Universidade Federal de Santa Catarina – UFSC
Programa de Pós-Graduação em Engenharia Elétrica – CTC
EEL510265 – Programação de Sistemas Embarcados – 2019/2

Cezar Antônio Rigo – 201902714

Relatório Final:
Máquina de Vendas Escalonada

Este relatório apresenta o processo de desenvolvimento da solução completa para o trabalho final da disciplina, programadado em C++.

1. Diagrama das classes implementadas

A Figura 1 apresenta o diagrama das classes implementadas a fim de cumprir as especificações funcionais e requisitos do trabalho final. As classes BancoDados e Machine (em verde) são **friends** e isso foi feito para que a classe BancoDados possa ter acesso a interface, objeto de Machine, especialmente para mostrar o relatório final de vendas.

Para atender ao requisito de portabilidade do código C++ para diferentes interfaces, foi criada uma classe Interface pai com funções virtuais e duas classes derivadas, a InterfaceAtl para o sistema embarcado alvo e a InterfacePC para execução no microcomputador. Dessa forma faz-se uso do **polimorfismo** e da condicional #ifdef para definir qual classe instanciar, de acordo com a plataforma alvo.

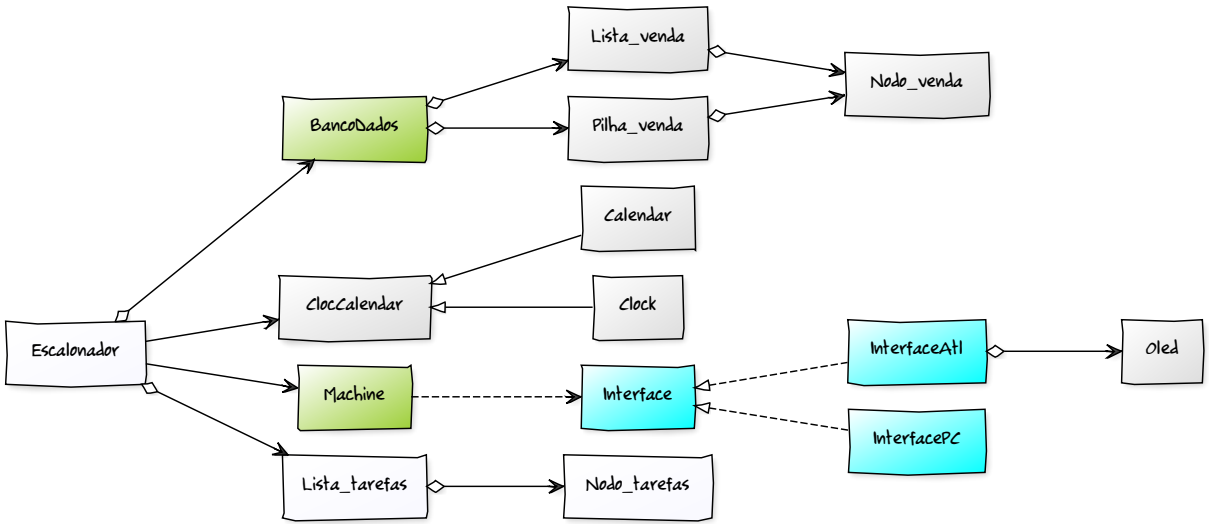

Figura 1: Diagrama das classes implementadas.

Nas próximas seções serão detalhados o funcionamento da máquina de vendas, do banco de dados, do escalonador e do ClockCalendar.

2. A máquina de vendas

A Figura 2 traz o diagrama expandido das classes relacionadas à venda de refrigerantes. Essa classe é responsável por gerenciar todo processo de venda, bem como permitir acesso às interfaces.

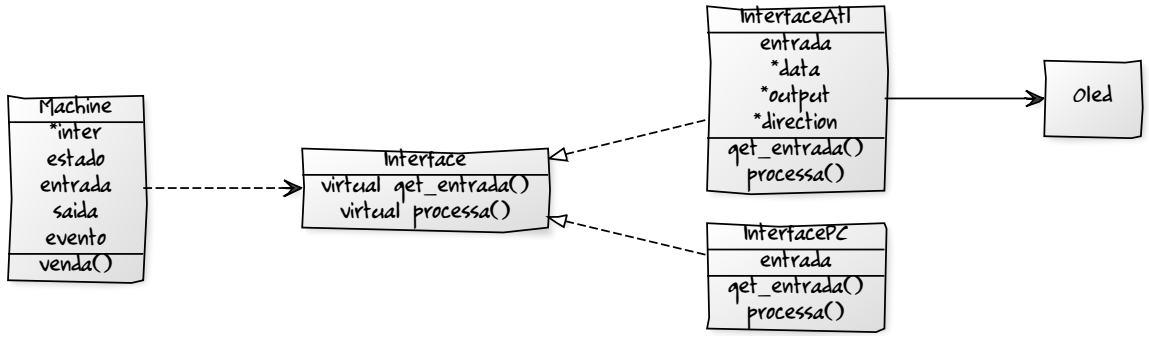

Figura 2: Diagrama expandido das classes relacionadas à venda de refrigerantes.

Esse acesso é dado através do ponteiro *inter, do tipo Interface, possibilitando executar as funções get_entrada(), que obtém input do usuário, e processa(), que mostra as saídas da máquina. O polimorfismo se dá através da criação de uma classe pai que possui as mesmas funções, porém declaradas como virtuais puras. O ponteiro inter aponta para InterfacePC ou interfaceATL conforme definição do usuário. O código a seguir mostra esta sintaxe:

```
#ifdef PC
    inter = new InterfacePc;
#endif

#ifdef ATL
    inter = new InterfaceAtl;
#endif

entrada = inter->get_entrada(); //chama objeto interface para
obter entrada
```

Para que o controlador da máquina de venda de refrigerantes funcionasse de acordo com as especificações de projeto, foi solucionada a Tabela 1 que abrange todas as possibilidades de entrada e correspondentes mudanças estados.

Tabela 1: Solução da tabela de estados

Estado Atual	Nada	M025	M050	M100	DEV	MEET	ETIRPS
S000	S000	S025	S050	S100	S000	S000	S000
S025	S025	S050	S075	S125	S000, D025	S025	S025
S050	S050	S075	S100	S150	S000, D050	S050	S050
S075	S075	S100	S125	S150, D025	S000, D025, D050	S075	S075
S100	S100	S125	S150	S150, D050	S000, D100	S100	S100
S125	S125	S150	S150, D025	S150, D025, D050	S000, D100, D025	S125	S125
S150	S150	S150, D025	S150, D050	S150, D100	S000, D100, D050	MEET	ETIRPS

A implementação das máquinas de estado em C++ se deu dentro da classe Machine, na função venda() – vide Figura 2. A classe Machine possui as variáveis estado, entrada, saida e evento para armazenar o estado atual da maquina bem como a entrada e assim determinar a saída. Após obtenção do valor da entrada, através da chamada da

classe Interface apropriada, a transição de estados é determinada através de switch-case, conforme apresentado no fragmento de código abaixo:

```
1     switch (estado)
2     {
3     case 0:
4         if (entrada == "m025")
5             estado = 25;
6
7         else if (entrada == "m050")
8             estado = 50;
9
10        else if (entrada == "m100")
11            estado = 100;
12
13        entrada = "nada";
14        break;
15    ...
16    }
```

A vantagem desta implementação é que na ocorrência de uma interrupção não há chance de perda do estado atual da máquina, tampouco da entrada mais recente, já que estes valores estão armazenados no objeto Machine, através de variáveis – vide Figura 2. Assim que um valor de entrada for obtido a máquina passa pelo switch-case, se necessário a Interface é chamada novamente para processar uma saída, e a execução da função venda() é terminada, permitindo retorno ao escalonador.

3. O banco de dados

A classe BancoDados é responsável por gerenciar uma pilha para armazenamento temporário dos dados de venda (classe Pilha_venda) e uma lista para armazenamento definitivo (classe Lista_venda). O diagrama expandido das classes relacionadas é apresentado na Figura 3.

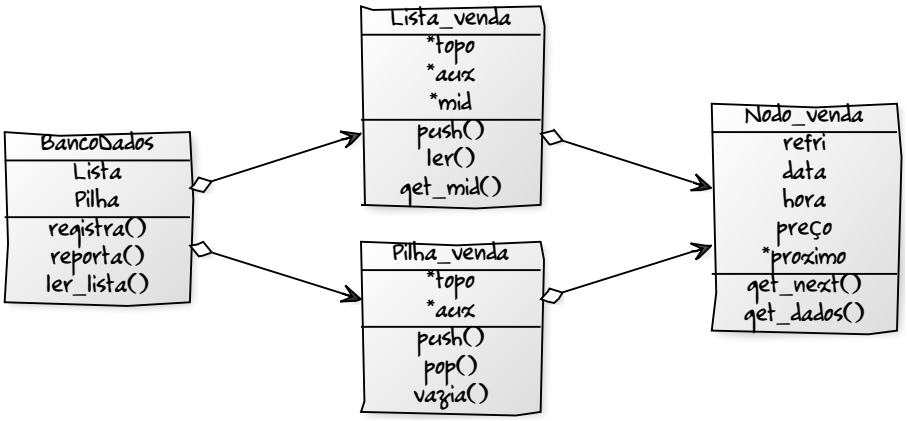

Figura 3: Diagrama expandido das classes relacionadas ao gerenciamento dos dados de venda.

O nodo contém todos os dados relacionados a uma venda, bem como o endereço de memória onde se encontra o próximo nodo, e é uma classe comum a ambas estruturas de dados (pilha e lista). A lista gerencia seus nodos com o auxilio de ponteiros do tipo

nodo, o ponteiro "topo", por exemplo, sempre aponta para o último nodo na sequência, sendo que nodos nunca são excluídos. A classe pilha gerencia seus nodos de forma similar, porém ao fazer a leitura de um nodo - pop() - este é excluído. A criação e exclusão de nodos é feita com alocação dinâmica de memória, através dos comandos "new"e "delete". Sempre que uma venda for realizada o fragmento de código abaixo é executado, a fim de salvar as informações da venda:

```
#include <exception>
using std::bad_alloc;

void Pilha::push(string refri, int preco)
{
    /*  Funçao para inserir valores em um novo nodo da Pilha.
    Argumentos:
    refri: O refrigerante vendido.
    g_data: A data da venda.
    g_hora: A hora da venda.
    precp: O preco do refrigerante.
   */
    try
    {   //aux recebe o endereco do novo nodo criado com o comando new.
        aux = new Nodo(refri, g_data, g_hora, preco, topo);
        topo = aux; //topo passa a apontar para o novo nodo.
    }
    catch (bad_alloc E)
    {
        cout << "Faltou Memoria...\n" << endl;
    }
}
```

Como pode ser visto, foi usado também o **tratamento de exceções** para monitorar se faltou memória para executar o comando new.

Quando há alocação dinâmica de memória em uma classe com o uso de ponteiros, como é o caso, deve-se fazer um destrutor para liberar essa memória antes que a instância seja destruída. Dessa forma, tanto a classe Pilha quanto Lista possuem destrutores. Abaixo o código destrutor da classe Pilha:

```
Pilha::~Pilha() //destroi a Pilha.
{
    while (topo) //enquanto houver um nodo.
    {
        aux = topo->get_next(); //salva o endereço para o proximo nodo.
        delete topo;            //deleta o nodo atual.
        topo = aux;             //atual = proximo.
    }

    //zera todos os ponteiros.
    topo = 0;
    aux = 0;
}
}
```

A classe BancoDados, por sua vez, tem uma função para registrar dados que chama a função push da pilha, criando um novo nodo. Também possui uma função reporta() que

lê toda pilha atual, efetivamente esvaziando-a, processa os dados lidos para determinar o período de maior venda, o refrigerante mais vendido e o valor total de vendas e salva as informações (que estavam na pilha) na lista - armazenamento permanente. Este algoritmo é apresentado no pseudocódigo abaixo:

```
 Enquanto a pilha nao estiver vazia

        Retira dados da Pilha

        Analisa qual refrigerante foi vendido e incrementa seu contador

        Verifica em qual periodo ocorreu a venda e incrementa seu contador

        Salva dados na Lista.

Analisa os contadores de periodo de vendas e define o maior

Analisa os contadores de refrigerante mais vendido e define o maior

Chama a interface para reportar
```

Também chama uma função da interface para mostrar as informações de venda. Há, ainda, uma função para ler toda a lista, e assim mostrar todo o histórico de vendas.

4. O Clock e Calendário

A função advance() da classe ClockCalendar é executada pelo escalonador a cada segundo, para atualizar a hora. Essa classe **herda** das classes Clock e Calendar, como apresentado na Figura 4.

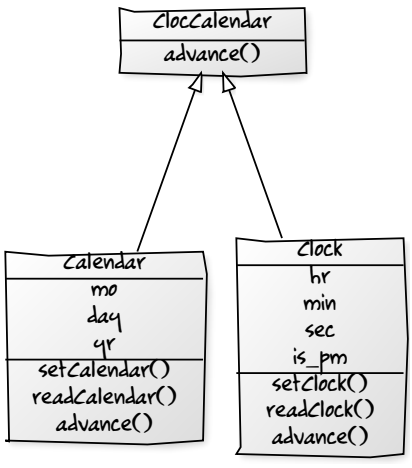

Figura 4: Diagrama expandido das classes hora e data.

As informações de data e hora são obtidas chamando readCalendar() e readClock(), também chamadas a cada segundo, e posteriormente salvas em variáveis globais para que as outras classes possam acessar. Há, ainda, uma **sobrecarga de operadores** para permitir avançar o calendário apenas digitando ++ObjetoClockCalendar;. Abaixo o fragmento de código usado para criar essa sobrecarga:

```
void operator++(ClockCalendar &t)
{
    t.advance();
}
```

5. O escalonador

Para atender aos requisitos de escalonamento do projeto, as classes da Figura 5 foram implementadas. O nodos, da classe Nodo_tarefas possuem as informações de cada tarefa, como: atraso, período, status (apta ou bloqueada) e ponteiro para função associada. A cada 1 milissegundo a função executa() da classe Escalonador (Figura 5) é chamada. Essa função incrementa um contador de tempo e chama a função decide(), da Lista_tarefas. Esta função tem acesso a todos os nodos, e baseado nessas informações e na passagem de tempo um algoritmo decide qual tarefa deve ser chamada e chama-a. O código desse algoritmo está anexado no APÊNDICE 1.

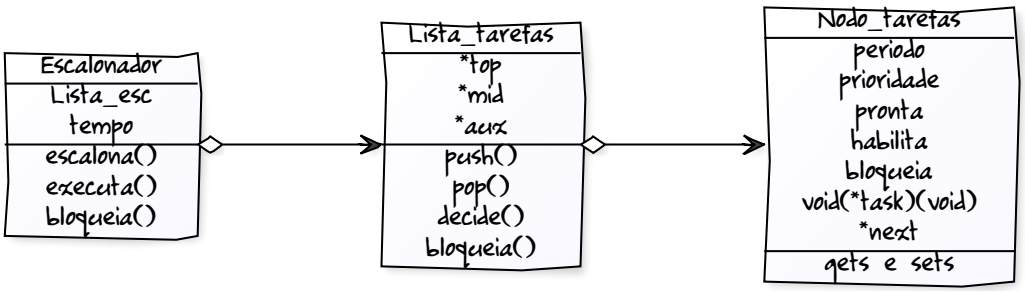

Figura 5: Diagrama expandido das classes envolvidas no escalonamento.

As tarefas que são inseridas nessa lista são às refentes à máquina de vendas, ao Banco de Dados, e às classes de manutenção do horário e calendário, conforme mostrado no código abaixo:

```
1  int main()
2  {
3      Escalonador Escalonador;
4
5      //periodo, prioridade, pronta, habilitada, bloqueada, funcao
6
7      Escalonador.escalona(30, 1, 0, 1, 0, &sell);
8
9      Escalonador.escalona(50, 2, 0, 1, 0, &repor);
10
11     Escalonador.escalona(1000, 3, 0, 1, 0, &atualiza_tempo);
12
13     while (true)
14     {
15         Escalonador.executa();
16         usleep(1000); //1ms
17     }
18 }
```

O escalonamento do ClockCalendar, por exemplo, na linha 11, recebeu um período de 1000 ms pois deve ser chamado a cada 1 segundo para atualizar a hora. Este

também recebeu a prioridade mais alta entre as tarefas para evitar erros cumulativos na manutenção do horáio. A função sell() chama a função venda() de machine, no entanto isso é feito através da criação de uma thread, pois no caso da interface com o microcomputador a função get_entrada() da classe InterfacePC utiliza o comando cin, o qual é impossível interromper sem que o usuário pressione enter. Dessa forma, chamando como uma thread, não há um boqueio do escalonador por esperar o retorno desta função.

6. Testes

Como estratégia para os testes foram utilizados diversos mains locais, que possibilitaram testar extensivamente as os diversos módulos do projeto acarretando na correção de diversos casos de borda e comportamentos inesperados. Foram criados mains locais para as classes do banco de dados, do escalonador, da máquina de vendas e do calendário e do clock.

No caso do Escalonador, por exemplo, foram criadas 4 funções de teste com períodos e prioridades distintas e passadas para a fila de tarefas através de um main local. Foi possível escalonar essas tarefas para 500 unidades de tempo e verificar o desempenho do algoritmo em casos de conflito de execução, precedência de prioridades e assim por diante. O código deste main bem como seu resultado é apresentado no APÊNDICE 3.

7. Considerações finais

A solução de escalonamento apresentada neste relatório não é preemptiva uma vez que não foi encontrada uma maneira de interromper e salvar contexto de uma tarefa em execução sem ter acesso aos registros de baixo nível. Para evitar que o escalonador fique preso majoritariamente na função de venda, esperando input, foi usada estratégia de criação de uma thread para esta função. Para as demais funções não foi considerado necessária a criação de threads, já que suas execuções são relativamente rápidas.

A estrutura modular e com possibilidade de reuso proporcionada pela orientação a objetos facilitou muito a organização e o desenvolvimento dos códigos. Foi possível implementar um código muito bem estruturado e "limpo", com detalhes das implementações ficando dentro das respectivas classes e funções. Com algumas modificações e adaptações foi possível reutilizar as classes de nodo e lista do banco de dados para o escalonador. Além disso, o uso de bibliotecas prontas para realizar tarefas específicas foi cruicial, como o uso da pthread para criação de threads.

Algumas funções importantes não puderam ser usadas na plataforma Atlys, como a função usleep(), to_string() e pthread_create(). Alternativas foram implementadas para substitui-las como, por exemplo, um for para o delay. Os relatórios de compilação, tanto no Linux como na plataforma alvo, estão incluidos nos Apêndices 4 e 5.

O projeto foi desafiador e possibilitou grande aprendizado ao longo do desenvolvimento, seja sobre as regras do C++, funcionamento de estruturas de dados ou mesmo sobre o paradigma de orientação a objeto.

APÊNDICE 1 - Manual do usuário

Para compilar o código atual em um sistema Linux com o g++, basta definir PC (#define PC) e digitar as linhas de comando abaixo:

```
g++ -o exe main.cpp -pthread
./exe
```

Para compilar o código atual no sistema embarcado alvo, basta definir ATL (#define ATL) salvar e seguir os passos do tutorial de uso da Placa Atlys. Note que não há necessidade do uso da pthread neste caso.

Com o programa em execução, para usar a máquina de refrigerantes basta inserir moedas seguindo o padrão "m"+ valor. Por exemplo, para inserir uma moeda de R$ 1,00 basta digitar "m100"e pressionar enter. Para moedas abaixo de R$ 1,00 deve-se inserir um "0"entre "m"e o valor, como em "m050". Para pedir um refrigerante do tipo MEET basta digitar "meet"e pressionar enter e, similarmente, "etirps"para um refrigerante do tipo Sprite. As devoluções de valores ocorrem no padrão "d"+ valor, sendo que para solicitar uma devolução de valores basta digitar "dev".

Para solicitar o relatório com o período de mais vendas, o refrigerante mais vendido e o valor de vendas, basta digitar "report"e pressionar enter. Há, ainda, a função de listar cada uma das vendas já realizadas desde que a máquina iniciou funcionamento, através do comando "report.all".

Para usar a classe Escalonador no seu projeto basta criar um objeto tipo Escalonador e e chamar a função escalona() com os atributos período, prioridade (quanto maior o número mais prioritária), flag de pronta, flag de habilitada, flag de bloqueada, e o endereço para a funão a ser escalonada (do tipo void (void)), respectivamente. Em seguida, para que haja execução é necessário chamar a função executa(). Sempre a que esta função for chamada haverá um incremento do tempo mantido pelo escalonador, portanto, se executa() for chamada a cada milissegundo o período das tarefas (passado como argumento da função escalona()) será dado em milissegundos.

O código abaixo é um exemplo de como usar a classee Escalonador:

```
#include "Escalonador.cpp"

int main()
{
    Escalonador Escalonador;

    Escalonador.escalona(int periodo, int prioridade, int pronta, int habilitada,
    int bloqueada, void()(void) funcao);

    while (true)
    {
        Escalonador.executa();
        usleep(1000); //1ms
    }
}
```

APÊNDICE 2 - Algoritmo de decisão do escalonador

```
1  void ListaEsc::decide(unsigned long int tempo)
2  {
3      void (*ttt)(void);
4      int exec = 0;
5      int ptr = 0;
6
7      //periodo, prioridade, pronta, habilitada, bloqueada, funcao
8
9      if (top) //se existe no na lista
10     {
11         //varre a fila para por na fila das prontas
12
13         mid = top; //meio aponta para o topo da lista
14
15         while (mid != 0) //enquanto meio nao chega no fim da lista
16         {
17             if (mid->get_habilita() && !mid->get_bloqueada()) // se
18             //a tarefa esta habilitada
19             {
20                 if ((tempo - mid->get_count()) == mid->get_periodo())
21                 // se seu periodo ja passou
22                 {
23                     mid->set_count(tempo); // momento de
24                     //execucao = momento atual
25
26                     ptr = mid->get_pronta(); // pega o valor de
27                     //quantas vezes a tarefa ficou pronta para
28                     //executar
29
30                     ptr++; //incrementa esse valor
31
32                     mid->set_pronta(ptr); // poe na fila das prontas
33
34                     ptr = 0; // zera variavel auxiliar
35
36                     if (mid->get_prioridade() > n) // se a prioridade
37                     //dessa tarefa for maior que a prioridade teto
38                     {
39                         n = mid->get_prioridade(); // seta para ser
40                         //a prioridade teto
41                     }
42                 }
43             }
44
45             mid = mid->get_next(); // meio aponta para o proximo
46             //nodo
47         }
48
49         //varre a fila novamente para executar a de maior
50         prioridade na fila das prontas
```

```
51
52          while (exec == 0) // enquanto nao executar algo varre
53          //a fila e diminui a prioridade teto
54          {
55              mid = top; //meio aponta para o topo da lista
56
57              while (mid != 0) //enquanto meio nao chega no fim da
58              //lista
59              {
60                  if (mid->get_pronta() > 0 && !exec) // entra as
61                  prontas
62                  {
63                      if (mid->get_prioridade() == n &&
64                      !mid->get_bloqueada()) // acha a que tem
65                      //prioridade igual a prioridade mais alta
66                      /entre as prontas
67                      {
68                          ptr = mid->get_pronta(); //pega o valor de
69                          pronta
70                          ptr = ptr - 1;           // diminui
71                          mid->set_pronta(ptr);    // seta
72                          ptr = 0;                 //zera
73                          //variavel auxiliar
74
75                          ttt = mid->get_void(); // pega endereco
76                          //da funcao
77                          ttt();                 // executa
78                          exec = 1;              //houve uma execucao
79                      }
80                  }
81
82                  mid = mid->get_next(); // meio aponta para o proximo
83                 // nodo
84              }
85
86              n = n - 1; // diminui a lista da maior prioridade
87
88              if (n < 0) // caso a prioridade teto desceu a zero e
89             // nada foi executado
90              {
91                  mid = top; //meio aponta para o topo da lista
92
93                  while (mid != 0) //enquanto meio nao chega no fim
94                  //da lista
95                  {
96                      if (mid->get_pronta() > 0 && !exec) // entra
97                      //as prontas
98                      {
99                          ptr = mid->get_pronta(); //pega o valor
100                        // de pronta
101                         ptr = ptr - 1;           // diminui
102                         mid->set_pronta(ptr);    // seta
103                         ptr = 0;                 //zera variavel
```

```
104                         //auxiliar
105
106                         ttt = mid->get_void(); // pega endereco
107                         //da funcao
108                         ttt();                 // executa
109                         exec = 1;              //houve uma execucao
110                     }
111
112                     mid = mid->get_next(); // meio aponta para
113                     //o proximo nodo
114                 }
115                 exec = 1;
116             }
117         }
118     }
119 }
```

APÊNDICE 3 - Main local para testes do escalonador

```
#include <iostream>
#include <string>
#include <stdlib.h>
#include <unistd.h>

using namespace std;

#include "Escalonador.cpp"

void foo()
{
    cout << "     T2  P2";
}

void foo2()
{
    cout << "     T5  P4";
}

void foo3()
{
    cout << "     T10  P3";
}

void foo4()
{
    cout << "     T1  P1";
}

int main()
{
    Escalonador Escalonador;

   //periodo, prioridade, pronta, habilitada, bloqueada, funçao
    Escalonador.escalona(1, 1, 0, 1, 0, &foo4);
    Escalonador.escalona(2, 2, 0, 1, 0, &foo);

    Escalonador.escalona(5, 4, 0, 1, 0, &foo2);
    Escalonador.escalona(10, 3, 0, 1, 0, &foo3);

    int b = 0;

    while (b < 500)
    {
        Escalonador.executa();
        b++;

        if (b == 100)
            Escalonador.bloqueia(&foo4);

        usleep(1000); //1ms
```

```
    }
}
```

Output:

```
t1      T1  P1
t2      T2  P2
t3      T1  P1
t4      T2  P2
t5      T5  P4
t6      T2  P2
t7      T1  P1
t8      T2  P2
t9      T1  P1
t10      T5  P4
t11      T10  P3
t12      T2  P2
t13      T1  P1
t14      T2  P2
t15      T5  P4
t16      T2  P2
t17      T1  P1
t18      T2  P2
t19      T1  P1
t20      T5  P4
...
```

APÊNDICE 4 - Relatório de compilação no Linux

```
ca_rigo@DESKTOP-0GOMJHJ:/mnt/d/OneDrive/_education/_doutorado/_materias/
_C++/_trabalho_final/vending_machine/vending_machine$ g++ -o exe main.cpp -pthread

In file included from Machine/../Dados/Pilha.cpp:2:0,
                 from Machine/../Dados/BancoDados.h:8,
                 from Machine/Machine.h:5,
                 from Machine/Machine.cpp:1,
                 from functions.cpp:6,
                 from main.cpp:5:
Machine/../Dados/../common.h:4:12: warning: ‘g_data’ initialized and declared
‘extern’
 extern int g_data = 0;
            ^~~~~~
Machine/../Dados/../common.h:5:12: warning: ‘g_hora’ initialized and declared
‘extern’
 extern int g_hora = 0;
            ^~~~~~
Machine/../Dados/../common.h:6:12: warning: ‘repo’ initialized and declared ‘extern’
 extern int repo = 0;
            ^~~~
Machine/../Dados/../common.h:7:12: warning: ‘repo1’ initialized and declared ‘extern’
 extern int repo1 = 0;
            ^~~~~
Machine/../Dados/../common.h:8:12: warning: ‘th’ initialized and declared ‘extern’
 extern int th = 0;
```

APÊNDICE 5 - Relatório de compilação na plataforma Atlys

```
alunos@LABSDG-03:~/Desktop/vending_machine$ sparc-elf-g++ main.cpp -o programa

In file included from Machine/../Dados/Pilha.cpp:2,
                 from Machine/../Dados/BancoDados.h:8,
                 from Machine/Machine.h:5,
                 from Machine/Machine.cpp:1,
                 from functions.cpp:6,
                 from main.cpp:5:
Machine/../Dados/../common.h:4: warning: 'g_data' initialized and declared 'extern'
Machine/../Dados/../common.h:5: warning: 'g_hora' initialized and declared 'extern'
Machine/../Dados/../common.h:6: warning: 'repo' initialized and declared 'extern'
Machine/../Dados/../common.h:7: warning: 'repo1' initialized and declared 'extern'
Machine/../Dados/../common.h:8: warning: 'th' initialized and declared 'extern'
alunos@LABSDG-03:~/Desktop/vending_machine$
```