Universidade Federal Rural do Semiarido
Departamento de Ciências Exatas e Naturais
Ciência da Computação
Prof. Sílvio Fernandes

Trabalho de Sistemas Operacionais – Unidade I

- O trabalho pode ser feito em dupla ou individual
- O código da implementação e um relatório técnico descrevendo a implementação deve ser enviado via SIGAA no dia: **30/08/2016**
- No mesmo dia o trabalho deve ser apresentado em sala de aula.
- O trabalho corresponde a 30% da nota da I Unidade
- A implementação deve ser feita no MARS versão 4.5, disponível em <http://courses.missouristate.edu/kenvollmar/mars/>
- A implementação corresponde a uma infraestrutura de sistema operacional para realizar uma troca de contexto em processos.
- Para se ter acesso ao código do MARS, após baixar o arquivo .jar utilize uma ferramenta de descompactação (ex: Winrar) e extraia seu conteúdo. Em seguida crie um projeto em uma IDE para Java (ex: Eclipse) com os arquivos extraídos.
- O "Apêndice A" do livro "PATTERSON, D. A. ; HENNESSY, J.L. Organização e projeto de computadores – a interface hardware software. 3. ed. Editora Campus, 2005" e o tutorial "SANDERSON, Pete; VOLLMAR, Ken. An Assembly Language I.D.E. To Engage Students Of All Levels. A Tutorial. CCSC, 2007. Disponível em: http://courses.missouristate.edu/KenVollmar/mars/tutorial.htm" apresentam detalhes sobre chamadas de sistema do MIPS.

Descrição da Implementação: Chamada de sistema – não preemptivo

O trabalho é para implementar um gerenciador de processos, para isso:

- Crie uma classe PCB (*Process Control Block*) que serve para armazenar todas as informações de contexto de um processo:
  - Informações do hardware: deve ter no mínimo o conteúdo de todos os registradores
  - Informações lógicas: deve ter no mínimo o endereço do início do programa, PID (Identificador do processo), estado do processo (pronto, executando ou bloqueado)
- A classe PCB deve está preparada para ser incrementada, recebendo novos atributos e métodos para acessá-los nos próximos trabalhos
- Crie uma classe Tabela de Processos que instancia um objeto da classe PCB para cada novo processo

- Crie uma classe Escalonador onde poderão ser implementados diferentes algoritmos para escolha dos processos que estão no estado de "pronto" na Tabela de Processos.
- Você deve criar 3 chamadas de sistema de acordo com o tutorial disponível em http://courses.missouristate.edu/KenVollmar/mars/tutorial.htm
- Cada uma das chamadas vai ganhar um número que deve ser incluído no arquivo "Syscall.properties"
- Cada syscall será uma classe Java que deve fazer parte do pacote "**`mars.mips.instructions.syscalls`**" e deve extender a classe abstrata "AbstractSyscall"
- A primeira syscall é a "Fork", com o objetivo de criar um processo
    - O número da chamada de sistema Fork deve ser passado no registrador $v0
    - O endereço inicial do processo (representado por um label) deve ser passado pelo registrador $a0
    - Essa chamada cria um PCB para o novo processo, preenchendo os atributos possíveis, e o inclui na Tabela de processos
- A segunda syscall é o ProcessChange
    - Um processo que invoque essa chamada de sistema abre mão da CPU voluntariamente
    - Para invocar essa nova syscall o programa deve apenas passar o código da syscall no registrador "$v0" e executar a instrução "syscall"
    - Essa chamada não possui parâmetros
    - Quando ProcessChange é chamado, é identificado o processo que está em execução (só há um processo executando), o qual deve ter seu contexto de processo salvo em sua PCB na tabela de processos, ou seja, os valores de registradores físicos devem ser copiados para os registradores lógicos na PCB do processo. Além disso, o estado do processo deve ser alterado de "executando" para "pronto" na PCB. Em seguida deve ser chamado o algoritmo de escalonamento que por sua vez deve escolher um outro processo que esteja no estado de "pronto" (o escalonador só indica o processo que irá executar). Em seguida o gerenciador de processo carregar o contexto do processo escolhido pelo escalonador a partir do seu PCB, copiando o conteúdo dos registradores lógicos da PCB para os registradores físicos da máquina e mudando seu estado para "executando". Por fim a execução é retomado (agora do outro processo escolhido)
- A terceira syscall é o TerminateProcess
    - Um processo que invoque essa chamada de sistema indica ao SO que concluiu sua execução e não deve mais se escalonado
    - Para isso o processo deve passar o código da chamada no $v0, sem parâmetros, e executar syscall
    - O efeito para o gerenciador de processos é para sua execução, excluir sua PCB da tabela de processos e carregar um novo processo que esteja "pronto"
    - Se não houver mais processos prontos seu gerenciador deve mostrar uma mensagem que não há mais processos e deve encerrar a simulação

(pode desviar para um código no programa principal que simplesmente finaliza chamando syscall exit)

- Para auxiliar e facilitar a vida do programador dos processos crie um arquivo com macros com as principais definições e configurações do seu SO
    - Crie macros para as chamadas de sistema mais usadas (impressão, fim de programa e as syscalls criadas neste trabalho)
    - Crie qualquer constante necessária e visível globalmente
    - Para usar as macros é necessário usar a diretiva .include no programa principal
- Para testar seu escalonador
    - Teste 1
        - Escreva um trecho de código assembly equivalente a um programa principal (main) que cria 2 processos (usando Fork)
        - Escreva dois trechos de código assembly identificados pelos labels: programa 1 e programa 2 (relativos aos processos)
        - Cada programa deve executar um laço infinito com um código qualquer mas que no final do laço execute o syscall ProcessChange
        - No código de cada programa faça-os utilizar um mesmo registrador (ex: $S0) para guardar o resultado de operações diferentes (ex: um processo faz uma soma e quarda em $s0 e outro faz uma subtração e quarda no mesmo registrador).
        - Um possível código seria

```
main:   fork(P1)
        fork(P2)
        ProcessChange

P1:     add $s0, $zero, $zero
loop1:  addi $s0, $s0, 1
        ProcessChange
        j loop1

P2:     add $s0, $zero, $zero
loop2:  addi $s0, $s0, -1
        ProcessChange
        j loop2

        done
```

    - Teste 2
        - Faça o mesmo que o teste anterior, exceto por criar 3 processos e que os laços deles não são infinitos. Cada processo deve ter um laço com quantidade de repetições diferentes (ex: primeiro processo faz 2 iterações, o segundo 3 e o terceiro 4).
        - No final de cada laço os processos devem chamar a syscall TerminateProcess