Pedro Henrique Bezerra Cavalcante

Relatório Final

Natal - RN
8 de junho de 2016

Pedro Henrique Bezerra Cavalcante

Relatório Final

Relatório final para obtenção parcial da nota da terceira unidade da disciplina Introdução à Ornanização e Arquitetura de Computadores do Bacharelado em Tecnologia da Informação, IMD/UFRN.

Universidade Federal do Rio Grande do Norte

Orientador: Mônica Magalhães Pereira

Natal - RN
8 de junho de 2016

Sumário

1 Introdução

1.1 Daisy Chaining

A Arbitragem por Daisy Chain tem como vantagem a simplicidade de implemetação. Por outro lado, é desvantajoso pelo motivo de que "um dispositivo de menor prioridade pode não conseguir acesso ao barramento"(SILVA, 2004), podendo ficar, assim, bloqueado indefinidamente. O uso desse esquema de arbitragem "também limita a velocidade do barramento"(SILVA, 2004).

1.2 Prioridade

A forma de árbitrariedade por prioridade é executada de acordo com a prioridade inerente ao dispositivo que solicitou execução.

1.3 Justiça

Arbritrariedade por tempo de justiça é dada da forma que: cada dispositivo tem seu tempo de execução. Por justiça, o árbitro é implementado com um tempo fixo de execução. Caso o dispositivo leve mais tempo que o árbitro permite, ele é executado, interrompido e passa para o final da linha de execução. Dessa forma,

2 Descrição e Organização do Projeto e Execução

O projeto foi desenvolvido em linguagem C++ e está distribuído em três arquivos:

1. main.cpp: contem a função principal do projeto
2. perifericos.h: contem os cabeçalhos de funções do projeto
3. perifericos.cpp: contém a implementação das funções.

Deve ser executado através do comando:

```
1 g++ -Wall main.cpp perifericos.cpp -o <arquivo de saida>
```

A primeira parte do programa irá solicitar a entrada do usuário para definir quantos periféricos irão solicitar execução.

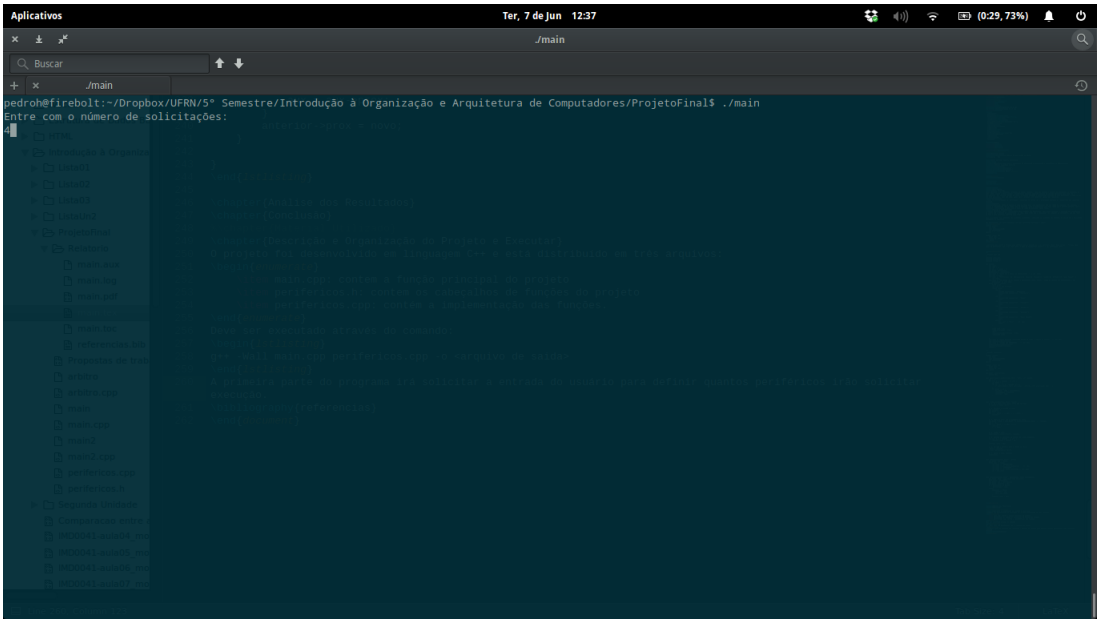

Figura 1 – Exemplo de entrada para quantidade de periféricos

Em seguida, irá solicitar ao usuário que entre com o ńúmero correspondente ao dispositivo e sua respectiva prioridade, como se pode ver na imagem a seguir:

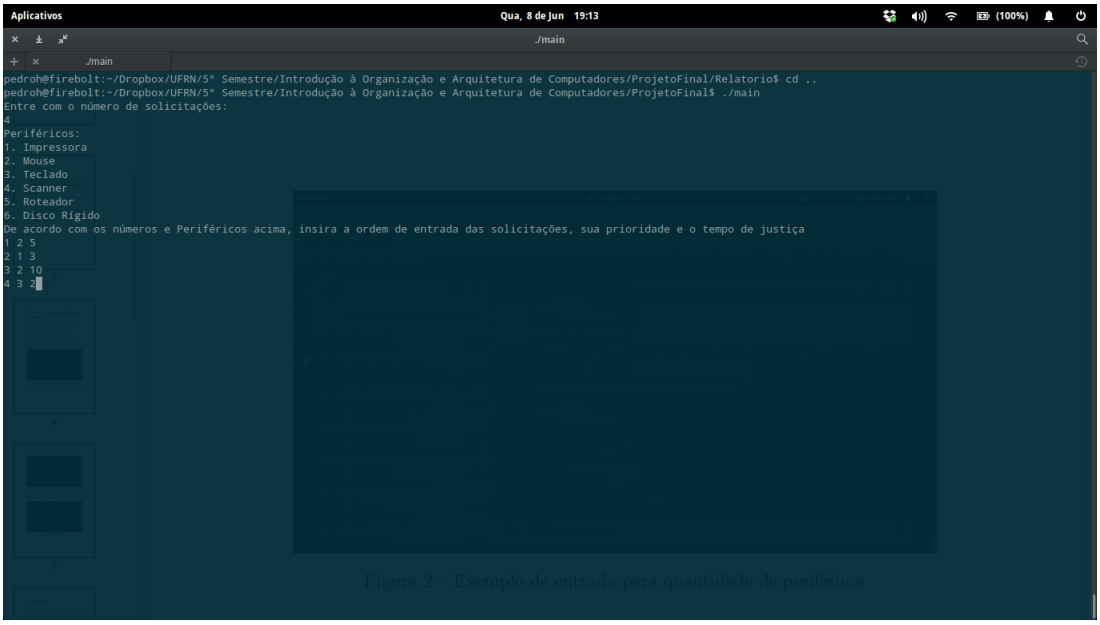

Figura 2 – Exemplo de entrada para os periféricos, sua prioridade e seu tempo de execução

Logo após isso, irá executar as rotinas do programa e gerar a saída de dados. Primeiramente a saída para Daisy Chaining, depois a saída para Prioridade e, em seguinda, a saída para Justiça (falta implementar). Veja a imagem.

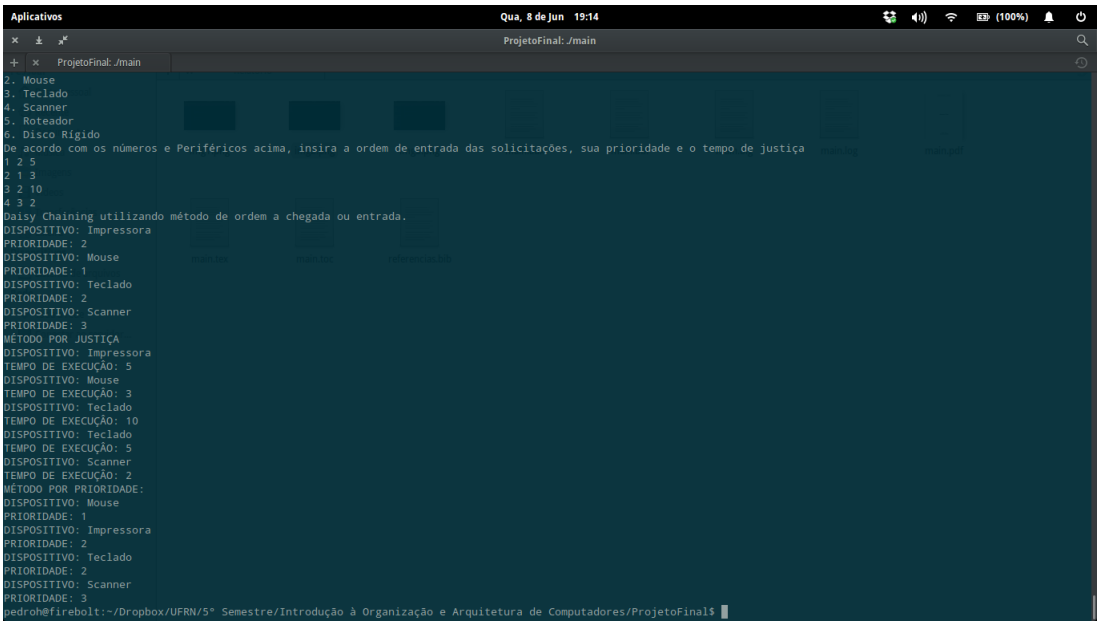

Figura 3 – Exemplo de saída

3 Implementação

A solução encontrada para o resolvimento desse projeto foi encontrada, para Daisy Chaining, a execução dos periféricos de acordo com a entrada da solicitação, ou seja, pelo ordem que a requisição vai chegando ao barramento, ela vai ficar na fila e será executada conforme essa ordem.

Para prioridade, foi realizada uma ordenação utilizando o algoritmo de Ordenação por Seleção, de acordo com a prioridade armazenada na estrutura.

Foi desenvolvida uma lista encadeada com duas estruturas,mostradas à seguir:

```
typedef struct{
    int tipo;
    char dispositivo[50];
    int prioridade;
}Perif;

typedef struct no{
    Perif info;
    struct no* prox;
}Aux_Perif;
```

A estrutura Aux_perif é formada pelo ponteiro para o próximo no e uma estrutura Perif. Essa última, é formada pelos dados do periférico, que é seu tipo (int), o nome do dispositivo (char) e a sua prioridade (int).

```
/**
* Arquivo main.cpp
*/
#include <iostream>
#include <string.h>
#include <stdlib.h>
#include "perifericos.h"


int main(){
    int solic;
    int cont = 0;
    int justica = 0;
    int aux = 0, prior = 0;
    Aux_Perif* perifericos;
    Aux_Perif* perif_just;
    perifericos = criarLista();
    perif_just = criarLista();
    std::cout << "Entre com o numero de solicitacoes: " <<
       std::endl;
    std::cin >> solic;
    std::cout << "Perifericos:\n1. Impressora\n2. Mouse\n3.
       Teclado\n4. Scanner\n5. Roteador\n6. Disco Rigido" <<
```

```
        std::endl;
22      std::cout << "De acordo com os numeros e Perifericos
           acima, insira a ordem de entrada das solicitacoes, sua
            prioridade e o tempo de justica" << std::endl;
23      while (cont < solic){
24          Perif ordem;
25          Perif justica_vet;
26          std::cin >> aux >> prior >> justica;
27          switch (aux){
28              case 1:
29                  //ordem.dispositivo = 'Impressora';
30                  strcpy(ordem.dispositivo, "Impressora");
31                  strcpy(justica_vet.dispositivo, "Impressora")
                        ;
32                  break;
33              case 2:
34                  strcpy(ordem.dispositivo , "Mouse") ;
35                  strcpy(justica_vet.dispositivo, "Mouse");
36                  break;
37              case 3:
38                  strcpy(ordem.dispositivo , "Teclado");
39                  strcpy(justica_vet.dispositivo, "Teclado");
40                  break;
41              case 4:
42                  strcpy(ordem.dispositivo , "Scanner");
43                  strcpy(justica_vet.dispositivo, "Scanner");
44                  break;
45              case 5:
46                  strcpy(ordem.dispositivo , "Roteador");
47                  strcpy(justica_vet.dispositivo, "Roteador");
48                  break;
49              case 6:
50                  strcpy(ordem.dispositivo , "Disco Rigido");
51                  strcpy(justica_vet.dispositivo, "Disco Rigido
                        ");
52                  break;
53              default:
54                  std::cout << "Error" << std::endl;
55                  break;
56          }
57
58          ordem.tipo = aux;
59          justica_vet.tipo = aux;
60
61          ordem.prioridade = prior;
62          justica_vet.prioridade = prior;
63
64          justica_vet.set_used = true;
65          ordem.set_used = true;
```

```
66
67          ordem.justica = justica;
68          justica_vet.justica = justica;
69
70          cont++;
71          inserirFinal(&perifericos, ordem);
72          inserirFinal(&perif_just, justica_vet);
73      }
74      std::cout << "Daisy Chaining utilizando metodo de ordem a
            chegada ou entrada." << std::endl;
75      imprimirLista(&perifericos);
76      std::cout << "METODO POR JUSTICA" << std::endl;
77      imprimirLista_Justica(&perif_just);
78      std::cout << "METODO POR PRIORIDADE: " << std::endl;
79      ordena_AuxPerif(&perifericos);
80      imprimirLista(&perifericos);
81      return 0;
82  }
83
84
85  }
86  /**
87  * Arquivo perifericos.h
88  */
89  #ifndef _PERIFERICOS_H_
90  #define _PERIFERICOS_H_
91
92  #include <iostream>
93  #include <string.h>
94  #include <stdlib.h>
95
96  typedef struct{
97      int tipo;
98      char dispositivo[50];
99      int prioridade;
100     int justica;
101     bool set_used = false;
102 }Perif;
103
104 typedef struct no{
105     Perif info;
106     struct no* prox;
107 }Aux_Perif;
108
109 Aux_Perif* criarLista();
110 void selectionsort_AuxPerif(Aux_Perif **vetor, int tamanho);
111 void ordena_AuxPerif(Aux_Perif **perif);
112 void imprimirLista(Aux_Perif** lista);
113 void inserirFinal(Aux_Perif** perifericos, Perif barramento);
```

```
114 void percorre_true(Aux_Perif** lista, int dispo, int
       prioridade);
115 void imprimirLista_Justica(Aux_Perif** lista);
116
117 #endif
118
119
120 /**
121 * Arquivo perifericos.cpp
122 */
123
124 #include <iostream>
125 #include <string.h>
126 #include <stdlib.h>
127 #include "perifericos.h"
128
129 Aux_Perif* criarLista(){
130     return NULL;
131 }
132
133 void selectionsort_AuxPerif(Aux_Perif **vetor, int tamanho) {
134     int i, j;
135     for (i = 0; i < tamanho - 1; i++) {
136         int menor = i;
137         for (j = i + 1; j < tamanho; j++) {
138             if (vetor[menor]->info.prioridade > vetor[j]->
                   info.prioridade) menor = j;
139         }
140         if (menor != i) {
141             Aux_Perif *temp = vetor[menor];
142             vetor[menor] = vetor[i];
143             vetor[i] = temp;
144         }
145     }
146 }
147
148 void ordena_AuxPerif(Aux_Perif **perif) {
149     // 1. Se a lista esta vazia, entao nem faz nada.
150     if (perif == NULL) return;
151
152     // 2. Descobre o tamanho da lista.
153     int tamanho = 0;
154     Aux_Perif *v;
155     for (v = *perif; v != NULL; v = v->prox) {
156         tamanho++;
157     }
158
159     // 3. Monta um vetor com os elementos da lista.
160     Aux_Perif **vetor = (Aux_Perif **) malloc(sizeof(
```

```
        Aux_Perif *) * tamanho);
    int i = 0;
    for (v = *perif; v != NULL; i++) {
        vetor[i] = v;
        v = v->prox;
    }

    // 4. Ordena o vetor.
    selectionsort_AuxPerif(vetor, tamanho);

    // 5. Corrige os ponteiros de acordo com a nova ordenacao
        .
    for (i = 0; i < tamanho - 1; i++) {
        vetor[i]->prox = vetor[i + 1];
    }
    vetor[i]->prox = NULL;

    // 6. Corrige o ponteiro para o inicio da lista.
    *perif = vetor[0];
    //void imprimirLista(*perif);

    // 7. Deleta o vetor auxiliar.
    free(vetor);
}

void imprimirLista(Aux_Perif** lista){
    Aux_Perif* aux = *lista;
    while (aux != NULL && aux->info.set_used == true){
    //while (aux != NULL){
        std::cout << "DISPOSITIVO: ";
        std::cout << aux->info.dispositivo << std::endl;
        std::cout << "PRIORIDADE: ";
        std::cout << aux->info.prioridade << std::endl;
        aux = aux->prox;
    }
}

void imprimirLista_Justica(Aux_Perif** lista){
    Aux_Perif* aux = *lista;
    while(aux != NULL){
        int teste = aux->info.justica;
        while(teste > 0){
            std::cout << "DISPOSITIVO: ";
            std::cout << aux->info.dispositivo << std::endl;
            std::cout << "TEMPO DE EXECUCAO: ";
            std::cout << teste << std::endl;
            teste = teste-5;
        }
        aux = aux->prox;
```

```
208     }
209 }
210 
211 void inserirFinal(Aux_Perif** perifericos, Perif barramento){
212     Aux_Perif* novo = (Aux_Perif*)new Aux_Perif;
213     novo->info = barramento;
214     Aux_Perif* aux = *perifericos;
215     Aux_Perif* anterior = NULL;
216     if (*perifericos == NULL){
217         *perifericos = novo;
218     }else{
219         while(aux != NULL){
220             anterior = aux;
221             aux = aux->prox;
222         }
223         anterior->prox = novo;
224     }
225 
226 }
227 
228 void percorre_true(Aux_Perif** lista, int dispo, int
       prioridade){
229     Aux_Perif* aux = *lista;
230     while (aux != NULL){
231 
232         if (aux->info.tipo == dispo){
233             aux->info.set_used = true;
234             aux->info.prioridade = prioridade;
235         }else{
236             aux->info.prioridade = prioridade;
237 
238 
239         }
240         aux = aux->prox;
241     }
242 }
```

4 Análise dos Resultados

5 Conclusão

Referências

SILVA, C. A. *Microcontroladores e Interfaces*. 2004.