**imagesUniversidade Federal Rural do Semi-Árido - UFERSA**

**Centro de Ciências Exatas e Naturais - CCEN**

**Departamento de Computação - DC**

**Curso Ciência da Computação**

**Disciplina: Sistemas Operacionais**

**Prof. Leiva Casemiro Oliveira**

# 1.2– Tarefa de Sistemas Operacionais

**INSTRUÇÕES:** Essa atividade pode ser em grupo e os códigos (projeto Eclipse) e respostas correspondentes as questões devem ser enviados em um único arquivo (.zip) até a data estabelecida no SIGAA pelo líder do grupo como parte da nota da Unidade em avaliação.

**Orientações gerais**

* A implementação deve ser feita no MARS versão 4.5, disponível em <http://courses.missouristate.edu/kenvollmar/mars/>
* A implementação corresponde de parte da infraestrutura de sistema operacional para realizar gerenciamento de 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” apresenta detalhes sobre o MIPS e chamadas de sistema.

**Criação das Syscalls do gerenciador 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” do projeto do MARS
* Cada syscall será uma classe Java que deve fazer parte do pacote “**mars.mips.instructions.syscalls**” e deve estender a classe abstrata “AbstractSyscall”
* A primeira syscall é a SyscallFork, com o objetivo de criar um processo
  + Os parâmetros para essa syscall são: o número da chamada de sistema SyscallFork passado no registrador $v0 e o endereço inicial do processo (representado por um *label*) passado no registrador $a0
  + Essa chamada cria um PCB para o novo processo, preenchendo os atributos possíveis (como o valor inicial de PC – indicado pelo *label*), e o inclui na Tabela de processos
* A segunda syscall deve ser chamada de SycallProcessChange com o objetivo de trocar o processo que está executando na CPU
  + Só possui um parâmetro: o número da sycall passado no registrador $v0
  + Um processo que invoque sua chamada de sistema abre mão da CPU voluntariamente, ou seja, esse processo para de executar e um novo processo é colocado para executar no seu lugar
  + A chamada de sistema implementada deve salvar o contexto do processo paralisado em seu respectivo PCB na tabela de processos (mudando seu estado de “Executando” para “Pronto”), chamar o algoritmo de roteamento do Escalonador (por um nome genérico independente do algoritmo que esteja sendo executado) que por sua vez deve escolher um outro processo que esteja no estado de “Pronto”, carregar o contexto do processo escolhido a partir do seu PCB (mudando seu estado para “Executando”) e retornar à execução (agora do outro processo escolhido)
    - Em outras palavras, salvar o contexto significa copiar os dados de que estão em todos os registradores físicos da CPU para os respectivos registradores virtuais da PCB do processo que está deixando a CPU
    - Enquanto que o processo que terá o contexto carregado acontecerá o inverso: os dados da PCB do processo escolhido são copiado dos registradores virtuais para os registradores físicos da CPU
  + Lembre-se que a classe “RegisterFile” do MARS é do tipo “static” e seus métodos e atributos devem ser acessados diretamente sem instanciação de objetos
    - Não é recomendado utilizar o método “getRegisters()” no processo de troca de valores entre os registradores físicos e os da PCB pois eles costumam vincular o mesmo valor em PCBs diferentes
    - É preferível usar os métodos de “get” e “update” individuais de cada registrador
* A terceira syscall é a SyscallProcessTerminate com o objetivo de finalizar um processo
  + O único parâmetro passado para essa syscall é seu código no registrador $v0
  + O processo que a invoca é finalizado, ou seja, tudo que estava alocado para ele é desalocado e sua PCB é removida da tabela de processos
  + Em seguida, o algoritmo de escalonamento deve ser invocado e um processo no estado de pronto deve ser colocado para executar
* O algoritmo de escalonamento será uma simples fila, ou seja, cada novo processo criado é colocado no final da fila. Todos os processos que estão nessa fila devem estar no estado de “Pronto”. Quando o escalonador é chamado o algoritmo de escalonamento é executado, e neste caso, o primeiro processo da fila é removido para ser então ser colocado no estado de “Executando”
* A classe que implementa o algoritmo de escalonamento deve estar preparada para implementar outros algoritmos, ou seja, quando houver outros algoritmos uma simples mudança de parâmetro deve invocar outro algoritmo sem a necessidade de mudar a implementação das outras classes.
* 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 (com as diretiva “.macro” e “.end\_macro”) para as chamadas de sistema mais usadas (impressão de string, 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”
* Para testar a criação e escalonamento de processo pode utilizar o código assembly a seguir

Programa1:

addi $s1, $zero, 1 # valor inicial do contador

addi $s2, $zero, 10 # valor limite do contador

loop1: addi $s1, $s1, 1

beq $s1, $s2, fim1

SyscallProcessChange

j loop1

fim1: SyscallProcessTerminate

Programa2:

addi $s1, $zero, -1 # valor inicial do contador

addi $s2, $zero, -10 # valor limite do contador

loop2: addi $s1, $s1, -1

beq $s1, $s2, fim2

SyscallProcessChange

j loop2

fim2: SyscallProcessTerminate

.include "macros.asm"

.data

.text

#criação dos processos

SyscallFork(Programa1)

SyscallFork(Programa2)

SyscallFork(Idle)

#escalonando o primeiro processo

SyscallProcessChange

Idle:

loop:

SyscallProcessChange

j loop

SEGUNDA PARTE