Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina
Departamento Acadêmico de Eletrônica
Curso de Engenharia Eletrônica

Trabalho final – simulador de escalonamento de tarefas

No contexto de sistemas computacionais embarcados modernos, temos alguns elementos principais:

- Algoritmo:

  É o procedimento lógico para resolver certo problema. O algoritmo estabelece informalmente uma sequência de passos que uma máquina deve seguir. Um algoritmo não necessariamente precisa ser expresso por uma linguagem de programação.

- Programa:

  Programa é a implementação de um algoritmo utilizando uma linguagem de programação. Pode ser executado diversas vezes com diferentes entradas.

- Processo ou tarefa (τ):

  Processo é uma instância de um programa, ou seja, dado um conjunto de entradas, esta instância produz um conjunto de saídas.

- Sistema operacional:

  É um programa que provê abstrações da máquina através de interfaces simples. Estas interfaces são conhecidas como serviços. O sistema operacional também pode ser visto como um gerente de recursos, sendo o tempo, um destes recursos. Dado que uma máquina executa inúmeros processos simultaneamente, é dever do Sistema Operacional elaborar uma escala de execução dos processos ativados como no exemplo da Figura 1.

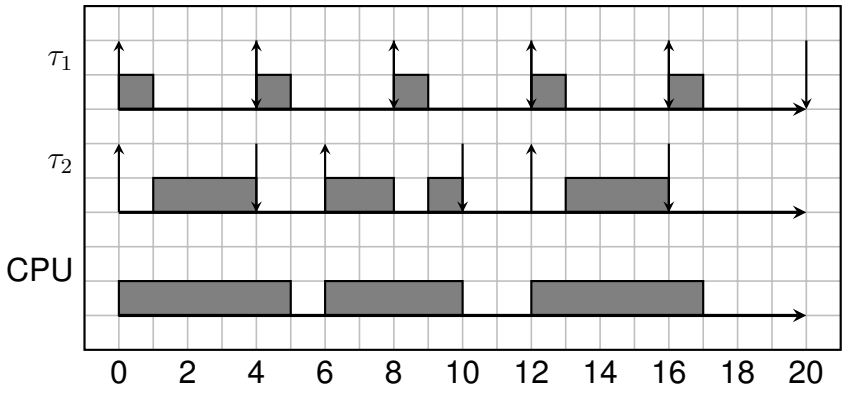

Figura 1: Exemplo de escalonamento para duas tarefas.

Neste sistema, temos apenas um processador que é compartilhado por duas tarefas: τ_1 e τ_2. Percebe-se que τ_1 possui prioridade superior a τ_2 pois ela é interrompida no instante de tempo 8 para execução de τ_1.

No contexto de sistemas embarcados, tarefas ou processos são modelados por:

- C: tempo de computação total. É o tempo que processo leva para ser executado completamente.
- T: período de ativação.

No caso da Figura 1, $C_{\tau_1} = 1$ e $T_{\tau_1} = 4$. Isto significa que τ_1 executa por 1 unidade cada 4 unidade de tempo, ou seja, suas ativações são dadas a cada 4 unidades de tempo. O mesmo vale para τ_2, com $C_{\tau_2} = 3$ e $T_{\tau_2} = 6$.

Na Figura 1, setas apontando para cima representam quando um tarefa está apta para executar, ($t_{sis} \bmod T_\tau == 0$), e setas apontando para baixo representam quando termina sua execução.

Conforme a contextualização acima, o trabalho final da disciplina consiste em elaborar um escalonador de tarefas. Este escalonador é um simulador que deve determinar a execução das tarefas. O tempo da simulação poderá evoluir a cada **um segundo**, caso deseja-se exibir na tela a progressão do sistema.

- Cada dupla criará um arquivo de texto com a especificação das tarefas para provar o funcionamento. Durante a avaliação, outro arquivo será fornecido para o simulador.
  - Cada tarefa será modelada por um C e um T.
  - A prioridade de cada tarefa é inversamente proporcional ao seu período, ou seja, quanto menor o período, maior a prioridade.
  - $C \in \mathbb{Z}$ e $T \in \mathbb{Z}$.
  - $\forall \tau_i, C_{\tau_i} < T_{\tau_i}$
  - Mínimo de 10 tarefas.
- O arquivo de entrada deve ser dinâmico especificado pela linha de comando: use *main(int argc, char** argv).*
- A simulação deve executar até, no mínimo, o hiper período das tarefas: Mínimo múltiplo comum dos períodos
- O diagrama de execução, conforme a Figura 1 **deve** ser gerado.

  - Este diagrama é facilmente criado utilizando LaTeX
  - Ver: exemplo-gannt.tex
  - Ver: rtsched-doc.pdf

- É **obrigatório** a utilização de estruturas de dados.
- É **obrigatório** que o projeto utilize o conceito de tipos de dados abstratos.
- Entrega e elaboração pelo GitHub. O projeto deve utilizar o repositório em **todo** o desenvolvimento. **Upload** somente no final degrada o conceito da avaliação.
- Escolha correta dos algoritmos e estruturas de dados faz parte da avaliação.
- Utilização de bibliotecas gráficas, *allegro* ou *ncurses*, eleva o conceito do trabalho.
- Utilização do *doxygen* para documentação do código eleva o conceito do trabalho.
- Entrega:
  - Código fonte
  - Manual de funcionamento

Sugestões:

- Utilize uma fila de prioridades, fila de tarefas aptas para executar.
- Utilize listas encadeadas.
- Sugestões de estrutura e estado:

```
// Estados das tarefas
enum TASK_STATE {
  P_STOPPED,            /* Nada a fazer */
  P_RUNNIG,             /* Rodando */
  P_READY,              /* Pronta para executar */
  P_BLOCKED             /* Bloqueda, tempo */
};

struct task
{
  // prioridade
  unsigned char uc_priority;

  // identificador
```

```
  unsigned char uc_id;

  //estado atual
  TASK_STATE uc_status;

  //tempo do sistema na última execução */
  unsigned long ul_ticks;

  //Período de execução, rotinas temporizadas, bloqueio por tempo */
  unsigned long ul_period_ticks;

  //Quantidade de tempo restante para bloqueio */
  unsigned long l_ticks2block;

  // Quantidade de tempo executado
  unsigned long l_exec_ticks;

  // Tempo de execução
  unsigned long ul_comp_time;
};
```

- Especificação do arquivo de tarefas (campos serão separado por ";"):

```
N;5        //Primeira linha especifica a quantidade de tarefas
id;C;T     //Segunda linha:  descrição das colunas
T1;2;7     //Lista das tarefas, não estará ordenado por prioridade
T2;1;10
T3;6;18
T4;4;16
T5;10;30
```