



## INF1316 - Laboratório 04 - Sinais e escalonamento de processos em tempo real

**Nome:**

- Guilherme Riechert Senko
- Pedro de Almeida Barizon

**Matrícula:**

- 2011478
- 2211350

**Professor:** Luiz Fernando Seibel

**Turma:** 3WA

**Data:** 22/09/2024

**Objetivo:** implementar e compreender um escalonador de tempo real (*REAL-TIME*).

**Observações preliminares**

- Para mais detalhes, todos os códigos fonte comentados encontram-se em anexo. Por isso, para evitar redundâncias, não serão aqui exibidos.
- Todos os arquivos fonte foram compilados com auxílio do *GNU C Compiler* (GCC) fazendo `gcc -Wall -o programa fonte1.c fonte2.c ... fonten.c` e executados com `./programa arg1 arg2 ... argn`. Caso haja particularidades, estas serão abordadas ao longo do relatório.

**Exercício 1)** Na política de escalonamento **REAL-TIME** cada processo deve executar periodicamente (uma vez por minuto), iniciando sua execução, em determinado momento de tempo (I) e deve permanecer executando apenas durante um certo período de tempo (D).

Escreva em C um programa escalonador que execute um loop infinito criando três processos filho, p1, p2 e p3, e que os escalona da seguinte maneira (a cada minuto):

- P1 inicia após 5 segundos (do início do minuto cheio) e executa por 20 segundos
- P2 inicia após 45 segundos (do minuto cheio) e executa durante 15 segundos
- P3 executa quando nem P1 e nem P2 executam

## 1.1 Estrutura

Criou-se apenas um módulo, `escalonador_real_time.c`, que continha as funções:

- `main`: responsável pela execução do programa.
- `escalona`: responsável pela interrupção de um processo ou (inclusive) pela continuação de outro. Atualiza os estados dos processos.
- `exibe`: responsável pela exibição do processo interrompido ou (inclusive) pela do processo continuado. Inclui nas saídas o instante de tempo passado como parâmetro.
- `escalona_exibe`: *wrapper* de `escalona` e de `exibe`.
- `verifica_escalona_exibe`: *wrapper* de `escalona_exibe`. Verifica se os processos passados como parâmetro possuem os estados adequados para escalonamento. Se tiverem, executa `escalona_exibe`; do contrário, ignora.

## 1.2 Solução

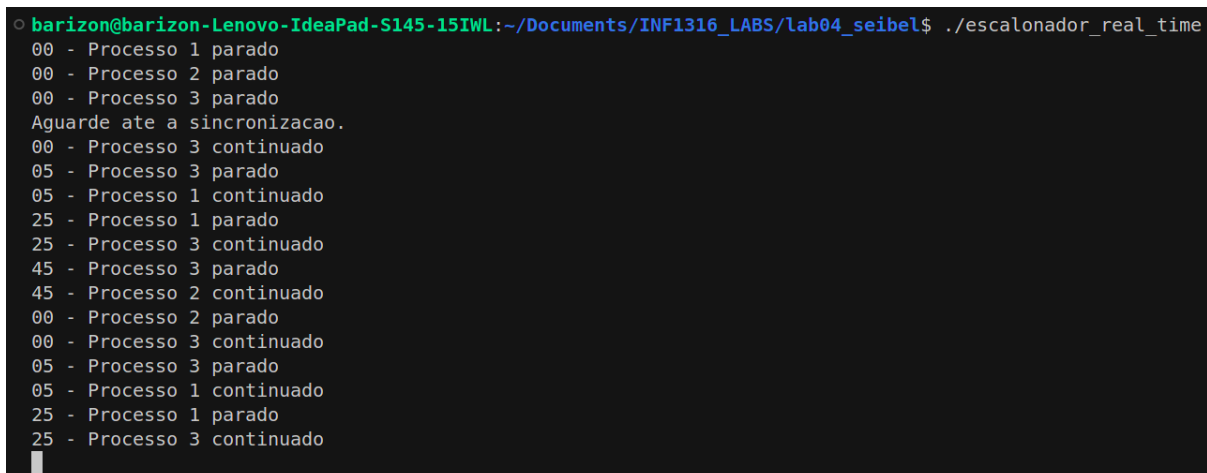
Inicialmente, define-se uma estrutura de tipo `Processo`, chamada `processo`, que contém um vetor de PIDs `pid` e outro de estados (`INATIVO` ou `ATIVO`), denominado `estado`. Além disso, define-se uma variável de tipo `timeval`, chamada `instante`, e outra inteira chamada `secs`. Isso feito, o processo coordenador cria três filhos com `fork`, cada qual a executar um *loop* eterno, e armazena seus PIDs em `pid`. Em seguida, são postos em espera e seus estados são exibidos com o uso de `escalona_exibe`.

Logo após, realiza-se a sincronização: o processo 3 deverá dar início ao ciclo **REAL-TIME** de escalonamento no instante 0 do novo minuto. Para tanto:

1. Armazena-se o número de segundos transcorridos desde 01/01/1970 com `gettimeofday` em `instante.tv_sec`.
2. Para obter a quantidade de segundos transcorridos no minuto atual, calcula-se o resto da divisão por 60 de `instante.tv_sec`, sendo o seu valor atribuído a `secs`.
3. Se `secs == 0`, estamos no início do novo minuto, logo se pode começar o escalonamento. Do contrário, repete-se o ciclo a partir da etapa 1.

Assim, o processo 3 é iniciado com `escalona_exibe`. A partir disso, entra-se em um *loop* infinito de escalonamento, no qual:

1. Repetem-se as mesmas etapas 1 e 2 vistas na sincronização.
2. Um `switch` baseado no valor de `secs` faz executar o caso de `verifica_escalona_exibe` adequado.



```
barizon@barizon-Lenovo-IdeaPad-S145-15IWL:~/Documents/INF1316_LABS/lab04_seibel$ ./escalonador_real_time
00 - Processo 1 parado
00 - Processo 2 parado
00 - Processo 3 parado
Aguarde ate a sincronizacao.
00 - Processo 3 continuado
05 - Processo 3 parado
05 - Processo 1 continuado
25 - Processo 1 parado
25 - Processo 3 continuado
45 - Processo 3 parado
45 - Processo 2 continuado
00 - Processo 2 parado
00 - Processo 3 continuado
05 - Processo 3 parado
05 - Processo 1 continuado
25 - Processo 1 parado
25 - Processo 3 continuado
```

Figura 1.1 - Saída de `escalonador_real_time`

### 1.3 Observações e conclusões

Vale ressaltar a necessidade da parte “`verifica`”: como o tempo de execução de uma iteração do *loop* de escalonamento é muito menor que um segundo, se não houvesse verificação, um mesmo caso de `escalona_exibe` seria executado centenas de vezes, o que, além de lotar o *log* na console (“`exibe`”), enviaria sinais redundantes aos processos (“`escalona`”). Afinal, mandaria parar um processo já parado e continuar um processo já ativo. Eis a relevância do vetor estado, que permitiu a execução do escalonamento sem redundâncias. Constatar tal necessidade foi a única dificuldade de implementação.

## **2 Conclusões finais**

Os programas foram compilados sem erro usando-se o GCC, o ambiente de desenvolvimento do Visual Studio Code e o sistema operacional Ubuntu. Ademais, conforme se observa na saída do programa, todos os testes obtiveram êxito. Por fim, devemos reconhecer a importância deste laboratório, uma vez que nos permitiu vislumbrar o grande poder do envio de sinais entre processos em tempo real, que nos possibilitou construir um pequeno escalonador com política *REAL-TIME*.