INE5412 – Sistemas Operacionais I
Profs. Giovani Gracioli e Márcio Bastos Castro
2022/01

Projeto de um Sistema Operacional em Nível de Usuário
Trabalho 1: Troca de Contexto

1) Apresentação do Projeto do Semestre

Durante este semestre iremos projetar e construir um pequeno sistema operacional (SO) em nível de usuário. Tal SO irá focar no gerenciamento de threads cooperativas, implementadas como um programa em nível de usuário dentro do Linux.

O desenvolvimento de todas as etapas do SO será realizado de forma incremental. Cada uma das etapas de desenvolvimento será disponibilizada aos alunos através de trabalhos. Isso significa que cada etapa (ou trabalho) é fundamental para a sequência do SO. Gradativamente, cada um dos trabalhos irá adicionar novas funcionalidades para o gerenciamento de threads. O projeto final da disciplina usará o gerenciamento de threads desenvolvido.

Essa abordagem de projeto e implementação de SO em nível de usuário diminui o nível de complexidade. Por exemplo, alguns detalhes internos do SO dependentes de código *assembly* não são necessários. Entretanto, não afeta a fixação dos conceitos de alto nível da disciplina e nem a prática no desenvolvimento de SOs.

O SO irá ser desenvolvido usando a linguagem de programação C++, obrigatoriamente. Será, portanto, avaliado também aspectos de projeto orientado a objetos e uso correto de OO.

2) Trabalho 1: Troca de contexto

O primeiro trabalho consiste na implementação do conceito de Contexto e Trocas de Contexto entre threads (ou tarefas ou processos).

O contexto de uma thread é formado por todos os registradores do processador, além de outras informações como arquivos abertos, sinais, etc. Desta forma, os procedimentos de salvar e carregar o contexto de uma thread são implementados usando linguagem de máquina (*assembly*) e são, portanto, dependentes do processador.

Para este trabalho, usaremos uma biblioteca POSIX que permite simplificar a manipulação de contextos de threads, eliminado toda a dependência com *assembly*. A biblioteca utilizada é acessível pelo arquivo header ucontext.h (*user context*) e contém as seguintes funções:

- `int  getcontext(ucontext_t *);`

  Utilizada para salvar o contexto atual no ponteiro recebido como parâmetro.

- `int  setcontext(const ucontext_t *);`

Utilizada para restaurar o contexto salvo previamente no ponteiro recebido como parâmetro.

- **`void makecontext(ucontext_t *, (void *)(), int, ...);`**

  Função que modifica um contexto recebido por parâmetro e previamente inicializado com getcontext(). Após o contexto modificado ser resumido com swapcontext() ou setcontext(), a execução do programa continua executando a função recebida como parâmetro. Note que a função recebe um inteiro como terceiro parâmetro e "..." como quarto parâmetro. O inteiro tem o número de parâmetros que a função a ser executada pelo contexto irá receber e o "..." representa esses parâmetros que podem ter qualquer número. Para entender como a passagem de N parâmetros com templates em C++, chamado de *variadic template* e usado neste trabalho, acesse o tutorial online: https://en.cppreference.com/w/cpp/language/parameter_pack.

- **`int  swapcontext(ucontext_t *a, const ucontext_t *b);`**

  Salva o contexto atual em "a" (primeiro argumento) e restaura o contexto previamente salvo em "b" (segundo argumento)

O seguinte website contém uma descrição das funções disponíveis na ucontext.h: https://pubs.opengroup.org/onlinepubs/7908799/xsh/ucontext.h.html

Maiores informações sobre as funções também podem ser encontradas nas suas respectivas páginas de manual, acessível através do comando man do terminal (ex: *man getcontext*).

O seguinte website contém exemplo de uso das funções ucontext.h: http://nitish712.blogspot.com/2012/10/thread-library-using-context-switching.html

O primeiro trabalho, portanto, será criar a infraestrutura mais baixa do SO que trata do gerenciamento e troca de contexto das threads. Para isso, os seguintes arquivos devem ser complementados seguindo a descrição abaixo:

- **Arquivo cpu.h**

  Este arquivo irá conter a definição da classe CPU e da classe Context. A classe CPU tem um método switch_context(Context *from, Context *to) que irá trocar o contexto usando os dois contextos recebidos como parâmetro (de "*from*" para "*to*").

  A classe CPU contém uma inner class (uma classe declarada dentro dela) chamada de Context. A classe Context contém um ponteiro para uma pilha (que deve ser alocada na criação de um novo contexto – construtor) e um contexto (_context) do tipo ucontext_t.

  A classe Context também os seguintes métodos:

  - Context(void (* func)(Tn ...), Tn ... an): construtor parametrizado que recebe uma função como primeiro parâmetro e os argumentos para essa função na seguinte. A declaração, para quem tem dificuldades com C++, usa *variadic templates*. A função passada como parâmetro é a função executada no contexto. Esse construtor deve alocar memória para a pilha do novo contexto a ser criado, além de inicializar todos os campos da estrutura ucontext_t (através do atributo _context) e criar um novo contexto usando makecontext().

- ~Context(): destrutor da classe usado para liberar a memória alocada para a pilha.

- save() e load(): métodos para salvar e carregar o contexto.

- **Arquivo cpu.cc**

  Este arquivo deve conter a implementação dos métodos declarados em cpu.h.

- **Arquivo traits.h**

  Este arquivo deve implementar as classes Trait do sistema, usadas para configuração dos parâmetros estáticos das classes durante a compilação. Um exemplo de parâmetro estático é o tamanho da pilha dentro da classe Context. Notem que o parâmetro é definido dentro de Traits<CPU>::STACK_SIZE.

  O trabalho neste arquivo é declarar STACK_SIZE dentro da classe template<> struct Traits<CPU>.

- **Arquivo main.cc**

  Arquivo com a função main. É disponibilizado pelo professor.

- **Arquivos main_class.h e main_class.cc**

  Esses arquivos contém um exemplo de execução de uma aplicação exemplo. A saída esperada para esse exemplo é a seguinte:

```
main: inicio
ping: inicio
ping0
pong: inicio
pong0
ping1
pong1
ping2
pong2
ping3
pong3
ping4
pong4
ping5
pong5
ping6
pong6
ping7
pong7
ping8
pong8
ping9
```

pong9
ping: fim
pong: fim
main: fim

Os arquivos descritos aqui estão disponibilizados na tarefa do moodle.

Um detalhe **MUITO IMPORTANTE** que deve ser seguido por todos, neste e nos demais trabalhos durante o semestre, é o uso de __BEGIN_API e __END_API no início e no final dos arquivos .h e .cc. Essas macros, definidas em traits.h não devem ser modificadas em nenhuma hipótese. Elas criam um namespace que é usado para fins de avaliação do código produzido pelos alunos.

3. Formato de Entrega

Todos os arquivos utilizados na implementação do trabalho devem ser entregues em um único arquivo .zip ou .tar.gz na atividade do moodle. Deve ser anexado um arquivo Makefile para compilar o código.

4. Data de Entrega:

Conforme tarefa definida no Moodle da disciplina.

5. Avaliação

A avaliação se dará em 3 fases

1. Avaliação de compilação: compilar o código enviado. Caso haja erros de compilação, a nota do trabalho será automaticamente zerada.

2. Avaliação de execução: para validar que a solução executa corretamente sem falhas de segmentação. Caso haja falhas de segmentação, a nota é zerada. Será também avaliado o uso de variáveis globais (-5 pontos) e vazamentos de memória (-20%). Dica: utilizar valgrind para detectar vazamento de memória.

3. Avaliação da organização do código: busca-se nesta fase avaliar a organização do código orientado a objetos. Deve-se usar classes e objetos e não estilo de programação baseado em procedimentos (como na linguagem C).

Este primeiro trabalho **não** precisa ser apresentado ao professor. Plágio não será tolerado em nenhuma hipótese ao longo dos trabalhos, acarretando em nota 0 a todos os envolvidos.