Trocas de contexto

Diogo Guilherme Garcia de Freitas

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
15 de Outubro de 2015

1. Introdução

Este relatório descreve o funcionamento de um programa com a função de troca de contextos de acordo com as normas POSIX. Este programa é baseado nas funções da biblioteca `ucontext` que tem por objetivo facilitar tarefas complexas na troca de contextos.

2. Funcionamento do programa

O programa fornecido (`contexts.c`) representa um caso onde são usadas trocas de contexto. O objetivo deste código é desviar a execução atual para as funções ali descritas, `main()`, `BodyPing()` e `BodyPong()`.
Primeiramente, `getcontext()` é chamada para salvar o contexto atual na respectiva variável global. Depois, configura-se a pilha de dados relevante ao contexto de execução atual. Após isto, se chama `makecontext()` para indicar a criação de um novo contexto para a função `BodyPing()`. Desse modo, `BodyPing()` passará a estar pronta para ser executada. O mesmo se repete para `BodyPong()`.
Após todas as configurações serem realizadas, o programa faz a troca de contexto através da função `swapcontext()`, que desvia a execução da `main()` para `BodyPing()`. Então, `BodyPing()` irá ser executada até que seja chamada novamente a função `swapcontext()`, o que ocorre dentro do próprio `BodyPing()`, que troca o contexto para `BodyPong()`. Então, o atual contexto em execução, agora `BodyPong()`, fará a mesma coisa e trocará o contexto para `BodyPing()` até que finalmente termine sua execução e troque novamente o contexto para a função `main()`. Neste caso, `BodyPing()` termina antes de `BodyPong()` e se faz necessário mais uma troca de contexto da `main()` para `BodyPong()`, para que esta última seja finalizada. Após o término de `BodyPong()`, o contexto é trocado novamente para `main()` e o programa termina sua execução.
A figura 1 mostra o diagrama de tempo da execução do programa.

3. Funcionamento da biblioteca

Esta seção descreverá o funcionamento da biblioteca `ucontext` através de suas funções e estruturas.

3.1. Funções

São presentes na biblioteca `ucontext` as seguintes funções:

- `getcontext(&a)`: Inicializa a estrutura `a` que receberá o contexto atual.
- `setcontext(&a)`: Restaura o contexto salvo na estrutura `b`.
- `swapcontext(&a, &b)`: Efetua a troca de contexto, salvando o contexto atual em `a` e restaurando o contexto salvo em `b`.
- `makecontext(&a, (void*)(*func), argc, ...)`: Modifica o contexto inicializado em `a` e faz o contexto em `a` ser o contexto da função `func()`. Os demais parâmetros `arcg` e reticências serão os parâmetros recebidos pela função `func()`.

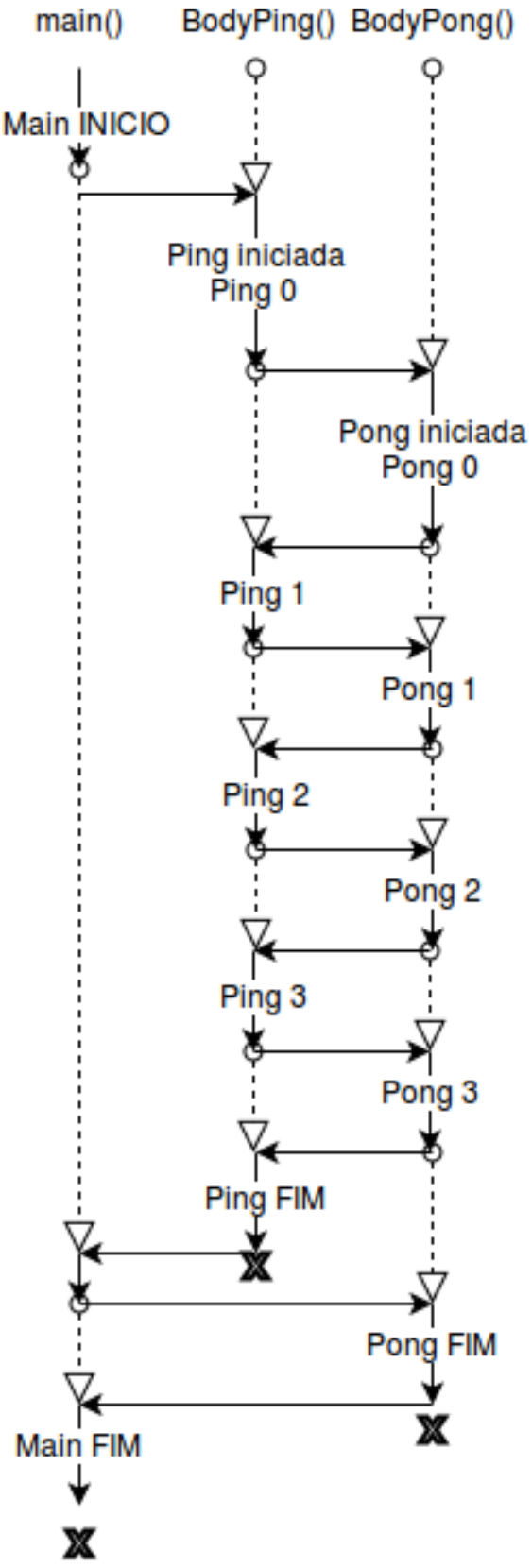

Figura 1. Diagrama de tempo de execução.

3.2. Estrutura ucontext_t

A estrutura `ucontext_t` é composta dos seguintes campos:

- `ucontext_t *uc_link`: Ponteiro para o próximo contexto que será executado ao final da execução do contexto atual.
- `sigset_t uc_sigmask`: Máscara de sinal usada para guardar informações sobre sinais a serem bloqueados, como sinais para matar processos por exemplo.
- `stack_t uc_stack`: Outra estrutura que guarda valores referentes a pilha. Veja seção 3.3.
- `mcontext_t uc_mcontext`: guarda os valores dos registradores.

3.3. Estrutura stack_t

Já a estrutura `stack_t` possui os seguintes campos:

- `void *ss_sp`: *Stack pointer*.
- `int ss_flags`: *Flags* associadas a pilha.
- `size_t ss_size;`: Quantidade de memória anteriormente alocada para o ponteiro para `void`.

4. Conclusão

Após os testes e verficação do funcionamento do programa fornecido, concluiu-se que a implementação de trocas de contexto na linguagem C não são tão complexas quanto parece quando se utiliza as funções da biblioteca `ucontext`, padronizadas pelas normas POSIX.