UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO – UFERSA

DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS

RODOLFO FELIPE MEDEIROS ALVES

JERFFERSON GOMES DUTRA

DESENVOLVIMENTO UM GERENCIADOR DE PROCESSOS E CHAMADAS DE SISTEA PARA O MARS 4.5

MOSSORÓ

2016

1. OBJETIVO

O objetivo desse trabalho é aplicar funcinalidades ao Mars 4.5, são três chamadas de sistemas inseridas, são elas: sendo a primeira chamada de *Fork,* na qual servirá para a criação de um novo processo, e a segunda chamada de *ProcessChange,* que permitirá trocar de processo em um determinado ponto do algoritmo e a terceira *ProcessTerminate* que tem como função terminar o processo que está executando no momento.

2. CLASSES ACRESCENTADAS

Para o desenvolvimento das novas funcionalidades foram criadas algumas classes para gerenciamento dos processos, são elas: ProcessControlBlock, ProcessManager, TableProcessors. Todas essas classes estão no pacote “mars.pluginRJ”.

2.1. ProcessControlBlock

A classe ProcessControlBlock(*ou PCB*) tem como objetivo armazenar as informações de contexto de um processo, como: informações do hardware: o conteúdo de todos os registradores, e informações lógicas: PID (Identificador do processo).

Essa classe também contém todos os métodos de *getters* e *setters* de todos os atributos.

```
public class ProcessControlBlock {
    |
    private int[] regValue;
    private int pc;
    private int hi;
    private int lo;
    private String pid;
```

2.2. TableProcessors

A classe TableProcessors representa a tabela de processos, no qual guarda todos os processos que estão executando, prontos ou bloqueados. Nesta classe são feitas todas

as operações referente aos processos, como: adicionar pcb, alterar pcb, remover pcb. Também nesta classe foi abstraido o **escalonador** em forma de métodos. Os algoritmos de escolha de processos são utilizados nesta classe, no qual só ela tem acesso(estão privados).

```
public class TableProcessors {

    //processo que está executando
    private String pidExec;
    //Todos os processos
    private HashMap<String,ProcessControlBlock> PCBTable;
    //Processos prontos
    private LinkedList<String> ReadyTable;
    //Processos bloqueados
    private ArrayList<String> BlockedTable;
```

2.3. ProcessManager

A classe ProcessManager tem como atributo uma TableProcessors, ou seja, é uma classe que gerencia e interliga as chamadas de sistemas com os processos que estão na tabela de processos(faz a comunicação).

```
public class ProcessManager {

    //numero do prox. pid de processo ou quanti
    public static int pidId = 0;
    //tabela de processos
    private static TableProcessors tableERB = new TableProcessors();

    /**
     * @param informações iniciais para criar um novo processo
     */
    public static void createProcess(int[] regValue, int pc , int hi, int lo) {
        tableERB.addPCB(new ProcessControlBlock(++pidId, regValue, pc, hi, lo));
    }

    /**
     * @param informações de guarda, ou seja, são as informações do processo que está executando
     * @return retorna um processo que irá executar.
     */
    public static ProcessControlBlock processChange(int[] regValue, int pc , int hi, int lo){
        if(tableERB.updatePCB(regValue, pc, hi, lo)){
            tableERB.processChange();
            return tableERB.getPcbExec();
        }else return null;
    }
```

3. CLASSES DE CHAMADA DE SISTEMA

Antes de criarmos nossas novas classes de chamadas de sistema, adicionamos três propriedades no arquivo *Syscall.properties:* Fork (19), e ProcessChange (20) e ProcessTerminate(21).

3.1. Fork

```java
public class Fork extends AbstractSyscall{

    public Fork() {
        super(19, "Fork");
    }

    @Override
    public void simulate(ProgramStatement statement) throws ProcessingException {
        ProcessManager.createProcess(   null,
                                        RegisterFile.getValue(4),
                                        0,
                                        0
                                    );
    }
```

As informações iniciais para criação de um novo PCB são repassadas para o gerenciador de processos(ProcessManager), o mesmo tem como função comunicar com a tabela de processos, a mesma se engarrega de instanciar um novo processo e guardar o processo criado na tabela de processos e adicionar o pid na fila de processos prontos para ser executados.

3.2. ProcessChange

A classe *ProcessChange* tem como objetivo informa ao gerenciador de processos os dados do processo que está em execução, e receber um novo processo para atualizar os registradores. Toda a mudança ocorre na tabela de processos, e só a mesma tem acesso e privacidade de fazer as trocas de processos.

```java
public class ProcessChange extends AbstractSyscall{

    public ProcessChange() {
        super(20, "ProcessChange");
    }

    @Override
    public void simulate(ProgramStatement statement) throws ProcessingException {

        //os argumentos são para guarda o estado do processo a ser mudado, o retorno é o novo processo
        ProcessControlBlock pcb = ProcessManager.processChange(     RegisterFile.getRegsValue(),
                                                                    RegisterFile.getProgramCounter(),
                                                                    RegisterFile.getValue(33),
                                                                    RegisterFile.getValue(34));
        //atualização do processo no registradores
        if(pcb != null){
            if(!pcb.isNull()){
                //debug
                OutputDebug.odProcessChange(pcb);

                for(int i = 0; i < 32; i++){
                    RegisterFile.updateRegister(i, pcb.getValueReg(i));
                }
                RegisterFile.setHi(pcb.getHi());
                RegisterFile.setLo(pcb.getLo());
            }

            RegisterFile.setProgramCounter(pcb.getPc());
```

3.3. ProcessTerminate

A Classe *ProcessTerminate* tem como objetivo termina o processo que está executando, e atualizar os registradores, caso ainda exista algum processo pronto para ser executado, está informação é feita pelo retorno da classe P*rocessManager.*

```
public class ProcessTerminate extends AbstractSyscall{

    public ProcessTerminate() {
        super(21, "ProcessTerminate");
    }

    @Override
    public void simulate(ProgramStatement statement) throws ProcessingException {
        ProcessControlBlock pcb = ProcessManager.processTerminate();

        //se existe algum processo para ser executado, as informações será adicionada nos registradore
        if(pcb != null){
            //debug
            OutputDebug.odProcessChange(pcb);

            for(int i = 0; i < 32; i++){
                RegisterFile.updateRegister(i, pcb.getValueReg(i));
            }
            RegisterFile.setProgramCounter(pcb.getPc());
            RegisterFile.setHi(pcb.getHi());
            RegisterFile.setLo(pcb.getLo());
        }else{
            new SyscallExit();
        }
    }
```