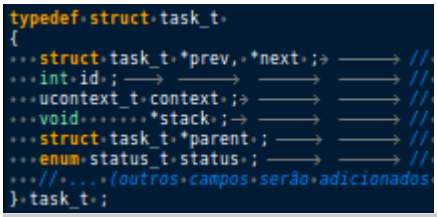


**Carlos Maziero**

# Gerente de disco

Este projeto tem por objetivo implementar operações de entrada/saída (leitura e escrita) de blocos de dados sobre um disco rígido virtual (simulado).

## O problema

A tarefa principal (`main`) deve inicializar o disco através da seguinte chamada:

```
int disk_init (&numblocks, &blocksize) ;
```

Ao retornar da chamada, a variável `numblock` contém o número de blocos do disco inicializado, enquanto a variável `blocksize` contém o tamanho de cada bloco, em bytes. A chamada retorna 0 em caso de sucesso ou -1 em caso de erro.

As tarefas devem acessar o disco virtual através das seguintes chamadas bloqueantes:

```
int disk_block_read  (int block, void* buffer) ;
int disk_block_write (int block, void* buffer) ;
```

- `block`: posição (número do bloco) a ler ou escrever no disco;
- `buffer`: endereço dos dados a escrever no disco, ou onde devem ser colocados os dados lidos do disco;
- retorno: 0 em caso de sucesso ou -1 em caso de erro.

Cada tarefa que solicita uma operação de leitura/escrita no disco **deve ser suspensa** até que a operação seja completada. As tarefas suspensas devem ficar em uma fila de espera associada ao disco. As solicitações de leitura/escrita devem ser atendidas na ordem em que foram geradas, de acordo com a política de escalonamento de disco FCFS (*First Come, First Served*).

## O disco virtual

O "disco virtual" simula o comportamento lógico e temporal de um disco rígido, com as seguintes características:

- O conteúdo do disco virtual é mapeado em um arquivo UNIX no diretório corrente da máquina real, com nome default `disk0.dat`. O conteúdo do disco virtual é preservado de uma execução para outra.
- O disco está estruturado em N blocos de mesmo tamanho. O número de blocos do disco dependerá do tamanho do arquivo subjacente, no sistema real.
- As operações de leitura/escrita em um disco são feitas sempre com um bloco de cada vez. Não é possível ler ou escrever bytes isolados, parte de um bloco, ou vários blocos ao mesmo tempo.
- Os pedidos de leitura/escrita no disco são assíncronos, ou seja, apenas registram a operação solicitada, sem bloquear a chamada. A finalização de cada operação de leitura/escrita é indicada posteriormente pelo disco, através de um sinal UNIX `SIGUSR1`.
- O disco só trata uma leitura/escrita por vez. Enquanto o disco está tratando uma solicitação, ele fica em um estado ocupado (*busy*); tentativas de acesso a um disco ocupado retornam um código de erro.
- O tempo de resposta do disco é proporcional à distância entre a posição atual da cabeça de leitura do disco e a posição da operação solicitada. Inicialmente a cabeça de leitura está posicionada sobre o bloco zero (0).

O código que simula o disco está em `harddisk.c` e sua interface de acesso está definida em `harddisk.h`; estes arquivos **não devem ser modificados**.

O acesso ao disco deve feito **somente** através das definições presentes em `harddisk.h`. As definições presentes em `harddisk.c` implementam (simulam) o comportamento interno do disco e por isso **não devem ser usadas** em seu código.

Atenção: o arquivo `harddisk.c` depende da biblioteca POSIX de tempo-real (`-lrt`) para ser ligado ao seu código:

```
cc pingpong.c queue.c harddisk.c pingpong-disco.c -lrt
```

# O que deve ser feito

A gerência das operações de entrada/saída em disco consiste em:

- Criar e manter uma fila de pedidos de acesso ao disco;
- Implementar as funções de acesso ao disco oferecidas às tarefas, definidas no arquivo de interface diskdriver.h;
- Tratar cada sinal `SIGUSR1` gerado pelo disco, acordando a tarefa cuja solicitação foi atendida;

Sua implementação deverá funcionar com este código de teste e este conteúdo inicial do disco virtual, que tem 256 blocos de 64 bytes cada ($b_0$, $b_1$, …, $b_{255}$), totalizando 16.384 bytes. Ao concluir a execução, o disco virtual deverá ter este conteúdo final, que corresponde ao disco inicial com os blocos escritos na ordem inversa ($b_{255}$, $b_{254}$, …, $b_0$). O conteúdo inicial do disco foi definido para facilitar seus testes, pois cada bloco contém uma string padrão com comprimento de 63 bytes, seguida por um *newline* (`\n`). A saída em tela de uma execução típica pode ser vista neste arquivo.

Como sugestão para simplificar a implementação, pode-se definir uma **tarefa** *gerente de disco*, responsável por tratar os pedidos de leitura/escrita das tarefas e os sinais gerados pelo disco. Essa tarefa de sistema tem +/- o seguinte comportamento:

```
void diskDriverBody (void * args)
{
   while (1)
   {
      // obtém o semáforo de acesso ao disco

      // se foi acordado devido a um sinal do disco
      if (disco gerou um sinal)
      {
         // acorda a tarefa cujo pedido foi atendido
      }

      // se o disco estiver livre e houver pedidos de E/S na fila
      if (disco_livre && (fila de pedidos de acesso ao disco não está vazia))
      {
         // escolhe na fila o pedido a ser atendido, usando FCFS
         // solicita ao disco a operação de E/S, usando disk_cmd()
      }

      // libera o semáforo de acesso ao disco

      // suspende a tarefa corrente (retorna ao dispatcher)
   }
}
```

A tarefa gerente de disco deve ser acordada (voltar à fila de tarefas prontas) sempre que:

- alguma tarefa pedir uma operação de leitura/escrita no disco; ou
- o disco gerar um sinal indicando que a última operação solicitada foi concluída.

Dessa forma, as funções `disk_block_read` e `disk_block_write` devem seguir +/- o seguinte comportamento:

```
disk_block_read (block, &buffer)
{
   // obtém o semáforo de acesso ao disco

   // inclui o pedido na fila de pedidos de acesso ao disco

   if (gerente de disco está dormindo)
   {
      // acorda o gerente de disco (põe na fila de prontas)
   }

   // libera semáforo de acesso ao disco

   // suspende a tarefa corrente (retorna ao dispatcher)
}
```

## Outras informações

- Duração estimada: 8 horas.
- Dependências:
    - Gestão de Tarefas
    - Dispatcher
    - Preempção por Tempo
    - Tarefa Main
    - Operador Join
    - Semáforos

---