5 פרויקט: לקוח ושרת HTTP

נכתוב תוכנית סוקטים שתוכל להוריד קבצים משרת אינטרנט! זו תהיה הלקוח שלנו לרשת. התוכנית תעבוד כמעט עם כל שרת אינטרנט, אם נכתוב אותה נכון.

ולא רק זאת, נכתוב גם שרת אינטרנט פשוט! תוכנית זו תוכל לטפל בבקשות מהלקוח שכתבנו… או מכל לקוח רשת אחר כמו Chrome או Firefox!

התוכניות ידברו פרוטוקול שאתם כנראה מכירים: HTTP, פרוטוקול העברת הטקסט ההיפרטקסטואלי.

ומכיוון שהתוכניות מדברות HTTP, ודפדפנים כמו Chrome מדברים HTTP, הן יוכלו לתקשר אחת עם השנייה!

5.1 מגבלות

כדי להבין טוב יותר את ה-API של סוקטים ברמה נמוכה יותר, בפרויקט זה אסור להשתמש באף אחת מהפונקציות העזר הבאות:

- הפונקציה `socket.create_connection()`
- הפונקציה `socket.create_server()`
- כל דבר במודולים urllib

לאחר כתיבת הפרויקט, יהיה ברור יותר כיצד פונקציות העזר הללו ממומשות.

5.2 קידוד תווים בפייתון

בסוקטים בפייתון שולחים ומקבלים רצפים של בתים (bytes), שהם שונים ממחרוזות פייתון. תצטרכו להמיר בין השניים כשתרצו לשלוח מחרוזת או להדפיס רצף בתים כמחרוזת.

רצפי הבתים תלויים בקידוד התווים שבו משתמשים במחרוזת. קידוד תווים מגדיר אילו בתים מתאימים לאילו תווים. אולי שמעתם על קידודים כמו ASCII ו-UTF-8. ישנם מאות כאלו.

קידוד התווים המוגדר כברירת מחדל ברשת הוא "ISO-8859-1".

זה חשוב מכיוון שתצטרכו לקודד את מחרוזות הפייתון שלכם לרצפי בתים, ותוכלו להגדיר את הקידוד כשתעשו זאת. (ברירת המחדל היא UTF-8).

כדי להמיר ממחרוזת פייתון לרצף בתים בקידוד ISO-8859-1:

```
s = "Hello, world!"  # מחרוזת
b = s.encode("ISO-8859-1")  # רצף בתים
```

רצף הבתים הזה מוכן לשליחה דרך הסוקט.

כדי להמיר מרצף בתים שהתקבל מסוקט בפורמט ISO-8859-1 למחרוזת:

```
("s = b.decode("ISO-8859-1
```

ואז זה מוכן להדפסה.

כמובן, אם הנתונים לא מקודדים ב-ISO-8859-1, תקבלו תווים מוזרים במחרוזת או שגיאה.

הקידודים ASCII, UTF-8, ו-ISO-8859-1 זהים עבור אותיות לטיניות בסיסיות, מספרים, וסימני פיסוק, כך שהמחרוזות שלכם יעבדו כמצופה, אלא אם תשתמשו בתווי Unicode מוזרים.

אם אתם כותבים זאת ב-C, כנראה עדיף פשוט לא לדאוג לכך ולהדפיס את הבתים כפי שהם מתקבלים. חלקם עשויים להיות זבל, אבל זה יעבוד ברוב המקרים.

5.3 סיכום HTTP

HTTP פועל על עקרון של בקשות ותגובות. הלקוח מבקש דף אינטרנט, והשרת מגיב על ידי שליחתו.

בקשת HTTP פשוטה מצד לקוח נראית כך:

```
GET / HTTP/1.1
Host: example.com
Connection: close
```

זוהי כותרת הבקשה, שמורכבת משיטת הבקשה, הנתיב, והפרוטוקול בשורה הראשונה, ולאחר מכן מספר שדות כותרת. ישנה שורה ריקה בסוף הכותרת.

בקשה זו אומרת "תביא את דף הבית מהשרת example.com ואני אסגור את החיבור מיד לאחר שאקבל את תשובתך."

סופי שורות מסומנים על ידי שילוב Carriage Return/Linefeed. בפייתון או ב-C, תכתבו זאת כך:

python
Copy code
```
"r\n\"
```

אם ביקשתם קובץ מסוים, זה יהיה בשורה הראשונה, לדוגמה:

```
GET /path/to/file.html HTTP/1.1
```

(ואם הייתה מטען נתונים בצמוד לכותרת, הוא היה מופיע מיד לאחר השורה הריקה. הייתה גם כותרת Content-Length שמציינת את אורך המטען בבייטים. אנחנו לא צריכים לדאוג לכך בפרויקט זה.)

תשובת HTTP פשוטה מצד שרת נראית כך:

```
HTTP/1.1 200 OK
Content-Length: 50
Content-Type: text/html

<html><body>Hello, world!</body></html>
```

זוהי כותרת התשובה, שמתחילה עם הגרסה, קוד המצב, ותיאור המצב בשורה הראשונה, ואחריה מספר שדות כותרת.
ישנה שורה ריקה אחרי הכותרת, ואחריה בא התוכן (במקרה הזה, HTML).

קוד המצב (לדוגמה, 200) ותיאור המצב (לדוגמה, OK) מראים אם התשובה הייתה הצלחה או כישלון.

הכותרת `Content-Length` מציינת את גודל התוכן שמגיע אחריה בבייטים.
הכותרת `Content-Type` אומרת מהו סוג התוכן, כדי שהלקוח יוכל להבין כיצד לפרש אותו.

5.4 תוכנית הלקוח

כתיבה של לקוח HTTP בסיסי.

1. הלקוח יתחבר לשרת דרך סוקט.
2. הוא ישלח בקשת HTTP בסיסית.
3. הוא יקרא את התשובה (כולל הכותרת והתוכן) מהשרת.
4. הוא יציג את התשובה.

עליכם לחשוב על הטעויות שעלולות להתרחש (לדוגמה: שם שרת לא קיים, חיבור שהשרת סגר מוקדם, קבצים שאינם נמצאים).

5.5 תוכנית השרת

כתיבה של שרת HTTP בסיסי.

1. השרת יאזין לסוקט ויחכה להתחברויות מלקוחות.
2. הוא יקרא בקשת HTTP בסיסית מהלקוח.
3. הוא ייצור תשובת HTTP פשוטה עם תוכן (לדוגמה, דף HTML פשוט).
4. הוא ישלח את התשובה ללקוח וינתק את החיבור.

השרת שלכם לא צריך להתמודד עם בקשות מורכבות; רק תשובות פשוטות מספיקות לפרויקט זה.

5.6 אתגרים נוספים

אם סיימתם את הפרויקט ורוצים לקחת אותו לשלב הבא:

- הוסיפו אפשרות ללקוח להוריד קובץ ולשמור אותו במערכת הקבצים.

- הוסיפו אפשרות לשרת להגיש קובץ אמיתי במקום HTML סטטי.
- אפשרו לשרת לטפל במספר לקוחות בו זמנית באמצעות `select` או שרשור (threads).