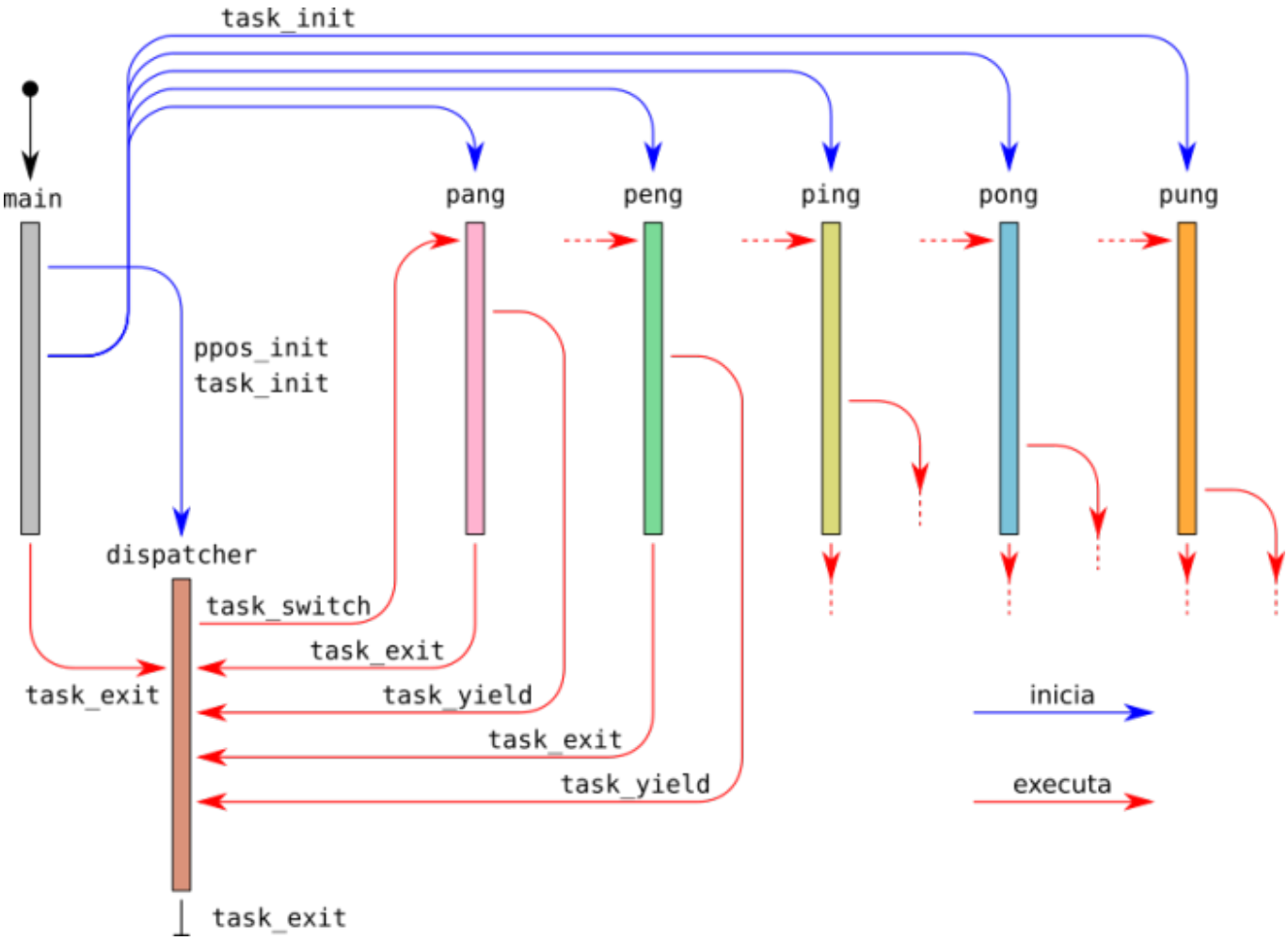
Despachante de tarefas

alterações na interface em 03/2023

Vídeo deste projeto

Você irá construir um despachante de tarefas baseado em duas entidades: uma tarefa *dispatcher*, responsável pelo controle geral, e uma função *scheduler*, responsável por determinar qual a próxima tarefa a executar a cada troca de contexto. A figura abaixo ilustra o funcionamento geral do sistema:



A seguinte chamada para a execução da tarefa atual e retorna ao *dispatcher*:

```
void task_yield () ;
```

O que deve fazer essa função:

1. coloca a tarefa atual no fim da fila de prontas
2. muda o estado da tarefa atual para PRONTA
3. devolve a CPU ao despachante

Sugestão: use a função `task_switch()` para implementar `task_yield()`.

Observações

- O *dispatcher* deve ser implementado como uma tarefa, a ser iniciada usando a chamada `task_init` durante a inicialização do sistema (execução de `ppos_init`).
- O programa principal inicia todas as tarefas de usuário e passa o controle para a tarefa *dispatcher*, que só encerra quando não existirem mais tarefas de usuário a executar.
- Será necessário implementar uma fila de tarefas prontas, usando a biblioteca de filas genéricas desenvolvida anteriormente.
- A **política de escalonamento** será definida por uma função `scheduler()`, chamada pelo *dispatcher* para decidir qual a próxima tarefa a ativar. Neste projeto, deve ser implementada uma política FCFS.
- Quando uma tarefa encerrar, o controle volta ao *dispatcher* e este libera as estruturas de dados alocadas para a tarefa.
- quando o *dispatcher* encerrar, a chamada `task_exit` deve encerrar o programa chamando `exit`.

O código do corpo da tarefa *dispatcher* deve seguir +/- o seguinte modelo (simplificado):

```
função dispatcher
início
   // retira o dispatcher da fila de prontas, para evitar que ele ative a si
próprio
   queue_remove (...)

   // enquanto houverem tarefas de usuário
   enquanto ( userTasks > 0 )

      // escolhe a próxima tarefa a executar
      próxima = scheduler ()

      // escalonador escolheu uma tarefa?
      se próxima ≠ NULO então

         // transfere controle para a próxima tarefa
         task_switch (próxima)
         // voltando ao dispatcher, trata a tarefa de acordo com seu estado
         caso o estado da tarefa "próxima" seja:
            PRONTA    : ...
            TERMINADA : ...
            SUSPENSA  : ...
            (etc)
         fim caso

      fim se

   fim enquanto

   // encerra a tarefa dispatcher
   task_exit(0)
fim
```

Sua implementação deve funcionar com este código. A saída da execução deve ser igual a este exemplo.

Uso do Valgrind

O Valgrind é uma ferramenta poderosa para a depuração de problemas de memória, que pode ser muito útil neste projeto. Entretanto, o uso de trocas de contexto em modo usuário confunde esse depurador e gera uma imensa quantidade de avisos errôneos.

Para poder usar o Valgrind corretamente neste projeto, deve-se informar ao Valgrind que estão sendo usados vários contextos com pilhas separadas. Isso é feito através de algumas alterações no código-fonte.

Primeiro, deve-se incluir o arquivo de cabeçalhos do Valgrind:

```
#include <valgrind/valgrind.h>
```

No descritor de cada tarefa, deve-se incluir um campo para uso do Valgrind:

```
struct task_t
{
  ...
  int vg_id ;     // ID da pilha da tarefa no Valgrind
  ...
}
```

Após a alocação da pilha de cada tarefa, deve-se **registrar** essa pilha junto ao Valgrind:

```
// aloca a pilha da tarefa
task->stack = malloc (STACKSIZE) ;
if (!task->stack)
  return (-1) ;

// registra a pilha da tarefa no Valgrind
task->vg_id = VALGRIND_STACK_REGISTER (task->stack, task->stack + STACKSIZE);
```

Finalmente, quando a pilha da tarefa for liberada, deve-se **desfazer o registro** dessa pilha no Valgrind:

```
// libera a pilha da tarefa
free (task->stack) ;

// dezfaz o registro da pilha no Valgrind
VALGRIND_STACK_DEREGISTER (task->vg_id);
```

Mais informações sobre essas macros e o uso avançado do Valgrind podem ser obtidas nesta página.

Outras informações

- Duração estimada: 5 horas.
- Dependências:
  - Biblioteca de Filas
  - Gestão de Tarefas

From:
https://wiki.inf.ufpr.br/maziero/ - **Prof. Carlos Maziero**

Permanent link:
https://wiki.inf.ufpr.br/maziero/doku.php?id=so:dispatcher

Last update: **2025/08/28 12:55**