Instituto Superior de Engenharia de Coimbra
Sistemas Operativos

Trabalho Prático
Meta 2

Licenciatura em Engenharia Informática
2023 / 2024

Alexandre Ramalho Silva
a2021145409@isec.pt

André Filipe Lopes Dias
a2021140917@isec.pt

Índice

Introdução

Nesta meta do trabalho prático, o foco está na finalização do projeto iniciado na meta anterior.

Programas e ficheiros utilizados

Durante a realização da 2ª meta, procedemos ao desenvolvimento de dois programas (motor.c e jogoUI.c) e os respetivos *header files (*motor.h e jogoUI.h*).*O programa "motor.c" trata de realizar a execução geral. O ficheiro "motor.h" faz a inicialização de estruturas, variáveis e funções utilizadas pelo programa motor.c. O programa "jogoUI.c" trata de criar a interface para o jogador. O ficheiro "jogoUI.h" inicializa as estruturas e funções utilizadas pelo programa jogoUI.c. Foi definido o ficheiro "utils.h" para a definição de estruturas e variáveis auxiliares aos programas. Foi também definido um ficheiro *MakeFile* para auxiliar na compilação dos programas acimas referidos.

```
typedef struct utilizador{
    char nomeJogador[50];
    int pid;
    int x,y;
    char comando[TAMANHO_STRING];
    int resposta;
    char labirinto[16][40];
    char mensagem[TAMANHO_STRING];
}Utilizador;

Utilizador newUtilizador;
```

```
typedef struct motor{
    char nomeJogador[50];
    int pid;
    int x,y;
    char comando[TAMANHO_STRING];
    int resposta;
    char labirinto[16][40];
    char mensagem[TAMANHO_STRING];
} Motor;
Motor m;

typedef struct infoJogadores{
    char nomeJogador[TAMANHO_STRING];
    int pid;
}InfoJogadores;
InfoJogadores arrayJogadores[MAX_UTILIZADORES];
```

```
typedef struct jogoUI{
    char nomeJogador[50];
    int pid;
    int x, y;
    char comando[TAMANHO_STRING];
    int resposta;
    char labirinto[16][40];
    char mensagem[TAMANHO_STRING];
}JogoUI;
JogoUI jogo;

typedef struct janelas{
    WINDOW *janelaJogo;
    WINDOW *janelaComandos;
    WINDOW *sub_janela;
}Janelas;
Janelas janela;
```

Descrição de funções implementadas

Funções de motor

valida():

A função valida() processa comandos inseridos pelo usuário na janela janelaComandos. Ela ativa a exibição dos caracteres digitados, solicita um comando, e executa ações específicas com base no comando. Reconhece comandos como "users", "kick", "test bots", "begin", "bmov", "rbm", "end", exibindo mensagens correspondentes. Desabilita a exibição dos caracteres digitados, atualiza a janela e retorna 1 em situações normais. Quando o comando é "end", exibe uma mensagem, aguarda uma tecla, remove o FIFO do motor, encerra o ncurses e retorna 0. Comandos inválidos exibem uma mensagem e retornam 1.

janelas():

Configura janelas ncurses de acordo com os parâmetros fornecidos. Se o segundo argumento for 1, ela habilita o scrollok na janela, caso contrário, configura a janela para aceitar entrada do teclado e desenha uma borda. A função finaliza atualizando e exibindo a janela.

carregarLabirinto():

Lê um arquivo de labirinto, armazenando as informações em uma matriz e exibindo o conteúdo em uma sub-janela ncurses. Ela cria uma sub-janela dentro da janela principal, lê as linhas do arquivo para preencher a matriz do labirinto, e imprime os caracteres na sub-janela, atualizando-a ao final.

atualizaMapa():

A função atualizaMapa() é responsável por enviar as coordenadas atualizadas dos jogadores para os respectivos processos filhos (bots) através de FIFOs. Ela percorre a lista de jogadores ativos, abre os FIFOs correspondentes, e envia as informações atualizadas. Essa função é utilizada para manter os bots informados sobre a movimentação dos jogadores no jogo.

*thread_funcao():

A função thread_funcao é executada em uma thread paralela e é responsável por receber informações do FIFO do motor, processar comandos específicos relacionados a jogadores e interagir com o jogo. Ela atualiza o labirinto, coordena movimentos de jogadores, envia mensagens entre jogadores e lida com a inicialização e encerramento do jogo. A função também pode lançar bots em processos filhos para realizar operações específicas no jogo.

recebe():

A função recebe() verifica se o nome de um jogador já existe na lista de jogadores ativos. Se o nome não existe, adiciona o novo jogador à lista e informa a conclusão. Se o nome já existe, envia uma resposta negativa ao jogador, utilizando FIFOs para comunicação. Essa função é utilizada para gerenciar a entrada de novos jogadores no jogo.

mandaLab():

A função mandaLab() envia informações do motor (labirinto) para os jogadores ativos, utilizando FIFOs individuais para cada jogador. Ela adquire um bloqueio antes de realizar a operação para garantir consistência na comunicação e, em seguida, envia as informações do labirinto para os jogadores. Essa função é usada para manter os jogadores atualizados sobre o estado do labirinto no jogo.

main():

A função main() inicia o jogo, configurando a interface ncurses, carregando um labirinto a partir de um arquivo, e criando um FIFO para comunicação entre os componentes do jogo. Uma thread é iniciada para processar a comunicação com o motor do jogo em segundo plano. O programa entra em um loop que aguarda comandos do usuário na interface ncurses, processa esses comandos, e continua até que um comando específico ("end") seja inserido, encerrando o jogo e liberando os recursos.

Funções de jogoUI

janelas():

A função janelas() recebe uma janela (WINDOW *janela) e um parâmetro inteiro (int x). Se x for igual a 1, configura a janela para exibir informações relacionadas ao jogo, como bordas e limpeza. Se x for igual a 2, configura a janela para permitir rolagem.

enviaNome():

A função enviaNome() tenta abrir um FIFO chamado FIFO_MOTOR para escrita. Se a abertura for bem-sucedida, ela envia informações sobre o jogador contidas na estrutura jogo para o FIFO e em seguida fecha o FIFO. Essa função é parte de um processo de comunicação entre processos em um jogo.

atualizaCoordenadas():

A função atualizaCoordenadas() recebe as coordenadas x e y, atualiza a estrutura jogo com essas coordenadas e, em seguida, envia a estrutura atualizada para um FIFO chamado FIFO_ATUALIZA. Este FIFO é usado para informar outras partes do programa sobre as novas coordenadas do jogador no jogo. Se ocorrer algum erro ao abrir o FIFO, a função exibe uma mensagem de erro e encerra o programa.

lerResposta():

A função lerResposta() lê dados de um FIFO chamado abrirFIFO_JOGO para atualizar a estrutura jogo. Se ocorrer um erro na abertura do FIFO, exibe uma mensagem de erro e encerra o programa. Em seguida, verifica se a resposta contida na estrutura jogo é igual a zero. Se for, indica que já existe um nome de jogador igual, exibe uma mensagem de erro, remove o FIFO e encerra o programa. Caso contrário, a função chama carregarLabirinto() para atualizar a interface com o labirinto contido na estrutura jogo.

carregarLabirinto():

A função carregarLabirinto() cria uma sub-janela na interface do jogo e a preenche com os caracteres do labirinto contidos na matriz labirinto. Essa função é responsável por exibir o labirinto na interface do usuário.

thread_funcaoLabirinto():

A função thread_funcaoLabirinto é uma thread que lê continuamente atualizações do estado do jogo de um FIFO chamado FIFO_ATUALIZA2. Se a mensagem do jogo indicar que um jogador foi expulso, exibe mensagens apropriadas e encerra o programa se o jogador expulso for o jogador principal. Se a mensagem for "Saiu", exibe uma mensagem indicando que um jogador saiu do jogo. Em seguida, atualiza a interface do usuário com o estado do labirinto. A thread opera em um loop infinito até que o programa seja encerrado, removendo o FIFO no final.

thread_funcaoTeclado():

A função thread_funcaoTeclado() é uma thread que lida com a entrada do teclado do jogador. Ela envia informações sobre o nome do jogador, PID e comandos para um processo motor através de um FIFO chamado FIFO_MOTOR. A função também processa comandos de movimentação do jogador e atualiza as coordenadas no jogo. O loop continua até que o comando "exit" seja inserido, momento em que o FIFO é removido e o programa encerrado.

executaComando():

A função executaComando() analisa e executa comandos do jogo, recebidos como argumentos na forma de uma string comando. Dependendo do tipo de comando, como "players" ou "msg", ela realiza operações específicas, interagindo com processos relacionados ao jogo. Se o comando não for reconhecido, exibe uma mensagem indicando que o comando não foi encontrado ou executado com sucesso.

recebeMensagem():

A função recebeMensagem() lê mensagens do FIFO chamado abrirFIFO_JOGO e as armazena na estrutura jogo. Se ocorrer um erro na leitura do FIFO, a função exibe uma mensagem de erro e encerra o programa. Se a mensagem na estrutura jogo for igual a zero, indica que já existe um nome de jogador igual, exibe uma mensagem de erro, remove o FIFO e encerra o programa. Caso contrário, exibe a mensagem recebida na janela de comandos da interface do jogo.

fazerPedido():

A função fazerPedido() recebe um comando como uma string comando e o envia para um processo motor através do FIFO chamado FIFO_MOTOR. Ela utiliza a estrutura jogo para transmitir informações relacionadas ao comando. Se ocorrer um erro ao abrir o FIFO, a função exibe uma mensagem de erro e encerra o programa. Essa função é parte do sistema de comunicação entre o jogo e o processo motor.

recebePedido():

A função recebePedido() lê informações sobre jogadores do FIFO chamado abrirFIFO_JOGO e exibe os nomes dos jogadores na janela de comandos da interface do jogo. Se ocorrer um erro na leitura do FIFO, a função exibe uma mensagem de erro e encerra o programa. Essa função é utilizada para atualizar a lista de jogadores na interface do usuário.

main():

A função main() serve como ponto de entrada para o programa. Ela inicializa a interface do usuário, cria threads para lidar com entrada de teclado e atualização do labirinto, configura janelas e verifica a existência de FIFOs essenciais. Posteriormente, ela coordena a configuração do jogador, inicia as threads e aguarda o término delas. Finalmente, encerra a interface do usuário, removendo os FIFOs temporários utilizados. Essa função é responsável pela execução principal do programa, abrangendo interação com o usuário, comunicação entre processos e manipulação da interface do jogo.