Pedro Henrique Bezerra Cavalcante

Relatório Final

Natal - RN
7 de junho de 2016

Pedro Henrique Bezerra Cavalcante

Relatório Final

Relatório final para a disciplina de Introdução à Organização e Arquitetura de Computadores

Universidade Federal do Rio Grande do Norte

Orientador: Mônica Magalhães Pereira

Natal - RN
7 de junho de 2016

Sumário

1 Introdução

1.1 Daisy Chaining

Daisy Chaining é uma topologia utilizada em vários aspectos, como por exemplo o árbitro de barramento ou topologia de rede. Ambos seguem a mesma lógica: o dispositivo solicita execução através do caminho destinado para esse fim (bus-request). Por estarem ligados em série, o árbitro responde à solicitação passando em cada periférico e vendo se ele foi quem solocitou. Caso não seja, passa para o próximo, até encontrar aquele que solicitou.

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 Solução e 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>

int main(){
    int solic;
    int cont = 0;
    int aux = 0, prior = 0;
    Aux_Perif* perifericos;
    perifericos = 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;
    std::cout << "De acordo com os numeros e Perifericos
       acima, insira a ordem de entrada das solicitacoes e
       sua prioridade" << std::endl;
    while (cont < solic){
```

```
19          Perif ordem;
20          std::cin >> aux >> prior;
21          switch (aux){
22              case 1:
23                  //ordem.dispositivo = 'Impressora';
24                  strcpy(ordem.dispositivo, "Impressora");
25                  break;
26              case 2:
27                  strcpy(ordem.dispositivo , "Mouse") ;
28                  break;
29              case 3:
30                  strcpy(ordem.dispositivo , "Teclado");
31                  break;
32              case 4:
33                  strcpy(ordem.dispositivo , "Scanner");
34                  break;
35              case 5:
36                  strcpy(ordem.dispositivo , "Roteador");
37                  break;
38              case 6:
39                  strcpy(ordem.dispositivo , "Disco Rigido");
40                  break;
41              default:
42                  std::cout << "Error" << std::endl;
43                  break;
44          }
45
46          ordem.tipo = aux;
47          ordem.prioridade = prior;
48          cont++;
49          inserirFinal(&perifericos, ordem);
50          //delete[] ordem;
51
52      }
53      std::cout << "Daisy Chaining utilizando metodo de ordem a
            chegada ou entrada." << std::endl;
54      imprimirLista(&perifericos);
55      std::cout << "METODO POR PRIORIDADE: " << std::endl;
56      ordena_AuxPerif(&perifericos);
57      imprimirLista(&perifericos);
58      return 0;
59  }
60  /**
61  * Funcoes utilizadas
62  */
63  typedef struct{
64      int tipo;
65      char dispositivo[50];
66      int prioridade;
```

```
67 }Perif;
68
69 typedef struct no{
70     Perif info;
71     struct no* prox;
72 }Aux_Perif;
73
74 Aux_Perif* criarLista(){
75     return NULL;
76 }
77 void selectionsort_AuxPerif(Aux_Perif **vetor, int tamanho) {
78     int i, j;
79     for (i = 0; i < tamanho - 1; i++) {
80         int menor = i;
81         for (j = i + 1; j < tamanho; j++) {
82             if (vetor[menor]->info.prioridade > vetor[j]->
                   info.prioridade) menor = j;
83         }
84         if (menor != i) {
85             Aux_Perif *temp = vetor[menor];
86             vetor[menor] = vetor[i];
87             vetor[i] = temp;
88         }
89     }
90 }
91
92 void ordena_AuxPerif(Aux_Perif **perif) {
93     // 1. Se a lista esta vazia, entao nem faz nada.
94     if (perif == NULL) return;
95
96     // 2. Descobre o tamanho da lista.
97     int tamanho = 0;
98     Aux_Perif *v;
99     for (v = *perif; v != NULL; v = v->prox) {
100         tamanho++;
101     }
102
103     // 3. Monta um vetor com os elementos da lista.
104     Aux_Perif **vetor = (Aux_Perif **) malloc(sizeof(
           Aux_Perif *) * tamanho);
105     int i = 0;
106     for (v = *perif; v != NULL; i++) {
107         vetor[i] = v;
108         v = v->prox;
109     }
110
111     // 4. Ordena o vetor.
112     selectionsort_AuxPerif(vetor, tamanho);
113
```

```
114     // 5. Corrige os ponteiros de acordo com a nova ordenacao
        .
115     for (i = 0; i < tamanho - 1; i++) {
116         vetor[i]->prox = vetor[i + 1];
117     }
118     vetor[i]->prox = NULL;
119 
120     // 6. Corrige o ponteiro para o inicio da lista.
121     *perif = vetor[0];
122     //void imprimirLista(*perif);
123 
124     // 7. Deleta o vetor auxiliar.
125     free(vetor);
126 }
127 
128 void imprimirLista(Aux_Perif** lista){
129     Aux_Perif* aux = *lista;
130     while (aux != NULL){
131         std::cout << "DISPOSITIVO: ";
132         std::cout << aux->info.dispositivo << std::endl;
133         std::cout << "PRIORIDADE: ";
134         std::cout << aux->info.prioridade << std::endl;
135         aux = aux->prox;
136     }
137 }
138 
139 void inserirFinal(Aux_Perif** perifericos, Perif barramento){
140     Aux_Perif* novo = (Aux_Perif*)new Aux_Perif;
141     novo->info = barramento;
142     Aux_Perif* aux = *perifericos;
143     Aux_Perif* anterior = NULL;
144     if (*perifericos == NULL){
145         *perifericos = novo;
146     }else{
147         while(aux != NULL){
148             anterior = aux;
149             aux = aux->prox;
150         }
151         anterior->prox = novo;
152     }
153 
154 }
```

3 Análise dos Resultados

4 Conclusão

5 Material Utilizado

6 Descrição e Organização do Projeto e Executar