11 ניתוח מנות

כבר ראינו כמה בעיות בקבלת נתונים מובנים משרת. אתה קורא ל-`recv(4096)`, ומקבל רק 20 בתים בחזרה. או שאתה קורא ל-`recv(4096)` ונתון שהתקבל ארוך יותר מזה, ואתה צריך לקרוא שוב.

יש גם בעיה חמורה יותר. אם השרת שולח לך כמה חתיכות נתונים, ייתכן שתקבל את החלק הראשון וחלק מהשני. יהיה לך חבילה מלאה ולקראת החבילה הבאה שהיא חלקית! איך תשחזר את זה?

אנלוגיה לכך עשויה להיות אם הייתי צריך שתפרק משפטים בודדים מחסימת טקסט שאני אתן לך, אבל אתה יכול לקבל רק 20 תווים בכל פעם.

אתה קורא ל-`recv(20)` ומקבל:

This is a test of th
זו לא משפט מלא, אז אתה לא יכול להדפיס את זה עדיין. אז אתה קורא שוב ל-`recv(20)`:

This is a test of the emergency broadcas
עדיין לא משפט. קורא שוב:

This is a test of the emergency broadcast system. This is on
היי! יש שם נקודה, אז יש לנו משפט מלא. עכשיו אנחנו יכולים להדפיס אותו. אבל גם יש לנו חלק מהמשפט הבא כבר התקבל!

איך אנחנו הולכים לטפל בזה בצורה אלגנטית?

11.1 מה יכול להפוך את זה לקל?

מה יכול להפוך את זה לקל? אם היינו מ抽יץ את זה אז היינו יכולים לעשות משהו כזה:

python
Copy code
```
:while the connection isn't closed
  ()sentence = get_next_packet
    (print(sentence
```

האם זה לא יותר קל לחשוב על זה? ברגע שיש לנו את הקוד שמחלץ את החבילה המלאה הבאה מתוך זרם הנתונים, אנחנו פשוט יכולים להשתמש בזה.

אם הקוד הזה מספיק מורכב, הוא יכול להחיל חבילות מסוגים שונים מתוך הזרם:

python
Copy code
```
()packet = get_next_packet

:if packet.type == PLAYER_POSITION
(set_player_position(packet.player_index, packet.player_position
```

```
elif packet.type == PRIVATE_CHAT:
display_chat(packet.player_from, packet.message, private=True)
```

וכן הלאה.

זה הרבה יותר קל מלהתעסק בחבילות כקולקציות של בתים שעשויים להיות או לא להיות מלאים.

כמובן, לעבד את הנתונים זה הטריק האמיתי. בוא נדבר על איך לגרום לזה לקרות.

11.2 עיבוד זרם ליחידות

הסוד הגדול להצלחה הוא זה: ליצור חיץ גלובלי גדול.

חיץ הוא רק מילה נוספת לאזור אחסון לבתים רבים. בפייתון, זה יהיה מיתר של בתים, שזה נוח כי אתה כבר מקבל אותם בחזרה מ-`recv()`.

החיץ הזה יחזיק את הבתים שאתה רואה עד כה. תבדוק את החיץ כדי לראות אם הוא מחזיק חבילה של נתונים מלאה.

אם יש שם חבילה מלאה, תחזיר אותה (כמיתר או מעובד). וגם, באופן קרדינלי, תסיר אותה מהחיץ.

אם לא, תקרא שוב ל-`recv()` כדי לנסות למלא את החיץ עד שיהיה לך חבילה מלאה.

בפייתון, זכור להשתמש במילת המפתח `global` כדי לגשת למשתנים גלובליים, לדוגמה:

python
Copy code

```
packet_buffer = b''

def get_next_packet(s):
    global packet_buffer

    # עכשיו אנחנו יכולים להשתמש בגרסה הגלובלית כאן
```

אחרת פייתון פשוט ייצור משתנה מקומי חדש שיצלצל על הגלובלי.

11.3 דוגמת המשפטים שוב

בואו נסתכל על דוגמת המשפטים מתחילת הפרק.

נקרוא לפונקציית `get_sentence()`, והיא תסתכל על כל הנתונים שהתקבלו עד כה ותראה אם יש נקודה בתוכה.

עד כה יש לנו:

שום דבר. אין נתונים שהתקבלו. אין שם נקודה, אז אין לנו משפט, אז אנחנו צריכים לקרוא שוב ל-`recv(20)` כדי לקבל יותר בתים:

text
Copy code
```
This is a test of th
```

עדיין אין נקודה. קורא שוב ל-`recv(20)`:

text
Copy code
```
This is a test of the emergency broadcas
```

עדיין אין נקודה. קורא שוב ל-`recv(20)`:

text
Copy code
```
This is a test of the emergency broadcast system. This is on
```

יש אחת! אז אנחנו עושים שני דברים:

1. מעבירים את המשפט כך שנוכל להחזיר אותו, ו:
2. מסירים את המשפט מהחיץ.

לאחר שלב שני, המשפט הראשון נעלם והחיץ נראה כך:

text
Copy code
```
This is on
```

ואנחנו מחזירים את המשפט הראשון "This is a test of the emergency broadcast system."

והפונקציה שקראה ל-`get_sentence()` יכולה להדפיס אותו.

ואז קוראים שוב ל-`get_sentence()`!

ב-`get_sentence()`, אנחנו בודקים את החיץ שוב. (זכור, החיץ הוא גלובלי אז הוא עדיין מחזיק את הנתונים מהקריאה הקודמת.)

text
Copy code
```
This is on
```

אין נקודה, אז אנחנו קוראים שוב ל-`recv(20)`, אבל הפעם אנחנו מקבלים רק 10 בתים:

text
Copy code
```
.This is only a test
```

אבל זה משפט מלא, אז אנחנו מסירים אותו מהחיץ, משאירים אותו ריק, ואז מחזירים אותו למתקשר להדפסה.

11.3.1 מה אם תקבל כמה משפטים בו-זמנית?

מה אם אני קורא ל-`recv(20)` ומקבל את זה:

text
Copy code
```
Part 1. Part 2. Part
```

ובכן, זה עדיין עובד! פונקציית `get_sentence()` תראה את הנקודה הראשונה שם, תסיר את המשפט הראשון מהחיץ כך שהוא יכיל:

text
Copy code
```
Part 2. Part
```

ואז תחזיר את Part 1..

בפעם הבאה שתקלוט את `get_sentence()`, כמו תמיד, הדבר הראשון שהיא עושה זה לבדוק אם החיץ מכיל משפט מלא. הוא מכיל! אז אנחנו מסירים אותו:

text
Copy code
```
Part
```

ומחזירים את Part 2.

בפעם הבאה שאתה קורא ל-`get_sentence()`, הוא לא יראה נקודה בחיץ, אז אין משפט שלם, אז הוא יקרא שוב ל-`recv(20)` כדי לקבל יותר נתונים.

text
Copy code
```
.Part 3. Part 4. Part 5
```

ועכשיו יש לנו משפט מלא, אז אנחנו מסירים אותו מהקדימה:

text
Copy code
```
.Part 4. Part 5
```

ומחזירים את Part 3 למתקשר. וכן הלאה.

11.4 התוכנית הגדולה

בסך הכל, אתה יכול לחשוב על האאבסטראקציה הזו כצינור מלא בנתונים. כשיש חבילה שלמה בצינור, היא נמשכת מהקדימה ומוחזרת.

אבל אם אין, הצינור מקבל יותר נתונים מאחור וממשיך לבדוק אם יש לו חבילה שלמה.

הנה כמה קוד פסאודו:

python
Copy code
```
''global buffer = b     # מיתר בתים ריק

:()function get_packet
    :while True
        if buffer מתחיל בחבילה שלמה
            חלץ את נתוני החבילה
            הסר את נתוני החבילה מהקדימה של החיץ
            החזר את נתוני החבילה

        קבל יותר נתונים

        אם כמות הנתונים שהתקבלו היא אפס בתים
            החזר אינדיקטור לסגירת החיבור

        הוסף את הנתונים שהתקבלו לחיץ
```

בפייתון, אתה יכול לחתוך את החיץ כדי להיפטר מנתוני החבילה מהקדימה.

לדוגמה, אם אתה יודע שנתוני החבילה הם 12 בתים, אתה יכול לחתוך אותם כך:

python
Copy code
```
[packet = buffer[:12    # קח את החבילה
[:buffer = buffer[12    # חתוך אותם מהקדימה
```

11.5 מחשבה

תאר את היתרונות מנקודת מבט תכנותית של הפשטת חבילות מתוך זרם נתונים.