Construção de semáforos

Video deste projeto

O objetivo deste projeto é implementar **semáforos clássicos** em nosso sistema. As funções a implementar são descritas na sequência.

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 `pos_data.h`.

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

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).

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).

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 retornar da operação *Down* correspondente com um código de erro (valor de retorno -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 , deve ser usado um mecanismo mais primitivo, baseado em espera ocupada. Para tal, Sugere-se usar operações atômicas.

Observações:

- 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.c` (opcionalmente, podem ser usado um arquivo separado `ppos_ipc.c`, para abrigar a implementação das funções de IPC).
- 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.
- Você deverá definir a estrutura `semaphore_t` (observe que cada semáforo deve ter seu próprio contador e sua própria fila).

Sua implementação deve funcionar com estes dois arquivos de teste:

- pingpong-semaphore.c e sua saída
- pingpong-racecond.c e sua saída

Outras informações

- Duração estimada: 4 horas.
- Dependências:
    - Gestão de Tarefas
    - Dispatcher
    - Preempção por Tempo
    - Tarefas suspensas
    - Tarefas dormindo