Trocas de contexto

Vídeo deste projeto

Implementar tarefas simultâneas dentro de um processo de usuário não é uma tarefa difícil, embora alguns detalhes de baixo nível possam assustar os iniciantes, como a manipulação de registradores para as trocas de contexto. Felizmente, algumas chamadas de sistema POSIX permitem simplificar a manipulação de contextos, eliminando as operações com registradores e tornando o código portável:

- `getcontext(&a)` : salva o contexto atual na variável a.
- `setcontext(&a)` : restaura um contexto salvo anteriormente na variável a.
- `swapcontext(&a,&b)` : salva o contexto atual em a e restaura o contexto salvo anteriormente em b.
- `makecontext(&a, …)` : ajusta alguns valores internos do contexto salvo em a.
- as variáveis a e b são do tipo `ucontext_t` e armazenam contextos de execução.

Mais informações sobre essas funções podem ser obtidas em suas respectivas páginas de manual (`man getcontext`, etc.).

Estude o código presente no arquivo contexts.c, execute-o e explique seu funcionamento.

A depuração passo-a-passo desse código pode apresentar alguma dificuldade, devido às trocas de contexto. Sugere-se inserir pontos de parada (*breakpoints*) nos trechos mais críticos e depurar “saltando” de um ponto ao próximo.

~~Elabore um relatório (no formato correto) cobrindo pelo menos os seguintes pontos:~~

Após estudar o código do arquivo, faça as seguintes atividades:

1. Explique o objetivo e os parâmetros de cada uma das quatro funções acima.
2. Explique o significado dos campos da estrutura `ucontext_t` que foram utilizados no código.
3. Explique cada linha do código de `contexts.c` que chame uma dessas funções ou que manipule estruturas do tipo `ucontext_t`.
4. Para visualizar melhor as trocas de contexto, desenhe o diagrama de tempo dessa execução.

Curiosidade

Trocas de contexto são implementadas dentro do núcleo, por código sucinto mas geralmente complexo. Veja um comentário do código de troca de contexto de uma versão inicial do UNIX (anos 1970):

```
/*
 * Switch to stack of the new process and set up
 * his segmentation registers.
 */
retu(rp->p_addr);
sureg();
/*
 * If the new process paused because it was
 * swapped out, set the stack level to the last call
 * to savu(u_ssav).  This means that the return
 * which is executed immediately after the call to aretu
 * actually returns from the last routine which did
 * the savu.
```

```
 *
 * You are not expected to understand this.      <-----------------
 */
if(rp->p_flag&SSWAP) {
    rp->p_flag =& ~SSWAP;
    aretu(u.u_ssav);
}
/*
 * The value returned here has many subtle implications.
 * See the newproc comments.
 */
return(1);
```

Outras informações

- Duração estimada: 2 horas.