Licenciatura em Engenharia Informática

Relatório de Trabalho Prático

[Sistemas Operativos]
[Trabalho Prático – Meta 2]

Daniel Tinoco - 2021132552
Rodrigo Lourenço - 2021155662

29 de dezembro de 2023

Instituto Superior de Engenharia
Politécnico de Coimbra

Índice

JogoUI

O programa jogoUI, tal como referido no enunciado do trabalho prático, é o programa que os jogadores irão utilizar para interagir com o jogo e as suas funcionalidades.

A funcionalidade deste é relativamente simples, começa por validar o nome de usuário, não aceitando nomes que contêm caracteres não-letras (para prevenir possíveis erros).

O cada processo do jogoui tem uma FIFO própria nomeada com base no seu PID.

O jogoui está dividido em duas threads, a thread principal (main) é responsável pela comunicação outbound como o envio de comandos ao motor/outros jogosui, a segunda thread é responsável pela leitura do seu FIFO e atualizações necessárias, por exemplo, ao display do mapa quando recebe uma versão mais atualizada do motor.

Após enviar os seus dados ao FIFO do motor, esta irá aguardar pelo jogo começar. E, após tal, o jogador pode jogar usando as arrow keys ou enviar comandos pressionando espaço

Figura 1 – Interface do jogoUI.

Motor

O motor serve como o “gestor” do jogo, sendo controlado por um administrador e tratando da grande maioria da lógica do programa. O motor tem apenas um FIFO, usado pelo jogoUI para o contactar.

Pondo o sistema do motor em resumo, este primeiro irá começar por receber jogadores, e guardar os dados dos mesmos num array, caso receba mais do que 5, os extras ficarão como espetadores.

Após os jogadores estarem em jogo, o tempo se esgotar ou o administrador usar o comando “begin”, o motor irá entrar num ciclo em que inicializará o nível, e lançará as threads que fazem o jogo funcionar.

O motor possui 4 threads, excluindo a main, a primeira, que trata do teclado e verificação de comandos é constante e existe apenas uma durante a execução toda. As outras 3 são criadas e destruídas (pthread_join) a cada nível, sendo estas a que recebe dados do jogador, a que cria bots e coloca as pedras no labirinto, e uma terceira, responsável por limpar as pedras que já expiraram e mover os bloqueios moveis.

Após o ciclo se repetir 3 vezes, (3 níveis) o jogo acaba e os processos são encerrados.

A interface do motor foi alterada significamente quando comparada à nossa primeira meta, nesta, tínhamos feito uso do ncurses, no entanto, devido às recomendações de professores assim como a realização que tal não só não faria muito sentido, também não era prático, e os problemas visuais que o ncurses já causava no jogoUI de quando em vez, optámos por remover esta funcionalidade, agora, caso o administrador queira ver o labirinto, terá de usar um novo comando “lab”.

Comunicação entre processos

A comunicação entre os dois programas é feita com base numa única struct “MSG”, esta tem um tipo, e, consoante este, os dados que carrega são diferentes. Reconhecemos que esta forma não é a mais elegante ou eficiente, seria melhor enviar, por exemplo, duas estruturas, uma com o tipo, e outra apenas com os dados relevantes, no entanto não o fizemos.

Algo importante notar é que a cada mensagem que envia o labirinto, para além de enviar este, envia também uma cópia do array de jogadores no jogo, isto é importante para o jogoUI ter um mais fácil acesso a estes dados, quer seja no comando “players” ou ao enviar a mensagem para outro jogador,

a alternativa seria pedir ao motor pelos dados sempre que preciso, no entanto, achámos mais lógico estes já estarem à disposição do jogoUI.

Pedras e Bloqueios Móveis

Voltando brevemente ao motor, gostaríamos de explicar melhor a lógica das threads deste no que toca aos obstáculos do jogo, para isso temos três threads relevantes, a primeira é “RockManager” (iremos referir-nos às threads pela sua declaração pthread_t), esta thread cria os bots no início de cada nível, a quantidade destes é apenas nível + 1 e passa-lhes os parâmetros relevantes.

Após a criação destes, fica a ler um unnamed pipe e a tentar colocar as pedras no labirinto. Uma curiosidade neste é a diferença de um segundo entre a criação de cada bot, notámos que ao criar os bots ao mesmo tempo, a função srand() dentro deles tinha o mesmo valor, logo ambos os bots iriam sempre produzir as mesmas coordenadas, por isso adicionamos um simples sleep(1) entre a criação de cada Bot.

A segunda thread relevante é a thread BMCleanup, que, no contexto das pedras, apenas verifica a sua duração comparada ao tempo atual e, caso já se tenha expirado, apaga-a do mapa.

Para além disso esta thread também cria e move todos os bloqueios moveis, estes são na verdade Pedras com outra letra e armazenados localmente à thread. A thread sabe quando criar ou apagar um bloqueio movel baseado num contador, que, é modificado na terceira thread relevante, a thread dos comandos, que ao receber o comando “bmov” ou “rbm” incrementa ou decrementa este valor, tendo cuidado para não passar do máximo ou ser menor que 0, valores quais causariam problemas relativamente sérios.

Arquitetura do Sistema

Como já mencionado os processos (JogoUI, Motor e bot) comunicam através de pipes (named ou unnamed), para alem disso o jogoUI e o Motor transferem informações através de uma struct do tipo MSG.

Cada jogoUI tem o seu próprio fifo, distinguido através do seu PID que é usado para receber dados do Motor.

O motor também tem um FIFO próprio, nomeado apenas “motor_fifo”, que é responsável, de forma similar ao jogoui, por receber toda a comunicação dos jogadores.

Para além disso, a comunicação entre o motor e os bots é feita através de um único unnamped pipe para onde os STDOUT dos processos bots são redirecionados, este pipe é gerido por uma thread do motor.