Objektovo orientované programovanie - *pokročilí*

Učiteľ:
Ing. Jozef Wagner, PhD.

Učebnica:
https://oop.wagjo.com/

OPGP
Pokročilí 22

1. Rámcovanie správ nad TCP
2. S pevnou dĺžkou správy
3. S oddeľovacím znakom
4. S dĺžkou správy v hlavičke

Opakovanie

TCP je spoľahlivý prúd bajtov. Keď pošleme `send("Ahoj")` a potom `send(" svet")`, na druhej strane to môže prísť ako:

- "Ahoj svet" naraz,
- "Ahoj" + " svet",
- "Aho" + "j svet",
- čokoľvek iné.

TCP nevie, kde končí jedna správa.

Preto musíme na aplikačnej vrstve (nad TCP) definovať vlastný protokol – pravidlá, podľa ktorých klient a server vedia, kde začína a končí jedna správa.

Rámcovanie správ

Rámcovanie je spôsob, ako rozdelíme nekonečný TCP stream na jednotlivé správy

Existujú tri základné techniky rámcovania správ:

- Rámcovanie s **pevnou dĺžkou správy** - fixed size
- Rámcovanie s **oddeľovacím znakom** - delimiter
- Rámcovanie s **dĺžkou správy** v hlavičke - length prefix

Rámcovanie nevytvára protokol TCP, ale je to úlohou aplikačnej vrstvy

Pevná dĺžka správy – fixed size

Správy majú vopred dohodnutú pevnú dĺžku.

Prijímateľ presne vie, koľko bajtov má načítať

Väčšie správy sa musia rozdeliť

Malé správy mrhajú priestorom

Je používaný iba zriedka.

```
FIXED_SIZE = 64  # môžeš zmeniť na 32, 128, 256...

# správa nemôže končiť znakom \0, ten sa používa na padding
def send_fixed(sock: socket.socket, message: bytes):
    """Pošle správu s pevnou dĺžkou – doplní nulami ak je kratšia"""
    if len(message) > FIXED_SIZE:
        raise ValueError(f"Správa je príliš dlhá")
    padded = message.ljust(FIXED_SIZE, b'\0')   # doplní nulami
    sock.sendall(padded)

def recv_fixed(sock: socket.socket):
    """Prijme presne FIXED_SIZE bajtov (ošetruje čiastočné recv)"""
    data = b""
    while len(data) < FIXED_SIZE:
        chunk = sock.recv(FIXED_SIZE - len(data))
        if not chunk:                               # spojenie zatvorené
            return None
        data += chunk
    # odstránime padding nuly
    return data.rstrip(b'\0')
```

Oddeľovací znak - delimiter

Odosielateľ správu zakončí oddeľovacím znakom, napr. `0x00`, alebo `'\n'` (nový riadok).

Oddeľovací znak sa môže skladať aj z viacerých bajtov

Prijímateľ dáta číta, až kým mu nepríde oddeľovací znak

Prijímateľ dopredu nevie, aká veľká bude správa

V tele správy sa nesmie vyskytovať oddeľovací znak

```
DELIMITER = b'\n'

def send_delimiter(sock: socket.socket, message: bytes):
    if DELIMITER in message: raise ValueError("Delimiter v správe!")
    sock.sendall(message + DELIMITER)

def recv_delimiter(sock: socket.socket, buffer: bytearray):
    while True:
        idx = buffer.find(DELIMITER)
        if idx != -1:
            message = bytes(buffer[:idx])
            del buffer[:idx + len(DELIMITER)]
            return message
        chunk = sock.recv(4096)
        if not chunk:
            if buffer:
                message = bytes(buffer)
                return message
            return None
        buffer.extend(chunk)
```

Dĺžka správy v hlavičke

Správy s pevnou dĺžkou hlavičky, v ktorej je uložená informácia o dĺžke celej správy.

Prijímateľ tak po načítaní hlavičky vie, koľko bajtov musí ešte načítať

Veľmi bezpečné a flexibilné, najčastejšie využívaný spôsob

Komplikovanejší spôsob - najprv číta hlavičku a až potom obsah správy

Neefektívne pri prenose extrémne malých správ

Správy majú maximálnu veľkosť (v praxi napr. 4 GB)

```
def send_length(sock: socket.socket, message: bytes):
    header = struct.pack("!I", len(message))
    sock.sendall(header + message)

def recv_length(sock: socket.socket) :
    header = b""
    while len(header) < 4:
        chunk = sock.recv(4 - len(header))
        if not chunk:
            return None
        header += chunk
    length = struct.unpack("!I", header)[0]
    data = b""
    while len(data) < length:
        chunk = sock.recv(length - len(data))
        if not chunk:
            return None
        data += chunk
    return data
```