**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**

# 1.3– Tarefa de Sistemas Operacionais

**INSTRUÇÕES:** Essa atividade pode ser feita 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.

**Descrição da Implementação:** Escalonamento preemptivo

PARTE 1 – ESCALONAMENTO PREEMPTIVO

* Crie uma uma nova ferramenta (*tool*) do MARS que funcionará como um *timer*
* A criação de ferramenta deve seguir o tutorial disponível em <http://courses.missouristate.edu/KenVollmar/mars/tutorial.htm>
* Há duas formas de criar ferramentas:
  + Implementando a interface **mars.tools.MarsTool**
  + Ou extendendo a classe abstrata **mars.tools.AbstractMarsToolAndApplication**
* A ferramenta deve permitir configurar de quanto em quanto tempo o *timer* deve gerar interrupção, um botão de iniciar a contagem do *timer* e os botões comuns a todas as ferramentas do MARS (conectar ao MIPS, reset, ajuda e fechar)
* O tempo do *timer* deve ser definido em quantidade de instruções executadas
  + Essa configuração (quantidade de instruções a ser contada) deve está disponível na interface gráfica para o usuário
  + Uma opção para contar a quantidade de instruções executadas é sua ferramenta adicionar a memória como “observável”. E sempre que um endereço de memória (dentro do limite do segmento de código) for acessado a ferramenta seria notificada para contar como uma instrução executada.
  + Dica: vejam o código da ferramenta “InstructionCounter” como inspiração
* Quando o tempo configurado do *timer* for atingido acontece uma “interrupção”, o contador do *timer* zerado e uma nova contagem iniciada.
  + O tratamento dessa “interrupção” deve corresponder a troca de processos, ou seja, o algoritmo de escalonamento é invocado.
  + Esse mecanismo de interrupção deve **substituir** a *syscall* SycallProcessChange. Assim, quando a interrupção do *timer* ocorrer o contexto do processo em execução deve ser salvo em sua PCB, colocando-o no estado “Pronto” e o algoritmo de escalonamento deve escolher outro processo “Pronto” para executar. Depois de escolhido, o processo é colocado no estado “Executando”, seu contexto é carregado de sua PCB para o processador, do mesmo modo quando *SycallProcessChange* era chamada.
  + A implementação dos processos **não deve realizar** a chamada de sistema *SycallProcessChange* nos testes, mesmo essa *syscall* continuando disponível
* Para criação de processos deve ser usada a mesma chamada de sistema (*SyscallFork*) e podem incluir alguma mudança se necessário.
* Teste a nova funcionalidade com o seguinte código

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

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

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:

NOP

j loop

PARTE 2 – NOVOS ALGORITMOS DE ESCALONAMENTO

* Utilize a implementação das classes PCB, Tabela de Processos e Gerenciador de Processos dos trabalhos anteriores.
* Na classe do Gerenciador de Processos, que já possui o algoritmo de escalonamento de fila, acrescente os algoritmos:
  + Por prioridade fixa
    - Quando o processo é criado, além do endereço inicial também deve ser informada a prioridade dele
    - Cada prioridade tem uma fila própria, de modo que o escalonador sempre busca os processos prontos de mais alta prioridade primeiramente
  + Por loteria
    - O escalonador escolhe aleatoriamente um processo na lista de prontos e o coloca para executar
* Realize as modificações necessárias na classe PCB ou outras
* Na tool *Timer*, além da configuração do tempo de geração de interrupções, inclua uma interface para escolha dos algoritmos de escalonamento: FIFO, Prioridade Fixa ou Loteria
* Para testar, utilize o mesmo código de teste da Parte 1 e:
  + Imprima o PID dos processos na ordem que são escolhidos
  + Compare as escolhas do mesmo teste para cada algoritmo de escalonamento
  + Se for utilizado o algoritmo de escalonamento por prioridade use o seguinte código para testar:

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

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

j loop2

fim2: SyscallProcessTerminate

.include "macros.asm"

.data

.text

#criação dos processos com prioridade

SyscallFork(Programa1, 1)

SyscallFork(Programa2, 2)

SyscallFork(Idle, 0)

#escalonando o primeiro processo

SyscallProcessChange

Idle:

loop:

NOP

j loop