Trabalho Final
SSC0140 - Sistemas Operacionais I

Docente: Kalinka Regina Lucas Jaquie Castelo Branco
Monitor: Marcelo Duchêne

Quem Quer Ser um Milionário?

César Henrique de Araujo Guibo - nUSP: 11218705
Leonardo Fonseca Pinheiro - nUSP: 11219241
Maria Fernanda Lucio de Mello - nUSP: 11320860

São Carlos
Dez/2020

1. O Jogo

O jogo “Quem Quer ser um Milionário” consiste numa releitura do clássico programa de auditório, cujo objetivo do único participante é acertar todas as perguntas e garantir um milhão de reais em sua conta.

A mecânica do jogo criado é parecida. O jogador precisa acertar 20 perguntas, em 30 segundos. Caso o tempo acabe antes dele responder, ou pressione a resposta errada, ele está eliminado.

Além disso, o sistema de pontuação é o seguinte:

- Se acertar as 5 primeiras questões, leva 5000 pontos;
- Se acertar as 10 primeiras questões, garante 75000 pontos;
- Se acertar as 15 primeiras questões, leva 250000 pontos;
- Se conseguir acertar todas as 20 questões, é o grande vencedor e leva 100000 de pontos!

Em todas as perguntas existe a alternativa ‘E’, que permite que o jogador pule a questão sem ser eliminado, mas isso também significa que ele nunca será o grande vencedor. Isso é feito como uma forma de releitura da possibilidade do participante sair do jogo com a quantia garantida a qualquer momento no jogo real.

2. Como Rodar

Para jogar, clone o repositório disponibilizado em https://github.com/mafemello/Operating-Systems-Game.git para seu computador. Após isso, abra o terminal e, na pasta em que o arquivo está, digite o seguinte comando:

```
make
```

Logo após, digite:

```
make run
```

Pronto, o jogo irá começar! Insira seu nome e se prepare! Basta escrever a letra da alternativa que você considera correta e torcer para acertar!

3. Implementação de Threads e Semáforos

3.1. *Display* e *Engine*

Para separar as funcionalidades implementadas no jogo em duas categorias, foram implementadas uma thread para o *display* e uma para se comportar como uma *engine* do jogo. A codificação da primeira foi feita dentro da classe Display e tem como única função gerenciar a impressão na tela do console. Já a segunda está presente em Engine e utiliza o padrão de *design* de máquinas de estado que é disponibilizado através da interface *Controller* para lidar com os diferentes tratamentos de *input* recebidos pelo teclado. Dessa forma, a thread presente em *Engine* lida com a maior parte da lógica do jogo. Ela controla o temporizador, a leitura das questões, pontuação e os restantes.

Porém, ainda era necessário que essa *thread* se comunicasse com aquela que está presente na classe Display. Então, com o fim de permitir essa comunicação, foram utilizados dois semáforos can_read e can_write para garantir a exclusão mútua do *buffer* compartilhado através da classe SharedBuffer. Ela faz com que o conteúdo presente em uma de suas instâncias, só possa ser lido após um up(can_write) e um down(can_read) serem realizados e só possa ser escrito após a execução de um down(can_write) de tal forma que após a escrita do *buffer* é realizado um up(can_read). Dessa forma, essa classe garante que o *buffer* sempre tenha sido escrito antes de poder ser lido e que, uma vez escrito, uma leitura deve preceder a próxima escrita.

3.2. Gerenciamento da leitura das questões

Além dos usos de threads e semáforos já descritos, outra *thread* foi criada para gerenciar a leitura do arquivo de questões. Com o fim de diminuir o tempo de gasto com a leitura do arquivo, essa *thread é* incumbida da tarefa de garantir que quando uma questão é requisitada, ela já está na memória primária armazenada em um buffer que é compartilhado com a *thread* de *engine*. A implementação dessa dinâmica é feita através da classe QuestionsManager que possui um SharedBuffer inicializado com a primeira questão, no qual a thread de leitura de questões sempre

escreve a próxima questão após o conteúdo do *buffer* ser lido pela Engine.

Tratando-se dessa parte do sistema do jogo, é importante mencionar a utilização do conceito de semáforos para interromper a execução das *threads*. Para forçar a interrupção da *thread* usada para ler o arquivo dentro do destrutor de QuestionsManager, é utilizado um semáforo inicializado como 0, no qual após atribuir o valor falso ao atributo is_running se realiza um down e, assim, garante que a o segmento de memória da instância não seja liberado antes do fim da execução da *thread* mencionada.

3.3. Temporizador

O último uso de *threads* foi feito na implementação do temporizador Timer que faz uso de um semáforo para interromper a contagem regressiva do tempo através de um mecanismo equivalente ao utilizado no destrutor de QuestionsManager.

Assim, resumindo, as *threads* são responsáveis pelo funcionamento de toda a mecânica do jogo, e os semáforos assumem o papel de regular os acessos à memória e interromper *threads* em execução.

4. Classes citadas

- Engine: include/control/engine.h e src/control/engine.cpp
- Display: include/display/display.h e src/control/display.cpp
- Controller: include/control/controllers.h
- SharedBuffer: include/shared_buffer.h
- QuestionsManager: include/questions_manager.h e src/questions.cpp
- Timer: include/timer.h e src/timer.cpp