MOSS 0.2 - MELHORIAS NA INTERFACE E NOVAS FUNCIONALIDADES

Antonio V. Costa, Silvio Fernandes, Leiva C. Oliveira

Departamento de Computação - Centro de Ciências Exatas e Naturais, Universidade Federal Rural do Semi-Árido (UFERSA), Mossoró, Rio Grande do Norte, Brasil.

{antonio.costa,silvio,leiva.casemiro}@ufersa.edu.br

Resumo. *Este artigo descreve as alterações feitas sobre a interface do MOSS e a implementação de novas funcionalidades para os gerenciadores de processos, memória e arquivos, trazendo mais informações sobre o que está sendo simulado pelo sistema operacional e permitindo uma melhor compreensão do seu funcionamento para os estudantes. Essa nova versão foi denominada de MOSS 0.2, a qual é apresentada em detalhes.*

Abstract. *This article describes the changes made to the MOSS interface and the implementation of new features for process, memory and file managers, bringing more information about what is being simulated by the operating system and allowing a better understanding of its operation for students. This new version was called MOSS 0.2, which is presented in detail.*

1. Introdução

A disciplina de Sistemas Operacionais (SO) está presente em praticamente todo curso de graduação em computação ou informática. Tal disciplina apresenta os conceitos e estratégias algorítmicas que servem como ponte entre o *hardware* e *software*.

As principais funções de um SO estão no gerenciamento do acesso ao processador, a memória, a entrada/saída e sistema de arquivos. O gerenciamento resolve a disputa dos recursos físicos e lógicos compartilhados pelos programas, que do ponto de vista do SO são chamados processos, da forma mais justa para os propósitos do SO. Para tais funções, o SO precisa implementar soluções otimizadas para o *hardware* e ainda oferecer interfaces abstratas tanto para o programador quanto para o usuário das aplicações.

Assim, o ensino de SO envolve desafios em relação ao detalhamento de conceitos e implementações na prática, de modo que o aluno experimente algo o mais próximo possível do real em um tempo limitado do curso dessa disciplina.

Diante disso, foi desenvolvido o MOSS (MIPS Operating System Simulator) [1], uma ferramenta didática desenvolvida para atuar como uma *tool* do MARS (MIPS *Assembler and Runtime Simulator*) [2] que possibilita a simulação das principais funções de um sistema operacional. O objetivo da ferramenta MOSS é fornecer auxílio didático de forma que os estudantes possam visualizar as respostas de um SO por meio dos gerenciadores básicos de processos executando programas em *assembly* MIPS no simulador do MARS.

Os resultados obtidos com a validação da primeira versão do MOSS, aqui denominado de MOSS 0.1, mostram que a ferramenta cumpriu seu papel em auxiliar os professores no processo de ensino/aprendizagem da disciplina SO. Com a análise dos resultados obtidos na validação do MOSS 0.1, foram observados alguns pontos negativos em sua interface com destaque para a disposição dos objetos na tela, os nomes dos elementos visuais utilizados, a apresentação das informações e dos dados.

Portanto, este artigo apresenta uma nova versão do MOSS - denominada de MOSS 0.2, contento as implementações das sugestões obtidas durante o processo de validação do MOSS 0.1. Essa nova versão traz melhorias na interface dos gerenciadores presentes na ferramenta original, bem como a adição de novas funcionalidades para a ferramenta, apresentado mais informações acerca das chamadas executadas pelo sistema operacional.

O artigo está organizado da seguinte forma: a seção 2 apresenta referencial teórico e alguns trabalhos relacionados; a seção 3 discute a ferramenta e as alterações realizadas e a seção 4 apresenta as conclusões e trabalhos futuros.

2. Referencial Teórico

2.1 MARS

O MARS é um ambiente de desenvolvimento interativo para programação em linguagem assembly MIPS, destinado ao uso educacional. É multiplataforma, disponibiliza um editor de texto, um montador e um simulador. Oferece um conjunto de chamadas de sistema (*syscall*) que simula diversos serviços de SO durante a execução dos programas e pseudo instruções. Integrado ao simulador há um conjunto de ferramentas (*tools*) que se conectam ao simulador e fornecem informações extras em tempo de simulação. O MARS ainda oferece em sua implementação classes abstratas que permitem a criação de novas *syscalls*, novas ferramentas, novas pseudo instruções e/ou novas instruções [3].

Vale mencionar que o MIPS, acrônimo para *Microprocessor without Interlocked Pipeline Stages*, é uma arquitetura de microprocessadores RISC desenvolvida pela MIPS Computer Systems e utilizada em diversas aplicações. Além disso, é uma das arquiteturas mais utilizadas para o ensino da disciplina Organização e Arquitetura de Computadores (OAC).

2.2 MOSS 0.1

O MOSS (MIPS *O*perating System *Simulator*) é uma ferramenta que integra *syscalls* e *tools* desenvolvidas para o MARS e permite simular as principais funções de um sistema operacional [1]. Foi concebido para ser parte integrante da metodologia interdisciplinar de ensino de adotada na UFERSA (Universidade Federal Rural do Semiárido) [4]. Na UFERSA as disciplinas de SO e de OAC estão intensamente relacionadas com o uso integrado do MARS e do MOSS. Dessa forma, os alunos são instruídos a entenderem o código do MARS para o desenvolvimento de extensões desse ambiente, possibilitando a implementação de funcionalidades de um SO valendo-se do MARS como ambiente de simulação e teste.

Conforme mostrado por [1] o MOSS implementa 3 (três) gerenciadores de um SO: o gerenciador de processo (GP), o gerenciador de memória (GM) e o gerenciador de arquivos (GA), bem como as *syscalls* relacionadas a eles. MOSS foi implementado como uma *tool* do MARS, sendo então encontrado no menu *tool* do ambiente, e quando conectado ao ambiente principal - por meio do botão "*Connect to MIPS*" presente na interface da ferramenta, passa a observar e atuar no estado interno do simulador MIPS. A Figura 1 mostra a janela principal do MOSS 0.1 com os 3 gerenciadores escolhidos. Para maiores detalhes dessa primeira versão consultar [1].

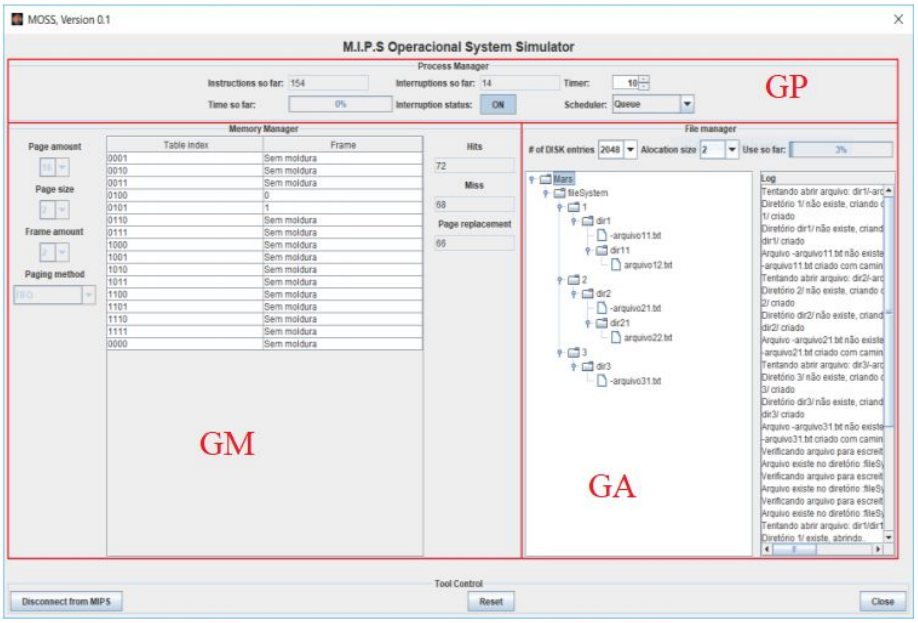

Figura 1. Interface do MOSS 0.1 com todos os gerenciadores iniciados.

2.3 Trabalhos relacionados

Um dos autores mais referenciados no estudo de Sistemas Operacionais (SO) é o Andrews S. Tanenbaum. Em seus livros [5, 6], ele apresenta os conceitos e a implementação do SO MINIX [7] e defende que para entender os conceitos na prática, um estudante de computação precisa "dissecar" o código de um SO assim como um estudante de biologia precisa dissecar um sapo. O MINIX evoluiu mas seu livro texto não conseguiu acompanhar, estando atualmente dessincronizados, o que dificulta uma metodologia de ensino usando os dois. Em [8] é apresentado um ambiente de laboratório virtual que utiliza uma infraestrutura híbrida de MINIX e Linux para o ensino de tópicos relacionados a segurança. Também usando um SO real, modificável em atividades durante uma disciplina, está o MiniOS [9], implementado por meio de um kernel virtual junto com um kernel Linux.

Os trabalhos de [10] tem o objetivo de melhorar o aprendizado em SO usando metodologia de gameficação com "palavras cruzadas" e o jogo perguntas e respostas "Jeopardy!". Eles também propõem "batalha de threads", inspirado em batalha naval e "jogo da transição de estado dos processos". Usando a abordagem baseada em problema [Yi-Ran, et all, 2010] foca em escalonamento de processos e o problema de *deadlock*.

O uso de simuladores é particularmente interessante no ensino de SO, no sentido de diminuir a abstração. O simulador SOsim é apresentado em [12], que aborda as principais funções de gerenciamento de forma visual e simplificada. Em [13] é apresentado o S^2O para simulação do escalonamento de processos. Já em [14] é apresentado um simulador multiplataforma, por ter sido implementado em Java, que já considera os aspectos de escalonamento e gerenciamento de memória. Simuladores mais tradicionais e mais acurados em relação às funções de um SO, por trabalharem em nível instrucional, como RCOS [15] e sua versão web RCOS.Java [16] podem ser encontrados. Uma ferramenta voltada para a arquitetura MIPS destaca-se o Nachos [17] que é um framework de SO funcional com simulador embutido da arquitetura MIPS.

3. O simulador MOSS 0.2

A nova interface do MOSS 0.2 é apresentada na Figura 2. É possível observar que foi realizado uma reorganização em como as informações antigas eram exibidas bem como foi feito uma melhora na nomenclatura dos *label* que definem e identificam os parâmetros de configurações de cada gerenciador, a fim de tornar o uso da ferramenta mais intuitiva e com uma maior possibilidade de auxiliar o ensino/aprendizagem. Também foram adicionadas as novas funcionalidades e informações nos gerenciadores. Nas próximas seções serão detalhadas as alterações dos gerenciadores para o MOSS 0.2.

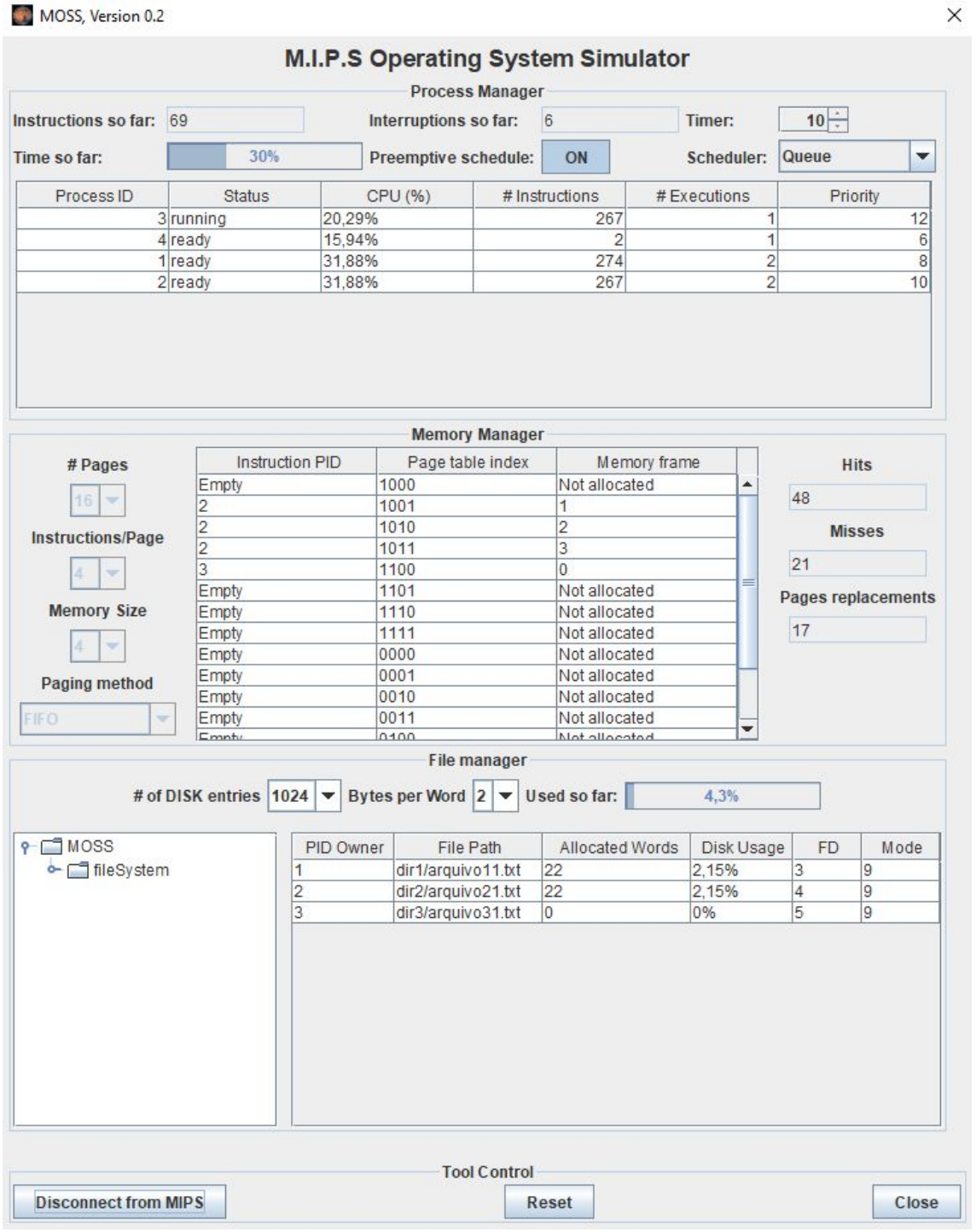

Figura 2. Nova interface do MOSS 0.2.

A estrutura da *syscall* e das macros presentes na versão 0.1 do MOSS foram mantidas na versão atual, devido a sua praticidade ao realizar as chamadas de sistemas que compõem a simulação de um SO. No MARS, para fazer uma chamada de sistema o programador deve incluir o código da chamada no registrador $v0, adicionar os parâmetro em outro registrador quando houver necessidade, e em seguida a palavra reservada **syscall**. Assim o arquivo de macros contendo todas as *syscall* utilizadas pelo

MOSS foi criado. No MOSS 0.2 foi mantida a *syscall* chamada de *SyscallMOSS* e definida pelo código 18, número disponível no MARS. O arquivo das macros deve ser incluído no código assembly que irá utilizar as funcionalidades do MOSS 0.2.

Um trecho do arquivo de macros é apresentado na Figura 3. Como mostra a figura, a *SyscallMOSS* tem o número 18 (linhas 60, 82 e 89). A combinação entre o código da *syscall* e o valor do parâmetro adicionado no registrado $t7 identificam a ação que será executada. O valor do registrador $t7 deve conter uma das opções: 1- para a criação de processo (*fork*), 2- para a troca de processos voluntária (*processChange*) e 3- para a finalização de processo (*processTerminate)*. Em seguida, basta chamar uma das macros passando os parâmetros, quando necessário, para utilizar as funções da *SyscallMOSS*. Como exemplo, para criar um novo processo deverá ser chamada a macro *fork* passando os seguintes parâmetros: uma label para a primeira instrução do processo, uma label para a última instrução do processo, o valor da prioridade inicial do processo, o valor da prioridade mínima em que o processo pode chegar e o valor da prioridade máxima que o processo pode ter. Já para as opções de ProcessChange e ProcessTerminate, basta chamar a macro referente a opção desejada sem passagem de parâmetro. Mais detalhes sobre o arquivo de macros do MOSS podem ser encontrados na referência [1].

Figura 3. Trechos de códigos assembly para definição das macros. Detalhes em A, B e C da combinação (código da syscall, parâmetros) usada para identificar qual ação deve ser executada.

3.1 O Gerenciador de Processos

O gerenciador de processos é encarregado de todos os aspectos do controle de "vida" dos processos e *syscalls* relacionadas, sendo responsável pela criação, escalonamento, manutenção das informações e término dos processos [1].

A Figura 4 apresenta em destaque o GP do MOSS 0.2. Para utilizar o gerenciador de processos, o usuário deve configurar os parâmetros que definem como se dará seu funcionamento. No MOSS 0.2 o escalonamento pode ser preemptivo e não-preemptivo, com possibilidade de escolha pelo usuário. O botão *preemptive schedule* inicialmente fica com o valor ***OFF*** indicando que a interrupção de sistema será não-preemptivo, ou seja, o gerenciador de processo só fará o escalonamento entre

os processos criados caso a *syscall processChange* seja chamada. Para ativar a interrupção de sistema o usuário deverá pressionar o botão para que ele fique em modo ***ON***, assim como mostra a Figura 4. Com a opção ativa, o usuário pode determinar o intervalo em quantidade de instruções que ocorrerá a interrupção. O parâmetro *Timer* determina o número de instruções executadas antes de acontecer um escalonamento preemptivo para outro processo (valor padrão é 10). O gerenciador fará o escalonamento (substituição) do processo por um outro que esteja no estado pronto - dentre os que estão na lista de processos criados, utilizando o algoritmo de escalonamento definido na opção *Schedule*. Os métodos de escalonamento disponíveis no MOSS 0.2 são: *queue* (fila, método padrão ao iniciar o gerenciador), *fixed priority* (prioridade fixa), *dinamic priority* (prioridade dinâmica) e *lottery* (loteria, escolha aleatório do processo).

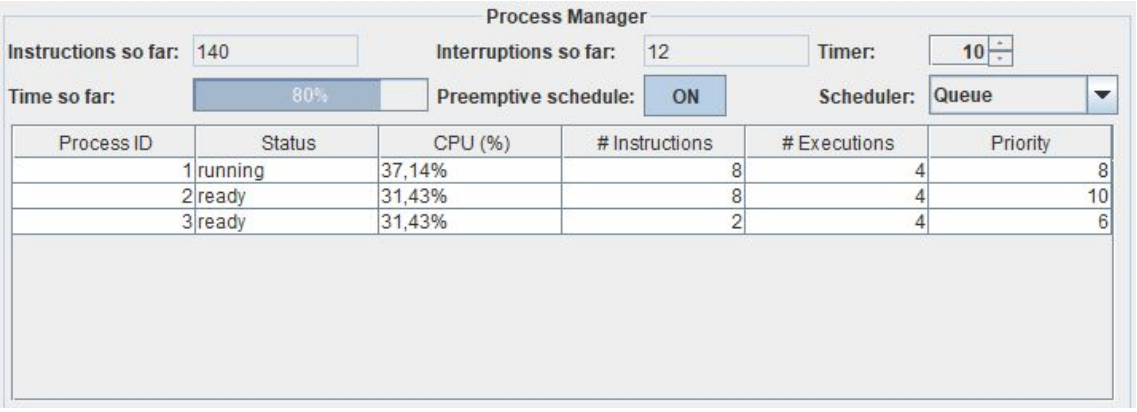

Figura 4. Nova interface do gerenciador de processos do MOSS 0.2.

Logo após a chamada de sistema para a criação de um novo processo for executada, o gerenciador criará esse processo e o exibirá na tabela, para cada processo, as informações pertinentes a ele. Essas informações são: *process ID* (um número que será o identificador único do processo); O *status* (o estado atual do processo, podendo esses serem *ready* (pronto) ou *running* (executando)); O *CPU (%)*, que será a porcentagem de CPU utilizada pelo processo, calculado da forma: $\frac{X}{Y} \times 100$, onde x é a quantidade de instruções executadas pelo processo e y é o total de instruções executadas desde que o gerenciador foi iniciado; O *#Instructions* representa o número de instruções que o processo possui; O *#Executions* representa o número de vezes que o processo foi executado e a *Priority* que é a prioridade atual do processo, já que o mesmo pode ter sua prioridade alterada por um método de escalonamento com base em prioridade dinâmica.

O intuito da inclusão dessa tabela foi o de trazer mais informações sobre os processos criados pelo código *assembly*, permitindo que o usuário possa saber qual processo está em execução, quais deles estão prontos e suas informações diretamente na interface da ferramenta, podendo sanar dúvidas geradas pelos conceitos estudados sobre o funcionamento de um gerenciador de processos, dos métodos de escalonamentos e de interrupção de sistema.

Além dos dados que foram adicionadas com a criação da tabela, outras informações podem serem vistas assim como na versão anterior do MOSS. Tais informações são: a quantidade de instruções executadas até o momento pelos processos

em execução (*Instructions so far*), a quantidade de interrupções feitas pela interrupção de sistema (*Interruptions so far*), caso a mesma esteja ativa uma barra de progresso que exibe o percentual de tempo (em número de instruções) até a próxima interrupção de sistema (*Timer so far*), caso ela esteja ativa.

Para testar o novo GP do MOSS 0.2 foi utilizado um código *assembly* conforme apresentado na Figura 5. Esse código é responsável pela criação de três processos, denominados de: programa1, programa2 e idle. O processo "programa1" (Figura 5, P1), realiza uma soma unitária dentro de um laço de repetição em um registrador até que o valor do mesmo atinja 10, fazendo uma chamada de sistema para o *processTerminate* após isso. O processo "programa2" (Figura 5, P2) realiza uma subtração unitária dentro de um laço de repetição em um registrador até que seu valor atinja -10, também chamando o *processTerminate* ao atingir essa condição. Já o processo "idle" (Figura 5, P0), realiza uma soma de zero com zero no registrador zero e salta para essa mesma instrução indefinidas vezes, fazendo dele um laço "infinito", além disso, esse processo não realizada a chamada responsável pela finalização de processos, fazendo com que o mesmo fique em execução mesmo após a finalização dos demais.

Ao efetuar uma troca de processos, o código possibilita a visualização das mudanças ocorridas nos registradores e na memória do MARS. Assim, foi possível observar que o MOSS 0.2 salva o contexto (valores de todos os registradores, endereços de memória e a instrução atual executada) do processo em execução antes de trocá-lo por um outro processo. A recuperação de tais valores referentes ao novo processo que será executado e a modificação do PC (*process counter*) do MARS - para que ele representa a próxima instrução a ser executada do novo processo, também foram satisfatoriamente realizadas.

As execuções ocorreram de forma sucinta, sem a constatação de erros, todos os processos foram criados, executados e finalizados em seu devido tempo. Além disso, foi observado que as informações contidas na tabela condizem com as presentes em cada processo, mostrando sua eficácia em representar essas informações na interface do GP.

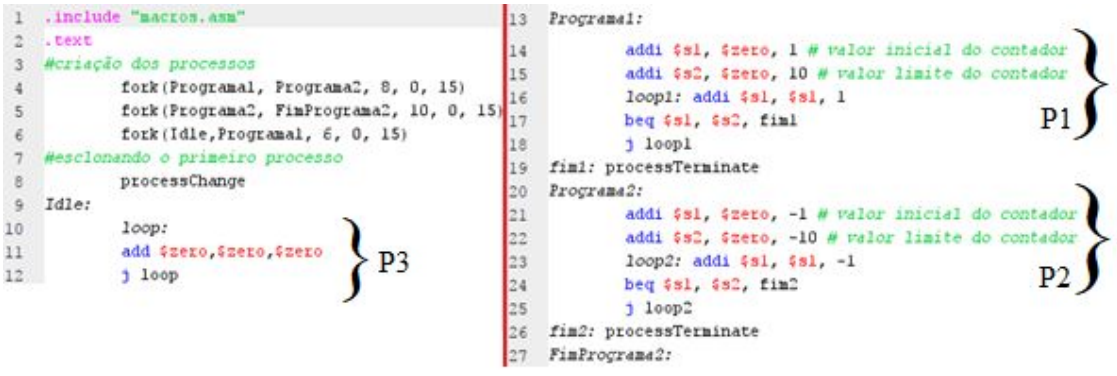

Figura 5. Código em *assembly* para testar o GP do MOSS 0.2. Código utilizada para a criação de processos.

3.2 O Gerenciador de Memória

O gerenciador de memória mantém o controle sobre quais partes da memória estão em uso e quais não estão, podendo alocar memória aos processos quando eles

precisam e liberando-nas quando estes terminam. Efetua também a paginação e a troca de páginas, ou parte delas (*swapping*), entre memória e disco quando a memória principal não é suficiente. Esse gerenciador não oferece *syscall*, mas todos os endereços de memória do segmento de código são monitorados assim que um processo é criado pelo gerenciador de processo, de modo que acessos fora do limites do espaço de endereçamento dos processos são proibidos pelo MOSS [1].

A Figura 5 apresenta em destaque o GM do MOSS 0.2. A configuração desse gerenciador se dá pela escolha dos valores referentes ao: número de páginas virtuais existentes (*# Pages*), o número de instruções que conterá em cada página (*Instructions/Page*), o número de molduras que simulará a memória física (*Memory Size*), ou seja, quantos espaços a memória física possuirá para armazenar páginas de instruções (o tamanho da memória principal) e por último, o método de paginação (*Paging method*) que define qual algoritmo será usado quando uma instrução a ser executada não estiver em uma página contida em uma das molduras de página da memória física - situação em que faz-se necessário a troca (*swap*) de uma dessas páginas por aquela que contém a instrução desejada. Os métodos implementados para substituição de paginação são: Não usada recentemente, fila, segunda chance (*NRU*, *FIFO*, *Second chance*, respectivamente).

Ao ser executada a primeira instrução de um dos processos criados, será atribuído a essa instrução uma página da tabela e á essa página uma moldura na memória física. Além disso, as informações sobre cada página será carregada para a tabela presente na interface do gerenciador de memória, exibindo o identificador do processo a quais as instruções da página pertencem (*Instruction PID*), o index da tabela de páginas (*Page table index*) e a moldura (*Memory frame*) daquela página na memória física, caso esteja alocada. Além dessas informações presentes na tabela, o gerenciador continua exibindo as informações presentes na versão anterior, sendo elas a quantidade de vezes que a instrução a ser executada estava presente em uma página que estivesse na memória física (*Hits*), a quantidade de vezes em que não estava (*Miss*) e a quantidade de vezes em que foi necessário efetuar uma substituição de páginas dentre aquelas que estavam na memória física e as que não estavam (*Page replacement*).

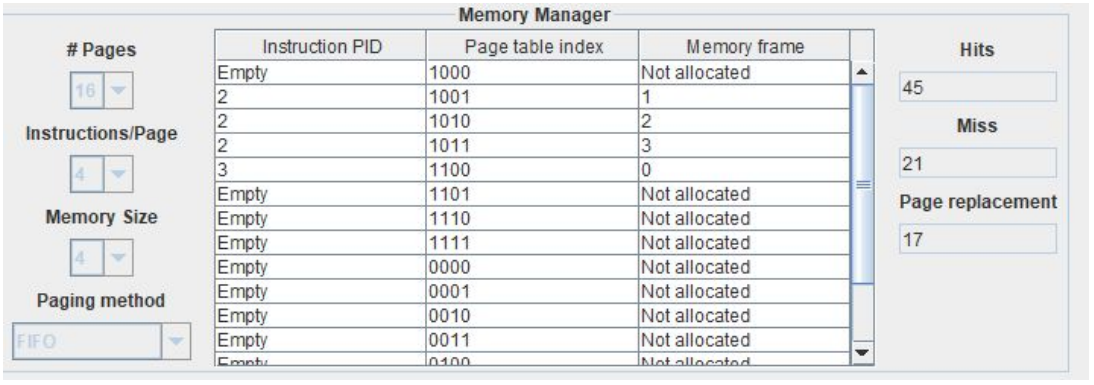

Figura 5. Nova interface do gerenciador de memória do MOSS 0.2.

Comparando com o GM da versão 0.1, nota-se que foi realizada uma modificação na nomenclatura das informações que são exibidas e das que são

solicitadas como parâmetros de configurações, a fim de tornar seu uso mais intuitivo para o usuário.

Assim como no gerenciador de processos, foram realizados testes a fim de verificar a existência de possíveis erros com as alterações realizadas. O GM foi, dentre os três, o que sofreu o menor número de modificações, sendo adicionado apenas uma nova coluna na tabela de informações e a renomeação de *labels*. Para efetuar os teste de gerenciamento de memória, código *assembly* apresentado na Figura 5 foi empregado, uma vez que sua execução ocasiona alterações na memória as quais são passíveis de gerenciamento.

Foi verificado que as modificações realizadas não introduz erros em seu funcionamento e que a nova informação adicionada à tabela condiz com a informação referente ao identificador do processo proprietário da instrução alocada na página.

3.3 O Gerenciador de Arquivos

O gerenciador de arquivos atua como um simplificador para a organização, compartilhamento e o gerenciamento de grandes volumes de informação em arquivos de forma abstrata para o programador. Esse gerenciador não implementa nenhuma nova *syscall*, ele utiliza as já existem no MARS para o controle e manuseio de arquivos [1].

Essas chamadas de sistema são responsáveis pela abertura de arquivos (*SyscallOpen*) com o código 13, leitura de arquivos (*SyscallRead*) com código 14, pela escrita em arquivos (*SyscallWrite*) com código 15 e pelo fechamento de um arquivo (*SyscallClose*) com código 16. Ao iniciar o gerenciador de arquivos ele será executado sempre que for feito uma das chamadas de sistemas responsáveis por manipular arquivos no MARS. Para realizar a abertura de um arquivo, o usuário chama a *syscall* correspondente passando os seguintes parâmetros nos registradores: o nome do arquivo (incluindo os diretórios) no registrador $a0, um número representando as *flags* de abertura do arquivo no registrador $a1, sendo esses valores: 0 para o modo de leitura, 1 para escrita sobrescrevendo o texto antigo no arquivo e 9 para escrita adicionando o novo texto ao final do arquivo (*append*). Caso o modo de abertura seja escrita (1 ou 9) o gerenciador de arquivo verifica se o arquivo e seus diretórios já foram criados. Caso não tenham sido, faz a criação dos mesmos e os adicionam para serem exibidos na árvore de arquivos presente na interface e na tabela de arquivos abertos. Caso o modo de abertura seja para leitura (0), o gerenciador espera que o arquivo já esteja criado. Após efetuar essas verificações e/ou criações de arquivos e diretórios, a chamada de sistema retorna o descritor do arquivo (*file descriptor*), que é um número único representando um identificador para o arquivo aberto.

Ao realizar uma chamada de sistema a fim de efetuar uma operação de leitura, escrita ou fechamento de um arquivo, o usuário deve fazer a chamada correspondente passando o *file descriptor* retornado na abertura do arquivo juntamente com os outros parâmetros solicitados por cada chamada. Para a leitura dos dados de um arquivo, além do *file descriptor*, o usuário deve passar um buffer onde será armazenado o texto lido e o número de *bytes* que serão lidos. A chamada realiza a leitura do arquivo e armazena o texto lido no *buffer* que foi passado, retornando o número de *bytes* lidos no registrador $v0. Já para a escrita em arquivos, a chamada de sistema pede o *file descriptor* do arquivo aberto, um *buffer* contendo o texto a ser escrito no arquivo e o número de *bytes*

a serem escritos. Com essas informações, o gerenciador verifica se existe espaço suficiente em disco para armazenar o número *words* que representam os *bytes* solicitados. O número de *bytes* por *word* é determinado como parâmetro de configuração na interface do gerenciador. Caso haja espaço suficiente, ele efetua a escrita do arquivo, subtrai a quantidade armazenada da quantidade de blocos do disco e retorna o número de *bytes* escritos. A fim de efetuar o fechamento de um arquivo aberto, o usuário deve informar apenas o file descriptor ao realizar a chamada do sistema.

A Figura 6 apresenta em destaque o GA do MOSS 0.2. Com o intuito de melhor representar as informações sobre quais arquivos estão abertos pelo MARS, foi adicionado uma tabela na interface do MOSS para que exibisse as informações desses arquivos. Essas informações representam o identificador do processo que criou o arquivo (*PID Owner*), o caminho até o arquivo, mostrando seus diretórios (*File Path*), a quantidade de palavras alocadas pelo arquivo no disco (*Allocated Words*), a porcentagem de uso de disco por aquele arquivo (*Disk Usage*), o identificador do arquivo (*FD*, ou *File Descriptor*) e o modo de abertura do arquivo (*Mode*).

Assim como feito nos demais gerenciadores, foi realizada uma modificação na nomenclatura das informações que são exibidas e das que são solicitadas como parâmetros de configurações, a fim de tornar seu uso mais intuitivo para o usuário. As informações solicitadas como parâmetros de configurações se restringem somente ao número de entradas (ou número de blocos) que o disco possuirá (*# of DISK entries*) e a quantidade de *bytes* que cada bloco armazenará (*Bytes per Word*), considerando um bloco como uma *word*. Além disso, as informações presentes na versão anterior do MOSS foram mantidas, sendo eles a barra de progresso que exibe o volume do disco usado até o momento (em porcentagem) e a árvore de arquivos contendo todos arquivos criados e seus diretórios, permitindo a navegação entres eles ao usuário.

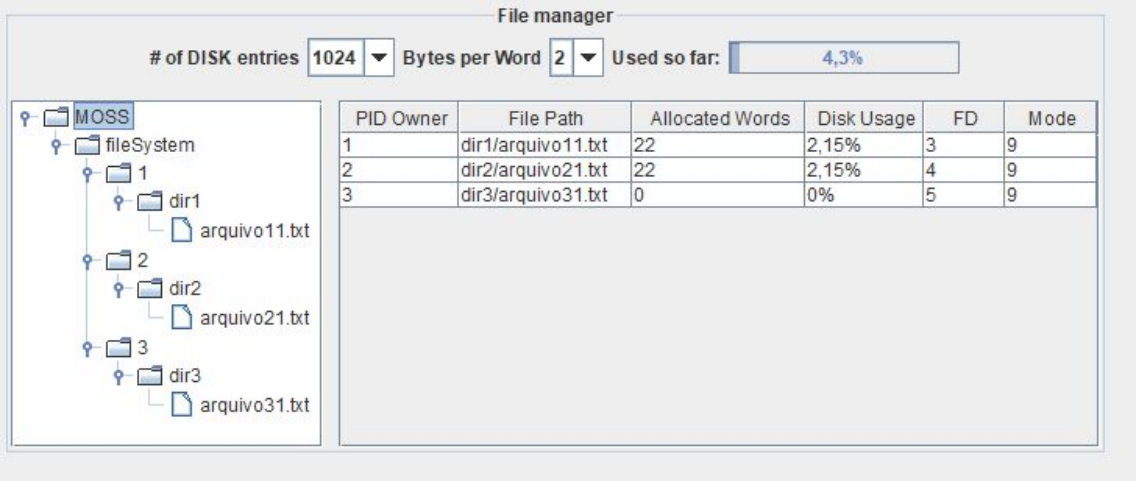

Figura 6. Nova Interface do gerenciador de arquivos do MOSS 0.2.

O código *assembly* que simula a criação, escrita, leitura e fechamento de alguns arquivos com seus respectivos diretórios é apresentado na Figura 7. Esse código é composto pela criação de três processos. Cada um deles efetua uma operação abertura de um arquivo, passando o valor do modo de abertura (valor 9 para o modo *append*) e seu caminho, cujo valor fora determinado na variável *file* com o número respectivo ao

processo. Após isso, realiza uma escrita nesse arquivo do texto contido na variável *body,* também nomeada com o número respectivo do processo proprietário. Depois de escrever no arquivo, cada processo irá efetuar chamadas a fim de fechá-lo e reabri-lo, alterando o modo de abertura para leitura (valor 0 no parâmetro referente ao modo de abertura). Em seguida, o processo efetua a chamada de sistema referente a leitura do deste arquivo, exibindo a informação lida na tela de *run I/O* do MARS. Caso não ocorra nenhum erro durantes as chamadas realizadas o processo efetua o fechamento do arquivo e a chamada referente a finalização de processo para terminar sua própria execução.

O código foi executado pelo MARS realizando as chamadas de sistemas para as funcionalidades do MOSS sem apresentar nenhum erro durante o processo. Também foi verificado que as informações apresentadas pela tabela de arquivos abertos introduzida no gerenciador de arquivos no MOSS 0.2 condizem com as informações dos arquivos que foram manipulados durante a execução.

```
.include "macros.asm"
.data
# Saida
output1: .space 1024
output2: .space 1024
output3: .space 1024

# DefiniÃ§Ã£o de arquivos
file1: .asciiz "dir1/arquivo11.txt"
file2: .asciiz "dir2/arquivo21.txt"
file3: .asciiz "dir3/arquivo31.txt"

# body writing
body1: .asciiz "The quick brown fox jumps over the lazy dog " # tem 44 caracteres
body2: .asciiz "The slowly red fox was taken by the lazy dog" # tem 44 caracteres
body3: .asciiz "The quick brown dog knocked the lazy red fox" # tem 44 caracteres

.text
#criaÃ§Ã£o dos processos
        fork(Programa1, Programa2, 8, 0, 15)
        fork(Programa2, Programa3, 10, 0, 15)
        fork(Programa3, FimPrograma3, 12, 0, 15)
#escalonando o primeiro processo
        processChange

Programa1: # escreve no dir1/arquivo11.txt
        opening(file1,9,rightOpen11,fimAll1) # abrindo aqruivo para escrita
rightOpen11:
        add $s0,$v0,$zero # salvando file description
        writing(body1,44,rightWrite11,fimAll1) # escrevendo no arquivo
rightWrite11:
        closing() # fechando arquivo
        opening(file1,0,rightOpenR1,fimAll1) # abrindo aqruivo para leitura
rightOpenR1:
        add $s0,$v0,$zero # salvando file description
        reading(output1,44,rightRead1,fimAll1) # lendo do arquivo
rightRead1:
        prinThis(output1) # exibindo arquivo lido
fimAll1:
        closing() # fechando arquivo
        processTerminate # finalizando o processo
Programa2: # escreve no dir2/arquivo21.txt
        opening(file2,9,rightOpen21,fimAll2) # abrindo aqruivo para escrita
rightOpen21:
        add $s0,$v0,$zero # salvando file description
        writing(body2,44,rightWrite21,fimAll2) # escrevendo no arquivo
rightWrite21:
        closing() # fechando arquivo
        opening(file2,0,rightOpenR2,fimAll2) # abrindo aqruivo para leitura
rightOpenR2:
        add $s0,$v0,$zero # salvando file description
        reading(output2,44,rightRead2,fimAll2) # lendo do arquivo
rightRead2:
        prinThis(output2) # exibindo arquivo lido
fimAll2:
        closing() # fechando arquivo
        processTerminate # finalizando o processo
Programa3: # escreve no dir3/arquivo31.txt
        opening(file3,9,rightOpen31,fimAll3) # abrindo aqruivo para escrita
rightOpen31:
        add $s0,$v0,$zero # salvando file description
        writing(body3,44,rightWrite31,fimAll3) # escrevendo no arquivo
rightWrite31:
        closing() # fechando arquivo
        opening(file3,0,rightOpenR3,fimAll3)  # abrindo aqruivo para leitura
rightOpenR3:
        add $s0,$v0,$zero # salvando file description
        reading(output3,44,rightRead3,fimAll3) # lendo do arquivo
rightRead3:
        prinThis(output3) # exibindo arquivo lido
fimAll3:
        closing() # fechando arquivo
        processTerminate # finalizando o processo
FimPrograma3:
```

Figura 7. Código *assembly* para testar o GA do MOSS 0.2. Colocar marcações explicativas na figura

4. Conclusões e trabalhos futuros

Este artigo apresentou alterações feitas na interface e a implementação de novas funcionalidades no MOSS 0.1, uma ferramenta didática para Sistemas Operacionais (SO) integrada ao MARS, a qual estende e cria novos *syscalls* e *tools*, denominada MOSS 0.2.

O MOSS 0.2 inclui de forma parametrizável os gerenciadores de processo, de memória e de arquivos, os quais são utilizados a partir do código em *assembly* do criado pelo o usuário e executado no MARS.

Após os testes realizados em cada gerenciador foi possível verificar que as alterações propostas no MOSS 0.2 não introduziram erros na execução de códigos *assembly* no MARS, além de trazerem mais informações acerca dos processos, memória e arquivos que podem ser manipulados pela ferramenta. Desse modo, novas atividades poderão ser produzidas com base nessas informações com a possibilidade de melhoras no processo de ensino/aprendizagem das disciplinas de SO que façam uso do MOSS.

Como trabalhos futuros, podemos usar a infraestrutura do gerenciador de processos para orientar outros alunos a desenvolverem soluções de mutexes, monitores, semáforos e implementação dos problemas clássicos de IPC para testá-los. É possível implementar novos algoritmos de escalonamento, bem como modificar os critérios de alteração de prioridade dinâmica já implementado.

O gerenciador de memória atualmente monitora apenas os endereços do segmento de código, assim, também é possível estender para o segmento de dados, onde existe permissão de leitura e escrita. Também é possível fazer uma integração maior entre o gerenciador de memória e o de arquivos durante o processo de substituição de página, envolvendo o bloqueio do processo, o acesso às informações de localização no disco e demais opções de segurança.

Com o gerenciador de arquivos pretende-se criar e exibir na interface gráfica uma forma mais clara e didática do acesso aos *inodes* e blocos do disco. Com exceção do *timer*, todo um gerenciamento de entrada/saída pode ser desenvolvido.

Ainda há espaço para integração com um compilador de uma linguagem em alto nível para *assembly* do MIPS, o qual pode aumentar a interdisciplinaridade. O trabalho prospecta o desenvolvimento de uma nova *tool* que funcione como um interpretador de comandos para o MOSS 0.2.

Outros planos incluem aprimorar o suporte à depuração e realce do conteúdo dos processos, memórias e arquivos modificados passo a passo na execução.

Além disso, o uso da ferramenta deve acompanhar uma série de atividades para serem feitas juntamente com ela em sala de aula, com intuito de melhorar o processo de ensino/aprendizagem da disciplina de SO. Essas melhorias realizadas no MOSS 0.2 estão cabíveis de passarem por uma validação com especialistas - incluindo o grupo mantenedor do MARS, a fim de demonstrar se as alterações propostas melhoram, de

fato, o acesso a informações que podem ajudar em um melhor entendimento sobre os conceitos de funcionamento de um SO.

5. Referências

[1] Costa, Antonio V. T., Silva, Arthur A. A., Fernandes, Silvio, Macedo, Francisco T. (2018). "MOSS - Uma Ferramenta para o Auxílio do Ensino de Sistemas Operacionais". XXIX Simpósio Brasileiro de Informática na Educação (Brazilian Symposium on Computers in Education), 2018, Fortaleza, 2018. p. 755.

[2] Vollmar, K. and Sanderson, P. (2006) "MARS: An Education-Oriented MIPS Assembly Language Simulator," in ACM SIGCSE.

[3] Vollmar, K. and Sanderson, P. (2007) "An Assembly Language I.D.E. To Engage Students Of All Levels," presented at the 2007 CCSC.

[4] Fernandes, Silvio and Silva, Ivan Saraiva. (2017). Relato de Experiência Interdisciplinar Usando MIPS. In *International Journal of Computer Architecture Education*, V.6, n. 1, pág. 52, SBC.

[5] Tanenbaum, A. S. (2008), Sistemas Operacionais. Projeto e Implementação, 3rd ed. Bookman.

[6] Tanenbaum, A. S. (2009) Sistemas Operacionais Modernos, 3a. ed. São Paulo: Pearson.

[7] Tanenbaum, A. S. "MINIX 3" (2006) [Online]. Disponível em: http://www.minix3.org/.

[8] Du, Wenliang, and Wang, R. (2008) "SEED: A Suite of Instructional Laboratories for Computer Security Education". J. Educ. Resour. Comput. 8, no 1 (março): 3:1–3:24. https://doi.org/10.1145/1348713.1348716.

[9] Otero, R., Rafael, and Aravind, A. A. (2015) "MiniOS: An Instructional Platform for Teaching Operating Systems Projects". In Proceedings of the 46th ACM Technical Symposium on Computer Science Education, 430–435. SIGCSE '15. New York, NY, USA: ACM. https://doi.org/10.1145/2676723.2677299.

[10] Hill, J., Ray, C. K., Blair, J. R., and Carver Jr, C. A. (2003) "Puzzles and games: addressing different learning styles in teaching operating systems concepts," in ACM SIGCSE Bulletin, vol. 35, pp. 182–186.

[11 Yi-Ran, H., Cheng, Z., Feng, Y., and Meng-Xiao, Y. (2010) "Research on teaching operating systems course using problem-based learning," in 5th International Conference on Computer Science & Education, 2010, pp. 691–694.

[12] Machado, Berenger, F., and Maia, L. P. (2007), Arquitetura de Sistemas Operacionais. 4o ed. LTC.

[13] Carvalho, D. S, Balthazar, G. R. , Dias, C. R. , Araújo, M. A. P. and Monteiro, P. H. R. (2006) "Simulador para a Prática de Sistemas Operacionais". Revista Eletrônica da FMG.

[14] Tonini, Gustavo Alexssandro, and Lunardi, S. C. (2006) “Simulador para o Aprendizado de Sistemas Operacionais” in conferencia latinoamericana de informática, 2006.

[15] Jones, D. and Newman, A. (2002). A Constructivist-based Tool for Operating Systems Education. In P. Barker & S. Rebelsky (Eds.), Proceedings of ED-MEDIA 2002--World Conference on Educational Multimedia, Hypermedia & Telecommunications (pp. 882-883). Denver, Colorado, USA: Association for the Advancement of Computing in Education (AACE). Retrieved March 29, 2018 from https://www.learntechlib.org/p/10276/..

[16] Jones, David, e Newman, A. (2001) “RCOS.java: a Simulated Operating System with Animations”. In Proceedings of the Computer-Based Learning in Science Conference. Rep. Tcheca

[17] Christopher, Wayne A., Procter, S. J., and Anderson T. E. (1992) “The Nachos Instructional Operating System”. EECS Department, University of California, Berkeley, novembro. http://www2.eecs.berkeley.edu/Pubs/TechRpts/1992/6022.html.