Projeto PingPongOS

versão 20241107

Autor Original do Projeto
Prof. Dr. Carlos Maziero
http://wiki.inf.ufpr.br/maziero/doku.php?id=so:pingpongos

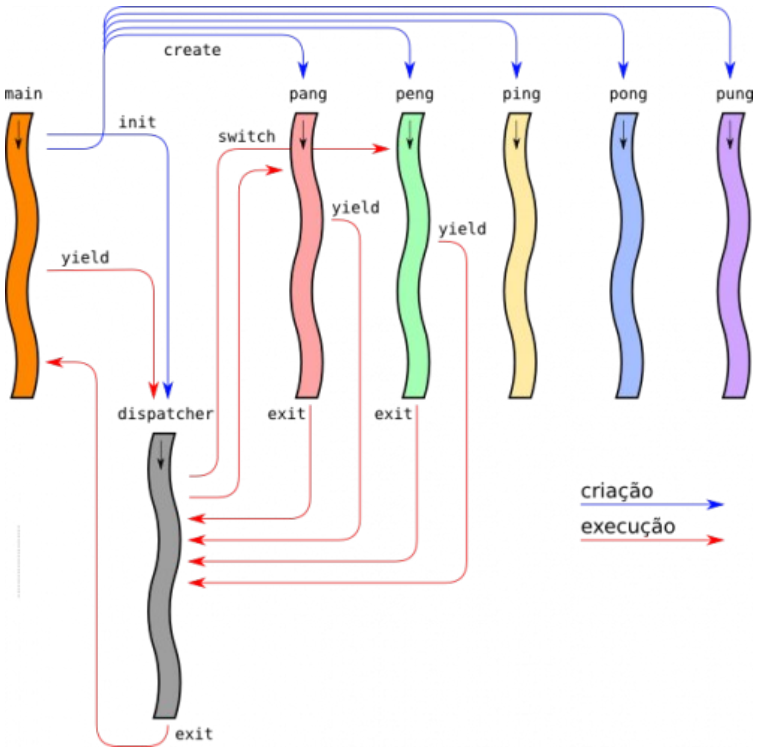
Adaptado por
Prof. Dr. Marco Aurélio Wehrmeister
para o contexto da disciplina CSO30 dos cursos de
Engenharia da Computação e Sistemas de Informação da UTFPR

INTRODUÇÃO

O projeto PingPongOS visa construir um pequeno sistema operacional didático. O sistema é construído inicialmente na forma de uma biblioteca de threads cooperativas dentro de um processo do sistema operacional real (Linux, MacOS ou outro Unix).

O PingPongOS deve fornecer algumas funcionalidades como preempção, contabilização, semáforos, filas de mensagens, gerenciamento de páginas e acesso a um disco virtual. **Algumas funcionalidades já estão implementadas e disponíveis para uso, enquanto outras precisam ser implementadas pelos estudantes.** Essa abordagem simplifica o desenvolvimento e depuração do núcleo, além de dispensar o uso de linguagem de máquina.

A figura abaixo mostra uma visão geral da execução do sistema multitarefa PingPongOS usando cinco tarefas concorrentes do sistema (pang, peng, ping, pong, pung); independentemente do número de tarefas do sistema, o esquema de execução é o mesmo.



No contexto das disciplinas "CSO30 – Sistemas Operacionais", turmas S71 (curso de Engenharia de Computação) e S73 (curso de Sistemas de Informação), a implementação de várias funções do PingPongOS é fornecida pelo professor. Os alunos deverão completar a implementação conforme a descrição dos dois projetos apresentada nas próximas seções.

Veja o vídeo que explica a estrutura do código fonte e do projeto fornecido pelo professor. LINK NO MOODLE

ATENÇÃO IMPORTANTE: LEIA OS COMENTÁRIOS no código fonte, em especial, em *ppos.h, queue.h, ppos-core-globals.h*. Várias das dúvidas frequentes dos alunos e das alunas, sobre a implementação, ocorrem por eles e elas não leem os comentários nos arquivos fornecidos.

PROJETO A – Implementação de parte do serviço de sincronização através de semáforos

Objetivos e Requisitos

1. Implementar semáforos clássicos no PingPongOS.

1. Cria um semáforo

```
int sem_init (semaphore_t *s, int value)
```

Inicializa um semáforo apontado por s com o valor inicial value e uma fila vazia. O tipo `semaphore_t` deve ser definido no arquivo *ppos-data.h*.

2. Requisita um semáforo

```
int sem_down (semaphore_t *s)
```

Realiza a operação *Down* no semáforo apontado por s. Esta chamada pode ser bloqueante: caso o contador do semáforo seja negativo, a tarefa corrente é suspensa, inserida no final da fila do semáforo e a execução volta ao dispatcher; caso contrário, a tarefa continua a executar sem ser suspensa.

Se a tarefa for bloqueada, ela será reativada quando uma outra tarefa liberar o semáforo (através da operação `sem_up`) ou caso o semáforo seja destruído (operação `sem_destroy`).

A chamada retorna 0 em caso de sucesso ou -1 em caso de erro (semáforo não existe ou foi destruído).

3. Libera um semáforo

```
int sem_up (semaphore_t *s)
```

Realiza a operação *Up* no semáforo apontado por s. Esta chamada não é bloqueante (a tarefa que a executa não perde o processador). Se houverem tarefas aguardando na fila do semáforo, a primeira da fila deve ser acordada e retornar à fila de tarefas prontas.

A chamada retorna 0 em caso de sucesso ou -1 em caso de erro (semáforo não existe ou foi destruído).

4. Destrói um semáforo

```
int sem_destroy (semaphore_t *s)
```

Destrói o semáforo apontado por s, acordando todas as tarefas que aguardavam por ele.

A chamada retorna 0 em caso de sucesso ou -1 em caso de erro.

As tarefas que estavam suspensas aguardando o semáforo que foi destruído devem ser acordadas e a operação *Down* correspondente de indicar um código de erro (valor de retorno -1).

Observações:

1. **Condições de Disputa**: Caso duas *threads* tentem acessar o mesmo semáforo simultaneamente, podem ocorrer condições de disputa nas variáveis internas dos semáforos. Por isso, as funções que implementam os semáforos devem ser protegidas usando um mecanismo de exclusão mútua. Como obviamente não podemos usar semáforos para resolver esse problema m(, deve ser usado um mecanismo mais primitivo.
2. O arquivo *ppos-data.h* contém os tipos de dados (incompletos) e o arquivo ppos.h os protótipos das funções a serem implementadas em *ppos-core-aux.c* (opcionalmente, podem ser usado um arquivo separado *ppos-ipc.c*, para abrigar a implementação das funções de IPC).
3. Os semáforos deverão ser totalmente implementados em seu código; sua implementação não deverá usar funções de semáforo de bibliotecas externas ou do sistema operacional subjacente.
4. Você deverá definir a estrutura *semaphore_t* (observe que cada semáforo deve ter seu próprio contador e sua própria fila).
5. Sua implementação deve funcionar com o programa de teste pingpong-semaphore.*c*, *pingpong-racecond.c*, e *pingpong-mqueue.c* e deve gerar um resultado similar ao da saída disponível nos respectivos arquivos *.*txt*.

PROJETO B – Implementação de um Gerenciador de Memória com Paginação em Disco

Objetivos e Requisitos

1. Implementar um gerenciador de memória com paginação em disco
2. Usar o mecanismo de acesso ao disco (virtual) já disponível no PingPongOS
3. Implementar um escalonador de requisições de acesso ao disco (virtual)

1. Implementar um gerenciador de memória com paginação em disco

Considerando um computador com uma quantidade limita de 640 bytes memória RAM, com páginas de 64 bytes. O computador possui um disco virtual com tamanho de 1280 bytes.

OS DEMAIS REQUISITOS DESTA PARTE SERÃO DISPONIBILIZADO ANTES DO INICIO DO RECESSO DE FIM DE ANO.

2. Usar o mecanismo de acesso ao disco (virtual) já disponível no PingPongOS

As operações de entrada/saída (leitura e escrita) de blocos de dados sobre um disco rígido virtual são realizadas através de um gerenciador de disco que está implementado no PingPongOS.

1.1 Interface de acesso ao disco

A tarefa principal de uma aplicação (*main*) inicializa o gerente/driver de disco através da chamada `int disk_mgr_init (&num_blocks, &block_size);`

Ao retornar da chamada, a variável *num_blocks* contém o número de blocos do disco inicializado, enquanto a variável *block_size* contém o tamanho de cada bloco do disco, em bytes. Essa chamada retorna 0 em caso de sucesso ou -1 em caso de erro.

As tarefas podem ler e escrever blocos de dados no disco virtual através das seguintes chamadas (ambas bloqueantes):

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

onde:

- *block*: posição (número do bloco) a ler ou escrever no disco (deve estar entre 0 e numblocks-1);
- *buffer*: endereço dos dados a escrever no disco, ou onde devem ser colocados os dados lidos do disco; esse buffer deve ter capacidade para block_size bytes.
- *retorno*: 0 em caso de sucesso ou -1 em caso de erro.
-

Cada tarefa que solicita uma operação de leitura/escrita no disco é suspensa até que a operação solicitada seja completada. As tarefas suspensas ficam em uma fila de espera associada ao disco. As solicitações de leitura/escrita presentes nessa fila devem ser atendidas na ordem conforme a política de escalonamento vigente (a implementação fornecida originalmente no PingPongOS é a FCFS – *First Come, First Served*)

1.2 O disco virtual

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

- O conteúdo do disco virtual é mapeado em um arquivo UNIX no diretório corrente da máquina real, com nome default *disk.dat*. O conteúdo do disco virtual é mantido de uma execução para outra.
- O disco contém N blocos de mesmo tamanho. O número de blocos do disco dependerá do tamanho do arquivo subjacente no sistema real. No caso deste exercício, o tamanho do disco é o tamanho do arquivo *disk.dat.*
- Como em um disco real, as operações de leitura/escrita 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 feitos ao disco são assíncronos, ou seja, apenas registram a operação solicitada, sem bloquear a chamada. A finalização de cada operação de leitura/escrita é indicada mais tarde pelo disco através de um sinal UNIX SIGUSR1, que é capturado e tratado.

- 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 inicial (zero).

Os arquivos *pingpong-disco1.c e pingpong-disco2.c* apresentam exemplos de aplicações que usam o gerenciador de discos. ***Você deve estudá-las para entender como usar o gerenciador de discos.*** *Esses arquivos de teste do disco s****ó funcionam se e somente se as funções do semáforo estiverem implementadas*** *corretamente.*

3. Implementar um escalonador de requisições de acesso ao disco (virtual)

ESTE ITEM AINDA ESTÁ COM A DESCRIÇÃO INCOMPLETA. NO ENTANTO, ELA SERÁ FORNECIDA ANTES DO INICIO DO RECESSO DE FIM DE ANO.

O objetivo é implementar diferentes algoritmos de escalonamento de acessos a um disco simulado. A política de escalonamento dos acessos a discos rígidos tem um impacto importante no *throughput* de um sistema (número de bytes lidos ou escritos no disco por segundo). Algumas políticas bem conhecidas são:

- *First Come, First Served* (FCFS): os pedidos são atendidos na ordem em que são gerados pelas tarefas; sua implementação é simples, mas não oferece um bom desempenho;
- *Shortest Seek-Time First* (SSTF): os acessos a disco são ordenados conforme sua distância relativa: primeiro são atendidos os pedidos mais próximos à posição atual da cabeça de leitura do disco.
- *Circular Scan* (CSCAN): os pedidos são atendidos sempre em ordem crescente de suas posições no disco; após tratar o pedido com a maior posição, a cabeça do disco retorna ao próximo pedido com a menor posição no disco.

3.1 Requisitos para a Implementação

Utilizando o gerente de disco desenvolvido anteriormente, o escalonador de requisições de acesso deve:

- Implementar as políticas de escalonamento de disco SSTF e CSCAN;
- Implementar uma função separada que implementa cada uma dessas políticas;
- Medir o número de blocos percorridos pela cabeça do disco em cada uma das políticas indicadas;
- Medir o tempo total de execução para cada uma das políticas.
- Use o código de teste e os conteúdos de disco do projeto de gerente de disco para efetuar seus testes e medições.