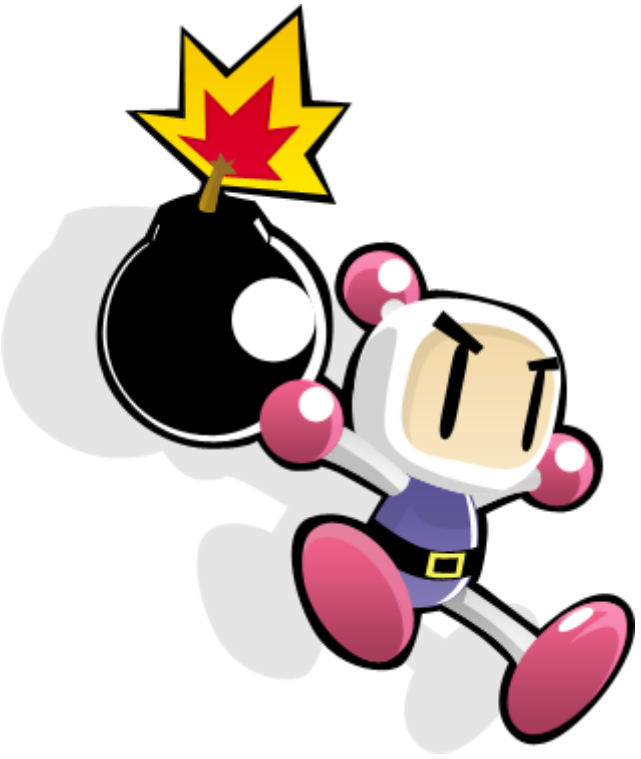
Departamento de Engenharia Informática e de Sistemas
Licenciatura em Engenharia Informática – Ramo de Desenvolvimento de Aplicações

Sistemas Operativos
2017/2018



Trabalho prático: Bomberman
Meta Final

Trabalho Realizado por:
Carlos André Eusébio Santana nº21240449
José Hugo Sousa Silva nº21240009

Índice

Introdução **3**

Estrutura de Dados **4**

Sinais **6**

Arquitetura e estratégia de named pipes **7**

Threads **8**

Mecanismo de sincronização **10**

Funcionalidades Realizadas **11**

Verificação **12**

Testes 12

Comportamentos Anômalos 12

Conclusão **13**

Introdução

No âmbito da cadeira de sistemas operativos foi proposto aos alunos a realização do jogo Bomberman para ambiente de consola (modo-texto).

Sendo esta a meta final, é requerido a conclusão do trabalho, a qual representa a junção das metas anteriores e a aplicação dos conhecimentos adquiridos neste semestre.

Estrutura de Dados

O servidorUtil.h é um ficheiro do tipo header onde encontram-se as estruturas necessária para o jogo, sendo estas:

- LocalizacaoI - tem como objectivo guardar toda a informação necessária para os inimigo, sendo estas a sua posição no array do tabuleiro, no ecrã e o seu estado;

```
struct LocalizacaoI{
    int antx, x;
    int anty, y;
    int antp, p;
    bool vida, bot;
};
```

- Jogador - contém toda a informação necessária para o servidor poder controlar o jogador, sendo esta o pid do jogador, o número de bombas, vidas, pontos, sua posição e uma pthread_t para controlar as bombas que o jogador colocar;

```
struct Jogador{
    int pid;
    int cara;
    int bombas, mBombas;
    int vidas;
    int pontos;
    int antx, x;
    int anty, y;
    int antp, p;
    pthread_t bombaPT;
    bool bombaLock;
};
```

- Labirinto - esta estrutura possui toda a informação que é utilizada durante o jogo. Têm um array de int onde será guardado o mapa, bool para usar como trincos, int para controlar o número de objectos apanhados pelos jogadores, array da estruturas mencionadas anteriormente e de seguida, 3 int para facilitar a passagem de informação para a função que controla as bombas;

```
struct Labirinto{
    int Tab[600];
    bool pronto, run, vitoria, final;
    int fim;
    Jogador jogadores[20];
    LocalizacaoI inimigos[20];
    int nBomba, pBomba, bPid;
};
```

- Clientes - Tem um int pid que vai conter o pid do cliente(esta variável vai servir para mais tarde o servidor poder comunicar com o cliente), nome[20] guarda o nome do utilizador, password[20] guarda a password do cliente, cmd[20] vai guardar a mensagem que o cliente pretende enviar ao servidor

```
struct Clientes{
    int pid;
    char nome[20];
    char password[20];
    char cmd[100];
    char msg1[100];
    int info;
};
```

- Informação - é a estrutura que o servidor manda para o cliente enquanto este estiver na fazer de jogo, contendo toda a informação necessária para conseguir fazer display do jogo.

```
struct Informacao{
    bool dentro, sair, fim, vitoria;
    int anty, antx, x, y, info, tipo, atualiza;
    int cara;
};
```

Sinais

No servidor são usados os sinais SIGUSR1 e SIGINT, os quais passam a usar a função shutdown que acede a uma variável global que guarda todos os pids que estão ligados ao ser e utilizando a variavel, faz um kill ao pids, mandando o sinal SIGUSR1.

No cliente também são utilizados os sinais SIGUSR1 e o SIGINT que tambem usam uma função chamada shutdown. o SIGUSR1 trata do sinal que este pode vir a receber do servidor e o SIGINT que quando recebe sinal desliga-se do servidor.

```
//Inicia o SIGUSR1 e verifica
if(signal(SIGUSR1,shutdown) == SIG_ERR)
{
    printf("Erro a tratar do SIGUSR1\n");
    exit(0);
}
//Inicia o SIGINT e verifica
if(signal(SIGINT, shutdown) == SIG_ERR)
{
    printf("Erro a tratar do SIGINT\n");
    exit(0);
}
```

Arquitetura e estratégia de named pipes

O objectivo da utilização de named pipes neste trabalho serve para possibilitar a comunicação entre o servidor e os seu clientes.

Quado servidor é iniciado, este verifica se já existe um FIFO com o mesmo nome que o seu para verificar se já existente um servidor ligado. De seguida, este pede um ficheiro de utilizadores e a seguir é criado um FIFO chamado “sss”, o qual fica aberto para leitura do servidor para que possa ler as informações que os clientes neste vão escrever para fazer pedidos ao servidor.

Quando o cliente é iniciado, vai verificar se o servidor já tem o seu FIFO. Após realizar a verificação o processo irá criar o seu próprio FIFO, o qual utilizará para receber/ler a informação que o servidor nele escreveu, sendo esta informação a resposta ao pedido anteriormente realizado pelo cliente.

Neste trabalho o servidor envia para o cliente dois tipos de informação/estruturas, a estrutura Cliente e Informação. No entanto o cliente limita-se a enviar só a estrutura do tipo Cliente.

Threads

No trabalho, tanto o servidor como o cliente requerem a utilização de threads para fornecer ao jogo um melhor desempenho.

O servidor utiliza várias threads, que servem para controlar os inimigos e controlar as bombas que os jogadores possuem.

As threads que controlam os inimigos/bots são criadas no momento que o primeiro cliente entra no jogo e dá o seu início (linha 1012).

```
pthread_create(&inimigoPT[j], NULL, &controlaBot, (void *) &game);
```

Sendo controlaBot a função que a thread irá realizar e &game (para mais informação sobre a estrutura verificar a pág.4) o parâmetro que vai receber. O número de Threads deste tipo irá depender do número de inimigos que existem no mapa. Após iniciar cada thread fica responsável por tratar da informação de um único inimigo, até que este morra ou o jogo acabe, assim terminando a thread.

O outro tipo de threads que o servidor utiliza, servem para controlar as bombas que os jogadores usam, tendo estes a possibilidade de “criar uma de cada vez.

```
pthread_create(&(game.jogadores[lugar].bombaPT),
NULL, &controlaBombas, (void *) &game);
```

A thread é criada sempre que o utilizador quer utilizar uma das suas bombas, tanto bombas normais como mega bombas(é o servidor que determina se é possivel ou não, através do número de bombas que este possui e por um bool), cada jogador possui uma variável do tipo pthread_t que fica responsavel pela thread. Cada uma percorre a função controlaBombas que recebe &games como parâmetro(para mais informação sobre a estrutura verificar a pág.4). A thread termina quando o processo de utilização acabar. O benefício de usar uma thread para controlar as bombas é o factos de estas terem de ficar 2 segundos inativas e 2 segundos ativas, assim como é um processo independente podemos utilizar usleep para controlar o tempo sem afetar o desempenho do servidor.

Ao Contrário do servidor o cliente só possui um thread, a qual tem como função tratar e receber a informação que é fornecida pelo servidor durante a fase de jogo.

```
pthread_create(&tarefa, NULL, &recebe_msg, (void*) pCli);
```

Como podemos ver a thread utiliza a função recebe_msg a qual está sempre a ler o FIFO criado pelo cliente à espera de informação do servidor, recebendo sempre informação do tipo Informação(para mais informação sobre a estrutura verificar a pág.5). A thread recebe como parâmetro pCli sendo este um ponteiro para uma estrutura do tipo Cliente (para mais informação sobre a estrutura verificar a pág.5) a qual é utilizada como moderador entre o processo principal e a thread.

O benefício de utiliza esta thread é o facto de possibilitar a leitura e envio de informação de forma independente.

Mecanismo de sincronização

Na totalidade do trabalho só é usada uma vez um mecanismo de sincronização, sendo este do tipo pthread_mutex_t.

Este mecanismo é utilizado no início da função controlaBot(sendo esta a função usada pelas threads que controlam os bot/inimigo), sendo o seu propósito permitir que cada thread saiba qual é informação que tem de aceder.

```
void *controlaBot(void *dados)
{
    Labirinto *game = (Labirinto *) dados;
    Informacao info;
    int ch;
    int lugar;
    bool moveLock = FALSE;
    pthread_mutex_lock(&trinco);
    for (int i = 0; i < 20; ++i)
    {
        if (game->inimigos[i].vida =TRUE && game->inimigos[i].bot == FALSE)
        {
            lugar = i;
            game->inimigos[lugar].bot = TRUE;

            break;
        }
    }

    printf("O inimigo %d iniciou com sucesso\n", lugar);

    pthread_mutex_unlock(&trinco);
```

controlaBot recebe como parâmetro uma estrutura do tipo Labirinto(o tipo de estrutura que controla o jogo), para este conseguir ir buscar o local onde se encontra a informação pela qual é responsável, decidimos utilizar um “trinco” para evitar possíveis erros.

Funcionalidades Realizadas

O trabalho entregue encontras-se concluído, no entanto há certas funcionalidades que não foram incorporadas sendo estas:

- carregar mapas a partir de um ficheiro, este trabalho só possui um mapa;
- o inimigos apesar de terem autonomia, serem afectados por bombas e após morrerem largarem drops, estes não realizam dano aos jogadores;
- não foram utilizadas variáveis de ambiente, sendo estas representadas por variáveis globais.

Todas as outros requisitos foram crumpidos.

Verificação

Testes

Após a conclusão do trabalho foram realizados testes ao código para verificar se este está operacional e sem nenhum bug.

Foram realizados testes às seguintes partes :

- Ficheiro de utilizadores - foram introduzidos nomes de ficheiro existentes e ficheiros não existentes;
- Input dos comandos - foram introduzidos múltiplos e variados tipos de input.
- Envio de sinais - foram testados os sinais anteriormente especificados

Não foram realizados mais testes, porque ao executar o código e jogar e possível ver se o código esta a correr como previsto.

Comportamentos Anômalos

Comportamentos encontrados após a realização dos testes:

- Ficheiro de utilizadores - nenhum bug foi encontrado durante a realização dos testes;
- Input de comandos - o servidor foi capaz de distinguir os comandos que foram introduzidos corretamente dos incorretos, sem apresentar nenhum problema.
- Envio de sinais - nenhum bug foi encontrado durante a realização dos testes;

O código está a correr da maneira que é pretendido.

Conclusão

Com a conclusão desta meta, concluimos o trabalho que nos foi proposto pelo âmbito da cadeira de Sistemas Operativos, assim, aplicando a maior parte dos conhecimentos adquiridos.

Apesar de o trabalho não possuir 100%, verificamos que este foi um teste das nossas habilidades, o qual, acreditamos que passámos, assim podendo aplicar os conhecimentos adquiridos em futuros projectos