Relatório Experimento 7
Sistemas Operacionais 1 – Turma 3 Grupo 8
Winux

Professor Tobar

Bruno de André Mazzoco RA: 02150522
César Henrique Kallas RA: 02099224

Introdução

Durante todo o curso de Sistemas Operacionais, estudou-se o funcionamento do mesmo, como processos, escalonamento de processos, comunicação entre os processos, gerencia de memória e sistemas de arquivos.

Como a complexidade de implementação de um Sistema Operacional é muito alta, em experimentos no laboratório, estudava-se apenas modificações relacionadas com a teoria, bem como implementações básicas de algoritmos.

Após passarmos por todos esses tópicos e estudos de algoritmos, esse experimento chega para demonstrar o funcionamento da integração entre todos os tópicos estudados, de uma forma mais simples e próxima do aluno, através do software Winux.

Com tudo isso, pode-se ter uma melhor noção do funcionamento e implementação, mesmo usando uma versão simplificada de um Sistema Operacional, também poder alterá-lo de maneira a implementar ou modificar estudos vistos na teoria, perceber a complexidade de ajustar um código não produzido por você mesmo e treinar a agilidade de trabalhar em equipe.

O Winux

O Winux é um projeto de Sistema Operacional simplificado feito na linguagem de progamação Delphi. É um software como outro qualquer, que funciona em conjunto com o Sistema Operacional proprietário Microsoft(r) Windows(r), sendo assim, ele compartilha o processamento com o mesmo e outros softwares que estejam ativos no momento na unidade de processamento central, processador.

O Winux é multiprogramado, executa programas de um usuário e tem uma interface gráfica agradável. Sua principal função é simular um Sistema Operacional de maneira mais simplificada.

Ele contém sete processos concorrentes que atuam sobre os periféricos, buffers, disco, memória e outros recursos, como blocos de controle

de programas e variáveis de controle, veja figura 1.

Processos Simples

- Spooling
  - Leitura
  - Spoll de entrada
  - Spool de saída
  - Impressão
- Entrada e Saída do Usuário
- Paginação
- Loader

Processos Especiais

- Tratamento de Interrupções
- Escalonamento
- Espera

Leitura : coloca nos buffers de entrada os conteúdos das páginas de informação, seja dado ou código, dado pelo usuário. É executado sempre que existem dados para serem lidos, a leitora não está sendo ocupada e existem buffers disponíveis.

Spool de entrada : É responsável pela interpretação das páginas de informações lidas dos buffers de entrada e pela colocação dos programas e dados no disco, bem como pela abertura e espaço para impressão, também no disco. Será executado quando houver buffers preenchidos com dados para leitura, espaço livre em disco, BCP's disponíveis e o disco não estiver sendo utilizado.

Spool de saída : é encarregado da saída de um programa do sistema. Carrega o conteúdo das páginas do disco (programa, dados, resultados) em buffers para impressão. Terminada a execução de um programa, com sucesso ou não, existindo buffers disponíveis e o disco não estando em utilização, este processo ser executado. - Impressão : sua tarefa é comandar a impressão do conteúdo dos buffers de impressão, sempre que houver buffer preenchido para impressão e o dispositivo de saída não estiver sendo utilizado.

Impressão : sua tarefa é comandar a impressão do conteúdo dos buffers de impressão, sempre que houver buffer preenchido para impressão e o dispositivo de saída não estiver sendo utilizado.

Entrada e Saída do usuário : permite trazer o conteúdo de páginas de dados do disco para a memória, ou remeter para o disco as páginas de impressão previamente preparadas na memória. Para ser realizado preciso que o disco não esteja sendo utilizado por nenhum outro processo e haja solicitação de leitura ou escrita por algum programa de usuário.

Paginação : processo encarregado de alocar uma página de memória disponível, para permitir a continuação de um programa, sempre que o sistema detectar falta de página durante a sua execução. Busca página do

disco quando precisar, se não houver espaço para alocar, ele escolhe uma para ser retirada.

Loader (carregamento de programas) : tem a função de alocar um número mínimo de páginas de memória e realizar a carga das páginas necessárias do disco, de forma a permitir o início da execução do programa.

Tratamento de Interrupção : fornece ao S.O. condições de retomar o controle dos recursos do sistema. Possui, por isso, a maior prioridade de execução atribuída pelo sistema operacional. É responsável pela análise das necessidades dos programas de usuários pelo posicionamento dos BCP's nas filas correspondentes e pelo controle do 'status' dos canais de E/S e disco, que habilitam ou bloqueiam a execução dos processos simples.

Escalonador : é responsável pela verificação de quais eventos estão em condições de serem executados e pela alocação dos recursos necessários para que eles ocorram. Quando não existirem mais processos simples em condição de serem executados, o escalonador tentará escolher um programa de usuário para execução.

Espera de Interrupção : ativado quando não há processos simples ou programas de usuários em condições de serem executados. O processo de espera representa a situação onde o sistema operacional aguarda somente o término da operação de algum canal de E/S que, ao terminar, irá gerar uma interrupção.

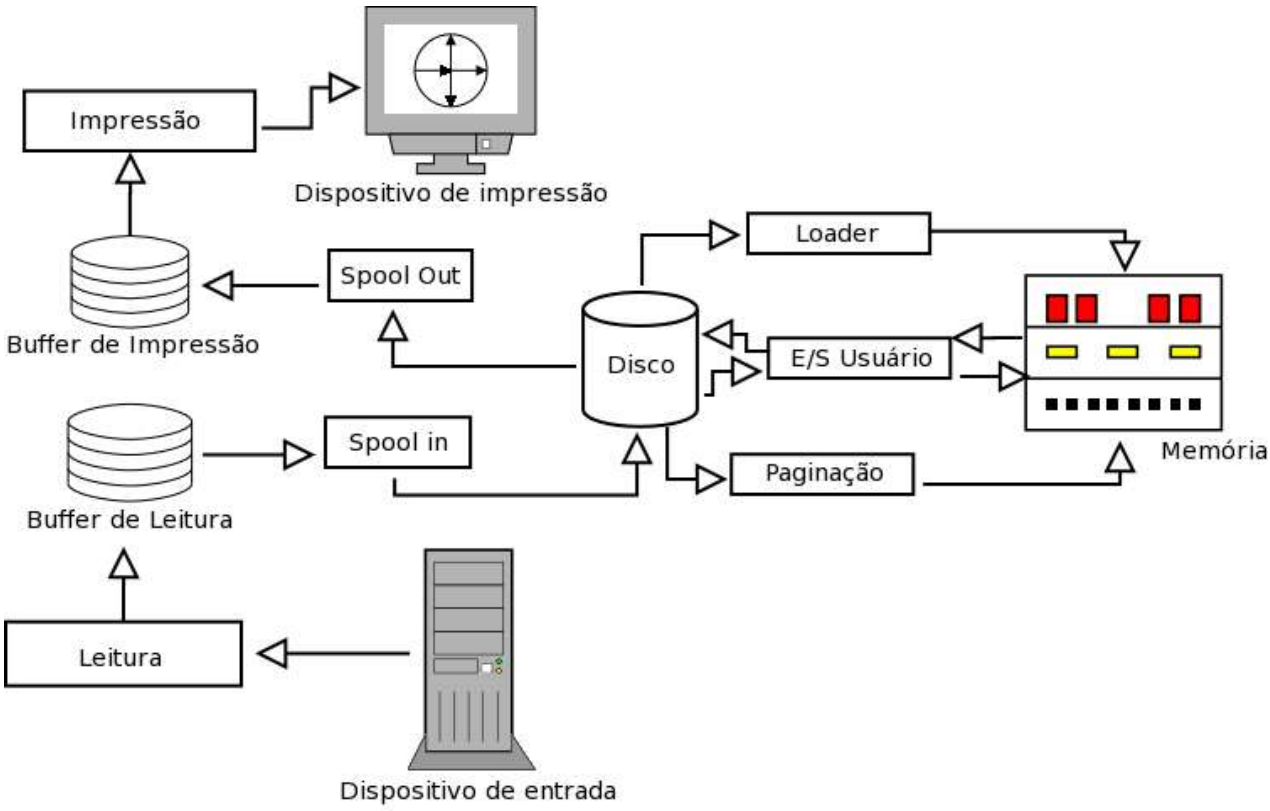

Figura 1 - Estruturação do sistema operacional e seus processos concorrentes.

Tarefas

As tarefas para este experimento são:

primeiro, execute o programa Winux com os jobs que estão disponíveis; procure entender o seu funcionamento. Para ajudar nessa tarefa, acesse os documentos existentes e responda às seguintes questões:

1)Quantos e quais são os comandos Assembly do Winux?

Podemos constatar que o numero de comandos no Winux pela documentação da versão 2.1 são de 9 instruções ('HLT', 'RD', 'PRN', 'LD', 'STR', 'SUB', 'ADD', 'JMP', 'JNG').

Há implementado no código outras 5 instruções ('SUI', 'ADI', 'MUL', 'DIV', 'LDI', 'STI').

Estrutura:

instrução	Página	palavra

Instruções:

HLT : Indica o fim de processamento de um programa.

0	-	-

RD : Lê a página corrente da área de dados do programa no disco e carrega o seu conteúdo na página XY da área de rascunho do programa.

1	xy	0

PRN Pág Desl: Escreve, na área de impressão do disco alocada pelo programa, o conteúdo da página XY da área de rascunho desse programa.

2	xy	0

LD Pág Desl: Carrega no acumulador (ACC) o conteúdo da palavra Z da página XY da área de rascunho do programa sendo executado. No caso do sistema simulado, a operação carrega no acumulador somente o conteúdo do primeiro campo da palavra de dados.

3	xy	z

STR Pág Desl: Armazena o conteúdo do acumulador na palavra Z da página XY da área de rascunho do programa. Atribui o valor do primeiro campo da palavra.

4	xy	z

SUB Pág Desl: Atribui ao acumulador o valor correspondente ao valor que este continha, subtraído do valor armazenado no primeiro campo da palavra Z

da página XY da área de rascunho do programa do usuário.

5	xy	z

ADD Pág Desl: Atribui ao acumulador o resultado da soma o valor do endereço.

6	xy	z

JMP Pág Desl: Atribui um novo valor para o Contador de Programa, do programa em execução, que passa a ser executado a partir dessa posição, um salto.
CP.PAG = XY
CP.DESL = Z

7	xy	z

JNG Pág Desl: O conteúdo do acumulador é verificado e, caso contenha umvalor negativo, o valor de XYZ é atribuído ao CP. Caso o acumulador não seja negativo, o programa continua sua execução a partir da próxima instrução em relação ao endereço em que se encontra.
Se ACC < 0
então (CP.PÁG = XY
CP.DESL = Z)

8	xy	z

Adcionais:
MUL XYZ : Atribui ao acumulador o valor correspondente ao valor que este continha, multiplicado pelo valor armazenado no primeiro campo da palavra Z da página XY da área de rascunho do programa do usuário.

DIV XYZ : Atribui ao acumulador o valor correspondente ao valor que este continha, dividido pelo valor armazenado no primeiro campo da palavra Z da página XY da área de rascunho do programa do usuário.

2)É possível realizar operações sobre números reais no Winux?

Com as nove instruções apresentadas na documentação não é possível fazer operações com números não inteiros, pois não há nenhuma instrução que manipule BIT a BIT os dados dos JOBS e também não há nenhuma instrução que simule a implementação de deslocamentos de BIT (como SHIFT , ROTATE) necessárias para operar números reais não inteiros.

3)Qual o tipo de escalonamento de processos implementado no Winux?

O tipo de escalonamento da versão 2.1 do Winux usado é escalonamento por fila FIFO (FistIn-FirstOut) onde o primeiro JOB que chega para na fila de prontos para execução e é o primeiro a ser escalonado para a mesma.

4) Qual o tipo de gerência de memória implementado no Winux? Caracterize essa gerência definindo estratégia de substituição e tamanho da memória.

O tipo de gerência de memória é feita através da simulação de esquema de paginação onde cada JOB tem sua respectiva tabela de pagina e existe um Mapa de Bits (bitmap) para mapear todos os frames (páginas) existentes da memória, dando as informações de que eles estão livres ou ocupados (com paginas de código ou dados ou com uma própria tabela de página).

A estratégia definida para substituição é de Página Menos Recentemente Utilizada (LRU) sendo que é usado um contador(que vai diminuindo conforme o tempo) para comparação de quando a página foi utilizada (se recentemente ou não).

A memória é dividida em 32 frames(de mesmo tamanho simulado que as páginas, numerados de 0 a 31). É possível expandir a memória para 64 frames.

Usa-se para gerenciar espaço livre de memória, uma tabela de bit, uma bitmap.

5) Como é caracterizado um programa (job) no Winux? Identifique e estabeleça limites para cada parte de um programa (job).

Job

Página Job
Página Inicial de Código
Página Código
Página Código
...
Página Inicial de Dados
Página Dados
Página Final Job

-Identificador do Job
-TIMER
-Número de páginas de código
-Número de páginas de dados
-Número de páginas de impressão

Segunda Tarefa: abra os fontes do Winux usando o ambiente Delphi e procure entender o funcionamento desse SO. Há um documento com os algoritmos básicos.

Feito.

Terceira Tarefa: especifique, projete, implemente e teste as seguintes mudanças no Winux :

1) Inclusão de duas novas instruções para envio e recebimento de mensagens.
2) Mudança da atual estratégia de escalonamento para SJF (Shortest Job First).
3) Mudança da atual estratégia de substituição de páginas para Segunda Chance.

Especificação

Estratégia de Escalonamento

Será modificado o tipo de escalonamento do Winux de FIFO (FisrtIn-FirstOut) para o tipo de escalonamento SJF(Shorstest Job First). No escalonamento FIFO o primeiro JOB que chega a fila de prontos é o primeiro a ser executado. O Winux utiliza-se de uma fila para garantir que essa seja a maneira de como os JOBS são escalonados.

O SJF é um tipo de escalonamento em que o SO atende o JOB que tiver menor tempo de execução (menor primeiro). Porém para isso, é preciso prever o tempo de execução de cada JOB (tempo de execução de cada uma de suas instruções), para que eles possam ser comparados e dentre eles, o menor ser escolhido. Embora possa parecer difícil prever o quanto cada execução do JOB demore, a implementação dos JOBs no Winux permite essa previsão sendo utilizado um campo do JOB para indicar qual será o seu tempo previsto. Esse valor pode ser mudado arbitrariamente no editor de JOBs e independe do tempo de execução de cada instrução, pois as mesmas não são implementadas por hardware e sim, simuladas em software que simula um Hardware “Virtual”.

Para implementação do SJF será modificada a estratégia FIFO colocando o processo de menor tempo previsto em primeiro na fila de prontos para que o mesmo seja o primeiro a ser escalonado e executado.

Exemplo de uma implementação SJF: (1 menor JOB)

1	2	3	4	5	6

Estratégia de Paginação

A estratégia de paginação do Winux versão 2.1 é de LRU (Least Recently Used), onde o SO, quando há falta de paginas na memória e um processo precisa ser executado, substitui as paginas desse novo processo a ser executado pelas paginas já alocadas de outros processos, porém utilizando a política de substituir as paginas que foram menos “usadas”(referenciadas ou modificadas) durante um período de tempo de execução.

Para isso, o Winux utiliza-se de um contador de tempo decrescente e compara com um contador de tempo existente de cada pagina residente em memória, atribuindo o valor do clock do SO para o da pagina quando essa for “usada” pela ultima vez. Assim ele desaloca a pagina com o menor contador, ou seja, a que foi menos usada recentemente.

Essa estratégia de troca de pagina, foi substituída na implementação pelo algorítimo de Segunda Chance Circular (Clock). Essa estratégia de substituição utiliza-se de um bit R, denominado de Bit de Referencia que é setado quando uma página é referenciada por um processo e também setado no Mapa de Bits referente aquela pagina referenciada. Quando ocorre uma falta de pagina por um processo que esta sendo executado, o SO percorre o Mapa de Bits verificando as paginas que estão setadas com o BitR (paginas que foram referenciadas) e “coloca” essa pagina numa fila de referencia(fila de segunda chance). Assim, tal pagina tem uma segunda chance de ser referenciada e assim retirada da dessa fila.O SO verifica de tempos em tempos percorrendo toda fila, se a pagina foi referenciada no Mapa de Bits e “retira” essa pagina da fila de referencia(segunda chance).Caso isso não ocorra, o SO desaloca a pagina da memória indicada pela primeira referencia da fila de segunda chance, ou seja, desaloca a pagina que mesmo recebendo uma segunda chance de ser referenciada, não o foi, permanecendo na fila de referencia e sendo escolhida como pagina a ser substituída.

Mecanismo de Sincronização de Processos Concorrentes

O tipo de mecanismo que será implementado será o de troca de mensagens, onde um processo enviará uma mensagem através de uma fila de mensagens que possuir uma chave única de identificação e outro processo recebera essa mensagem (se possuir a chave de identificação). Para o mecanismo de troca de mensagens serão implementadas duas novas instruções (uma de envio e outra de recebimento de mensagem).

Projeto

Implementação do SJF:

Para o algorítimo de SFJ, foram feitas as seguintes modificações no arquivo SOUnit.Pas de código fonte do Winux 2.1:

TIPO TBCP:

- foi adicionado um campo “TimerInicial” para guardar o tempo previsto inicial do JOB (TTIMER antes da execução);

Procedure SPOOL IN:
começo
...
PBCP // Ponteiro para BCP do JOB
PBCP : = Estado[1].Itens[0] // Ponteiro aponta para primeiro da fila de SPOOL IN

```
...
com PBCP faça
começo
    ...
      TimerInicial : = PBCP.Timer  // completa o campo TimerInicial com o Timer Inicial
                                   //  do JOB
    ...
fim
...
Fim // fim de SPOOL IN
```

Procedimento criado

Foi criado um procedimento que procura na fila de prontos quem é o menor e coloca em primeiro da fila. Ele é chamado antes do JOB ser escalonado para execução

```
Procedure Ordena_Fila_Prontos;  // método BOLHA modificado
Começo
  variaveis
    Aux   pontero_BCP;      // sera usado para guardar e atualizar as posicoes
     P1     pontero_BCP;   //aponta para o  BCP de numero "i" da fila de Pronto
     P2      pontero_BCP;  //aponta para o ultimo BCP da fila de Pronto
     T1,T2  tipo TTIMER(long int)
     laço i:=0 ate TamanhodaLista faça
    começo
      P1:=FilaProntos.Posição [i];
      T1:=P1^.TimerInicial;
      P2:= FilaProntos.Posição [ultima];
      T2:=P2^.TimerInicial;
      Se (T1 < T2) entao          // compara os tempos para pegar o
menor
          aux:=P1;   //auxiliar recebe o que P1 aponta
          P1:=P2;    //P1 aponta para onde P2 apontava
          P2:=aux;    //P2 aponta
     Fim do laço
   Fim
```

Procedimentos modificados

```
Proced Escala_Programa
    começo
      ...
      chama Ordena_Fila_Prontos;  //chamada do ordena vetor antes da mudança de estado
      Troca_FilaBCP(3,4);              //muda da fila de pronto para fila de execução
       ...
    fim
```

Implementação SegundaChance:

A implementação do algorítimo de Segunda Chance foi adaptada da implementação do algorítimo de SegundaChance implementada nos códigos fontes do WINUX versão 4.0, onde esse tipo de estratégia de paginação pode ser setada pelo usuário nas configurações do WINUX.

Foram implementados as seguintes modificações:

- Criação Fila SegundaChance: Criado Ponteiro para um novo tipo chamado "**segundachance**" com os seguintes campos:

```
r: booleano;       //campo que representa bitR (True=Setado False=Ñsetado)
endereco: inteiro;          //endereço na memoria
pagcod: inteiro;           //pagcod
pospagcod: inteiro;        //posição da pagina de codigo
prox: ponteiro_filasegundachance;    //aponta para o próximo da fila
```

- Criação de duas variáveis globais: **PFSInicio**, **PFSFim** ponteiro_filasegundachance;
//vão apontar para o primeiro e ultimo nó da fila respectivamente.

Procedimentos Criados

Proced AdicinionarNoFS(inteiros posição, ppagcod, ppospagcod)
//adiciona um novo No que contem os endereços da pagina referenciada
Começo
Cria novo no(NEW);
Se FilaVazia -> faz **PFSInicio** e **PFSFim** apontar para NEW
Senao UltimoNo aponta para NEW e **PFSFim** aponta par NEW;
Atualiza em no NEW:
BitR = false;
endereço = posição;
prox:=nil;
pagcod:=ppagcod;
pospagcod:=ppospagcod**;**
Fim

Proced RemoverNoFS(inteiros no)
//remove um No que contem o endereço de certa pagina
variaveis
Patras ponteiro_segundachance //ponteiro auxiliar guarda end No anterior a Ppercorre
Ppercorre ponteiro_segundachance; // percorre a fila atrás do no correto

Começo
Ppercorre aponta para o primeiro da fila
Se Ppercorre for o No que contem o endereço entao
Se Ppercorre.prox não apontar para ninguém então PFSFim aponta p/ ninguem
Desaloca (Ppercorre);
Senão Laço ate achar no certo

Aponta Patras para Ppercorre;
Caminha Ppercorre apontando p/ próximo
Se encontaNo então
PSFim aponta para atras
Liga No anterior a Ppercorre com o posterior
Desaloca Ppercorre; break laço

Fim

procedure ModificarBitR(no:inteiro; bitR:booleano);
variaveis
percorre:ponteiro_filasegundachance;
Comeco
percorre aponta para PFSInicio;
Laço até achar No certo até o final da lista (fila)
percorre aponta para o próximo;
Se percorre^.endereço for o certo
Seta o bit R do No apontado por percorre com valor de bitR (true) ;
Fim

Funções Criadas

```
// desaloca da memória uma pagina que já foi dada SegundaChance e não foi referenciada          // até o presente momento
function TrocaFilaSegundaChance inteiro;
variaveis
  volta:boolean;
 Começo
  result:=-1 //valor de retorno;
  volta:=true;
   laço ate volta ficar false
    comeco laço
      Se PFSInicio ñ apontar para ninguém entao
         result:=-1;
         terminafuncao;
      Se bit R de PFSInicio = false entao
        Se PFSInicio aponta p/ alguem entao
          result recebe endereço do primeiro no da Fila
       Senao  result:=-1 e fim de funcao

       // Marca na tabela de páginas (da página de código a ser retirada) a pagina de código como ausente

      //marca a pagina como ausente na tabela de pagina(campo c1)
        Memoria.Items[EndpagcodPriFilaSC] [PospagcodPriFilaSC].c1:=0;

        //marca a pagina (na tabela de pagina)como ausente da memoria
        Memoria.Items[EndpagcodPriFilaSC] [PospagcodPriFilaSC].c2:=-1;

       // marca esta página de código como ausente no bitmap de memória
       BitMem[EndPrimeiroFilaSC]:=0;
```

chama RemoverNoFS passando o endereço do Primeiro da FilaSC;
volta:=false; // condição de parada do laco

Senao
chama AdicionarNoFS
(EndPriFilaSC,EndPagCoddoPriFilaSC,PosPagCoddoPriFilaSC);
chama RemoverNoFS(EndPriFilaSC);

fim laço
Fim

Procedimentos Modificados

// essa função é chamada pela função “encontra_pagina_livre_memória” quando não há mais
// paginas livres na memória e é preciso substituir as paginas da memória.
function Troca (inteiro)
Começo
return(valor de retorno) recebe o resultado da função TrocaFilaSegundaChance;
Fim

// essa função faz a troca de paginas
procedure Paginação
Começo
...
aponta para primeiro BCP da fila Espera por Falta de Pagina;
...
//coloca essa pagina recém-paginada na fila de segunda chance para que a mesma não corra o
// risco de sofrer paginação logo que entra na memória.
chama AdicionaNoFS(EndPagASerPaginada, EndTabPag, EndFaltaPagina)
...
salva pagina em disco();
...
Fim

procedure Loader
Começo
...
aponta para primeiro BCP da fila SPOLL IN(paginas a serem carregadas do disco p/memória)
...
//coloca essa pagina recém-carregada do disco na fila de segunda chance para que a mesma não
// corra o risco de sofrer paginação logo que entra na memória.
chama AdicionaNoFS(EndPagASerPaginada, EndTabPag, EndFaltaPagina)
...
salva pagina em disco();
...

Fim

//busca instrução dentro da tabela de pagina de codigos
Procedure BuscaInstrução
Começo
Aponta para primeiro BCP da Fila de Execução(pagina a ser executada)
...
Verifica se a pagina esta presenta na memória:
Se Ausente: interrupção falta de pagina
Se Presente: - copia instrução, página e deslocamento para RI
- chama ModificarBitR(PagAtual,ValorParaSetar);
// cada fez que uma instrução for buscada a pagina que a contem vai ser referenciada e portanto é
// necessário que seu bit seja setado
...

Plano de testes

SJF

Condição a ser testada: tempo de execução dos jobs. O tempo é definido na criação do Job pelo editor de criação do Winux 2.1 .

Nome do arquivo do Job	Tempo previsto
Ordena Vetor1.job	**900**
Fatorial10.job	**250**
Encontra Menor1.job	**3000**

Maneira de testar: escolhe-se Jobs aleatórios e com tempos diferentes. Compara-se o primeiro Job com todos os outros (o tempo deles), se for o menor, ele será o primeiro, caso contrário, compara-se o segundo com todos os outros, se for o menor, será o primeiro, e assim adiante, sempre mantendo o menor na frente do maior (antes). Caso tenha tempos iguais, o primeiro a ser pego para testes, ficará na frente (antes).

Resultado esperado: Colocação do menor Job na posição inicial:

Colocação	Nome do Job
1	Fatorial10.job
2	Ordena Vetor1.job
3	Encontra Menor1.job

Como observar o resultado esperado: Os Jobs com menor tempo, serão escalados e acabaram primeiro.

Segunda Chance

Condição a ser testada: Qual página será utilizada na paginação. As páginas a passarem pelo teste, são todas que estiverem na memória.

Maneira de testar: Essa estratégia de substituição utiliza-se de um bit R, denominado de Bit de Referencia que é setado quando uma página é referenciada por um processo e também setado no Mapa de Bits referente aquela pagina referenciada. Quando ocorre uma falta de pagina por um processo que esta sendo executado, o SO percorre o Mapa de Bits verificando as paginas que estão setadas com o BitR (paginas que foram referenciadas) e "coloca" essa pagina numa fila de referencia(fila de segunda chance). Assim, tal pagina tem uma segunda chance de ser referenciada e assim retirada da dessa fila. O SO verifica de tempos em tempos percorrendo toda fila, se a pagina foi referenciada no Mapa de Bits e "retira" essa pagina da fila de referencia(segunda chance).Caso isso não ocorra, o SO desaloca a pagina da memória indicada pela primeira referencia da fila de segunda chance, ou seja, desaloca a pagina que mesmo recebendo uma segunda chance de ser referenciada, não o foi, permanecendo na fila de referencia e sendo escolhida como pagina a ser substituída.

Carrega-se vários Jobs na memória, principalmente os que se utilizarem a instrução RD, pois referencia com dados páginas da memória.

Carrega-se o Winux com o mesmo Job várias vezes (por exemplo Ordena Vetor 1) até encher a memória, assim quando começar a paginação por falta de memória as paginas menos usadas serão retiradas.

Resultado esperado: Que escolha a página certa para ser substituída.

Implementação

Arquivo SO_Unit.pas do Winux 2.1

SJF:

Linha 616:
```
TimerInicial:=PBCP^.Timer;
```

Linha 1550:
```
Ordena_Fila_de_Prontos;
```

Linha 1497:
```
procedure Ordena_Fila_de_Prontos;
//ordena a fila de prontos pelo metodo Bolha modificado

var i:integer;
var aux:pointer_BCP;
var P1:pointer_BCP;
var P2:pointer_BCP;
var T1:TTIMER;
var T2:TTIMER;
```

```
begin
   for i:=0 to Estado[3].Count-1 do
   begin
     P1:=Estado[3].Items[i];
     T1:=P1^.TimerInicial;
     P2:=Estado[3].Items[Estado[3].Count-1];
     T2:=P2^.TimerInicial;
     if (T1 < T2) then begin
       aux:=P1;
       P1:=P2;
       P2:=aux;
     end
   end;//fim for
end;
```

Segunda Chane

Linha 97: PFSInicio, PFSFim: pointer_filasegundachance;

Linha 255:

```
procedure AdicionarNoFS(posicao,ppagcod,ppospagcod:integer);
procedure RemoverNoFS(no:integer);
procedure ModificarBitR(no:integer; bitR:boolean);
function  TrocaFilaSegundaChance:integer;
```

Linha 751:

```
AdicionarNoFS(poslivre,PBCP^.Tab_Pag,0);
```

Linha 810:

```
AdicionarNoFS(poslivre,PBCP^.Tab_Pag,PBCP^.FP);
```

Linha 1175:

```
function Troca:integer;
begin
    result:=TrocaFilaSegundaChance;
  //  else result:=-1;
end;

function TrocaFilaSegundaChance:integer;
var
  volta:boolean;
begin
    result:=-1;
    volta:=true;
    while volta do begin
         // ** Adicionado por DLMN **
         if PFSInicio = nil then begin
            result:=-1;
            exit;
```

```
          end;
          // **********************
          if PFSInicio^.r=false then begin
            if PFSInicio<>nil then result:=PFSInicio^.endereco
            else begin
                result:=-1;
                exit;
            end;

            { Marca na tabela de páginas (da página de código a ser retirada)
             a pagina de código como ausente }
            Memoria.Items[PFSInicio^.pagcod][PFSInicio^.pospagcod].c1:=0;
            Memoria.Items[PFSInicio^.pagcod][PFSInicio^.pospagcod].c2:=-1;

            // Marca esta página de código como ausente no bitmap de memória
            FMain.BitMem[PFSInicio^.endereco]:=0;

            RemoverNoFS(PFSInicio^.endereco);
            volta:=false;
          end
          else begin
              AdicionarNoFS
(PFSInicio^.endereco,PFSInicio^.pagcod,PFSInicio^.pospagcod);
              RemoverNoFS(PFSInicio^.endereco);
          end;

          // mudanças 12/6
         // OrdenaVetorMemoria(modotrocapagina);
          // **********************************************
    end;
end;

// Adiciona pagina na fila com segunda chance
procedure AdicionarNoFS(posicao,ppagcod,ppospagcod:integer);
var FS: pointer_filasegundachance;
begin
    new(FS);
    if PFSInicio=nil then PFSInicio:=FS;
    if PFSFim=nil then PFSFim:=FS
    else begin
        PFSFim^.prox:=FS;
        PFSFim:=FS;
    end;
    FS^.r:=false;
    FS^.endereco:=posicao;
    FS^.prox:=nil;
    FS^.pagcod:=ppagcod;
    FS^.pospagcod:=ppospagcod;
end;

// Remove pagina da fila com segunda chance
```

```
procedure RemoverNoFS(no:integer);
var
  percorre,atras:pointer_filasegundachance;
  continua:boolean;
begin
    continua:=true;
    percorre:=PFSInicio;
    //atras:=nil;
    if percorre^.endereco=no then begin
      if percorre^.prox=nil then PFSFim:=nil;
      PFSInicio:=percorre^.prox;
      dispose(percorre);
    end
    else
       while (percorre^.endereco<>no) and continua do begin
            atras:=percorre;
            percorre:=percorre^.prox;
            if percorre^.endereco=no then begin
              if percorre^.prox=nil then PFSFim:=atras;
              atras^.prox:=percorre^.prox;
              dispose(percorre);
              continua:=false;
            end;
       end;
end;

procedure ModificarBitR(no:integer; bitR:boolean);
var
  percorre:pointer_filasegundachance;
begin
    percorre:=PFSInicio;
    while (percorre^.endereco<>no)and(percorre^.prox<>nil) do
percorre:=percorre^.prox;
    if percorre^.endereco=no then percorre^.r:=bitR;
end;
```

Linha 1661:

```
ModificarBitR(PagAtual,true);
```

Conclusão

O Winux é um software que abrange bastante o conhecimento e seus conceitos em relação a Sistemas Operacionais. Com o acesso a implementações como essa, podemos ter uma noção de como realmente é feito um sistema operacional.

Tivemos bastante dificuldade para entender e modificar o código, um dos motivos e talvez o de maior dificuldade, é o contato com a linguagem Delphi, visto que nunca tínhamos programado nela antes, muito menos encontrar material de apoio derivados de conceitos como escalonamento de

processos.

Podemos concluir que trabalhar com um SO simulado é não é tão fácil, porém interessante, pois, é como andar com um carro em cima de uma carreta. Aprendemos bastante com a implementações de algoritmos como o SJF.

Visto esse tema de trabalho, podemos agora a devastar os códigos de sistemas como Linux ou Unix e ver realmente o funcionamento na prática do dia-a-dia.

Agradecemos aos autores do Software Winux, por proporcionar esse contato com um Sistema Operacional Simulado.

Bibliografia

Sistema Operacional Simplificado – Freitas, Ricardo Luís
ftp://ftp.ii.puc-campinas.br/pub/professores/ceatec/ricardo/SO/SOLab.PDF
Data do último acesso: 21/06/2004 às 15:00h

Algoritmos SO – Freitas, Ricardo Luís
ftp://ftp.ii.puc-campinas.br/pub/professores/ceatec/ricardo/SO/Algoritmos%20SO.doc
Data do último acesso: 21/06/2004 às 15:00h