Relatório da aplicação

Relatório de explicação e documentação da aplicação

Eduardo Brasil Araujo
Cesar Henrique Cícero

Como Executar a Aplicação

A aplicação, primeiramente, necessita ser compilada. No diretório do repositório existe um arquivo script de **Makefile** que é o sistema de compilação que utilizamos no projeto.

Para compilar o programa utiliza-se o comand `make build`, que compilará a aplicação utilizando 4 threads.

Porém, como a aplicação é dividida em compilações diferentes para o cliente e para o servidor, existem dois comandos no script **Makefile** que compilam separadamente a parte de cliente e a parte de servidor da aplicação.

Para compilar o binário de cliente, basta executar `make client`. Já para compilar a parte do binário do servidor, bata executar `make server`.

Porém, como estamos utilizando os mesmos arquivos para compilar os dois binários diferentes, após compilar um dos binários, faz-se necessário o uso da tag `clean` (para apagar os intermediários de compilação) ao invocar uma chamada do script para compilar tanto a parte do cliente, quanto da parte do servidor. Ex.: `make clean client`, `make clean server`.

Existe ainda a possibilidade de compilar e executar a aplicação em um mesmo comando, com as tags `run_client` e `run_server`. As mesmas regras de limpeza de intermediários se aplicam a essas tags. Ex.: `make clean run_client` e `make clean run_server`.

Quando os binários são compilados, eles são armazenados separadamente, o cliente e o servidor, respectivamente, ficam em bin/client.out e bin/server.out.

No servidor, é possível passar um único argumento, que é um inteiro que reprensenta a quantidade de jogadores que terá no jogo. Por padrão esse número é 2.

Ressalva Importante

- É essencial de que a **conexão não seja perdida** durante o decorrer da partida, pois **não existe função de reconexão** na parte do servidor, ou seja, caso a conexão à algum cliente seja perdida, o servidor encerrará suas atividades.
- Devido à problemas técnicos, optamos por não codificar a aplicação para ser rodada em sistemas **Windows**, portanto, ela funcionará **somente em sistemas linux**.

O funcionamento do jogo de Poker

Ao instanciar o objeto poker, é passado o número de jogadores que participarão do jogo. Para cada jogador é pedido o nome para identificação e é atribuído um saldo de R$ 200.000,00, além de um objeto `deck` com 52 `cartas` (ambos classes no programa).

O primeiro é jogador é escolhido para ser o *small Blind*[1]. Cada partida é iniciada com o método `newGame()`, e, inicialmente, todas as cartas volta para o baralho que é embaralhado e os jogadores que estão falidos (com `dinheiro == 0`) são expulsos da mesa e o *pot*[2] da mesa é iniciado com 0.

[1] Fica localizado à esquerda da pessoa que dá as cartas, é o jogador que dá a menor aposta da mesa.
[2] O pot é onde ficam armazenadas as apostas de todos os jogadores da mesa.

A mesa recebe 5 cartas aleatórias, sem repetição, e cada jogador recebe 2 (duas) cartas cada. Cada jogador pode ver as próprias cartas e, inicialmente, apenas as 3 primeiras cartas da mesa.

O poker consiste em 3 rodadas de apostas. Na primeira, o *small Blind* é obrigado à apostar metade da aposta mínima, e o próximo jogador, em sequência, é obrigado a apostar a aposta mínima; todas as apostas vão para o **pot**. Após a aposta inicial, para cada jogador é oferecido 3 opções:

- Fold (abandonar a partida)
- Call (cobrir as apostas ateriores)
- Raise (aumentar a aposta pela quantidade que será especificada pelo jogador, ou zero para apostar tudo)

Caso os outros jogadores abandonem a partida, o jogador que permanecer vence a partida e recebe todo o pot. Depois que todos fizerem suas apostas, 1 nova carta da mesa é revelada e outra rodada de apostas segue.

Após a última carta da mesa ser revelada, e a última rodada de apostas ocorrer, todos os jogadores que continuarem na partida revelam suas mãos e o jogador que conseguir formar com a mesa a melhor mão (maior rank) vence a partida e recebe todo o dinheiro no *pot*; caso haja empate, o *pot* é dividido.

Para cada nova rodada é chamada a função de membro `newGame()` no objeto `Poker`.

Como rodar

O cliente e o servidor são aplicações diferentes, para compilar um ou o outro é necessário definir o define `CLIENT` nos arquivos de código fonte do **C++**. No Makefile isso já é realizado automaticamente.

Cada jogador deverá rodar o programa compilado com as diretrizes de cliente. Ao ser iniciado, o programa pedirá o nome do **jogador** que deverá ser escrito e enviado pelo terminal da aplicação. Após todos os jogadores se conectaram ao servidor as rodadas se iniciam.

O servidor mandará mensagens para cada jogaor atualizando-os de suas cartas e das ações de outros jogadores. No turno do jogador específico será pedido para ele digitar um número de tomada de ação:

- 0 - para sair da rodada
- 1 - pagar o valor da aposta
- 2 - aumentar a aposta

Caso ele escolha a opção 2 (aumentar a aposta), será pedido qual o valor que ele deseja aumentar (0 seria apostar todo o dinheiro). Ao final de cada rodada, o jogador que tiver a melhor mão leva o pot e uma nova rodada é iniciada.

O jogo no total é composto por 3 rodadas, e todas elas finalizam somente quando obtivermos um único jogador vencedor na rodada.

O que mais poderia ser feito para implementar:

Uma possível implementação seria que os jogadores pudessem se conectar a qualquer instante do jogo, não só no início. Outra importante implementação seria salvar as informações de dinheiro de cada jogador em um arquivo, para que, caso o mesmo jogador se desconectasse e retornasse para a partida, ele teria o dinheiro de antes da conexão.

Dificuldades encontradas

Calcular a melhor mão foi uma busca grande, mas achamos um algorítimo que registrou o ranking de cada grupo de 7 cartas em lookup table para facilitar o calculo da melhor mão.

Uma outra grande dificuldade foi a implementação de sockets no sistema Windows, o que infelizmente acarretou no abandono da implementação da aplicação na plataforma, pois, percebeu-se que na plataforma Linux a implementação de sockets foi bem mais sucinta e previsível, além de consistente.

Outro dificuldade é o fato de que quando o servidor enviava varias mensagens seguidas (send) o cliente recebia (recv) as mensagens juntas no mesmo char*, então era nescessário separar as mensagens posteriormente