Relatório Final - Simulador de Árbitro de Barramento

Fernanda Laís P. de Souza[1], Wellyson Gustavo S. Oliveira[2]

Instituto Metrópole Digital - Universidade Federal do Rio Grande do Norte (UFRN)
59.078-970 - Natal - RN – Brasil

{fernandalaisouza@gmail.com, wellysongustavo@gmail.com

1. Introdução

Um computador consiste em um conjunto de componentes ou módulos que possuem três tipos básicos: Processador, Memória e E/S (entradas/saídas). Para que seja possível a transferência de informações, necessita-se de uma estrutura de interconexão que irá ligar cada um desses módulos. Atualmente, das inúmeras estruturas já utilizadas, a mais comum é o barramento.

O barramento é um meio de transmissão compartilhado, responsável por ser o caminho por onde os dados trafegam e onde ocorre a comunicação dos módulos. Devido a ser um meio de transmissão compartilhado, o barramento permite que múltiplos dispositivos conectem-se a ele. Normalmente, sua estrutura consiste em cerca de 50 até centenas de linhas separadas, onde essas linhas, transmitem sinais que podem ser de dados, endereços e controle. Um sinal transmitido por um dispositivo ficará disponível para recepção de todos os outros que também estiverem conectados ao barramento. Em caso de outro dispositivo tentar emitir um sinal simultaneamente, irá ser sobreposto, dessa forma, apenas um dispositivo por vez pode emitir o sinal com sucesso.

Quando dois ou mais dispositivos do módulo E/S tentam acessar ao mesmo tempo o barramento, ocorre o que chamamos de disputa. Para evitar esse problema, utiliza-se o mecanismo de arbitragem de barramento com arranjo master-slave. Nesse tipo de arranjo, somente o mestre do barramento (bus master) controla o acesso e o escravo (bus slave) responde as requisições. Dessa forma, somente um dispositivo será atendido por vez. A arbitragem pode ser classificada como estática, com controle compartilhado de forma pré-determinada, ou dinâmica, onde o barramento só é alocado em resposta a uma requisição.

Na arbitragem dinâmica, devido ao barramento ser alocado apenas a partir de requisições, são necessárias algumas políticas de alocação que serão as responsáveis por definir como e qual dispositivo deve ter permissão de acesso ao barramento primeiro. Este relatório, consiste no resultado da implementação de um simulador de árbitro de barramento com as seguintes políticas de alocação: Prioridade Fixa, Prioridade Rotativa e Com Justiça.

2. Funcionamento

Cada política de alocação, possui determinados parâmetros que define qual ordem de atendimento pelo barramento das solicitações dos periféricos/dispositivos. O funcionamento das políticas implementadas no simulador se dá da seguinte forma:

- ➔ **Prioridade fixa**: Cada periférico é associado a um número fixo de prioridade e o árbitro decidirá qual periférico terá uso do barramento exatamente de acordo com a sua prioridade. O periférico com a maior prioridade no momento sempre terá acesso ao barramento.
- ➔ **Prioridade rotativa**: Como o próprio nome já indica, nessa política, a prioridade do periférico não é fixa. Existem várias maneiras de mudança da prioridade, como de função por tempo de espera ou se um dispositivo acabou de usar o barramento, ele terá menor prioridade.
- ➔ **Com justiça**: Quando periféricos de maior prioridade fazem constantes solicitações de acesso ao barramento, pode fazer com que periféricos de menor prioridade não tenham acesso algum ao barramento, o que chamamos de *starvation*. Para evitar que isso ocorra, existe a política de alocação com Justiça que não utiliza de prioridades. Nessa política, o árbitro cede o barramento ao periférico por um tempo determinado e igual para todos eles.

3. Solução Implementada

A aplicação desenvolvida foi pensada de acordo com os critérios transmitidos em sala de aula, primeiramente sendo escolhida a linguagem de programação C++, entendida como suficiente ao propósito do problema. Em segundo lugar, o ambiente de desenvolvimento a ser utilizado não precisaria possuir ferramentas elaboradas, visto que a lógica utilizada seria suficiente às nossas necessidades, assim, escolhemos proceder utilizando o Sublime Text. Como interface de interação, utilizamos o terminal próprio do sistema, artefato que supriu a comunicação alinhada na forma do requerimento e da entrada de dados.

A tarefa adotada foi a de simular árbitros de barramento, cada árbitro possui uma maneira única de atender solicitações, diferenciando-se em seus critérios de atenção a cada estrutura. As estruturas são entendidas como periféricos, aparelhos ou placas de expansão que enviam ou recebem informações do computador, eles possuem características próprias que são levadas em conta pelos árbitros na forma de critérios durante o processamento. Precisamos utilizar um formulário destinado à coleta dos valores antes de haver qualquer processamento.

O formulário demanda informações cruciais, são elas: a quantidade de periféricos em jogo, distinguidos na forma numérica de 0, 1, 2, …, N; a prioridade atribuída a cada periférico, utilizando os níveis 1, 2 e 3, nos quais a prioridade cresce da esquerda para a direita; a ordem das solicitações, representada pelos inteiros que simbolizam cada periférico dispostos da esquerda para a direita; e o tempo de duração de cada solicitação, que foi reproduzido por unidades múltiplas de 5 (5, 10, 15…).

Estruturas de dados são responsáveis pela organização, métodos de acesso e opções de processamento para coleções de itens de informação manipulados pelo programa. Para amparar nossos dados, escolhemos implementar a quantidade de periféricos no tipo <int>, os níveis de prioridade no tipo <int> e guardando-as respectivamente num <array de int>, a ordem de solicitações e o tempo de cada são armazenados em containers <vector>.

Foram apresentados em sala de aula alguns tipos de arbitragem, dentre eles, implementamos funções para representar a Prioridade Fixa, a Prioridade Rotativa e a Justiça. Todas as funções possuem complexidade quadrática, devido aos loops contidos, usados para

percorrer as solicitações na ordem escolhida pelo usuário e garantir a correta impressão dos periféricos, de acordo com os critérios próprios de cada árbitro.

A implementação se encontra apenas o arquivo simulador.cpp, pois, como explicado anteriormente, não houve necessidade de modularização para aplicação tão simples, visto que o funcionamento de acesso e interação se dá via terminal. A compilação deve ser realizada utilizando a linha de comando

```
g++ -std=c++11 -g -O0 simulador.cpp -o simulador
```

e a execução, também feita em linha de comando, é realizada ao inserir como próximo comando

```
./simulador
```

sendo possível a utilização do caso de teste disponibilizado pela mentora da disciplina e, assim, a observação das saídas de cada função/árbitro.

4. Análise de resultados

Como já foi mencionado, o ambiente de interação escolhido foi o próprio terminal, disponibilizando como saída qual a ordem dos periféricos atendidos de acordo com cada uma das políticas de alocação.

Com a biblioteca <ctime>, nós utilizamos a função std::clock() para nos auxiliar e coletar qual o tempo de compilação em clock por segundo de cada chamada da função da política específica, de modo que enxergamos um período menor para a prioridade rotativa e maiores para prioridade fixa e com justiça, dependendo, claro, do número de solicitações de periféricos.

5. Conclusão

A dinâmica utilizada por cada árbitro de barramento faz perceber que as demandas são variantes em critério, de modo que o desenvolvimento de novas técnicas sempre é capaz de tornar possível o surgimento de uma nova forma de realização.

Diante deste objeto de avaliação concluímos que tarefas como essa são de imprescindível importância após a apresentação de um conteúdo denso acerca da arquitetura dos computadores. Torna a compreensão de mais fácil alcance, visto que o estudo precisa ser detalhado antes de pôr em prática qualquer desenvolvimento por parte do programador.

6. Referências

- STANDARD Containers: Vector. [*S. l.*], 200-?. Disponível em: http://www.cplusplus.com/reference/vector/vector/. Acesso em: 19 jun. 2019.

- STANDARD Containers: Array. [S. l.], 200-?. Disponível em: http://www.cplusplus.com/reference/array/array/. Acesso em: 19 jun. 2019.

- RICARTE, Ivan Luiz Marques. Estrutura de dados. UNIVERSIDADE ESTADUAL DE CAMPINAS: UNICAMP, 2008. Disponível em: http://calhau.dca.fee.unicamp.br/wiki/images/0/01/EstruturasDados.pdf. Acesso em: 19 jun. 2019.

- SANTOS, Elias. Barramentos Entrada e Saída (E/S). [S. l.], 2017. Disponível em: https://docplayer.com.br/24213831-Barramentos-entrada-e-saida-e-s.html. Acesso em: 19 jun. 2019.

- STALLINGS, William. Arquitetura e Organização de Computadores. [S. l.]: Pearson, 2009.

- PATTERSON, David A.; HENNESSY, John. Organização e Projeto de Computadores. [S. l.]: Campus, 2013.