CAMADA DE ENLACE – SLIP

NESTA PRÁTICA, vamos implementar o SLIP, um protocolo de camada de enlace muito simples que funciona sobre linhas seriais. O SLIP é definido pela RFC 1055, que tem apenas 6 páginas (e isso incluindo exemplos de código em C)!

Sua implementação deve ser realizada no arquivo `slip.py`, que já veio com um esqueleto em cima do qual você vai construir o seu código.

Para testar seu código, execute `./autograde.py`. Cada um dos testes vai usar a sua implementação como uma biblioteca, verificando se ela apresenta o comportamento esperado.

Passo 1 — 2 pontos

Implemente o método `enviar` da classe `Enlace`. O envio no SLIP é muito simples: basta inserir um byte `0×C0` no começo e no fim do datagrama e enviá-lo para a linha serial chamando `self.linha_serial.enviar`.

É por meio desse byte especial que o protocolo sabe onde o quadro começa e onde termina. A rigor, só seria necessário inserir esse byte no final do quadro, mas é recomendável inseri-lo também no começo para aumentar a tolerância do protocolo a ruídos que possam ocorrer enquanto a linha estiver ociosa (leia a RFC para mais detalhes). Por isso, o teste exige que você siga essa recomendação.

Passo 2 — 2 pontos

E se o datagrama a ser enviado contiver o byte `0×C0` – como o SLIP vai saber se ele é o término de um quadro, ou se é apenas parte do datagrama? Para resolver esse problema, o SLIP exige que, quando o `0×C0` referir-se ao dado `0×C0` (em vez do término de um quadro), ele seja transmitido como uma sequência dos dois bytes `0×DB 0×DC`. Essas sequências especiais são comumente chamadas de *sequências de escape*.

Mas essa solução gera um outro problema – e se o datagrama contiver a sequência `0×DB 0×DC`? A representação continuaria ambígua. Para que esse problema não persista recursivamente e para resolvê-lo de forma relativamente eficiente, resolveu-se criar uma sequência de escape também para o byte `0×DB`, que é a `0×DB 0×DD`.

Complete sua implementação do passo anterior para lidar com as sequências de escape para os bytes `0×C0` e `0×DB`.

Passo 3 — 2 pontos

Para receber dados no SLIP, você vai enfrentar um problema muito similar ao que enfrentou no servidor de bate-papo do Trabalho 1. Os bytes podem ser recebidos pelo sistema operacional na outra ponta da linha serial de uma forma bastante imprevisível. Pode ser que você receba só um byte por vez, ou pode ser que receba vários de uma vez mas não chegue um quadro inteiro de uma vez só, ou pode acontecer até mesmo de receber o final de um quadro junto com o começo do próximo.

Implemente o método `__raw_recv` da classe `Enlace`. Ao receber um quadro completo, extraia o datagrama e chame `self.callback`, passando-o como argumento.

Além de tomar cuidado com os quadros que podem chegar *quebrados* em pedaços, siga a recomendação da RFC de **não** repassar à camada superior datagramas vazios. Eles ocorrem com muita frequência devido ao byte `0×C0` ser colocado no início e no fim do quadro. Descarte-os para melhorar a eficiência da implementação.

Passo 4 — 2 pontos

Complete sua implementação do passo anterior para receber e lidar corretamente com as sequências de escape correspondentes aos bytes `0×C0` e `0×DB`.

Passo 5 — 2 pontos

Caso o datagrama esteja mal formado e cause erros nas camadas superiores (nos protocolos IP, TCP ou mesmo na camada de aplicação), o Python irá abortar a execução da sua função `__raw_recv` em maio à chamada do *callback* e devolver o controle para o *loop* principal do asyncio. Dependendo de como você tiver implementado o tratamento de dados residuais solicitado no Passo 3, pode ser que sobre algum lixo que faça com que você nunca se livre desse datagrama mal formado.

Caso a sua implementação esteja falhando no quinto teste, modifique-a para limpar os dados do datagrama mal formado, evitando que eles sejam repassados novamente às camadas superiores. O exemplo abaixo pode ser útil.

```
try:
    self.callback(datagrama)
except:
    # ignora a exceção, mas mostra na tela
    import traceback
    traceback.print_exc()
finally:
    # faça aqui a limpeza necessária para garantir que não vão sobrar
    # pedaços do datagrama em nenhum buffer mantido por você
```

Opcional

O arquivo `exemplo_integracao.py` gruda todas as camadas implementadas até o momento, disponibilizando uma implementação completa de TCP/IP/SLIP que funciona totalmente à parte da implementação do *kernel* Linux.

O Linux também possui uma implementação de SLIP, então é possível colocar ambas as implementações para conversar, como se fossem máquinas diferentes (mesmo que estejam executando no mesmo computador). Para fazer isso, execute `./exemplo_integracao.py` e siga as instruções da tela.

Como exercício opcional, modifique o arquivo `exemplo_integracao.py` para implementar a sua camada de aplicação, assim como você talvez tenha feito no exercício opcional dos Trabalhos 2 e 3.