EACH

Sistemas Operacionais

Gerenciamento de Memória

Leila Chan Choimei

Murilo Galvão Honorio

4º Semestre - Turma 02 – Matutino

São Paulo, 27 de Novembro de 2009

Sumário

Introdução

Devido aos Sistemas Operacionais apresentarem características diversas, determinados procedimentos tornam-se por vezes semelhantes ou próprios de cada Sistema. O gerenciamento de memória é um procedimento fundamental na objetividade de todos os Sistemas Operacionais.

É responsabilidade do Sistema Operacional executar inúmeras tarefas, independente da quantidade de tarefas e do tamanho das mesmas.

Neste contexto, o Sistema Operacional deve gerenciar tarefas que exigem uma quantidade de memória superior ao que é disponibilizado pela máquina. Cada Sistema possui suas características próprias para complementar estas tarefas e principalmente gerenciar a memória disponível para estas execuções.

Existem diversos esquemas de gerência de memória que implicam em diferentes formas de abordagem, a eficácia de cada algoritmo implementada depende de situações específicas. É necessário garantir memória às tarefas além de oferecer execução em alto grau de segurança.

Neste trabalho, iremos apresentar um simulador de gerenciamento de memória o qual visa proporcionar uma análise da interação dos processos alocados na memória e como estes interagem com o escalonamento pelo algoritmo de round robin no sistema operacional. Mostraremos características particulares de cada método adotado, e qual seu comportamento frente a política de escalonomento e gerenciamento de memória.

Os algoritmos desenvolvidos nesse trabalho seguem conceitos apresentados no artigo de MacDOUGALL (1970) e tem como referência de Silbertschatz (2002) como base de informação para formulação dos algoritmos que serão apresentados mais a frente neste trabalho.

Base para o trabalho

Abaixo mostraremos uma análise das etapas de um processamento.

Descrevemos diversas características que devem ser especificadas na construção de um modelo de simulador do processos :

1 - O processo chega ao sistema.

2 - A memória principal é requisitada pelo processo, caso o espaço esteja livre, o processo é alocado, caso contrário, entra numa fila.

3 - A CPU é requistada e se estiver livre, este executa a solicitação de I/O até ser preemptada ou até a execução se concluir. Caso esteja ocupada, o processo entra numa fila.

4 - Quando o processo emite uma solicitação de I/O, ele libera o processador central. (Se houver outro processo à espera na fila de CPU, é agora atribuída ao processador central). Se o disco está livre, é atribuído ao programa de processar a solicitação I/O; se o disco está ocupado, o pedido é inserido em uma fila.

5 – Ao fim do processamento, o disco é liberado e o processador central é solicitada mais uma vez. (Se houver algum processo na fila do disco, este é atribuído ao disco).

6 - Quando o processo completa sua execução, este libera a CPU e a Memória Principal. (A fila de processos é analisada para determinar se existe algum processo na fila em espera que possa acessar a CPU).

7 – O programa sai do sistema.

As etapas apresentadas fornecem a base para a implementação do simulador, outras técnicas serão detalhadas mais a frente no trabalho.

Estrutura do Simulador

Eventos e Lista de Eventos

Em um simulador, o processo é representado por uma fila de processos.

A simulação de um processo está preocupado com o começo e fim dessas tarefas, esses pontos compõem uma seqüência de eventos, a ordenação de cada uma é determinada pelo tempo e quando cada evento está a ocorrer.

Podemos estabelecer uma sequência de eventos usando uma lista de eventos. Nosso simulador assume que cada event esteja dentro de uma Linked List (Lista Ligada) e cada elemento eh determinado por :

PID(Process Indentifier) : É uma entrada única atribuído pelo sistema operacional que é utilizado para se fazer referência a um processo que está a executar.

Instante de Chegada : Instante em que o processo chega à CPU

Burst (Surtos de CPU) : Tempo de uso da CPU

Seção Texto : Tamanho das Seções de Texto

Seção Dados : Tamanho das Seções de Dados

Seção Pilha : Tamanho das Seções de Pilha

Página : Tamanho da página

Os passos básicos implementados nos métodos do programa são os seguintes :

1 – O processo solicita memória e se está tiver espaço livre, o processo é alocado na fila de memória.

1 - O programa remove a entrada da cabeça da fila de memória para a fila de eventos.

2 - O programa avança o relógio para o evento especificado no primeiro passo.

3 - O programa transfere o controle para o evento de rotina designado pelo indentificador do evento.

4 – A rotina de evento executa o processamento, determina o seu próximo evento e hora do evento, e retorna o controle para o programa.

Filas

Outras maneiras de se usar uma Linked List é manipulação de filas. É uma lista linear em que a inserção é feita numa extremidade e a eliminação na outra. (**FIFO:** first in, first out).

Para alocação de processos na fila de memória adotamos o política First-Come First-Served (FCFS). A fila de processos é criada a partir dos processos alocados na memória, e aguardam os recursos do sistema operacional, sendo o primeiro elemento inserido , o primeiro a ser executado.

Inplementação & Classes

Foram criadas várias classes que constituem o simulador, utizando-se a linguagem java. Segue em detalhes cada uma das classes :

Cenario : Classe que contém um método que lê os eventos contidos em um arquivo texto tabulado com as características dos processos e retorna uma lista de processos. Esta classe também se encarrega de verificar se as entradas são potências de 2.

Evento : Classe que contém os contrutores que alocam os recursos necessários para o funcionamento do objeto Evento, define inicialmente as variáveis de estado(atributos), e também contém método para gerar a descrição do estado do processo.

Log : Classe que contém métodos que armazenam uma lista de eventos, o nome do algoritmo utilizado, o tempo médio de espera, e retornam um arquivo com esses dados gravados.

Processo : Classe que contém os contrutores que alocam os recursos necessários para o funcionamento do objeto Processo, além de definir inicialmente as variáveis de estado(atributos).

Paginação : Casse que contém métodos para execução do algoritmo de paginação. Ao alocar blocos de páginas , a memória desperdiça as fragmentações que possam ser geradas no processo de alocação.

Segmentação : Casse que contém métodos para execução do algoritmo de segmentação.

Segmento : Classe que contém construtor para definir a base por onde deverá ser alocado na memória, o tamanho do segmento, o tipo de dado (seção), atributos os quais serão alocados com política de segmentação.

Round Robin : Classe que contém o algoritmo de escalonamento Round Robin a qual calcula os tempos de resposta e tempo de espera do processo em execução.

Simulação : Classe que contém função main para executar o programa.

Ao adotarmos alocação de memória dinâmica aos algorimtos de paginação e segmentação, utilizamos a estratégia first-fit, a qual aloca o primeiro bloco de memória da fila de eventos (ordenada por PID).

Para os processos que requisitam mais memória do que memória total disponível, o programa simplesmente trata de rejeitá-lo.

O simulador não possui mecanismos de verificar situações de starvation, uma vez que definimos processos finitos para solicitação de memória. Caso houvesse a possibilidade de inserir processos novos indefinidamente, teríamos grandes chances de observar situações de starvation, caso o processo fique na fila esperando pelo recurso, enquanto que novos chegam e alocam memória.

Paginação & Segmentação

Paginação

A paginação da memória é um processo que consiste na subdivisão da mesma em pequenas partições, permitindo que o espaço de endereçamento físico de um processo seja não contíguo, ou seja, cada programa pode ser espalhado por áreas não contíguas da memória, eliminando assim a fragmentação externa.

A memória física é quebrada em blocos de tamanho fixo chamados blocos (frames). A memória lógica também é quebrada em blocos de mesmo tamanho chamado páginas.

Quando um processo vai ser executado, suas páginas são carregadas em qualquer quadro de memória disponível . Ao adotarmos alocação de memória dinâmica por paginação, utilizamos a estratégia first-fit, a qual aloca o primeiro bloco de memória da fila de eventos (ordenada por PID).

Quando utilizamos um esquema de paginação, não existe fragmentação externa : qualquer quadro livre pode ser alocado a um processo que precisa dele, no entanto, devemos considerar as fragmentações internas.

Quando um processo chega no sistema para ser executado, seu tamanho expresso em paginas é executado examinado. Cada página precida de um quadro. Assim, se o processo precisar de 'n' páginas, deve haver pelo menos 'n' quadros disponíveis em memória. Se houver 'n' quadros disponíveis em memória, eles são alocados a esse processo que está chegando, caso contrário entra pra fila de memória, ou se as páginas estourarem o tamanho da memória , o programa simplesmente retorna valor 'false'.

Em resumo, a implementação da tabela de páginas traz as seguintes vantagens:

– A tabela cresce de pedaço em pedaço, à medida que a alocação de páginas física acontece;

– Apenas o necessário é usado, com uma pequena **fragmentação interna** na própria tabela, quando não são necessárias todas as entradas do pedaço alocado;

– O processo pode "povoar" diferentes regiões do espaço lógico, sem dificuldades.

Segmentação

A segmentação é um esquema de gerência de memória que dá suporte a seguinte visão do usuário :

– Espaço de endereçamento lógico é uma coleção de segmentos.

– Cada segmento tem nome e tamanho.

– Os endereços especificam o nome (número do segmento) e posição (deslocamento do segmento).

– Elementos dentro do segmento são identidicados pelo deslocamento a partir do início.

Tabela de Segmento :

- Base : Contém o endereço físico inicial do segmento na memória.
- Limite (tamanho) : especifica tamanho do segmento.

O processo é dividido em segmentos as quais solicitam alocação na memória, caso exista base livre com limite (tamanho) suficiente para o segmento, será alocado espaço lógico na memória. Caso contrário, armazena-se em fila até a base ser desalocada e existir tamanho suficiente para alocação. Em caso do número de páginas ser maior que a memória, o programa retorna valor 'false', impedindo que o processo estoure o tamanho da memória , causando possíveis erros.

Como a segmentação é por natureza um algotimo de relocação dinâmica, podemos compactar a memória sempre que desalocarmos, ou seja, a compactação pode ser usada para criar um bloco de memória livre maior.

Se o escalonador de CPU precisa esperar por um processo, devido a um problema de alocação de memória, poderá percorrer a fila de CPU procurando um processo menor que se adeque à base e limite em bloco de memória livre.

MANUAL DO USUÁRIO

Geramos um **Java Archive** – JAR (memoria.jar) : um arquivo compactado usado para distribuir o conjunto de Classes Java. O JAR armazena as classes compiladas e os métodos associado que constituem nosso programa.

Segem os seguintes passos :

1 - Ao abrirmos o arquivo JAR, nos deparamos com um ícone com a seguinte opção :

– Entre com o nome do arquivo : O usuário deve inserir o nome do arquivo '.txt' a qual armazena as entradas dos processos que pretende analisar.

– O arquivo cenário deve respeitar a formatação como indicada no passo a passo a seguir :

```
64          //tamanho da memoria em bytes
16          //tamanho da pagina em bytes
3           //quantum de tempo em ms
0           //estratégia (0 = paginação, 1 = segmentacao pura)

//respectivamente: PID, instante de chegada, tempo de burst, tamanho das secoes texto, dados e pilha
//separar cada campo com TAB.
//A quantidade de memória a ser alocada na paginação será a soma das três seções (todos numéricos)

Processo/tempoChegada/Burst/texto/dados/pilha
001  0    10   10   10   10
002  0    15   20   10   20
003  0    10   5    4    1
004  50   10   5    5    5
005  23   3    1    1    1
```

Caso o arquivo de entrada não seguir corretamente à formatação descrita acima, o programa não irá funcionar, e neste caso o programa emite a seguinte mensagem ao usuário : “Erro no arquivo. Verifique a digitação e tente novamente.”

– Entre com o nome do arquivo de saída : O usuário deve inserir o nome do arquivo '.txt' a qual aramazenará as saídas do programa.

– Caso o usuário tenha errado o nome do arquivo de entrada, o programa emitirá uma mensagem de erro : “Erro no arquivo. Verifique a digitação e tente novamente”. A mensagem de inserção de arquivo de entrada é novamente aberta, ou o usuário pode simplesmente cancelar a ação e encerrar as atividades.

O arquivo de saída contém o log com todas as descrições do processamento, que vai da etapa de alocação de memória da fila de processos, do processamento pelo algoritmo de round robin, até a saída de cada processo concluído. Todo o passo a passo é descrito minuciosamente pelo arquivo de saída, mostrando a alocação na memória , ou na fila de memória, todos os tempos de entrada e saída pelo uso de CPU, até a conclusão do processamento.

Cenário de Execução

A seguir segue dois exemplos de execução e os respectivos cenários :

Cenário 1 : Mostra a execução de vários processos utilizando algoritmo de paginação.

```
64          //tamanho da memoria em bytes
16          //tamanho da pagina em bytes
3           //quantum de tempo em ms
0           //estrategia (0 = paginação, 1 = segmentacao pura)

//respectivamente: PID, instante de chegada, tempo de burst, tamanho das secoes texto, dados e pilha
//A quantidade de memória a ser alocada na paginação será a soma das três seções (todos numéricos)

Processo/tempoChegada/Burst/texto/dados/pilha
001  0    10   10   10   10
002  0    15   20   10   20
003  0    10   5    4    1
004  50   10   5    5    5
005  23   3    1    1    1
```

Na saída verificamos os seguintes dados, o nome do arquivo de entrada '.txt', o algoritmo de alocação de memória, tamanho da memória, tamanho da página, número de páginas, e quantum de tempo que será usado no algoritmo round robin.

Saída

Arquivo de entrada: cenario.txt
Gerencia de memoria: Paginacao
Tamanho da memoria: 64 bytes
Tamanho da pagina: 16 bytes
Numero de paginas: 4
Quantum de tempo: 3 ms

– Abaixo o arquivo de saída mostra que a fila de processos(já ordenado por PID) vai sendo alocada na memória de acordo com os espaços livres contidos nela ou se entra em fila de espera pela memória.

– É mostrado o tempo de chegada de cada processo em ms.

– E quando um processo ganha memória, o programa mostra tamanho do espaço de endereçamento em bytes e as páginas que foram alocadas pelo processo.

– A fragmentação interna será indicada em bytes.

0 ms: processo 1 chegou ao sistema.
0 ms: processo 1 obteve memoria.
*Espaco de Enderecamento: 30 bytes. Paginas: 0, 1. Fragmentacao interna: 2 bytes.
0 ms: processo 2 chegou ao sistema.
0 ms: processo 2 entrou na fila de memoria.
0 ms: processo 3 chegou ao sistema.
0 ms: processo 3 obteve memoria.
*Espaco de Enderecamento: 10 bytes. Paginas: 2. Fragmentacao interna: 6 bytes.

– Aqui o arquivo mostra a primeira alocação em memória :

– O formato em paginação é [Página][Processo]

– Caso não haja nenhum processo alocada em página, será assinalado com 'X'.

Memoria: [0][1], [1][1], [2][3], [3][X]

– Abaixo o arquivo mostra o processamento de todos os processos alocados em memória na CPU pelo algoritmo round robin.

– Mostra o instante de chegada na CPU (representado em ms), e os estados dos processos que ganham a CPU : qual o processo ganha a CPU, ou é preemptado dela, ou é concluído.

```
0 ms: processo 1 executando na CPU.
3 ms: processo 1 preemptado.
3 ms: processo 3 executando na CPU.
6 ms: processo 3 preemptado.
6 ms: processo 1 executando na CPU.
9 ms: processo 1 preemptado.
9 ms: processo 3 executando na CPU.
12 ms: processo 3 preemptado.
12 ms: processo 1 executando na CPU.
15 ms: processo 1 preemptado.
15 ms: processo 3 executando na CPU.
18 ms: processo 3 preemptado.
18 ms: processo 1 executando na CPU.
19 ms: processo 1 concluido.
```

– Nesse trecho observamos o tempo de resposta do processo concluído, turnaround que representa o tempo total que um processo executa, do instante de chegada ate sua conclusão, incluindo as esperas nas filas, e o tempo de espera que é a soma dos tempos em que um processo está na fila de prontos mas não está sendo executado.

*Tempo de resposta: 0 ms.

*Turnaround: 19 ms.

*Tempo de Espera: 9 ms.

– Neste instante o processo 1 é concluído e sai da memória.
– Novamente mostra-se a atualização das bases livres em memória.
– Neste caso, esvazia-se os quadros onde continham o processo 1.
– O processo 2 tenta alocar em memória, porém não há espaço suficiente.
– Continua o processamento dos processos em memória.

Memoria: [0][X], [1][X], [2][3], [3][X].

– Abaixo mostra-se a continuação dos processos em CPU pelos processos alocados em memória.

19 ms: processo 3 executando na CPU.
20 ms: processo 3 concluido.

– Processo 3 concluído.
– Nesse trecho observamos o tempo de resposta do processo concluído, turnaround que representa o tempo total que um processo executa, do instante de chegada ate sua conclusão, incluindo as esperas nas filas, e o tempo de espera que é a soma dos tempos em que um processo está na fila de prontos mas não está sendo executado.

*Tempo de resposta: 3 ms.
*Turnaround: 20 ms.
*Tempo de Espera: 10 ms.

– Nova atualização é feita pela saída do processo 3 da memória.
– Processo 2 ganha memória no instante indicado em ms, é indicado as páginas a serem alocadas pelo mesmo, e a fragmentação interna medida em bytes, e as páginas que este necessitou alocar.

20 ms: processo 2 obteve memoria.
*Espaco de Enderecamento: 50 bytes. Paginas: 0, 1, 2, 3. Fragmentacao interna: 14 bytes.

Memoria: [0][2], [1][2], [2][2], [3][2].

– Abaixo mostra-se a continuação dos processos em CPU pelos processos alocados em memória.

– O processo 2 é preemptado, e o processo 5 requisita memória e entra para fila pois não há espaço suficiente para alocação da mesma.

20 ms: processo 2 executando na CPU.
23 ms: processo 2 preemptado.
23 ms: processo 5 chegou ao sistema.
23 ms: processo 5 entrou na fila de memoria.
Memoria: [0][2], [1][2], [2][2], [3][2].
23 ms: processo 2 executando na CPU.
26 ms: processo 2 preemptado.
26 ms: processo 2 executando na CPU.
29 ms: processo 2 preemptado.
29 ms: processo 2 executando na CPU.
32 ms: processo 2 preemptado.
32 ms: processo 2 executando na CPU.
35 ms: processo 2 concluido.

– O processo 2 é executado até o final.

– O programa informa os tempos de resposta, o turnaround, o tempo de espera do processo concluído em ms.

*Tempo de resposta: 20 ms.
*Turnaround: 35 ms.
*Tempo de Espera: 0 ms.

– Ao concluir o processo 2 , este desaloca espaço em memória, deixando livre para alocação do processo 5.

– É indicado o instante que o processo chega a CPU, o espaço de endereçamento, as páginas alocadas pelo processo, e a fragmentação interna indicada em ms, e as páginas que este necessitou alocar.

35 ms: processo 5 obteve memoria.

*Espaco de Enderecamento: 3 bytes. Paginas: 0. Fragmentacao interna: 13 bytes.
Memoria: [0][5], [1][X], [2][X], [3][X].

– Segue o processamento em CPU pelo algoritmo round robin.

35 ms: processo 5 executando na CPU.

38 ms: processo 5 concluido.

– O processo 5 é concluído e desaloca memória.

– O programa informa os tempos de resposta, o turnaround, o tempo de espera do processo concluído em ms.

*Tempo de resposta: 12 ms.

*Turnaround: 15 ms.

*Tempo de Espera: 0 ms.

– O processo 4 é alocado em memória no instante de chegada indicado.

– É indicado o instante que o processo chega a CPU, o espaço de endereçamento, as páginas alocadas pelo processo, e a fragmentação interna indicada em ms.

50 ms: processo 4 chegou ao sistema.

50 ms: processo 4 obteve memoria.

*Espaco de Enderecamento: 15 bytes. Paginas: 0. Fragmentacao interna: 1 bytes.
Memoria: [0][4], [1][X], [2][X], [3][X].

– Segue o processamento em CPU pelo algoritmo round robin.

50 ms: processo 4 executando na CPU.

53 ms: processo 4 preemptado.

53 ms: processo 4 executando na CPU.

56 ms: processo 4 preemptado.

56 ms: processo 4 executando na CPU.

59 ms: processo 4 preemptado.
59 ms: processo 4 executando na CPU.
60 ms: processo 4 concluido.

– O processo 4 é concluído e desaloca memória.
– O programa informa os tempos de resposta, o turnaround, o tempo de espera do processo concluído em ms.

*Tempo de resposta: 0 ms.
*Turnaround: 10 ms.
*Tempo de Espera: 0 ms.

Processos que requerem mais memória do que memória total disponível são descartados pelo programa.

Cenário 2 : Mostra a execução de vários processos utilizando algoritmo de segmentação.

64 //tamanho da memoria em bytes
16 //tamanho da pagina em bytes
3 //quantum de tempo em ms
1 //estrategia (0 = paginação, 1 = segmentacao pura)

//respectivamente: PID, instante de chegada, tempo de burst, tamanho das secoes texto, dados e pilha
//A quantidade de memória a ser alocada na paginação será a soma das três seções (todos numéricos)

Processo/tempoChegada/Burst/texto/dados/pilha

001 0 10 10 10 10

002 0 15 20 10 20

003 0 10 5 4 1

004 50 10 5 5 5

005 23 3 1 1 1

Na saída verificamos os seguintes dados, o nome do arquivo de entrada '.txt', o algoritmo de alocação de memória, tamanho da memória, e quantum de tempo que será usado no algoritmo round robin.

Arquivo de entrada: cenario.txt

Gerencia de memoria: Segmentacao pura

Tamanho da memoria: 64 bytes

Quantum de tempo: 3 ms

– Abaixo o arquivo de saída mostra que a fila de processos(já ordenado por PID) vai sendo alocada na memória de acordo com os espaços livres contidos nela ou se entra em fila de espera pela memória.

– É mostrado o tempo de chegada de cada processo em ms.

– E quando um processo ganha memória, o programa mostra o seguinte formato : [Processo,secao][base:limite].

– Em caso de espaço não alocado – buracos, será indicado por 'HOLE'.

0 ms: processo 1 chegou ao sistema.

0 ms: processo 1 obteve memoria.

*Secoes: (texto) base 0, limite 10; (dados) base 10, limite 20; (pilha) base 20, limite 30.

0 ms: processo 2 chegou ao sistema.

0 ms: processo 2 entrou na fila de memoria.

0 ms: processo 3 chegou ao sistema.

0 ms: processo 3 obteve memoria.

*Secoes: (texto) base 30, limite 35; (dados) base 35, limite 39; (pilha) base 39, limite 40.

-->Memoria: [1,T][0:10],[1,D][10:20],[1,P][20:30],[3,T][30:35],[3,D][35:39],[3,P][39:40],[HOLE][40:64].

– Segue o processamento em CPU pelos processos alocados me memória.

– Mostra o instante de chegada na CPU (representado em ms), e os estados dos processos que ganham a CPU : qual o processo ganha a CPU, ou é preemptado dela, ou é concluído.

0 ms: processo 1 executando na CPU.
3 ms: processo 1 preemptado.
3 ms: processo 3 executando na CPU.
6 ms: processo 3 preemptado.
6 ms: processo 1 executando na CPU.
9 ms: processo 1 preemptado.
9 ms: processo 3 executando na CPU.
12 ms: processo 3 preemptado.
12 ms: processo 1 executando na CPU.
15 ms: processo 1 preemptado.
15 ms: processo 3 executando na CPU.
18 ms: processo 3 preemptado.
18 ms: processo 1 executando na CPU.
19 ms: processo 1 concluido.

– Nesse trecho observamos o tempo de resposta do processo concluído, turnaround que representa o tempo total que um processo executa, do instante de chegada ate sua conclusão, incluindo as esperas nas filas, e o tempo de espera que é a soma dos tempos em que um processo está na fila de prontos mas não está sendo executado.

*Tempo de resposta: 0 ms.

*Turnaround: 19 ms.

*Tempo de Espera: 9 ms.

– Com a conclusão do processo 1, este é desalocado da memória, dando espaço para o processo 2.

– A memória é atualizada, mostrando os novos segmentos sendo alocados em memória pelo processo 2.

19 ms: processo 2 obteve memoria.

*Secoes: (texto) base 0, limite 20; (dados) base 20, limite 30; (pilha) base 40, limite 60.

-->Memoria: [2,T][0:20],[2,D][20:30],[3,T][30:35],[3,D][35:39],[3,P][39:40],[2,P][40:60],[HOLE][60:64].

– Segue a execução round robin pelos processos alocados em memória.

19 ms: processo 3 executando na CPU.

20 ms: processo 3 concluido.

– Com a conclusão do processo 3, este é desalocado da memória.

– O programa informa os tempos de resposta, o turnaround, o tempo de espera do processo concluído em ms.

*Tempo de resposta: 3 ms.

*Turnaround: 20 ms.

*Tempo de Espera: 10 ms.

- O processo 2 continua em execução, e no tempo de chegada do processo 5, este ganha memória, como indicado abaixo em formato : [Processo,secao][base:limite].

-->Memoria: [2,T][0:20],[2,D][20:30],[HOLE][30:40],[2,P][40:60],[HOLE][60:64].

20 ms: processo 2 executando na CPU.

23 ms: processo 2 preemptado.

23 ms: processo 5 chegou ao sistema.

23 ms: processo 5 obteve memoria.

*Secoes: (texto) base 30, limite 31; (dados) base 31, limite 32; (pilha) base 32, limite 33.

-->Memoria: [2,T][0:20],[2,D][20:30],[5,T][30:31],[5,D][31:32],[5,P][32:33],[HOLE][33:40],[2,P][40:60],[HOLE][60:64].

23 ms: processo 2 executando na CPU.

26 ms: processo 2 preemptado.

26 ms: processo 5 executando na CPU.

29 ms: processo 5 concluido.

– Com a conclusão do processo 5, este é desalocado da memória.

– O programa informa os tempos de resposta, o turnaround, o tempo de espera do processo concluído em ms.

*Tempo de resposta: 3 ms.

*Turnaround: 6 ms.

*Tempo de Espera: 3 ms.

– O processo 2 continua em execução.

-->Memoria: [2,T][0:20],[2,D][20:30],[HOLE][30:40],[2,P][40:60],[HOLE][60:64].

29 ms: processo 2 executando na CPU.

32 ms: processo 2 preemptado.

32 ms: processo 2 executando na CPU.

35 ms: processo 2 preemptado.

35 ms: processo 2 executando na CPU.

38 ms: processo 2 concluido.

– Com a conclusão do processo 2, este é desalocado da memória.

– O programa informa os tempos de resposta, o turnaround, o tempo de espera do processo concluído em ms.

*Tempo de resposta: 20 ms.

*Turnaround: 38 ms.

*Tempo de Espera: 4 ms.

– O processo 4 é alocado em memória no instante de chegada indicado.

– É indicado o instante que o processo chega a CPU, e é alocado nos segmentos indicados abaixo :

50 ms: processo 4 chegou ao sistema.

50 ms: processo 4 obteve memoria.

*Secoes: (texto) base 0, limite 5; (dados) base 5, limite 10; (pilha) base 10, limite 15.

-->Memoria: [4,T][0:5],[4,D][5:10],[4,P][10:15],[HOLE][15:64].

50 ms: processo 4 executando na CPU.

53 ms: processo 4 preemptado.

53 ms: processo 4 executando na CPU.

56 ms: processo 4 preemptado.

56 ms: processo 4 executando na CPU.

59 ms: processo 4 preemptado.

59 ms: processo 4 executando na CPU.

60 ms: processo 4 concluido.

– Com a conclusão do processo 4, este é desalocado da memória.

– O programa informa os tempos de resposta, o turnaround, o tempo de espera do processo concluído em ms.

*Tempo de resposta: 0 ms.

*Turnaround: 10 ms.

*Tempo de Espera: 0 ms.

Processos que requerem mais memória do que memória total disponível são descartados pelo programa.

Conclusão

O desenvolvimento do simulador consiste em analisar de forma detalhada o funcionamento dos métodos propostos no trabalho. As etapas do processo inclui testes e validação do modelo proposto, planejamento de experimentos(eventos de rotina), implementação do programa e análise de resultados.

O simulador foi implementado em java e caracteriza-se por sua simplicidade, uma vez que a interface gráfica proporciona maior interação com o usuário. O simulador exibe a alocação de memória pelos algoritmos de paginação e segmentação e executa com os eventos descritos pelo usuário. O resultado é apresentado por um arquivo de saída gerado pelo programa, retornando todos as informações necessárias para a análise de resultado.

O trabalho teve por finalidade proporcionar a associação da teoria e da prática oferencendo melhor entendimento dos conceitos aplicados em cada método de alocação de memória, dando uma ánalise mais detalhada sobre os resultados, determinando o comportamento de cada método sobre o uso da memória e também pelo uso da CPU pelo algoritmo round robin.

Bibliografia

MacDOUGALL, M. H. Computer System Simulation : An Introduction. Computing Surveys, Vol.2, No. 3, Setembro 1970.

SILBERSCHATZ, A. GALVIN, P.B., GAGNE, G.; Sistemas Operacionais : Conceitos e Aplicações. Editora Campus, 2001.