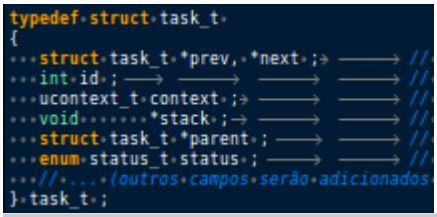

Carlos Maziero

Gestão de Tarefas

Objetivo: construir as funções básicas de gestão de tarefas usando as funções de troca de contexto vistas no projeto anterior.

Descritor de tarefa

Esta estrutura é o descritor de cada tarefa (*TCB - Task Control Block*), a ser definida no arquivo `datatypes.h`:

```
typedef struct task_t
{
   struct task_t *prev, *next ;   // para usar com a biblioteca de filas (cast)
   int tid ;                      // ID da tarefa
   ...                            // demais informações da tarefa
} task_t ;
```

Interface

As seguintes funções devem ser implementadas:

Inicializa o sistema

```
void pingpong_init ()
```

Esta função inicializa as estruturas internas do SO. Por enquanto, conterá apenas algumas inicializações de variáveis e a seguinte instrução, que desativa o buffer utilizado pela função `printf`, para evitar condições de disputa que podem ocorrer nesse buffer ao usar as funções de troca de contexto:

```
/* desativa o buffer da saida padrao (stdout), usado pela função printf */
setvbuf (stdout, 0, _IONBF, 0) ;
```

Cria uma nova tarefa

```
int task_create (task_t *task, void (*start_routine)(void *),  void *arg)
```

- `task`: estrutura que referencia a tarefa criada
- `start_routine`: função que será executada pela tarefa
- `arg`: parâmetro a passar para a tarefa que está sendo criada
- retorno: o ID da task ou valor negativo, se houver erro

Atenção: deve ser previsto um descritor de tarefa que aponte para o programa principal (que exercerá a mesma função da variável `ContextMain` no programa `contexts.c`).

Transfere o processador para outra tarefa

```
int task_switch (task_t *task)
```

- `task`: tarefa que irá assumir o processador
- retorno: valor negativo se houver erro, ou zero

Esta é a operação básica de troca de contexto, que encapsula a função `swapcontext`. Ela será chamada sempre que for necessária uma troca de contexto.

Termina a tarefa corrente

```
void task_exit (int exit_code)
```

- `exit_code` : código de término devolvido pela tarefa corrente (ignorar este parâmetro por enquanto, pois ele somente será usado mais tarde)

Quando uma tarefa encerra, o controle deve retornar à tarefa `main`. Esta chamada será implementada usando `task_switch`.

Informa o identificador da tarefa corrente

```
int task_id ()
```

- retorno: Identificador numérico (ID) da tarefa corrente, que deverá ser 0 para `main`, ou um valor positivo para as demais tarefas. Esse identificador é único: não devem existir duas tarefas com o mesmo ID.

Observações

A implementação completa deste projeto compreende definir os tipos e estruturas de dados necessários para gerenciar as tarefas e implementar as funções acima descritas, comentando detalhadamente o código.

A convenção de estruturação de código em C deverá ser respeitada:

- `datatypes.h`: definições de dados globais (esqueleto: datatypes.h)
  - constantes e macros globais
  - definições dos tipos e estruturas de dados globais
- `pingpong.h`: interface do SO (fornecido: pingpong.h, **não alterar**)
  - protótipos das funções públicas
- `pingpong.c`: contém as definições internas do sistema:
  - constantes e macros internas
  - definições dos tipos e estruturas de dados internas
  - funções internas
  - implementações das funções públicas

Capriche na implementação, pois esse código será a base de todos os projetos posteriores.

Validação

Seu código deve funcionar adequadamente com os programas de teste abaixo indicados, fornecendo as saídas esperadas:

- programa de teste 1 e sua saída esperada
- programa de teste 2 e sua saída esperada
- programa de teste 3 e sua saída esperada

O primeiro programa de teste corresponde ao código contexts.c do projeto anterior, reescrito com as novas funções. Comparar os dois códigos pode ajudar a compreender o que deve ser implementado em cada função.

Compile da seguinte forma:

```
cc pingpong.c pingpong-tasks1.c
```

Se desejar mais detalhes da compilação, pode adicionar os flags `-Wall` e `-Wextra`:

```
cc -Wall -Wextra pingpong.c pingpong-tasks1.c
```

Depuração

Todas as funções implementadas devem gerar mensagens de depuração, que permitam acompanhar a execução das tarefas, como no exemplo a seguir:

```
task_create: criou tarefa 23
task_switch: trocando contexto 14 -> 23
task_exit: tarefa 23 sendo encerrada
...
```

Essas mensagens de depuração somente deverão ser geradas se a contante global `DEBUG` estiver definida, usando compilação condicional:

```
#ifdef DEBUG
printf ("task_create: criou tarefa %d\n", task->id) ;
#endif
```

A constante `DEBUG` pode ser definida no código-fonte:

```
#define DEBUG
```

ou na linha de comando, no momento da compilação:

```
host:~> gcc -o teste -DDEBUG pingpong.c pingpong-tasks1.c
```

Outras informações

- Duração estimada: 4 horas.
- Dependências:
    - Trocas de contexto
