TP1 - Redes

O primeiro trabalho de redes foi implementar um servidor de *pokedex* com um cliente, ambos baseados em `sockets` de `C`. A implementação deveria ser feita tanto em `IPv4` quanto em `IPv6`, com validação dos nomes dos *Pokémons*.

Estratégia de implementação

Estrutura de dados

Para o armazenamento tanto do dicionário de *Pokémons* quanto dos *Pokémons* já existentes na *pokedex*, foi utilizada uma `trie` com *flags* de existência e de já adicionado.

> Devido à implementação com uma `trie`, o *Pokémon* Porygon2 foi considerado como inexistente, pois 2 não faz parte do alfabeto da `trie`.

```
int insert_into_existing(trie_node_p trie, const char* word) {
  trie_node_p cur = find_in_trie(trie, word);
  if (!cur) return -1;

  // more than maximum capacity
  if (trie->n >= MAX_N_POKEMON) return 2;

  // is not in dictionary
  if (!(cur->flags & IS_EOW_MASK)) return -1;

  // already in pokedex
  if (cur->flags & IS_IN_TRIE) return 1;

  cur->flags = cur->flags | IS_IN_TRIE;

  strcpy(cur->word, word);
  trie->n++;

  return 0;
}
```

Como pode ser observado no código da `trie`, a implementação é similar a uma `trie` normal, porém com uma *flag* extra de além de estar no dicionario já estar na *pokedex*.

Essa estrutura é interessante para o armazenamento de `strings` pois sua complexidade é $\mathcal{O}(n)$ para inserção e *lookup*, e como n é no máximo 10, o tempo fica praticamente constante.

Processamento de strings

O processamento das mensagens foi feito utilizando-se a função `strtok`, da biblioteca `strings.h` de `C`. Essa função retorna uma `string` parcial a partir de uma `string` principal que é basicamente uma `substring` até determinado caractere, na primeira chamada. Nas chamadas subsequentes, a função retorna outras `substrings` até o delimitador em questão, até quando não existem mais `substrings`, quando a função retorna `NULL`.

Essa função foi utilizada tanto para se identificar múltiplas mensagens em um mesmo `recv` quanto para identificar múltiplos *Pokémons* a serem adicionados ou removidos.

```
    while (msg) {
      kill = process_message(cdata->socket, msg);
      if (kill) break;
      msg = strtok(NULL, "\n");
    }
```

O código anterior mostra a lógica de processamento de diversas mensagens em um mesmo recover.

Servidor

O servidor foi implementado com uma *pool* de *threads*. Essa foi a parte mais complexa da implementação, dado que o manejamento de lixo de memória com *threads* persistentes é complexo em `C`.

Foi utilizado o código de referência para parsear a entrada da versão do `IP` a ser utilizado e o endereço em si.

Escolhas de implementação

Para a implementação do *handler* de mensagens, foi utilizado um vetor de ponteiros para função e um tradutor de prefixos para número, sendo assim minimizado o código na função principal.

Dificuldades de implementação

A maior dificuldade de implementação foi o tratamento de `strings` em `C`, que é relativamente chato e "burocrático". O manejamento de *threads* para múltiplos clientes também foi complexo, porém manejável.

Problemas ocorreram também após a implementação completa do TP ter que ser mudada pois a descrição do mesmo foi alterada, mudando os limites, alguns comportamentos de mensagens e suas formatações, que envolveu uma grande mudança na implementação do servidor.