Relatório Laboratório 1 : Trocas de Contexto

Equipe:

João Pedro da Matta Galera da Silva

Marina Pereira de Souza

Chamadas de sistema POSIX:

1. getcontext(&a)
2. setcontext(&a)
3. swapcontext(&a,&b)
4. makecontext(&a, ...)

Objetivos das funções:

1. Salvar o contexto significa salvar todas as informações do referentes ao estado de uma tarefa em determinado momento. São salvos o identificador da tarefa, seu estado, valores dos registradores do processador, lista de áreas de memória utilizadas e arquivos abertos, e também informações de gerência e contabilização.
2. Esta função carrega no um contexto salvo, ou seja, ele restaura no processador o estado de uma tarefa que foi suspensa ou interrompida para que o processador continua sua execução.
3. O swap serve para mudar o contexto do processador, suspendendo uma tarefa em execução e carregando outra que havia sido suspensa para execução novamente.
4. Serve para modificar o contexto apontado, fazendo isso por ucp.

Parâmetros das funções:

1. Esta função recebe como parâmetro a variável onde o contexto atual do processador será salvo.

2. Recebe a variável onde o contexto a ser carregado está armazenado.

3. Recebe como parâmetro duas variáveis, uma variável “a” onde o contexto da tarefa atual ser salvo e uma variável “b” onde está salvo o contexto de uma tarefa que irá ser carregada no processador para continuar sua execução.

4. O objetivo dessa função é alterar um determinado contexto, e, antes de fazer isso, é necessário alocar uma nova pilha para o contexto através de ucp->uc_stack e definir um contexto sucessor, salvo em ucp->uc_link. Ela recebe a variável do contexto a ser modificado, uma função a ser executada, o número de argumentos que serão passados, e por fim, os argumentos em si.

As estruturas ucontext_t nas linhas 64 a 67 para “Ping” e das linhas 82 a 85 para “Pong” são as informações do contexto de “Ping” e “Pong”. Definem o seguinte:

- stack.ss_sp: Define o endereço de início da nova pilha para o contexto salvo.

- stack.ss_size: Guarda o tamanho da nova pilha.

- stack.ss_flags: São as flags de controle das pilhas envolvidas.

- link: Guarda o endereço do contexto sucessor, que é para onde o processador irá retornar depois de executar.

Explicação das linhas do código que utilizam essas funções:

Linha 26: swapcontext (&ContextPing, &ContextPong); Salva o contexto da Tarefa ping na variável ContextPing, Carrega a tarefa Pong, cujo contexto estava na variável ContextPong .

Linha 30: swapcontext (&ContextPing, &ContextMain) ; Salva o contexto da Tarefa ping na variável ContextPing, Carrega a tarefa Main, cujo contexto estava na variável ContextMain.

Linha 44: swapcontext (&ContextPong, &ContextPing); Salva o contexto da Tarefa Pong na variável ContextPong, Carrega a tarefa Ping, cujo contexto estava na variável ContextPing.

Linha 48: swapcontext (&ContextPong, &ContextMain) ; Salva o contexto da Tarefa Pong na variável ContextPong, Carrega a tarefa Main, cujo contexto estava na variável ContextMain.

Linha 59: getcontext (&ContextPing); O contexto da tarefa Ping está sendo salvo na variável ContextPing.

Linha 64: ContextPing.uc_stack.ss_sp = stack ; Define o endereço de início da nova pilha para o contexto de ping salvo.

Linha 65: ContextPing.uc_stack.ss_size = STACKSIZE; Define o tamanho da nova pilha.

Linha 66: ContextPing.uc_stack.ss_flags = 0; Seta as flags da pilha para 0.

Linha 67: ContextPing.uc_link = 0; É para onde o código retorna depois de executar, ou seja, a próxima pilha onde contém o próximo contexto a executar.

Linha 75: makecontext (&ContextPing, (void*)(*BodyPing), 1, " Ping"); Depois de definir a nova stack ContextPing, podemos utilizar a função Makecontext para modificar o contexto. Aqui, o contexto modificado é o Contextping. Essa modificação é feita através da função bodyping, que passa "Ping" como parâmetro, que é usado para imprimir na função bodyping. O valor "1" simplesmente diz quantos argumentos serão passados.

Linha 76: getcontext (&ContextPong); Salva o contexto atual na variável ContextPong.

Linha 81: ContextPong.uc_stack.ss_sp = stack ; Define o endereço de início da nova pilha para o contexto de ping salvo.

Linha 82: ContextPong.uc_stack.ss_size = STACKSIZE; Define o tamanho da nova pilha.

Linha 83: ContextPong.uc_stack.ss_flags = 0; Seta as flags da pilha para 0.

Linha 84: ContextPong.uc_link = 0; É para onde o código retorna depois de executar, ou seja, a próxima pilha onde contém o próximo contexto a executar.

Linha 92: makecontext (&ContextPong, (void*)(*BodyPong), 1, " Pong"); Depois de definir a nova stack ContextPong, podemos utilizar a função Makecontext para modificar o contexto. Aqui, o contexto modificado é o ContextPong. Essa modificação é feita através da função bodyping, que passa "Pong" como parâmetro, que é usado para imprimir na função bodypong. O valor "1" simplesmente diz quantos argumentos serão passados.

Linha 94: swapcontext (&ContextMain, &ContextPing); Salva o contexto da tarefa Main em uma variável ContextMain e carrega o contexto salvo em ContextPing.

Linha 95: swapcontext (&ContextMain, &ContextPong); Salva o contexto da tarefa Main em uma variável ContextMain e carrega o contexto salvo em ContextPing.