Universidade São Judas Tadeu

Sistemas Operacionais

Pesquisa sobre Escalonamento de processos

e estratégias de alocação de memória

Butantã, São Paulo - fevereiro de 2019

Universidade São Judas Tadeu

Lucas de Lima Ferreira – 817121961

Grupo 5 – Professora Milkes Yone Alvarenga

Ciência da Computação

Sistemas Operacionais

Pesquisa sobre Escalonamento de processos e estratégias de gerenciamento de memória

Butantã, São Paulo - fevereiro de 2019

Sumário

1. Introdução

Este trabalho aborda sobre escalonadores de processos, seus tipos e os algoritmos que ele possui, além dos conceitos de alocação de memória do sistema.

A construção de um escalonador para trabalhar com processos de tempo real é uma tarefa muito complexa devido as considerações e decisões a serem feitas no seu desenvolvimento.

Alguns critérios foram levados em consideração para escolha dos escalonadores propostos, onde os principais são:

- Justiça, cada processo recebe uma parcela suficiente da CPU;
- Eficiência, manter a CPU ocupada o máximo possível;
- Tempo de Resposta, tempo entre a submissão e a primeira resposta;
- Tempo de Turnaround, intervalo entre a submissão e o termino da execução de um trabalho;

2. O que é escalonamento de processos

O escalonador de processo é um subsistema do SO responsável por decidir em qual momento cada processo fará uso da CPU. O sistema operacional possui variados algoritmos para decidir fazer essa decisão e cabe ao mesmo, avaliar o cenário para decidir qual algoritmo utiliza, dentre eles estão os de I/OBound e os CPU Bound.

Os processos de I/O Bound é quando os processos passam a maior parte do tempo em estado de espera, pois realizam um número elevado de operações de entrada e saída, como leitura ou gravação.

Os processos de CPU Bound são classificados quando a maior parte do tempo os processos estão em execução, ou seja, utilizando o processador. Esse tipo de processo é encontrado em aplicações que efetua muitas operações.

3. Algoritmos preemptivos e não preemptivos

Os algoritmos de escalonamento podem ser não preemptivos (que executa a tarefa até que ela seja finalizada sem interrupções) ou preemptivos (que executa a tarefa em um tempo determinado pela SO, que não necessariamente é o mesmo tempo que o processo levará para executar totalmente).

4. Prazo de escalonadores

4.1 Escalonador de longo prazo

Determina quais programas podem ser admitidos no sistema para o processamento, controlando o número de Jobs* na memória. Mesmo que o escalonador esteja selecionando aquele Job* significa que ele será executado imediatamente.

4.2 Escalonador de médio prazo

Determina a adição de um número de processos que estão parcialmente ou completamente na memória.

4.3 Escalonamento de curto prazo

Decide qual dos Jobs deve utilizar o processador e quando utilizará. Quando o Job se torna um processo, ele é adicionado à fila do escalonador.

Nota: Um Job é um programa que foi selecionado para a execução, mas ainda não foi executado.

1. Escalonador por alternância circular – Round Robin

O Round Robin é um dos algoritmos de escalonamento mais antigos. Esse algoritmo atribui um quantum (unidade de tempo definida pelo SO, normalmente esse tempo é em milissegundos), o de tempo igual para cada processo de forma circular. Se nesse tempo atribuído, o processo não terminou, a CPU corta o processo atual (sofre preempção) e o próximo processo da fila passa a executar. O processo anteriormente não terminado, vai para o fim da fila para ser executado novamente. Essa forma de execução não dá prioridade aos processos. O algoritmo RR é imune a problemas de "STARVATION" (estagnação), evitando que uma tarefa que demore muito tempo a ser executada, atrapalhe as demais. Esse escalonador é usado em sistemas operacionais multitarefas, e foi feito principalmente para sistemas Time Sharing (tempo compartilhado), pois o algoritmo necessita de um temporizador.

Exemplo:

O processo "B" leva um tempo de 60ms para ser executado, e o algoritmo de escalonamento atribuiu um tempo de 25ms para execução de cada processo (quantum). O processo "B" será executado em 25ms e passará para o fim da fila. As próximas execuções do processo "B" levarão 25ms + 10ms. A última execução levará menos tempo que o escalonador atribuiu, então a tarefa passará por um processo de preempção, que significa que a tarefa será interrompida, já que a mesma não precisará do tempo que sobrou.

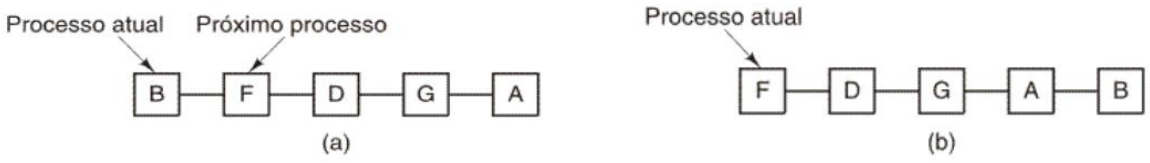

Imagem: Indicando fila Circular.

Para calcularmos o tempo de espera médio de cada processo, é necessário sabermos o tempo de espera de cada processo. Para isso devemos simular o algoritmo para pegar todas as informações.

PROCESSO	TEMPO DE CHEGADA	DURAÇÃO
P1	0	6
P2	2	4
P3	5	8
P4	6	1
P5	8	3

Imagem: Exemplo de processos em seu tempo de chegada e sua duração.

A fórmula de tempo de espera é: TempoEspera = (FinalProcesso - TempoChegada – Duração).

Com a tabela anteriormente mostrada, temos informação do tempo de chegada e duração do processo. Para sabermos o final do processo, devemos entender a maneira de como o processo aconteceu.

Digamos, que o sistema operacional decidiu que o quantum que será executado os seguintes processos será de 2. Montamos abaixo uma tabela de acordo com a tabela anterior, evidenciando cada processo executado, seu tempo de duração e a chegada do processo na fila. Da maneira que a tabela foi montada, conseguimos observar o momento no qual cada processo foi finalizado.

Tempo	Processo em execução	Fila		
0	P1			
1	P1			
2	P2	P1		
3	P2	P1		
4	P1	P2		
5	P1	P2	P3	
6	P2	P3	P1	P4
7	P2	P3	P1	P4
8	P3	P1	P4	P5
9	P3	P1	P4	P5
10	P1	P4	P5	P3
11	P1	P4	P5	P3
12	P4	P5	P3	
13	P5	P3		
14	P5	P3		
15	P3	P5		
16	P3	P5		
17	P5	P3		
18	P3			
19	P3			
20	P3			
21	P3			

Imagem: Exemplo do tempo de cada processo, o tempo de execução e as filas que se formam até o final do processo.

Observamos que, o tempo no qual o P1 terminou sua tarefa foi no tempo 11, o P2 no tempo 7, o P3 no tempo 21, o P4 no tempo 12 e o P5 no tempo 17. Dessa forma, conseguimos todos os valores necessários para obter o tempo de espera de cada processo.

TEP1 = 11 – 0 – 6 = 5

TEP2 = 7 – 2 – 4 = 1

TEP3 = 21 – 5 – 8 = 8

TEP4 = 12 – 6 – 1 = 5

TEP5 = 17 – 8 – 3 = 6

Agora podemos obter o tempo médio dos processos:

Tempo Médio = (TEP1+TEP2+TEP3+TEP4+TEP5+TEPn) / N° de Processos

Tempo médio = (5+1+8+5+6) /5= 5

2. Escalonador FCFS (First-Come First-Served)

O processador é alocado seguindo a ordem de chegada dos processos à fila de processos prontos, ou seja, o primeiro a chegar é o primeiro a ser atendido pelo escalonador, quando um processo requer a CPU ele é executado, quando chegam mais processos, estes são inseridos em uma fila conforme a ordem de chegada. Nesse escalonamento todos os processos tendem a serem atendidos, exceto se algum processo possuir erro ou loop infinito, caso ocorra o loop infinito o processador irá ser encerrado, pois não terá como dá continuidade a execução dos outros processos que se encontram na fia. A grande vantagem desse algoritmo é que ele é de fácil entendimento e também de fácil implementação, porém a maior desvantagem pode ser o tempo médio de execução, pois nesse algoritmo os processos de longo tempo levam vantagem, se comparado aos outros. Em um algoritmo com preempção a cada 10 milissegundos (em vez de cada um segundo), o processo terminaria em 10 segundos.

Exemplo:

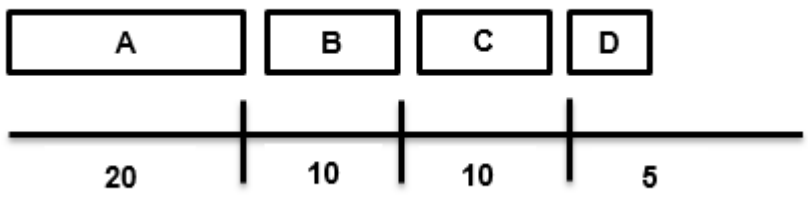

Imagem: Tempo de processo.

Conforme podemos ver no exemplo acima, o primeiro processo que é o de maior duração (10 segundos), leva vantagem em relação aos outros, principalmente se comparado ao último processo que é o de menor duração. O tempo médio de execução é de 11,25 segundos.

Tarefa	T1	T2	T3	T4
Ingresso	0	0	1	3
Duração	5	2	4	3

Imagem: exemplo de funcionamento do algoritmo FCFS, é através da tabela.

O diagrama abaixo mostra o funcionamento do algoritmo FCFS, onde os quadros em cinza são os que representam o uso do processador (conforme podemos notar, a cada instante apenas uma tarefa ocupa o processador) já os quadros em branco representam as tarefas que estão na fila aguardando.

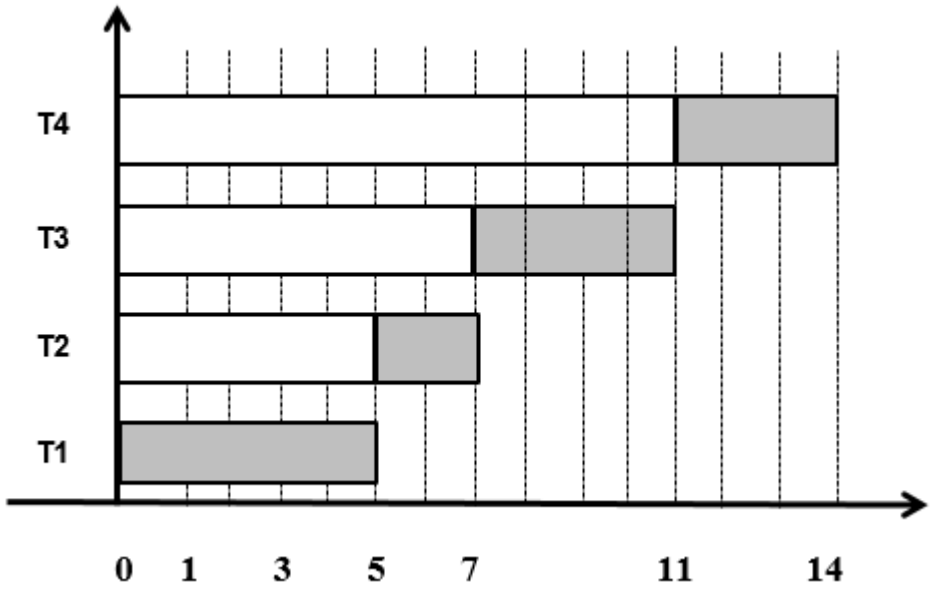

Imagem: tarefas respeitando seu tempo de execução de cada uma .

3. Escalonador SJF (Shortest Job First)

O SJF é um algoritmo de escalonamento que é um caso especial de algoritmo por prioridade que é definida em função do surto de CPU (uso da CPU) ele privilegia processos de tamanho menor.

O Algoritmo pressupõe o conhecimento prévio dos tempos de execução de todos os processos. E com esse conhecimento o processo com o menor tempo de execução sai da fila de pronto e entra em execução na CPU. E Atrás do mesmo forma uma fila de processos por ordem crescente de tempo de execução.

Processo com tempo iguais, utiliza–se a ordem de chegada (FCFS) o primeiro a entrar na fila que será o primeiro a ser executado.

3.1 O SJF tem dois tipos:

SJF não-preemptivo: esse tipo, uma vez atribuída um processo a CPU, este não pode ser desalojado antes de consumir o tempo previsto

SJF preemptivo – esse tipo, se chegar um novo processo com tempo de uso de CPU inferior ao tempo que resta ao processo em execução, então desalojar o processo que esta em execução e entra o outro processo com tempo menor.

3.2 Características:

- Sempre executa o processo da fila que demanda o menor tempo

- Cada processo é relacionado ao seu tempo de execução

- Quanto menor o tempo de execução, maior a prioridade dada pelo escalonador para execução.

3.3 Problemas:

Determinar o tempo de execução de um processo antes de executá-lo

(Pode ser feito através de análise histórica de execução do processo).

Emprego mais curto não preventivo primeiro

Considere os processos abaixo disponíveis na fila pronta para execução, com o **tempo de chegada** como 0 para todos e os **tempos de burst**.

O processo mais baixo é executado primeiro.

Processo	Tempo
P1	21
P2	3
P3	6
P4	2

Imagem: processos e tempo de execução.

Assim, o gráfico de Gantt será seguido.

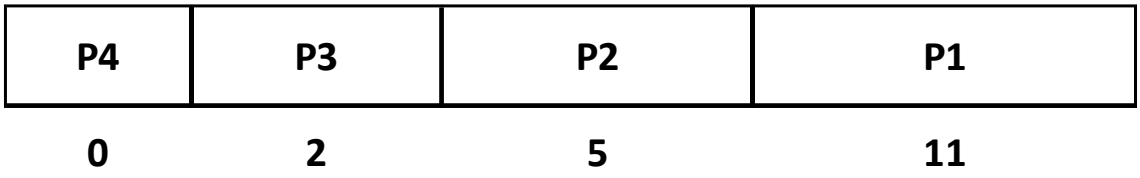

Imagem: indicando qual processo vai ser executado primeiro.
Agora, o tempo de espera médio será =(0+2+5+11)/4 = 4.5 ms.

Como você pode ver no **gráfico GANTT** acima, o processo **P4** será escolhido primeiro, pois tem o menor tempo de burst, depois **P2**, seguido por **P3** e, finalmente, **P1**.

Agendamos o mesmo conjunto de processos usando o algoritmo First come first first no tutorial anterior, e obtivemos tempo médio de espera 18.75 ms, enquanto com o SJF, o tempo médio de espera é atingido 4.5 ms.

Preempção no emprego mais curto primeiro

No Agendamento de Tarefa Mais Curta Preemptiva, as tarefas são colocadas na fila pronta à medida que chegam, mas à medida que um processo com **tempo de burst curto** chega, o processo existente é eliminado ou removido da execução e o job mais curto é executado primeiro.

Processo	**Busrt Time Curto**	**Tempo de Chegada**
P1	**21**	**0**
P2	**3**	**1**
P3	**6**	**2**
P4	**2**	**3**

Imagem: Cada processo, com seu tempo curto prazo e seu tempo de chegada

O gráfico GANTT para o esquema preventivo de trabalho mais curto em primeiro lugar será:

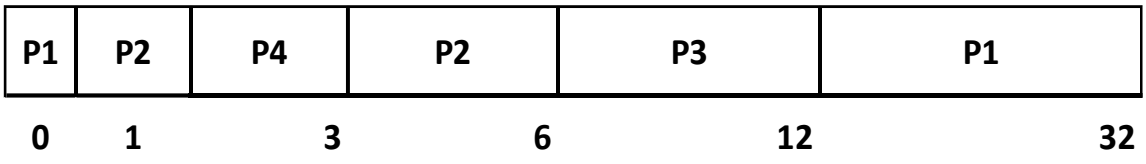

Imagem: Tempo de execução de cada processo.

O tempo de espera médio será =((5-3)+(6-2)+(12-1))/4 = 4.25 ms.

Como você pode ver no **gráfico GANTT** acima, como **P1** chega primeiro, portanto sua execução começa imediatamente, mas logo após 1ms, o processo **P2** chega com um **tempo** de **burst** 3ms que é menor que o tempo de burst de **P1**, daí o processo **P1** (1ms 20ms restantes) é preemptado e o processo **P2** é executado.

Conforme o **P2** está sendo executado, após 1ms, o **P3** chega, mas tem um tempo de burst maior que o do **P2**, portanto a execução do **P2** continua. Mas depois de outro milésimo de

segundo, **P4** chega com um tempo de burst de 2ms, como resultado **P2** (2ms terminado, 1ms restante) é preemptido e **P4** é executado.

Após a conclusão de **P4**, o processo **P2** é captado e finalizado, então **P2** será executado e, finalmente, **P1**.

O SJF preventivo também é conhecido como **Tempo Restante Mais Curto Primeiro**, porque em qualquer ponto do tempo, o trabalho com o menor tempo restante é executado primeiro.

4. Escalonador por prioridade

O algoritmo por prioridade funciona da seguinte maneira, escolhe o processo da fila de processos prontos que tenha maior prioridade. A fila de processos prontos é ordenada pela prioridade de cada processo. Favorece os processos com maior prioridade.

Cada processo tem um valor inteiro associado, que representa a prioridade da execução. Podem ser atribuídas de forma crescente (0 - prioridade baixa) ou decrescente (0 - prioridade alta).

O algoritmo é implementado por um clock, que interrompe o processador em determinados intervalos de tempo, reavaliando prioridades e possivelmente escalonando outro processo.

As prioridades são definidas pelo tipo de Sistema Operacional, como Linux, Microsoft entre outros.

Cada fila pode ter uma prioridade diferente, ou seja, ter a própria política de escalonamento a partir dos algoritmos FIFO, SJF e RR; (podem ser desempatados pelos algoritmos descritos); isso ocorre em um algoritmo de múltiplas filas.

Prioridade estática, não é modificada durante um processo, de simples implementação, pode ocorrer tempo de resposta elevado.

Prioridade dinâmica, pode ser modificada durante a execução do processo, o processo recebe um acréscimo na sua prioridade ao sair do estado de espera, processos I/O-Bound terão mais chances de serem escalonados compensando o tempo que passam no estado de espera e os processos CPU-Bound podem ser executados enquanto os processos I/O-Bound esperam por algum evento.

O tempo de resposta compensa o maior overhead e complexidade algorítmica.

Um processo de baixa prioridade pode ser deixado de lado, ocorrendo de nunca ser executado, para solucionar este problema o ideal é mudar a prioridade com o passar do tempo.

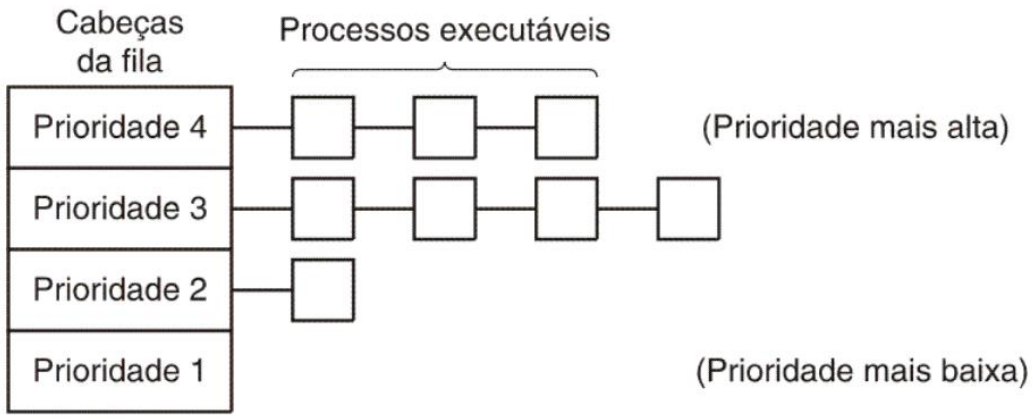

Imagem: A figura mostra o exemplo de quatro tipo de classes de prioridade.

5. **SRT**

Shortest remaining time ("tempo remanescente mais curto" em inglês; sigla: **SRT**) é a variante preemptiva do escalonamento SJF. A fila de processos a serem executados pelo SRT é organizada conforme o tempo estimado de execução, ou seja, de forma semelhante ao SJF, sendo processados primeiros os menores jobs. Na entrada de um novo processo, o algoritmo de escalonamento avalia seu tempo de execução incluindo o job em execução, caso a estimativa de seu tempo de execução seja menor que o do processo concorrentemente em execução, ocorre a substituição do processo em execução pelo recém chegado, de duração mais curta, ou seja, ocorre a preempção do processo em execução.

Cada processo suspenso pelo SRT deverá ser recolocado na fila em uma posição correspondente apenas ao seu tempo restante de execução e não mais o tempo de execução total, tornando-se necessário registrar os tempos decorridos de execução de cada processo suspenso.

A principal diferença para o algoritmo SJF é a preempção, digamos que um processo X esteja em execução na CPU e nesse meio tempo chegue um processo Y menor do que o restante do processo X, nesse momento ocorrerá uma preempção, ou seja, o processo X irá parar sua execução e ceder lugar para o processo Y executar. Caso chegue um processo ainda menor que o

restante do processo Y, esse processo ganhará a CPU e o processo Y retornará para a fila de prontos antes de terminar e irá aguardar um momento para ser executado.

Para que a preempção ocorra é necessário um timer que determine corretamente o tempo em que uma interrupção de hardware possa alternar os processos.

Como o trabalho mais curto em primeiro lugar, ele tem potencial para inanição do processo; longos processos podem ser mantidos indefinidamente se processos curtos forem continuamente adicionados. Essa ameaça pode ser mínima quando os tempos de processo seguem uma distribuição de cauda pesada.
Características:

- Preempção: interrompe um processo e inicia outro menor.
- Tempo de resposta: possuirá um tempo de resposta muito bom se o processo não for muito grande, caso seja demorará muito para começar a ser executado.
- Tempo de espera: caso comece a ser executado e de repente volte à fila de prontos, terá um tempo de espera maior que o tempo de resposta.
- Starvation: possível de ocorrer em processos longos.
- Throughput: possuirá um alto número de processos por unidade de tempo.

Quando um processo qualquer começa a executar e nenhum outro processo é menor que ele em nenhum momento, ele irá possuir o tempo de espera igual ao tempo de resposta.

6. Alocação Contígua Simples

A alocação contígua simples foi implementada nos primeiros sistemas operacionais desenvolvidos, porém, ainda está presente em alguns sistemas monoprogramáveis.

Nesse tipo de organização, a memória principal é dividida em duas partes:

- uma para o sistema operacional;
- outra para o programa do usuário.

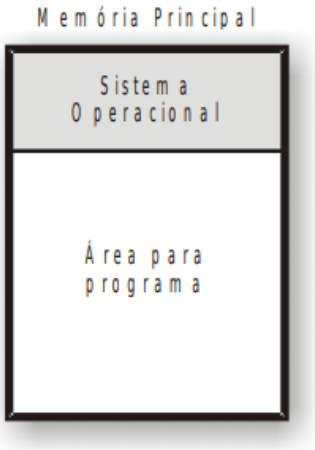

Figura 6.1: Memória principal.

O usuário tem controle sobre toda a memória principal, podendo ter acesso a qualquer posição de memória, inclusive alterar e destruir o sistema operacional.

Para protegê-lo desses ataques, conscientes ou não, alguns sistemas operacionais implementam proteção através de um registrador, que delimita as áreas do sistema operacional e do usuário.

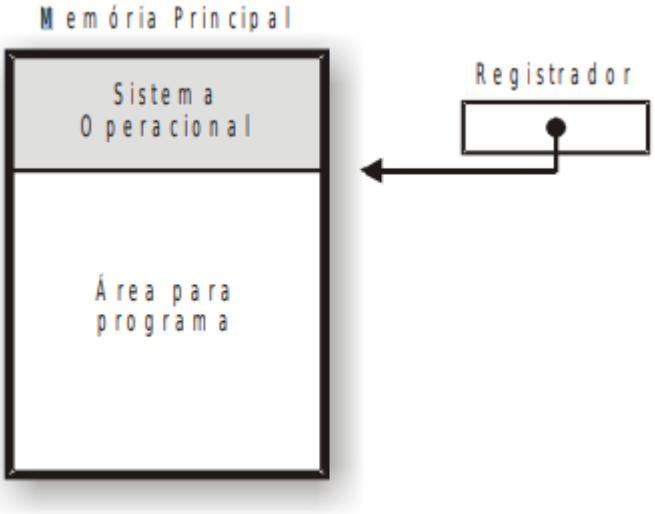

Figura 6.2: Memória Principal (Registrador)

Apesar de sua simplicidade de implementação e código reduzido, a alocação contígua simples não permite a utilização eficiente do processador e da memória pois apenas um usuário pode utilizar este recurso.

Em relação à memória, caso o programa não a preencha totalmente, existirá um espaço de memória sem utilização.

Fragmentação.

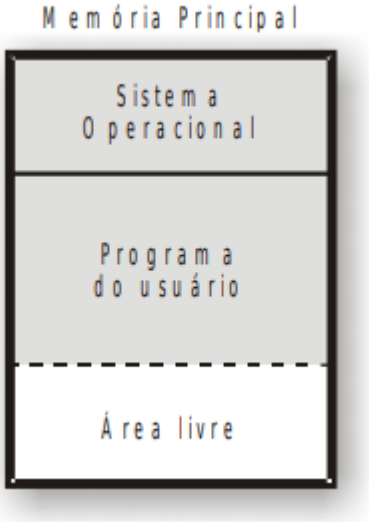

Figura 6.3: Memória Principal (Fragmentação).

No princípio, os programas dos usuários estavam limitados ao tamanho da memória principal disponível.

A solução encontrada para o problema foi dividir o programa em partes (módulos), de forma que pudessem executar independentemente uma da outra, utilizando a mesma área da memória.

Esta técnica recebeu o nome de overlay.

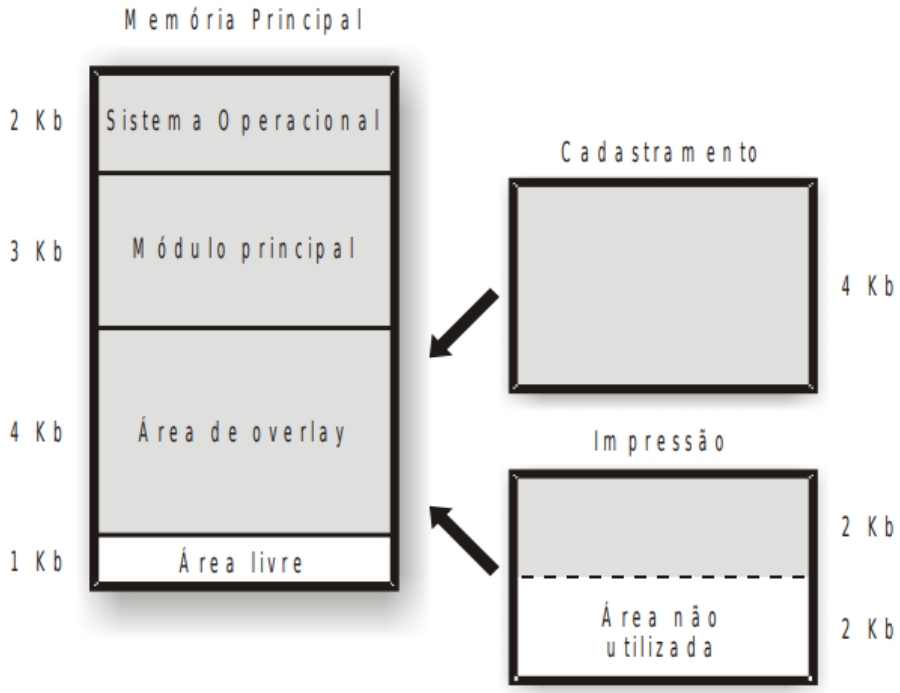

Figura 6.4: Memória Principal (Área não utilizada)

7. Alocação Particionada

Os sistemas monoprogramáveis permitem que o processador permaneça ocioso e que a memória seja subutilizada, enquanto um programa aguarda o término de uma operação de I/O, por exemplo.

A multiprogramação vem resolver este problema, pois enquanto aguarda algum evento, outros processos podem ser executados pela CPU nesse intervalo de tempo.

Para a multiprogramação ser eficiente, é necessário que vários programas estejam na memória ao mesmo tempo, daí a necessidade de uma nova forma de organização da memória principal.

8. Alocação Particionada Estática

Nos primeiros sistemas multiprogramáveis, a memória foi dividida em pedaços de tamanho fixo, chamados partições.

O tamanho das partições era estabelecido na fase de inicialização do sistema (boot), em função do tamanho dos programas que seriam executados no ambiente.

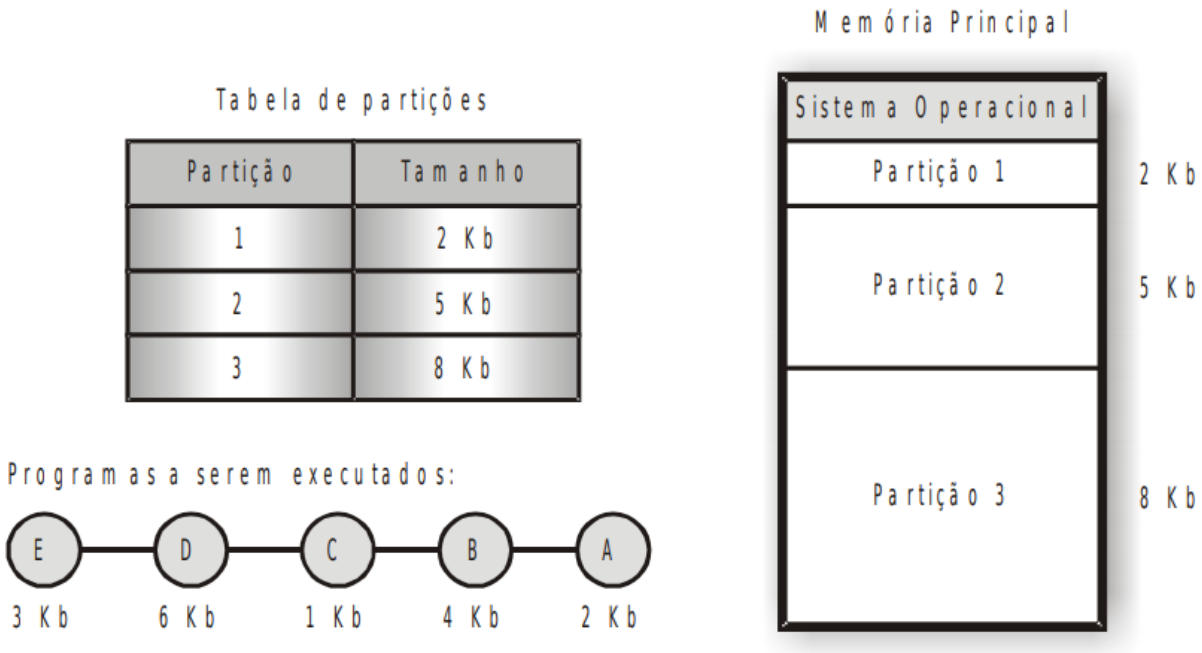

Figura 8.1: Tabela de partições e memória principal.

No princípio, os programas só podiam ser executados em uma das partições, mesmo que outras estivessem disponíveis.

Essa limitação era devido aos compiladores utilizados na época, que geravam códigos absolutos.

Este tipo de alocação recebeu o nome de alocação particionada estática absoluta.

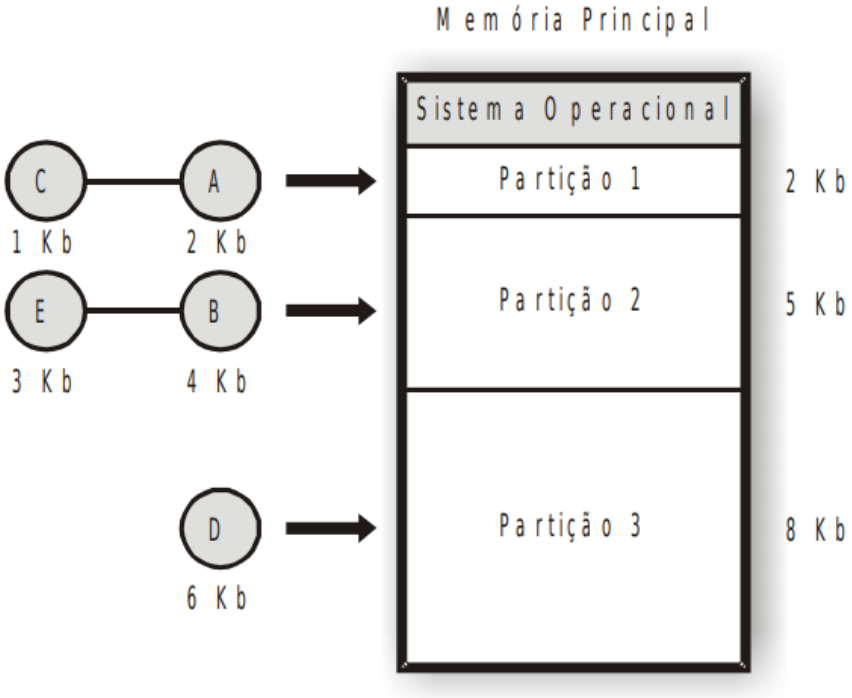

Figura 8.2: Estática Absoluta

Com a evolução dos compiladores, linkers e loadres, a geração de código relocável foi possível e os programas puderam ser carregados em qualquer partição.

A esse novo tipo de alocação deu-se o nome de alocação particionada estática relocável.

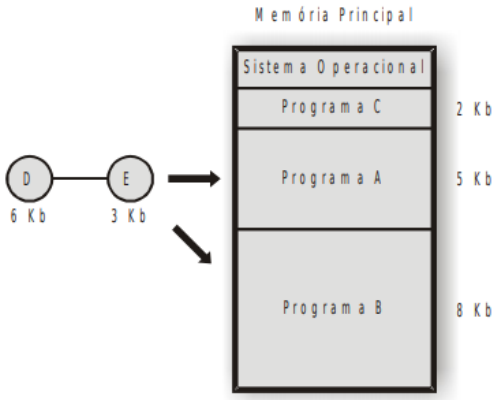

Figura 8.3: Partição Relocavél

Para manter o controle sobre quais as partições estavam alocadas ou não, os sistemas possuíam uma tabela delimitando cada partição, seu tamanho e se estava em uso ou não.

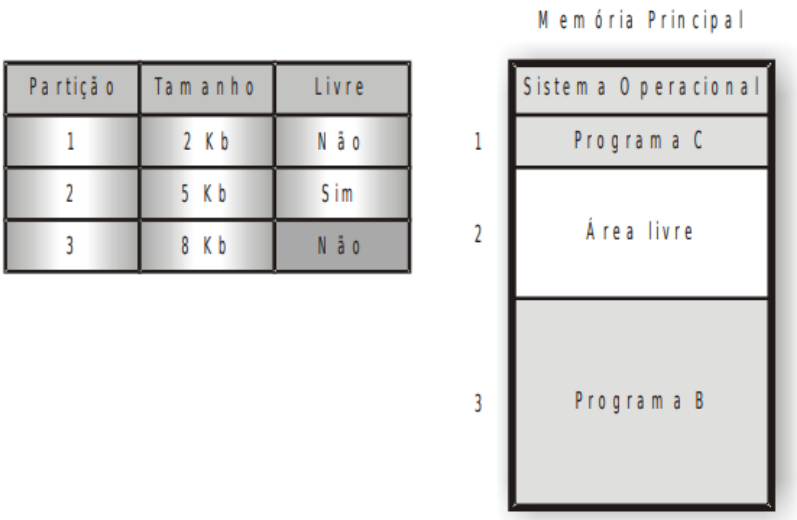

Figura 8.4: Verificação de tamanho em uso

A proteção baseava-se em dois registradores, que indicavam os limites inferiores e superiores da partição onde o processo seria carregado.

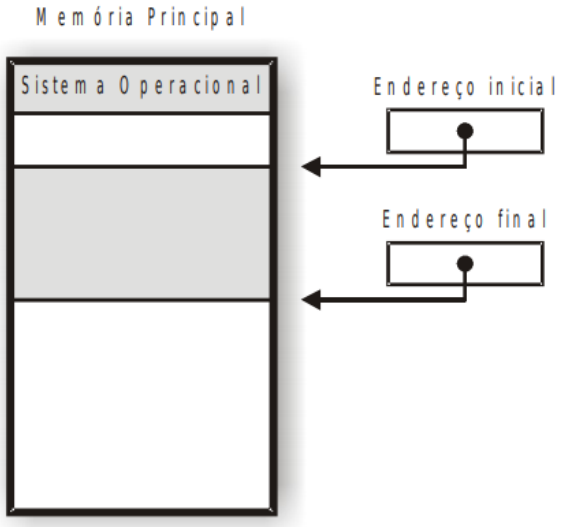

Figura 8.5: Partição onde o processo seria carregado

Tanto nos sistemas de alocação absoluta como nos sistemas com alocação relocável, os processos não preenchiam totalmente as partições onde eram alocadas. Desta forma, a fragmentação também estava presente nestes esquemas.

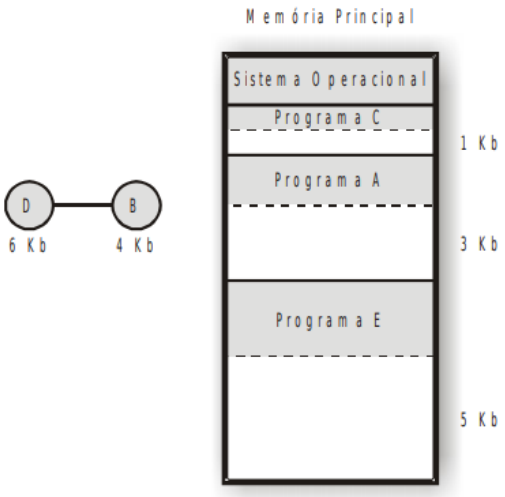

Figura 8.6: Partições não preenchidas totalmente.

9. Alocação Particionada Dinâmica

Aumenta o grau de compartilhamento de memória.

Diminui o grau de fragmentação.

Neste esquema foi eliminado o conceito de partições com tamanho fixo.

Cada programa utiliza o espaço que necessitasse, desde que existisse este espaço na memória, transformando-o em uma partição.

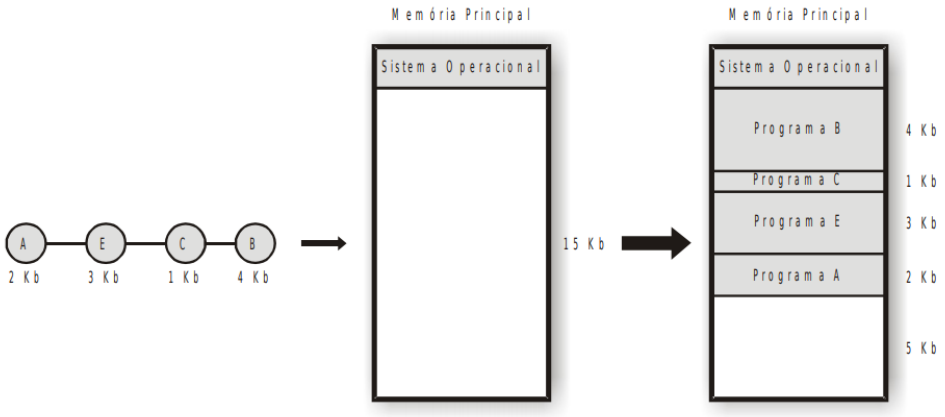

Figura 9.1: Transformando espaço em partição

Neste esquema também ocorre a fragmentação e ela aparecerá na medida em que os processos forem terminado e deixando espaços cada vez memores na memória, não permitindo o carregamento de outros processos.

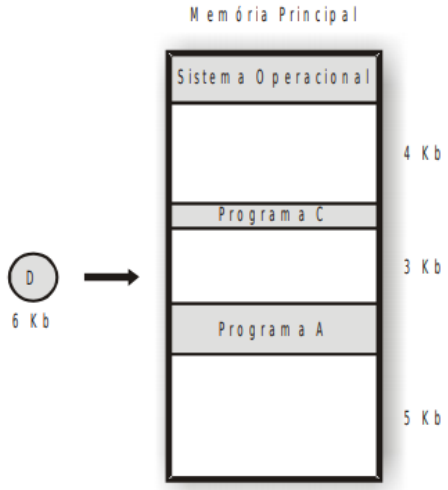

Figura 9.2: Fragmentação

Uma solução seria fazer com que os espaços adjacentes fossem reunidos, produzindo um único espaço de tamanho maior.

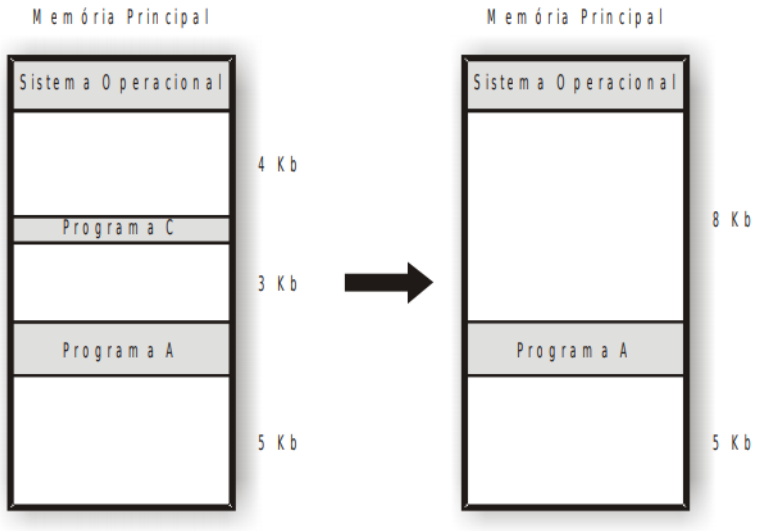

Figura 5.3: Solução de adjacente

A segunda maneira seria realizar uma relocação de todas as partições ocupadas, eliminando todos os espaços entre elas e criando-se uma única área livre contígua.

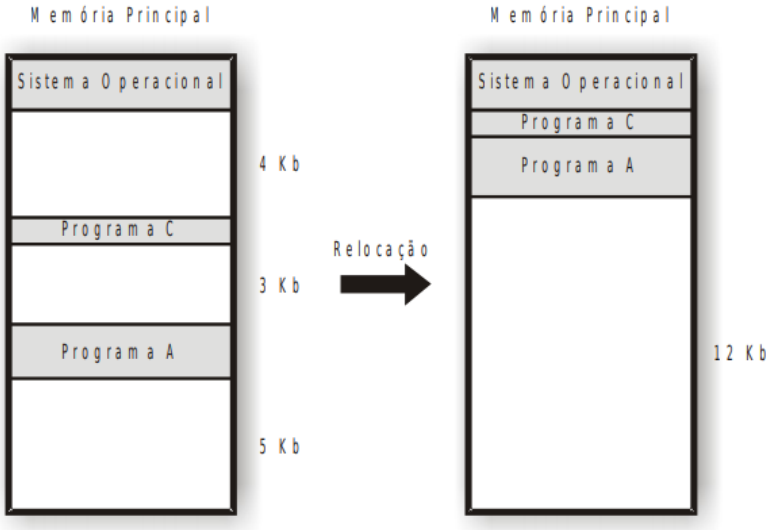

Figura 9.4: Criação área livre contígua

10. Memórias de alocação

10.1 First Fit

A busca de espaço livre, começa a varrer a memória do início, escolhe o primeiro bloco que seja grande o suficiente.

O método tenta usar primeiro as áreas livres de endereços baixos, podendo obter uma grande partição livre nos endereços altos. A lista de áreas livres é ordenada em ordem crescente

Este algoritmo é o mais rápido, consome menos recursos do sistema.

10.2 Next Fit

Algoritmo inferior ao First Fit, a diferença está na busca, ocorrendo a partir do endereço da última posição.

Aloca blocos livres no fim da memória, desta forma blocos grandes de memória são quebrados em blocos menores. Para se obter grandes blocos no fim da memória.

10.3 Best Fit

Escolhe o bloco onde o processo deixa menor espaço sem utilização.

Seu objetivo é garantir a melhor escolha de partição livre, assim todos os blocos vão ser avaliados.

Como ordena por tamanho, diminui o tempo de busca por área desocupada.

Algumas desvantagens que podem vir a ocorrer são, ter grande quantidade de pequenas áreas livres não-continuas e ocorrer o problema de fragmentação de memória.

10.4 Worst Fit

Escolhe o maior espaço livre na memória, a lista de área livre é ordenada por tamanho, otimizando a busca, reduz o problema de fragmentação de memória, em comparação ao Best Fit.

Como deixa espaço livres maiores, permite um maior número de programas usar a memória.

Mas seu tempo de busca é grande, não tendo bons resultados.

11 Bibliografia

https://www.youtube.com/watch?v=YoToRdqa6us

http://ctd.ifsp.edu.br/~marcio.andrey/images/Escalonamento-Processos-IFSP.pdf

https://pt.slideshare.net/rodrigomuribec/aula-4-ok

https://www.oficinadanet.com.br/post/12781-sistemas-operacionais-o-que-e-escalonamento-de-processos

http://www.univasf.edu.br/~andreza.leite/aulas/SO/ProcessosEscalonamento.pdf

https://pt.wikipedia.org/wiki/Escalonamento_de_processos

https://pt.wikipedia.org/wiki/FIFO_(escalonamento)

http://ctd.ifsp.edu.br/~marcio.andrey/images/Escalonamento-Processos-IFSP.pdf

https://joaoricardao.files.wordpress.com/2012/07/algoritmos_escalonamento.pdf

http://www.univasf.edu.br/~andreza.leite/aulas/SO/ProcessosEscalonamento.pdf

http://ctd.ifsp.edu.br/~marcio.andrey/images/Escalonamento-Processos-IFSP.pdf

http://www.inf.ufrgs.br/~johann/sisop1/aula10.scheduling.pdf

https://www.gsigma.ufsc.br/~popov/aulas/so1/cap8so.html

https://pt.slideshare.net/rodrigomuribec/aula-4-ok

https://acessaferpa.wordpress.com/escalonamento-fifo/

https://www.studytonight.com/operating-system/shortest-job-first

http://rossano.pro.br/fatec/cursos/soi/gerenciamento_memoria_fatec.pdf