Preempção e compartilhamento de tempo

alterações na interface em 03/2023

Video deste projeto

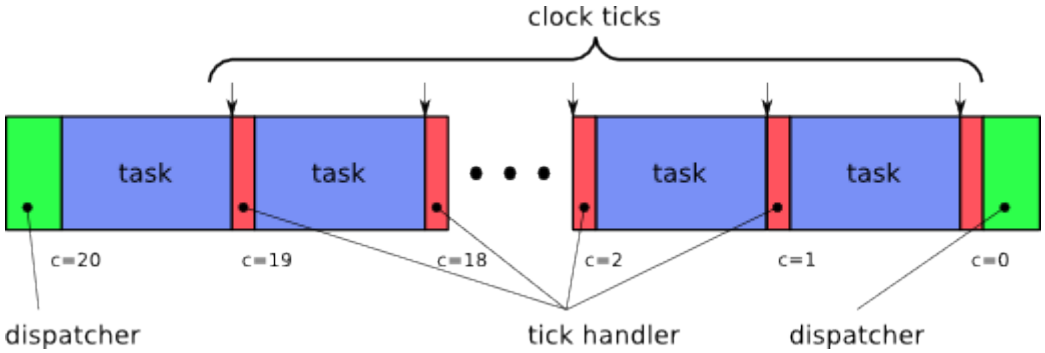
Até agora, nosso sistema suporta apenas tarefas cooperativas. O objetivo deste projeto é adicionar preempção por tempo ao sistema. Com essa modificação, nosso sistema passará a suportar tarefas preemptivas, que podem se alternar no uso do processador sem a necessidade de trocas de contexto explícitas via `task_yield`.

Preempção

Em sistemas de tempo compartilhado (*time-sharing*), cada tarefa de usuário recebe uma pequena fatia de tempo de processador, denominada *quantum*. Valores típicos de quantum estão entre 1 ms e 100 ms. Ao acabar seu quantum, a tarefa em execução retorna à fila de prontas para ceder lugar à próxima tarefa da fila de prontas.

Em um sistema real, a implementação da preempção por tempo tem como base as interrupções geradas pelo temporizador programável do hardware. Esse temporizador é programado para gerar uma interrupção a cada 1 milissegundo, que é tratada por um *interrupt handler* (tratador de interrupção) ou ISR (*Interrupt Service Routine*); essas ativações periódicas do tratador de interrupção são normalmente chamadas de *ticks* do relógio.

Quando uma tarefa recebe o processador, o *dispatcher* ajusta um contador de *ticks* que essa tarefa pode usar, ou seja, seu quantum definido em número de *ticks*. A cada *tick*, esse contador deve ser decrementado; quando ele chegar a zero, o processador deve ser devolvido ao *dispatcher* e a tarefa volta à fila de prontas. A figura a seguir ilustra esse conceito:



Como um processo UNIX não tem acesso direto aos temporizadores e interrupções do hardware, vamos simular o temporizador de hardware usando um temporizador UNIX, e o mecanismo de interrupção será simulado através de *sinais UNIX*, que serão explicados a seguir.

Sinais UNIX

O mecanismo de sinais do UNIX é similar às interrupções (IRQs) geradas pelo hardware: ao receber um sinal, um processo desvia sua execução para uma função que ele previamente registrou no sistema operacional.

A página de manual `signal` (seção 7) relaciona os principais sinais disponíveis em um sistema UNIX e as ações que cada sinal pode desencadear no processo que o recebe. Através da chamada de sistema `sigaction` é

possível registrar uma função de tratamento para um determinado sinal (*signal handler function*).

Um exemplo do uso de sinais está no arquivo `signal.c`. Nele, uma função é registrada para tratar o sinal `SIGINT`, que corresponde ao *Control-C* do teclado. Analise atentamente seu código, execute-o e observe seu comportamento.

Temporizadores UNIX

Para simular as interrupções de relógio do hardware, faremos uso do mecanismo de sinais (para implementar a preempção) e de temporizadores UNIX (para gerar os *ticks* de relógio). O UNIX permite definir temporizadores através das chamadas de sistema `getitimer` e `setitimer`. Ao disparar, um temporizador gera um sinal para o processo, que pode ser capturado por uma função tratadora previamente registrada por ele. O arquivo `timer.c` apresenta um exemplo de uso do temporizador.

Implementação

O mecanismo a ser implementado pode ser resumido nos seguintes passos:

1. Durante a inicialização do sistema, um temporizador deve ser programado para disparar a cada 1 milissegundo;
2. Os disparos do temporizador devem ser tratados por uma rotina de tratamento de *ticks*;
3. ao ganhar o processador, cada tarefa recebe um quantum de 20 *ticks* de relógio (experimente com diferentes tamanhos de *quantum* para ver seu efeito);
4. ao ser acionada, a rotina de tratamento de *ticks* de relógio deve decrementar o contador de quantum da tarefa corrente, se for uma tarefa de usuário;
5. se o contador de quantum chegar a zero, a tarefa em execução deve voltar à fila de prontas e o controle do processador deve ser devolvido ao *dispatcher*.

Sua implementação deve funcionar com este código e deve gerar um resultado similar ao desta saída. Um teste de *stress*, para verificar se seu sistema se comporta bem com muitas tarefas, também está disponível:

pingpong-preempcao-stress.c

.

A rotina de tratamento de *ticks* de relógio é **crítica**, pois vai ser executada com muita frequência (1000× por segundo). Essa rotina deve ser pequena e rápida, para não prejudicar o desempenho do sistema e garantir sua estabilidade.

Condições de disputa

É importante evitar preempções dentro do *dispatcher* ou de funções do nosso sistema, pois estas podem ter resultados imprevisíveis, como **condições de disputa** e instabilidade. Pode-se controlar a ocorrência de preempções de várias formas. Uma forma básica de implementar esse controle usa o conceito de **tarefa de sistema**:

- Tarefas críticas como o *dispatcher* são consideradas **tarefas de sistema**, pois sua execução correta é fundamental para o bom funcionamento do sistema; sua preempção deve ser evitada.
- As demais tarefas (*Main*, *Pang*, ...*Pung*) são consideradas **tarefas de usuário**, pois executam código definido pelo usuário do sistema, e podem ser "preemptadas" quando necessário.

- O tratador do temporizador deve sempre verificar se a tarefa corrente é de usuário ou de sistema, antes de preemptá-la devido ao fim de um *quantum*. Pode ser adicionado um *flag* na estrutura de controle de cada tarefa para indicar se é uma tarefa de sistema ou de usuário.

Uma solução mais “radical” para esse problema consiste em impedir completamente as preempções enquanto a execução estiver dentro das funções do núcleo. Uma forma simples de obter isso consiste em definir um *flag* global que seja TRUE quando uma tarefa de usuário estiver executando seu próprio código e FALSE quando a execução estiver dentro de uma função do sistema (`task_init`, `task_switch`, etc). Esse *flag* deve ser testado pela rotina de tratamento de *ticks* de relógio e tem de ser ligado/desligado explicitamente em cada função.

Versões mais antigas do núcleo Linux possuíam uma trava global chamada *The Big Kernel Lock* para fazer esse controle.

Outras informações

- Duração estimada: 4 horas.
- Dependências:
  - Gestão de Tarefas
  - Dispatcher