Instituto Superior de Engenharia de Coimbra

Unidade Curricular de Sistemas Operativos

Turma P3 - Ano Lectivo 2017/2018

Trabalho Prático - Meta 1

Bomberman

António Daniel Neves Faneca nº21250021

Ricardo Jorge Pinho Marques nº21250193

Índice

1. Introdução

Para a Unidade Curricular de **Sistemas Operativos** foi proposto aos alunos a realização do jogo **Bomberman** em modo multijogador com uma interface em modo-texto em que todos os jogadores estão ligados à mesma máquina UNIX.

Este projeto tem como guia um enunciado onde estão explícitos todos os requisitos necessários para a obtenção da nota máxima, assim como um breve sumário das regras do jogo.

Este relatório consiste somente na abordagem do grupo à primeira meta, onde nos é proposto a produção de estruturas de dados com características particulares, tanto do labirinto como dos clientes. A validação dos comandos do servidor é também um ponto importante desta meta.

O projeto é realizado em grupos de dois elementos.

2. Estruturas de Dados

As estruturas de dados estão incluídas no header file do projeto (*header.h*).

2.1. Labirinto

Na elaboração desta estrutura de dados entendemos que seriam necessário três membros:

- **maze -** Uma matriz bidimensional que guarda a posição todos os caracteres do mapa de jogo;
- **numObjetosPontos** - Variável que diz o número inteiro de objetos de pontuação que o mapa tem durante o jogo. Segundo as regras quando este for 0, a saída do labirinto abre e o jogador passa ao próximo nível;
- **numObjetosDest** - Variável que dá conta do número inteiro de objetos por destruir que ainda existem no mapa;

Numa futura meta, este ponto poderá ser revisto de forma a que a estrutura se aproxime mais do pretendido, visto que não vamos cobrir 100% dos requisitos de imediato.

```
typedef struct Labirinto {
    char maze[20][30];
    int numObjetosPontos;
    int numObjetosDest;
} labirinto;
```

2.2. Jogador

No decorrer da composição deste trabalho, o grupo chegou a um consenso de colocar quatro membros nesta estrutura de dados:

- **username** - Array de caracteres onde vai ser guardado o username do jogador;
- **password** - Array de caracteres onde vai ser guardada a password do jogador;
- **online** - Variável inteira, que varia entre 0 e 1, que define se o jogador está ou não online. Como exemplo muito rápido se o utilizador levar "*kick*" esta variável terá de ser alterada de imediato;
- **pontuação** - Variável inteira que vai guardar a pontuação do jogador;

```
typedef struct Jogador {
    char username[100];
    char password[100];
    int online;
    int pontuacao;
} jogador;
```

3. Leitura e Validação de Comandos

A validação dos comandos é feita entre a função main e o ficheiro processamento.c.

3.1. Processamento de comandos

Para o método de processamento de comandos é necessário em primeira instância apanhar tudo o que o cliente vai inserir na linha de comandos. Para isso é necessária uma variável ponteiro tipo caractere que vai guardar em memória a totalidade do comando inserido pelo cliente através da função *fgets().* É necessário também outra variável ponteiro para ponteiro, do tipo caractere, que futuramente armazenará as palavras dessa mesma string dividida em posições (com recurso à função *strtok()* da biblioteca *string.h*).

A variável ponteiro do tipo caractere vai ser igualada ao retorno de uma função de nome *processaComando()* que vai passar como parâmetros a variável que contém a totalidade da string e o endereço de uma variável tipo inteiro inicializada anteriormente a 0 que nos vai indicar o número de palavras da string.

Na função *processaComando()* são iniciadas três variáveis de grande importância:

- Variável ponteiro para ponteiro do tipo caractere, que vai ser o nosso "array de strings", iniciada a NULL que se espera no final da função seja retornada já com uma perfeita divisão da string principal por espaços;
- Variável ponteiro do tipo caractere igualada à função *strtok()* que tem de parâmetros, primeiro a totalidade da string inserida pelo cliente na linha de comandos em segundo o seu delimitador, que no neste caso é um espaço;
- Variável tipo inteiro inicializada a 0 que vai contar o número de palavras existente na string;

De seguida temos um ciclo *while()* que vai estar a correr sempre exista sempre mais uma palavra, ou seja sempre que exista um delimitador espaço na string. Caso isto se verifique é alocada nova memória para a nova palavra do array, através da função *realloc()*. Agarramos na palavra e igualamos-la à primeira posição do nosso array de strings.

Quando não existirem mais delimitadores é retornado o nosso array de strings para a função main, onde vamos comparar sempre o primeiro elemento do nosso array de strings com o nome do comando e se o número de palavras bate certo com a sintaxe do comando.

```
char ** processaComando(char *comando, int *tamCMD) {

    char ** cmd = NULL;
    char *  p = strtok(comando, " ");
    int n_espacos = 0;
    int i = 0;


    while (p) {     /* divide a string em palavras */
        cmd = realloc(cmd, sizeof(char*) * ++n_espacos);

        if (cmd == NULL)
            exit(-1); /* se alocação de memória falhar */

        cmd[n_espacos - 1] = p;
        p = strtok(NULL, " ");
    }

    char lastWord[25];
    char lastWord2[25];
    strcpy(lastWord, cmd[n_espacos -1]);

    for(i=0;i < strlen(lastWord) - 1; i++){
        lastWord2[i] = lastWord[i];
    }
    strcpy(cmd[n_espacos - 1], lastWord2);
    *tamCMD = n_espacos;

    return cmd;
}
```

Em caso positivo são chamadas funções que tentam preencher todos os requisitos necessários do comando em caso negativo é apresentada uma mensagem de erro de sintaxe.

3.1.1. Vetor dinâmico de Jogadores

Ao criar um vetor dinâmico de tipo *"jogador"* necessitamos de usar a função "*contaPlayers()",* para aceder ao ficheiro de texto que contém os logs dos jogadores, que retorna o total de jogadores do ficheiro de texto. Cada jogador tem quatro propriedades sendo estas, username, password, pontuação e um número inteiro que nos indica se o jogador está online ou não.

De seguida, alocamos para o vetor dinâmico memória correspondente ao número total de jogadores, multiplicado pelo tamanho de uma estrutura de jogador em memória, usando a função *malloc()*.

Agora que temos o vetor dinâmico, criamos a função *"buscaLogs()"* para o preencher, que passa como

```
void buscaLogs(jogador *v) {

    int k = 0;
    FILE *f = fopen("logs.txt", "r");
    if (f == NULL) {
        printf( "Erro a abrir ficheiro2.");
        return 1;
    }
    while (!feof(f)) {
        fscanf(f, "%s", v[k].username);
        fscanf(f, "%s", v[k].password);
        fscanf(f, "%d", &v[k].online);
        fscanf(f, "%d", &v[k].pontuacao);
        k++;
    }
    fclose(f);
}
```

```
jogador *v;
int total = contaPlayers();
v = malloc(sizeof(jogador)*total);
buscaLogs(v);
```

argumento o vetor dinâmico e que é uma função que tem por base ler o ficheiro até ao fim e preencher o vetor dinâmico com os dados encontrados.

3.1.2. USERS

Consequentemente à inserção do comando "*users*" na função main (sem mais nenhum argumento, pois aí apareceria uma mensagem de erro de sintaxe) é chamada uma função “users()” que recebe como parâmetros o vetor dinâmico dos jogadores e uma variável inteira com o número total de jogadores existentes no mesmo vetor dinâmico.

Dentro de um ciclo que imprime todos os *usernames* é imposta a condição de que o seu nome só é imprimido se este estiver online. Esta verificação é imposta pelo enunciado.

São imprimidos assim todos os utilizadores online.

```
void users(jogador *v, int conta) {

    int k = 0;
    if (conta == 0) {
        printf("Nao existem Users a jogar\n");
    }
    printf("USERS ONLINE: \n");
    for (k = 0; k < conta; k++) {
        if(v[k].online){
            printf("%s\n", v[k].username);
        }
    }
}
```

```
if (strcmp(cmd[0], "add") ==0) {
    if (tamCMD == 3) {
        v = add(v, cmd, &total);
    }
    else {
        printf("Erro de Sintaxe. <add 'username' 'password'>\n");
    }
}
```

3.1.3. ADD

O comando "add" precedido de dois argumentos (*username e password*) dá-nos a possibilidade de inserir um novo jogador no sistema de jogo através da função "*add()*" do tipo jogador que recebe como parâmetros o vetor dinâmico, o array de strings que é detentor da totalidade dos argumentos inseridos na linha de comandos e uma variável inteira por ponteiro que representa o total de jogadores do vetor dinâmico.

A primeira condição que temos que observar é a de que não existe mais nenhum *username* igual ao que foi inserido, portanto é feita uma comparação um a um do *username* inserido com o *username* de todos os jogadores inseridos no vetor dinâmico. Se o username já for existente é retornado de imediato o vetor dinâmico sem alterações.

Caso contrário, é criado um vetor dinâmico temporário com o conteúdo do vetor dinâmico principal com mais um espaço onde vai ser inserido o novo jogador. Neste processo usamos as funções strings de cópia tanto para o username como a password e colocamos o online a pontuação a 0, o que nos indica que o jogador tem a conta criada, mas ainda não está a jogar.

De seguida removemos o ficheiro de texto que contêm a informação sobre os jogadores (*Logs.txt*) e criamos um novo com o mesmo nome e passamos a informação atualizada guardada no vetor dinâmico, tudo isto para evitar o risco de sobreposição de informação ao qual os ficheiros de texto estão sujeitos.

Por último, é retornado o vetor dinâmico temporário que por sua vez assume o lugar de vetor dinâmico principal.

```
jogador* add(jogador* v, char *cmd[], int *conta) {

    int i = 0;
    int flag = 0;
    char lastWord[25];

    for (i = 0; i < *conta; i++) {
        if (strcmp(cmd[1], v[i].username) == 0) {
            printf("Username existente!!\n");
            flag = 1;
        }
    }

    strcpy(lastWord, cmd[2]);

    if (flag == 0) {
        *conta = *conta + 1;
        jogador *temp = realloc(v, sizeof(jogador)*(*conta));
        printf("Adicionado User No. %d --> %s\n", *conta, cmd[1]);
        strcpy(temp[*conta - 1].username, cmd[1]);
        strcpy(temp[*conta - 1].password, lastWord);
        temp[*conta-1].online = 0;
        temp[*conta-1].pontuacao = 0;

        remove("logs.txt");

        FILE * f = fopen("logs.txt", "w");
        for(i = 0; i < *conta ; i++){
            fprintf(f, "%s\n", temp[i].username);
            fprintf(f, "%s\n", temp[i].password);
            fprintf(f, "%d\n", temp[i].online);
            fprintf(f, "%d\n", temp[i].pontuacao);
        }

        fclose(f);
        return temp;
    }
    else {
        return v;
    }
    return v;
}
```

3.1.4. KICK

Para a validação do comando "*kick*" é necessário também, como segundo argumento na linha de comandos, um username de jogador que esteja em jogo.

Foi criado então uma função "*kick()*" do tipo jogador que recebe como parâmetros o vetor dinâmico que contém toda a informação sobre os jogadores, o array de strings detentor do conteúdo da linha de comandos e uma variável de tipo inteiro com o número de jogadores inseridos no vetor dinâmico.

A função começa por fazer um ciclo que percorre todos os usernames dos jogadores e compara-os ao segundo argumento anteriormente introduzido na linha de comandos agora presente na posição 1 do array "*cmd*". Se esta comparação for positiva vamos confirmar se o utilizador está realmente em jogo ou não, e caso esteja alteramos o valor da variável "**online**", passamos a mensagem de que o jogador levou "*kick*" e retornamos o vetor dinâmico. Senão nada é alterado, imprimimos a mensagem de que o username existe mas está offline e retornamos o vetor dinâmico sem alterações.

Se não existir retornos no ciclo, temos no fim da função a indicação de que o utilizador que era procurado não existe e devolvemos o vetor dinâmico sem quaisquer alterações.

```
else if (strcmp(cmd[0], "kick") == 0) {
    if (tamCMD == 2){
        v = kick(v, cmd, total);
    }else{
        printf("Erro de Sintaxe. kick <username>\n");
    }
}
```

```
jogador* kick(jogador* v, char *cmd[], int conta){

    int i = 0;

    for(i = 0; i < conta ; i++){
        if(strcmp(cmd[1], v[i].username)==0 && v[i].online == 1){
            v[i].online = 0;
            printf("%s --> Foi Kickado.\n" , v[i].username);
            return v;
        }

        if(strcmp(cmd[1], v[i].username) == 0 && v[i].online == 0){
            printf("%s --> O utilizador a kickar está offline\n", v[i].username);
            return v;
            }
        }

        printf("Não existe nenhum user com o nome de %s\n", cmd[1]);
        return v;
}
```

3.1.5. GAME

O comando "*game*" tem o objetivo de ser um processo informativo sobre o jogo que está a decorrer, por isso assim que ativado é passado como parâmetro na função *"gameInfo()"* o vetor dinâmico que contém os dados dos jogadores. Primeira versão básica (e incompleta) deste comando já criada. A ser desenvolvido em mais detalhe nas próximas fases.

```
void gameInfo(jogador *v, int conta){

    int i = 0;
    printf("-----------GAMEINFO-----------\n");
    for(i = 0; i < conta ; i++){
        if(v[i].online == 1){
            printf("USER:  %-15s", v[i].username);
            printf("SCORE: %d\n", v[i].pontuacao);
        }
    }
}
```

3.1.6. MAP

Reconhecimento e Validação da Sintaxe realizados. O desenvolvimento deste módulo fica agendado para as próximas fases de desenvolvimento.

3.1.7. SHUTDOWN

A introdução do comando "*shutdown*" para além de libertar a memória dos vetores dinâmicos vai quebrar o ciclo principal da função main encerrando assim o servidor.

```
if (strcmp(cmd[0], "shutdown") == 0) {
    if (tamCMD == 1){
        free(v);
        free(mz);
        break;
}
```

4. Funcionalidades Realizadas

Eis uma lista das funcionalidades realizadas aquando da submissão deste relatório e de todos os *deliverables*:

- **Estrutura de Dados para representar o labirinto e o seu conteúdo** - parcialmente cumprido. Estrutura Base criada. Faltam alguns mecanismos que só se revelarão necessários aquando da realização da segunda meta.
- **Informação dos Clientes** - parcialmente cumprido. Estrutura Base criada. Faltam alguns mecanismos que só se revelarão necessários aquando da realização da segunda meta.
- **Armazenamento dos pares *username/password*** - totalmente cumprido
- **Leitura e Validação (Sintaxe) dos comandos** - totalmente cumprido

5. Verificação e Validação

Uma vez que nesta meta são “erguidas” as fundações que irão suportar todos os mecanismos a serem desenvolvidos ao longo do projeto, é de extrema importância fazer um estudo exaustivo de todos os procedimentos e suas funcionalidades, de forma a garantir que não possuem *bugs* ou vulnerabilidades que possam revelar-se fatais no futuro.

Nesse sentido, estes foram os principais testes realizados de forma sistemática sempre que foram efetuadas alterações significativas na estrutura do código:

- Teste individual de todos os comandos na linha de comandos
    - Teste de Sintaxe
    - Teste de ações realizadas pelos comandos
- Teste de Escrita/leitura dos logs
    - Listagem dos utilizadores através do comando “users”
    - Adição/*kick* de jogadores
    - Verificação manual do que acontece ao ficheiro de texto com essas edições
- Teste de vulnerabilidades
    - Tentativa premeditada de “estourar” o programa, ao fornecer-lhe dados que ele não está à espera de receber. Permite-nos encontrar principalmente problemas de validação e de alocação de memória.