SHELL

O que é um comando?

Um conjunto de *strings*!

Sendo assim, nosso shell começa identificando todas as strings inseridas na linha de comando, armazenando-as em um vetor de *strings*, ou uma matrix de *chars*, para os entendidos.

Caso não se enquadre em nenhum dos formatos a seguir, o comando inserido é considerado um comando desconhecido.

Comandos

Com comandos sem argumentos, como “*exit*” ou nosso “*date*”, basta verificarmos se a linha corresponde a tal *string*.

Para todo o resto, verificamos a primeira das strings da linha escrita no *shell* para identificar o que deve ser executado.

Binários como */bin/ping* e */usr/bin/cal* são processados da mesma forma, através de um *execl*:

A primeira *string* representa nome/endereço do comando

A última *string* representa o argumento obrigatório do comando (“google.com” após um */bin/ping*, por exemplo)

As *strings* intermediárias são argumentos adicionais ao comando (“-c” e “5” após um */bin/ping, anterior ao endereço)*

Com isso, somos capazes de interpretar comandos como **“/bin/ping -c 3 youtube.com”** com o mesmo código que interpretamos **“/bin/ping google.com.br”**

Executáveis

Com executáveis, o processo é similar, embora mais simples. Usamos o *execvp*, onde:

A primeira strings representa “./ + 'nome do executável”
(com “./ep1sh” podemos rodar o shell em si mesmo, por exemplo)

Todas as outras *strings* são argumentos extras que serão recebidos pelo executável

Simulador de Processos

Todos os escalonadores recebem um vetor de *lines*, onde uma *line* é uma *struct* contendo todas as informações de uma linha do arquivo de trace (t0, dt, deadline e nome do processo).

Uma *line* é então convertida em um *process*, que possui as informações adicionais do tempo real de execução restante para a finalização do processo, e o *index* de sua *thread* (usado para controle das *threads*).

Interrupção de Threads

Inicialmente, *threads* eram interrompidas utilizando a chamada thread_cancel, tendo seu *canceltype* para um cancelamento assíncrono.

Esta abordagem, infelizmente, deixava incerto o momento em que um processo era realmente cancelado, o que criava problemas para o escalonador.

A solução foi determinar na própria estrutura do *process* o seu *quantum* de processamento. Sendo assim, quando uma *thread* ultrapassava seu quantum em tempo real de processamento, ela se cancela independentemente.

A partir disso, o escalonador é responsável principalmente por definir o quantum de cada *thread*, já que seu cancelamento é automático.

Tipos de Escalonadores

Shortest Job First

Neste escalonador foi usada uma *pilha* ordenada pelos *dt's* dos processos. Sempre que um novo processo chegava, ele era inserido na pilha de **forma a ela continuar ordenada**, com a execução priorizando o processo no topo da pilha. Esse regime era obedecido até todos os processos serem terminados, sem cancelamentos dos processos.

Round Robin

Neste escalonador foi usada uma estrutura de *filas*, onde os processos que chegavam sempre entravam no final da fila. Um processo que venha a ser cancelado (por atingir seu quantum de execução) era reinserido ao final da fila.

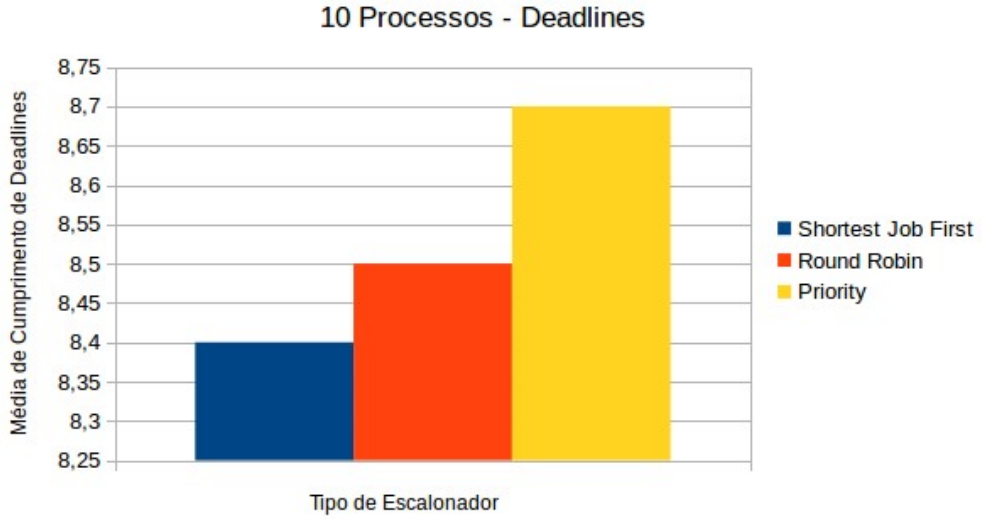
Escalonador com Prioridade

Também usando *filas*, este escalonador tem funcionamento similar ao **Round Robin**, com a diferença de que cada processo possui um tempo limite de execução dependente de sua **prioridade**.

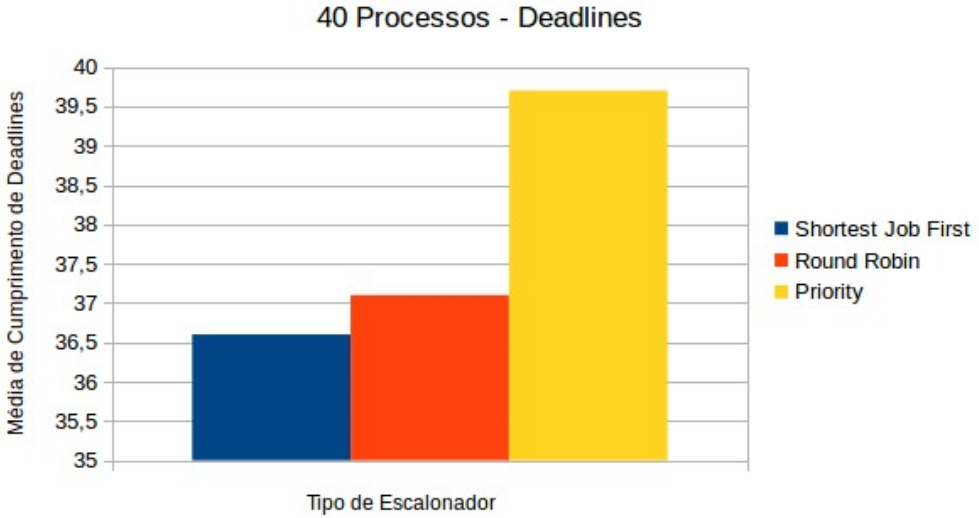
Processos de maior prioridade recebem um *quantum* maior, e logo podem executar por mais tempo sem serem interrompidos.

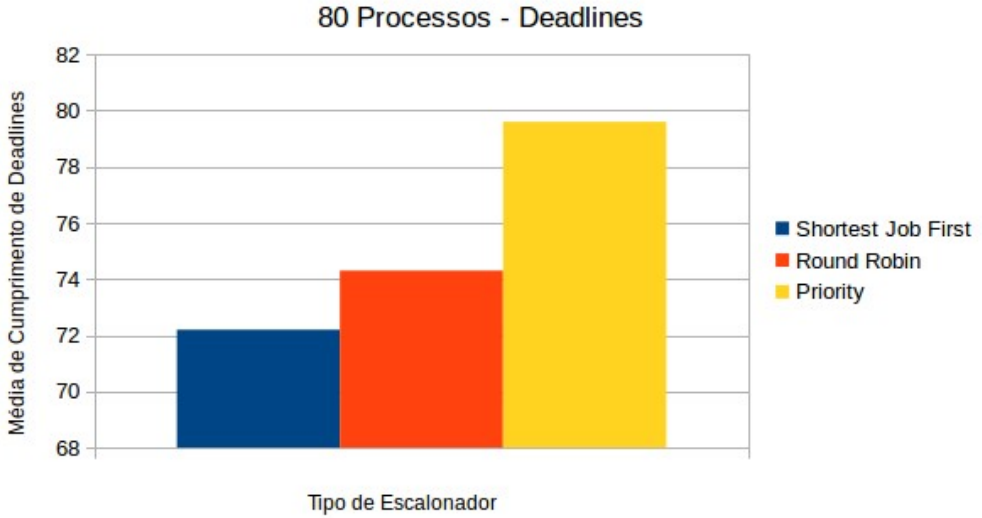
Alguns aspectos influenciam na prioridade de um processo, como tempo restante de execução, proximidade à deadline e quão justa a deadline do processo é em relação ao seu tempo de execução real.

Cumprimento de Deadlines



Os testes tiveram resultados semelhantes em ambas as máquinas, sendo uma 32-bits *dual core* e outra 64-btis *quad core.*





Mudanças de Contexto

Os testes tiveram resultados semelhantes em ambas as máquinas, sendo uma 32-bits *dual core* e outra 64-btis *quad core.*