

Trabalho Prático 1 - Servidor de Exploração de Labirintos

Thaís Ferreira da Silva - 2021092571

Novembro 2024

1 Introdução

Os labirintos, estruturas fascinantes que desafiam a mente humana há séculos, encontram aplicação em diversas áreas da computação, como jogos, algoritmos de busca e inteligência artificial. Este trabalho prático propõe o desenvolvimento de um sistema de exploração de labirintos, composto por um cliente e um servidor que se comunicam via sockets em C. O servidor gerencia o estado do jogo, enquanto o cliente permite a interação do jogador, enviando comandos e recebendo informações. Esta aplicação não só reforça o aprendizado em redes, mas também introduz conceitos de interação cliente-servidor, processamento em tempo real e gerenciamento de estados dinâmicos.

2 Implementação

O sistema foi desenvolvido em C, utilizando a interface POSIX de sockets. A estrutura segue um modelo cliente-servidor, no qual o cliente envia comandos (como iniciar o jogo, mover-se no labirinto ou solicitar dicas) e o servidor processa as solicitações, mantendo o estado atualizado e retornando respostas.

É importante reforçar que toda a base da comunicação cliente-servidor foi feita de acordo com a implementação deixada nas referências do trabalho.

Temos no total 4 tipos de arquivos desenvolvidos:

- common: contendo as funções utilizadas em comum pelo servidor e pelo cliente
- client: contendo as funções relacionadas com o cliente
- server: contendo as funções relacionadas com o servidor
- Makefile: contendo as instruções para compilar automaticamente os outros arquivos utilizando 'make' para criar e 'make clean' para deletar

3 Desafios, dificuldades e imprevistos do projeto

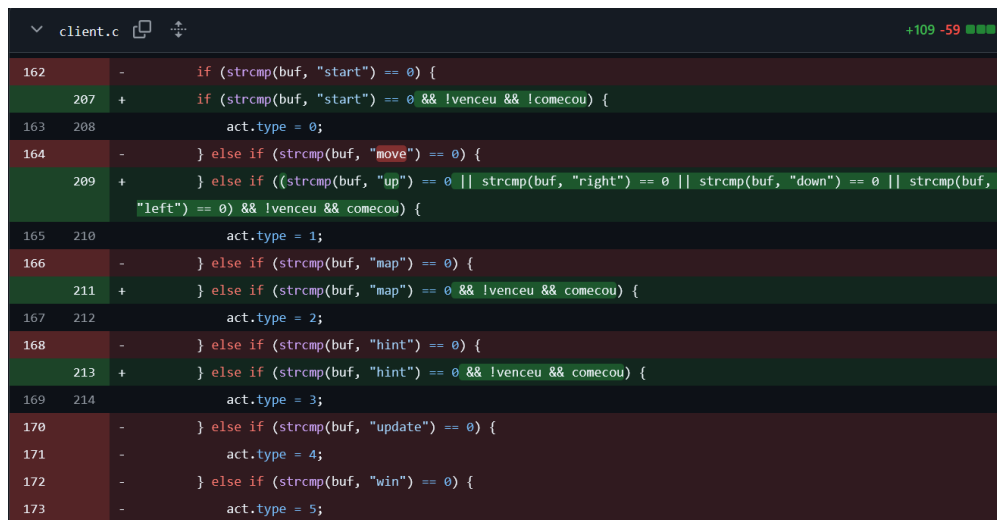
Durante o desenvolvimento do projeto, surgiram diversos desafios e imprevistos que exigiram ajustes no planejamento e soluções criativas. Neste capítulo, serão apresentados os principais obstáculos enfrentados e como eles foram superados, destacando o aprendizado adquirido ao longo do processo.

3.1 Desenvolvimento do cliente e servidor

Acredito que vários dos desafios não encontrados ao longo desse projeto foram devidos a uma lista de tarefas que construir previamente ao desenvolvimento em si do cliente e do servidor. Ela me serviu como guia inicial, dos principais passos a serem realizados para um bom funcionamento do labirinto, desta forma, os desafios a serem explicado a seguir estão na ordem das ações apresentadas na documentação e de acordo com a minha lista de tarefas desenvolvida.

O primeiro desafio encontrado foi relacionado a construção básica do cliente e servidor com o protocolo TCP. Seguindo os passos do professor Ítalo Cunha [1], montei a base necessária, e após isso fiz as alterações necessárias para que a comunicação fosse realizada utilizando a struct action. Como fazia um tempinho que não programava em C, controlar os buffers foi esse primeiro desafio, mas até que vem simples de se resolver.

O segundo desafio foi relacionado aos comando básicos a serem desenvolvidos. Optei por ir desenvolvendo o cliente e o servidor ao mesmo tempo, sendo assim capaz de testar o funcionamento do sistema como um todo a medida que o código foi evoluindo. Comecei pelo "start", pelo "map", e pelo "exit" que me pareceram mais simples, onde esbarrei com a dificuldade de trabalhar com o vetor e o mapa de tamanho fixo mas que poderiam possui dimensões menores, e isso dependeria da implementação de cada um. O start foi até que tranquilo de implementar, onde temos o armazenamento do input inicial do tabuleiro, e a criação de duas matrizes para desenvolvimento do jogo, uma para controle interno, e uma para enviar para o cliente de acordo com o protocolo. Já o vetor moves, foi o mais fácil pois para mim bastou ler até encontrar o primeiro 0. Já para o board, percebi que cada um estava preenchendo o "resto" do tabuleiro com algum tipo de carácter diferente, então optei por preencher no meu servidor com 8's (valor foram do esperado), e no cliente simplesmente ignorar tudo que fosse diferente do padrão esperado. O exit já havia sido desenvolvido previamente na criação da comunicação cliente servidor, então só adequuei ao comando do jogador.



```
162 - if (strcmp(buf, "start") == 0) {
207 + if (strcmp(buf, "start") == 0 && !venceu && !comecou) {
163 208     act.type = 0;
164 - } else if (strcmp(buf, "move") == 0) {
209 + } else if ((strcmp(buf, "up") == 0 || strcmp(buf, "right") == 0 || strcmp(buf, "down") == 0 || strcmp(buf,
    "left") == 0) && !venceu && !comecou) {
165 210     act.type = 1;
166 - } else if (strcmp(buf, "map") == 0) {
211 + } else if (strcmp(buf, "map") == 0 && !venceu && !comecou) {
167 212     act.type = 2;
168 - } else if (strcmp(buf, "hint") == 0) {
213 + } else if (strcmp(buf, "hint") == 0 && !venceu && !comecou) {
169 214     act.type = 3;
170 - } else if (strcmp(buf, "update") == 0) {
171 -     act.type = 4;
172 - } else if (strcmp(buf, "win") == 0) {
173 -     act.type = 5;
```

Figure 1: Modificações no move, e novas condicionais

Depois avancei para o "move", que me deu uma baita dor de cabeça (Figura 1). No início, acreditei que antes de receber a direção do movimento escolhido pelo jogador, tínhamos o comando "move", e comecei a desenvolver o código com essa ideia em mente. No entanto a medida que fui relendo a documentação realizei a primeira modificação, pois não existe um comando chamado

"move" e sim as direções de movimentação "up", "right", "down" e "left", o que não foi complexo de arrumar, mas aumentou consideravelmente o tamanho do código. Depois vieram os pequenos tratamentos de erros relacionados a isso, pois era necessário distinguir qualquer coisa diferente das ações disponíveis, e reportar o erro adequado.

Também descobri que ação "update" não era uma ação separada, mas sim uma forma de retorno do servidor atualizando o cliente sobre algo específico do desenvolvimento do jogo (Figura 2). Por isso foi necessário adequar todos os comando anteriores a essa regra preestabelecida para lidar com o imprevisto encontrado. Para isso no cliente separei todas as ações em duas parte, uma chamada commands para as ações pre comunicação com o servidor, e o update que lida com as respostas do servidor. Além disso, ajustei o servidor para retornar o type de update ao invés do type recebido do cliente.

```
273 +         if (act.type == 4 || act.type == 6) {
274 +             update(&act, comando_enviado);
275 +             if(comando_enviado == 0 || comando_enviado == 1 || comando_enviado == 6){
276 +                 for (int i = 0; i < 4; i++){
277 +                     movimentos_valido[i] = act.moves[i];
278 +                 }
279 +             }
280 +         }
281 +         if (act.type == 5) {
282 +             win(&act);
283 +         }
```

Figure 2: Modificação para utilizar o update

Após a atualização do "update" desenvolvi o "win" e "reset" que me fizeram adicionar novas variáveis globais para lidar com o imprevisto das condições de fim de jogo. Essas variáveis foram responsáveis por controlar quais ações o jogador poderia ter após encontrar a saída do labirinto, ou simplesmente desistir do jogo no meio da partida. Desta forma adicionei as variáveis venceu e começou no servidor e no cliente, e adicionei essas condições no tratamento de erros. Essa pequena adição já criou vários problemas em relação aos outros comandos, principalmente em relação ao start, que por possuir o type=0, toda vez que um jogador chegasse nesse ponto, e ficasse em loop até receber uma entrada válida, por conta da limpeza do buffer (colando tudo como 0 no vetor), qualquer coisa digitada após a vitória era tratada como "start".

Esse fim de jogo também gerou um grande impacto no tamanho do código, tendo em vista que adicionei algumas condicionais para tratar desse loop de fim de jogo, e o lançamento dos erros necessários.

Por fim, implementei o "hint" utilizando uma bfs. A maior dificuldade aqui não foi a logica por trás de encontrar o caminho, já que já vimos o algoritmo em disciplinas anteriores, mas passei por um pequeno problema onde as vezes o algoritmo entrava em loop, indo e voltando para o mesmo local. Algo que só veio a parar de ocorrer quando modifiquei a ordem das direções exploradas ao longo do jogo.

3.2 Testes realizados com cliente e servidor de outra pessoa

Para mim a parte mais importante dos testes realizados foi a de fazer a conexão ao servidor ou cliente de outra pessoa. Foi desta forma que percebi que o meu desenvolvimento não estava de acordo com o protocolo estabelecido. Felizmente a maior parte do meu desenvolvimento estava funcionando apresentando apenas alguns pequenos imprevistos a serem solucionado.

Nesta parte precisei realizar ajustes no comportamento do meu servidor ao desconectar um cliente, pois ele não estava reiniciando os parâmetros do jogo para receber um novo jogador diferente.

Além disso, me avisaram que quando o meu cliente solicitava map, o padrão da impressão na tela não estava de acordo com o esperado, pois ao invés de separar os valores com o tab, acabei desenvolvendo utilizando espaço simples.

Por fim, voltei a ter o problema com o comando start ao receber entradas inválidas, e precisei ajustar, colocando mais uma condicional de validação. Alguns bugs do start estão nas figuras abaixo (Figura 3a, 3b, 3c).

```
You escaped!
> _ # _ _ _
_ _ _ # # #
# _ # # # #
# _ _ _ _ #
# # # # _ # X
# # _ _ _ # _
# # _ # _ # _
# # _ _ _ _ start
error: start the game first
reset
starting new game
Possible moves: right, down.
start
error: start the game first
Erro ao receber resposta: Success
```

(a) Erro ao dar start após uma vitória

```
right
error: start the game first
start
right
error: start the game first
```

(b) Erro de não reconhecimento do start

217	-	printf("error: start the game first\n");
217	+	if(comecou == 0){
218	+	printf("error: start the game first\n");
219	+	} else if (venceu){
220	+	continue;
221	+	}

(c) Uma das manutenções do start

Figure 3: Imprevistos do start

4 Conclusão

Por fim, gostaria de concluir que desenvolvimento do trabalho prático sobre o protocolo TCP foi bem interessante, me ajudando a consolidar o conteúdo visto em sala de aula.

Acredito que o motivo de ter encontrado pouquíssimos erros na parte de realizar o teste com o código desenvolvido por outros alunos, seja devido ao controle de versões realizados pelo git, e a lista de tarefas a serem desenvolvidas.

Uma melhoria que poderia ter sido feita, mas que fica de lição para o próximo trabalho, seria dividir as funções entre diversos arquivos. Ao invés de lidar apenas com 3 arquivos principais, contendo os métodos em comum e os métodos específicos de cada parte, acredito que o ideal seria pelo menos ter mais 2 arquivos, separando o main do servidor e do cliente, de suas funções auxiliares. Mas quando vi já era tarde de mais para realizar novas alterações, tendo em vista o prazo de entrega e outros trabalhos a serem desenvolvidas para a semana seguinte.

References

- [1] ÍTALO CUNHA. Introdução à programação em redes. <https://www.youtube.com/playlist?list=PLyrHOCFXIM5Wzmbv-1C-qvoBejsa803Qk>, Ago 9, 2020.