתן לבצע שאילתות DNS ב-Python עם `socket.getaddrinfo()`, פשוט לקרוא ל-`socket.connect()` עם שם המחשב יבצע את השאילתה עבורך, כך שניתן לדלג על שלב זה.
   טיפ ל-C (אופציונלי):
   השתמש ב-`getaddrinfo()` כדי לבצע את השאילתה.
3. **חבר את ה-Socket לכתובת ה-IP ולפורט מסוים.**
   חשבו על מספר פורט כמו דלת פתוחה שניתן להתחבר דרכה.
   הם מספרים שלמים בטווח 0–65535.
   דוגמה טובה היא פורט 80, הפורט הסטנדרטי המשמש לשרתי HTTP (לא מוצפן).
   חשוב שיהיה שרת שמאזין לפורט זה במחשב המרוחק, אחרת החיבור ייכשל.

4. **שלח וקבל נתונים.**
זהו השלב שחיכינו לו!
נתונים נשלחים כרצף של בתים (Bytes).
5. **סגור את החיבור.**
כשסיימנו, נסגור את ה-Socket ונודיע לצד המרוחק שאין לנו מה לומר עוד.
הצד המרוחק יכול גם הוא לסגור את החיבור בכל עת.

3.2 תהליך האזנה בשרת

כתיבת תוכנית שרת שונה מעט.

1. **בקש ממערכת ההפעלה Socket.**
כמו בצד הלקוח.
2. **שייך את ה-Socket לפורט.**
בשלב זה מגדירים מספר פורט לשרת שהלקוחות יוכלו להתחבר אליו. לדוגמה: "אני מאזין בפורט 80!"
אזהרה:
תוכניות שלא רצות כ-root/מנהל לא יכולות להשתמש בפורטים נמוכים מ-1024 – הם שמורים.
בחר מספר פורט גדול וייחודי, למשל בטווח 15,000–30,000.
אם תנסה לשייך פורט שבו משתמש שרת אחר, תקבל שגיאת "Address already in use".
עובדה מעניינת:
גם לקוחות משויכים לפורט. אם לא תשייך להם אחד מפורש, הם יקבלו פורט פנוי אוטומטית בעת החיבור – וזה לרוב מה שנרצה.
3. **האזן לחיבורים נכנסים.**
יש ליידע את מערכת ההפעלה שכאשר מתקבלת בקשת חיבור לפורט שבחרת, היא תעביר אותה לתוכנית.
4. **קבל חיבורים נכנסים.**
השרת יחסום (יירדם) כאשר תנסה לקבל חיבור חדש אם אין בקשות ממתינות.
הוא יתעורר כאשר מישהו ינסה להתחבר.
הערה:
הפונקציה `accept()` מחזירה Socket חדש!
ה-Socket המקורי שנוצר בשלב הראשון עדיין קיים ומאזין לחיבורים חדשים.
החיבור הנוכחי מטופל על ידי Socket ייעודי שנוצר עבורו.
לעיתים השרת יפעיל Thread או Process חדש כדי לטפל בכל לקוח. אך אין חוק שקובע שחייבים לעשות זאת.
5. **שלח וקבל נתונים.**
זהו השלב שבו השרת יקבל בקשה מהלקוח וישיב לה.
6. **חזור וקבל חיבור נוסף.**
שרתים נוטים להיות תהליכים ארוכי טווח ומטפלים בבקשות רבות לאורך חייהם.