Trocas de contexto

Este projeto faz parte do PPOS v2.

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.

Este projeto visa explicar o funcionamento básico das trocas de contexto, usando uma pequena **biblioteca de trocas de contexto** em modo usuário. Essa biblioteca foi escrita em linguagem de máquina, pois a manipulação direta de registradores é difícil de fazer em linguagens de alto nível.

A biblioteca usada é específica para processos Linux executando em processadores Intel/AMD de 64 bits. Sua execução em outros ambientes exige diversas adaptações. O padrão POSIX definiu uma biblioteca portável para trocas de contexto em modo usuário, chamada ucontext, que foi descontinuada.

Tarefa

- Estude o conteúdo dos arquivos em context.tgz :
  - `contexts.c` : programa de teste
  - `ctx.h` : definições
  - `ctx.s` : implementação (assembly)
- Compile e execute o programa (`make`) e explique seu funcionamento.
- Explique o objetivo da estrutura `ctx_t` definida em `ctx.h`.
- Explique o objetivo e os parâmetros das funções definidas em `ctx.h`.
- Explique cada linha do código de teste que chame uma dessas funções.
- Para visualizar melhor as trocas de contexto, desenhe o diagrama de tempo dessa execução.

A depuração passo a passo desse código com o `gdb` pode apresentar alguma dificuldade, devido às trocas de contexto. Sugere-se inserir pontos de parada (*breakpoints*) nos trechos mais críticos (antes e depois das chamadas a `ctx_swap`, por exemplo) e depurar “saltando” de um ponto ao próximo.

A entregar

 Nada!

 Basta estudar e entender o código da biblioteca e do teste...

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.