



Bomberman

Trabalho prático – meta 3

Sistemas Operativos

Amadeus Alves Nº 21260399

Filipe Silva Nº 21260367

# Índice

[Índice 2](#_Toc503766273)

[Introdução 3](#_Toc503766274)

[Ficheiros 3](#_Toc503766275)

[Estruturas de Dados 4](#_Toc503766276)

[Estruturas de Dados 4](#_Toc503766277)

[Código 5](#_Toc503766278)

[Sinais 5](#_Toc503766279)

[Arquitectura e Estratégia de Named Pipes 6](#_Toc503766280)

[Named pipes criados: 6](#_Toc503766281)

[Threads 7](#_Toc503766282)

[Threads usadas no servidor: 7](#_Toc503766283)

[Funcionalidades Realizadas 7](#_Toc503766284)

[Validação 9](#_Toc503766285)

[Testes realizados: 9](#_Toc503766286)

[Comportamentos anómalos conhecidos: 9](#_Toc503766287)

# Introdução

Este trabalho tem como objetivo implementar o jogo “Bomberman” para o ambiente de consola. O jogo será implementado em linguagem C num sistema UNIX.

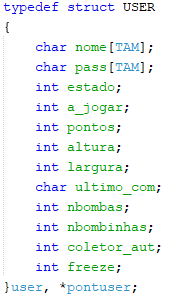
Neste documento irão ser apresentados os objetivos da meta 2 (todos os requisitos da meta 1; comunicação com named pipes a funcionar, toda a gestão dos clientes como: expulsar um jogador, ver jogadores online e terminar a sessão a todos os utilizadores quando o servidor é desligado; e login dos jogadores a funcionar), para além das alterações feitas em relação à meta anterior.

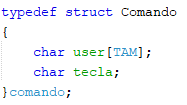
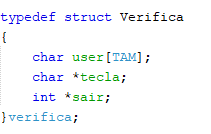
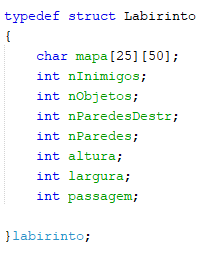
# Ficheiros

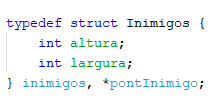
O trabalho está dividido em 4 ficheiros: servidor.c, que contém todo o código referente ao servidor do jogo; users.txt que vai guardar os utilizadores registados pelo administrador; cliente.c, que tem o código que os clientes necessitam para comunicar com o servidor; e por fim o header.h que contém as estruturas e bibliotecas utilizadas entre os dois programas.

# Estruturas de Dados

# Estruturas de Dados



* Estrutura USER :
* nome: Nome do jogador.
* pass: Password do jogador.
* estado: 0 caso esteja off ou 1 caso esteja conectado ao servidor(podendo ou não estar a jogar).
* a\_jogar:0 caso não esteja dentro do jogo e 1 caso esteja a jogar.
* pontos: Numero de pontos do jogador adquiridos no jogo atual;
* altura e largura: Posição atual do jogador no mapa.
* ultimo\_com: Guarda o ultimo comando de movimento do jogador.
* nbombas: Indica o numero de bombas atuais do jogador, começando com 2.
* nbombinhas: Indica o numero de bombinhas atuais do jogador, começando com 3.
* coletor\_auto:Apanha automaticamente 5 objetos quando adquirido no mapa.
* Freeze: Congela os inimigos por 5 segundos quando adquirido no mapa.
* Estrutura comando: Guarda o nome do ‘user’ e a ‘tecla ‘pressionada durante o jogo.
* Estrutura Verifica: Guarda o nome do ‘user’, a ‘tecla ‘utilizada e ‘sair’ para o utilizador sair do jogo caso tenha morrido.
* Estrutura labirinto:
* mapa: tem como função definir o tamanho máximo do mapa.
* nInimigos: Define o numero de inimigos no mapa.
* nObjetos: Define o numero de objetos no mapa.
* nParedesDestr: Define o numero de paredes destrutíveis.
* nParedes: Define o numero de paredes fixas.
* altura: Altura do labirinto.
* largura: largura do labirinto
* passagem: verifica se é possível concluir o jogo ou não.



# Código

## Sinais

Apenas implementamos um sinal, o SIGUSR1, o objetivo é terminar o programa. É utilizado quando o utilizador escreve “shutdown”. Apenas o servidor usa este sinal para comunicar consigo próprio.

Quando é recebido este sinal, o servidor encerra.

# Arquitectura e Estratégia de Named Pipes

O objetivo dos named pipes é estabelecer uma comunicação entre os clientes e o servidor.

Os clientes escrevem a sua informação na estrutura “mensagem” a partir do named pipe “fifologin” criado pelo servidor no inicio da sua execução, assim todos os clientes comunicam para o servidor a partir do mesmo named pipe sendo que o servidor apenas lê este named pipe.

O servidor recebe os dados dos utilizadores e para comunicar com estes cria um named pipe individual para cada utilizador atribuindo o nome do user, ou seja, o servidor escreve a mensagem a enviar a cada utilizador e estes apenas leem a mensagem recebida.

Fifo(cliente1)

fifologin

Fifo(cliente2)

## Named pipes criados:

“fifologin” – é criado pelo servidor, com o objetivo de o cliente saber que o servidor está online e este poder enviar informações com o formato da estrutura “mensagem”. É escrito por clientes e lido pelo servidor;

“fifo\_comandos” – é criado pelo servidor, serve para receber todos os comandos de todos os utilizadores, com o formato da estrutura comando. É escrito por clientes e lido pelo servidor;

Named pipe com o nome do utilizador, por exemplo “Pedro” – é utilizado para enviar mensagens aos clientes, como, se entrou ou não no cliente ou se o jogo já está cheio de jogadores (mais de 20). É escrito por clientes e lido pelo servidor;

Named pipe chamado “labirinto\_” seguido de o nome de um utilizador, por exemplo: “labirinto\_Pedro” – é criado pelos clientes, recebe informação com o formato da estrutura “labirinto”. Serve para manter o ecrã dos jogadores atualizados. É escrito pelo servidor e lido pelos clientes.

# Threads

## Threads usadas no servidor:

“recebe” – serve para receber mensagens dos clientes a partir do “fifologin”. Existe porque a qualquer momento um novo cliente pode crer aceder ao servidor, entrar no jogo, sair do servidor e verificar se o jogo já está cheio de jogadores. Num ciclo infinito, lê o “fifologin”, esperando que algum cliente tente entrar ou sair do servidor, e, depois de verificar que existe slot, entrar no jogo.

“envia\_lab” - serve para dar a posição inicial a cada jogador e adicionar esse mesmo jogador ao vetor de jogadores a jogar. Para além disso, caso um jogador abandone a partida vai alterar também as coordenadas desse jogador que saiu para [0,0]. Quando a variável do user “a\_jogar” fica a “1”, a função percorre o mapa e dá uma posição vazia a esse jogar.

“comandos” – serve para, a todo o instante ler os comandos dos vários jogadores, e aplica-los. Existe porque o servidor não pode “parar” para ler cada comando de cada jogador.

# Funcionalidades Realizadas

|  |  |
| --- | --- |
| Funcionalidades | Comentários: |
| Requisitos da meta 1 | Tudo implementado: estruturas, informação dos clientes, armazenamento dos pares username/password e leitura e validação (sintaxe) dos comandos do servidor.  Conclusão: plenamente cumprido. |
| Comunicação com named pipes | Os clientes comunicam e recebem respostas do servidor.  Conclusão: plenamente cumprido. |
| Gestão dos clientes | O servidor lê os utilizadores do ficheiro users.txt, expulsa jogadores online, dá uma lista dos jogadores online, envia mensagens aos clientes aquando do login/tentativa do login.  Conclusão: plenamente cumprido. |
| Login dos jogadores | Os jogadores podem fazer login no servidor caso este esteja online.  Conclusão: plenamente cumprido. |

# Validação

## Testes realizados:

Iniciar o servidor sem o nome do ficheiro de texto;

Inicializar o servidor com e sem utilizadores guardados no ficheiro users.txt;

Adicionar novos utilizadores com o servidor;

Fazer login com o servidor desligado;

Fazer login com nome e password corretas;

Fazer login com nome correto e password incorreta;

Fazer login com nome e password erradas;

Fazer login de um utilizador que já está online;

Expulsar jogadores ativos do servidor;

Tentar expulsar jogadores offline;

Iniciar o servidor, criar um novo utilizador e fazer login do mesmo;

Mostrar a lista de utilizadores online;

Encerramento do servidor.

## Comportamentos anómalos conhecidos:

Caso seja adicionado um novo jogador a partir do servidor, quando um cliente entra com essas credencias, ao iniciar o jogo, o mapa não inicia, a menos que entre outro cliente que foi registado manualmente no ficheiro de texto.

Os inimigos quando vão para a posição do jogador não o matam, no entanto, o jogador morre se for para a posição do inimigo.