Relatório do Projeto 1

Giuliana Martins Silva
Alexandre Herrero Matias
13 de abril de 2019

1. As funções definidas em `ucontext.h`

Está biblioteca define quatro funções principais descritas a seguir.

a. **`getcontext(&a)`**

Esta função inicializa a estrutura apontada por `(&a)` para o contexto de usuário atual do processo de chamada. A estrutura `ucontext_t` para a qual `(&a)` aponta define o contexto do usuário e inclui os conteúdos dos registradores, da máscara de sinais e da pilha. Essa função pode ser usada para salvar o contexto atual para retomá-lo em outro momento através da função `setcontext()` e pode servir para criar um molde de contexto que será posteriormente modificado utilizando a função `makecontext()`.

b. **`setcontext(&a)`**

Esta função restaura o contexto apontado por `(&a)`. A execução continua a partir do ponto no qual o contexto foi salvo anteriormente em `(&a)`.

c. **`swapcontext(&a, &b)`**

Esta função efetua a troca de contexto onde o contexto atual de execução é salvo em `(&a)` e o contexto apontado por `(&b)` é restaurado. Esta função é equivalente a fazer uma chamada para `getcontext()` tendo `(&a)` como argumento e, em seguida, chamar `setcontext()` tendo `(&b)` como argumento.

d. **`makecontext(&a, &func(), int argc, ..)`**

Esta função modifica o contexto especificado por `(&a)` o qual foi inicializado usando `getcontext()`. Antes dessa função ser chamada é necessário modificar `(&a)` para definir uma pilha e inicializar o campo `uc_link` para indicar o contexto a ser executado após o término da execução do contexto definido por `makecontext()`. A função `func()` é a que será executada pelo contexto, `argc` define a quantidade de argumentos da lista de argumentos que serão passados para o contexto.

2. A estrutura `u_context`.

Essa estrutura é definida através dos membros descritos a seguir.

a. **`ucontext_t *uc_link`**

É um ponteiro para o contexto que será retomado quando o contexto atual retornar, se o contexto foi criado com makecontext().

b. **`sigset_t uc_sigmask`**

É um conjunto de sinais que são bloqueados quando o contexto está ativo.

c. **`stack_t uc_stack`**

É a pilha usada pelo contexto. É uma estrutura definida pelos membros descritos a seguir.

i. **`void *ss_sp`**

É a base da pilha ou ponteiro.

ii. **`size_t ss_size`**

Define o tamanho da pilha.

iii. **`int ss_flags`**
São as flags definidas.

d. `mcontext_t uc_mcontext`

É uma representação específica da máquina do contexto salvo. É responsável por salvar o estado de execução, todos os registradores e flags da CPU e o ponteiro para a pilha.

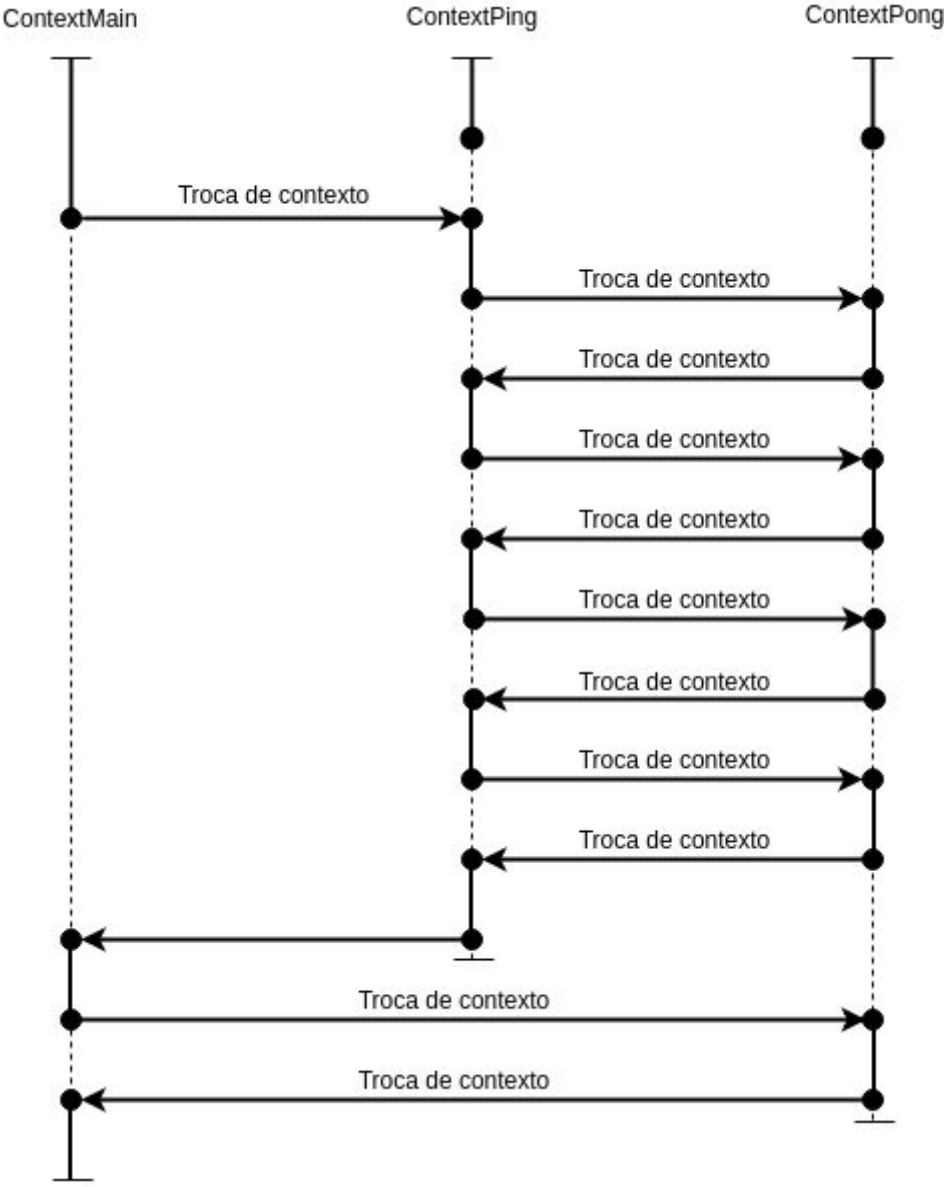
3. O código de `pingpong.c`.

No código `pingpong.c`, primeiro, cria-se três contextos `ContextPing`, `ContextPong`, `ContextMain`. Em seguida, são definidas duas funções:

- `void BodyPing (void* arg)`
  Esta função troca quatro vezes o contexto de `ContextPing` para `ContextPong`.
- `void BodyPong (void* arg)`
  Esta função troca quatro vezes o contexto de `ContextPong` para `ContextPing`.

Inicialmente, `ContextPing` é inicializado através da função `getcontext()` e uma pilha é alocada e inicializada em `ContextPing`. Após isso, a função `makecontext()` é utilizada para modificar `ContextPing`, adicionando a função `BodyPing` e o argumento "
Ping". O mesmo é feito para `ContextPong` com a diferença de que se é adicionada a função `BodyPong()` e o argumento " Pong". Em seguida, a função `swapcontext()` muda o contexto de `ContextMain` para `ContextPing`. `BodyPing()` começa a ser executada e troca de contexto para `ContextPong` através da função `swapcontext()`. `BodyPong()` passa a ser executada para depois trocar o contexto novamente para `ContextPing` através da função `swapcontext()`. `BodyPing()` volta a ser executada a partir do lugar que parou da última vez. Há uma outra troca de contexto e `BodyPong()` também é retomada a partir do lugar que parou. Isso se repete outras duas vezes (por conta do `for` nessas duas funções), até que `BodyPing()` termina a sua execução e o contexto retorna para `ContextMain`. Na `main()`, a função de troca de contexto é chamada de novo e troca o contexto para `ContextPong` para que, assim, a execução de `BodyPong()` seja finalizada. Ao término da execução de `BodyPong()`, o contexto, por fim, retorna para a `main()`, onde o programa pode ser totalmente executado e finalizado.

4. O diagrama de tempo



Referências

http://www.inf.ufrgs.br/~asc/sisop/pdf/2016-02/aulaTP1.pdf
https://en.wikipedia.org/wiki/Setcontext
http://pubs.opengroup.org/onlinepubs/007908799/xsh/ucontext.h.html
http://pubs.opengroup.org/onlinepubs/007908799/xsh/signal.h.html