Sistemas Operacionais I - INE5412

Relatório - Trabalho 1

Lottery Scheduling

Universidade Federal de Santa Catarina

Felipe de Campos Santos [17200441]

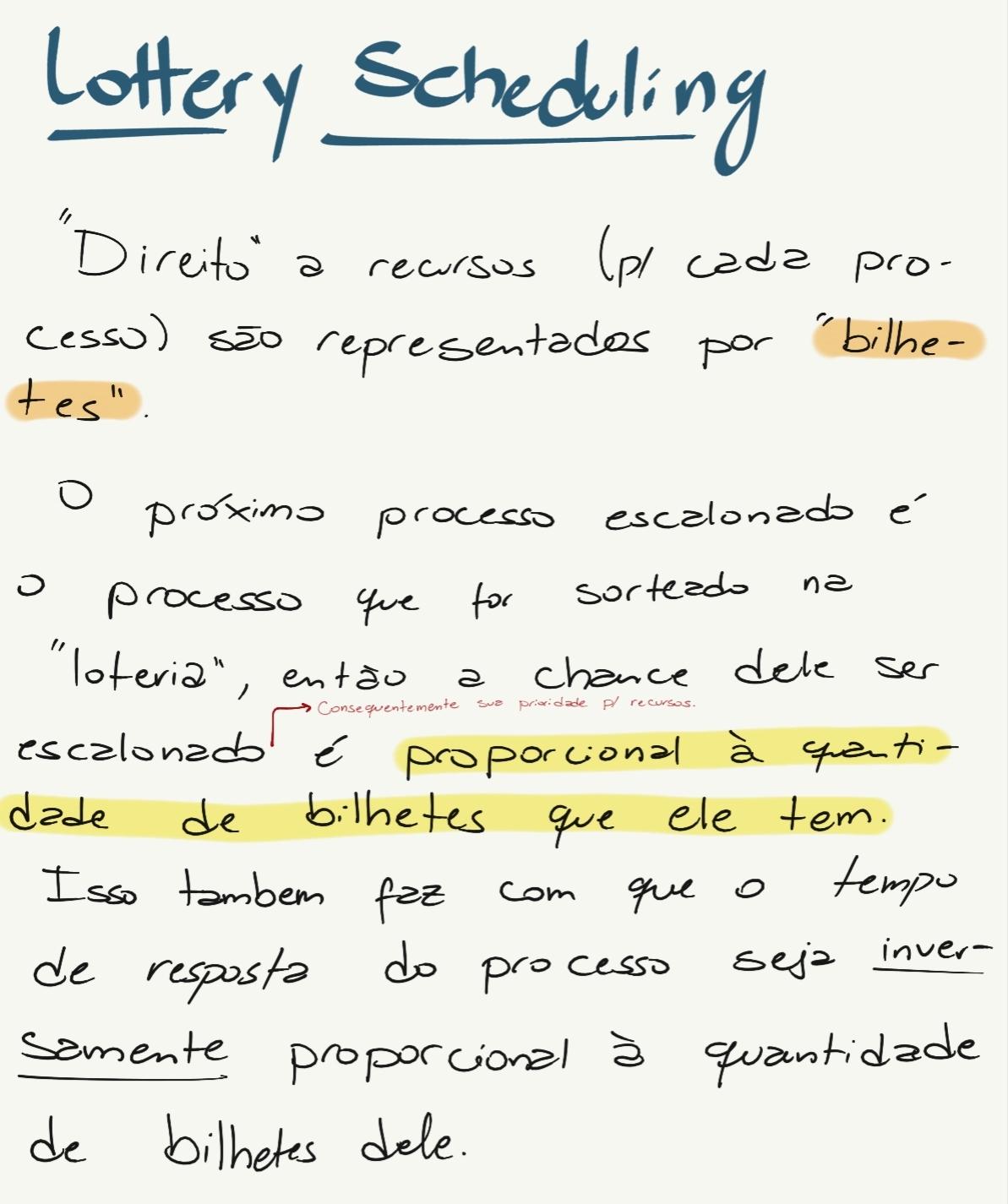
Thiago Martendal Salvador [16104594}

06 de dezembro de 2021.

Esse relatório tem como objetivo demonstrar como foi feita a implementação do Lottery Scheduling como escalonador padrão para o nanvix.

Primeiro, foi feita a leitura do artigo original [1] para entender o funcionamento teórico por trás do escalonador por Lottery Scheduling.

Durante a leitura, foram feitas as anotações abaixo, permitindo uma maior clareza para consultas durante a implementação.

****

Após o entendimento da teoria por trás do escalonador, pudemos começar a implementação.

Primeiro, deveriamos encontrar no código-fonte do nanvix, aonde eram definidos os processos, onde eram criados e onde eram escalonados.

Pudemos encontrar os respectivos arquivos:

* **pm.h** (include > nanvix) - Arquivo onde é feita a definição da estrutura dos processos. Aqui, foi alterado o quantum para 10ms (explicado mais a frente) e foi adicionado à estrutura do processo a propriedade “tickets”, que contem a quantidade de tickets atribuidos àquele processo.
* **pm.c** (src > kernel > pm) - Arquivo onde é feita a criação da variável global ntickets, que guarda o número total de tickets na loteria do sistema
* **fork.c** (src > kernel > sys) - Arquivo onde são adicionados processos à tabela de processos, de maneira que possam ocupar quantums na CPU. Nesse momento, adicionamos 1 ticket ao processo, pois esse passa a estar apto a participar na loteria. Também somamos 1 ao número global de tickets
* **sched.c** (src > kernel > pm) - Arquivo responsável por escalonar processos, onde foram feitas as maiores mudanças. Adicionado método que calcula um número pseudo-aleatorio (usando Gerador congruente linear), faz o sorteio da loteria, escalona um processo e faz o gerenciamento de tickets (adiçao e compensação)
* **sleep.c** (src > kernel > pm) - Arquivo responsável pela gerência de processos que foram interrompidos.
* **die.c** (src > kernel > pm) - Arquivo responsável por terminar os processos, onde é feita a remoção dos tickets do total do sistema.

Abaixo, uma explicação mais detalhada das mudanças comentadas acima.

Primeiro, tinhamos que preparar a “estrutura” de cada processo para que pudessem participar da loteria, ou seja, criar um atributo que guarde o número de tickets que o processo está “segurando”. Isso foi feito em *pm.h*, onde adicionamos

| int tickets; |
| --- |

ao *struct process.*

Além de alterar a estrutura do processo em si, precisamos de um lugar que guarde o número total de tickets dentro do sistema. Isso também foi feito no arquivo *pm.h*, onde criamos uma variável externa chamada ntickets:

| EXTERN unsigned ntickets; |
| --- |

Pronto. Agora nossos processos podem segurar tickets, e podemos contar quantos tickets totais temos no sistema.

Ainda nesse arquivo, vamos alterar o quantum de 50, valor original do nanvix, para 10, **valor proposto no artigo original do Lottery Scheduling**.

| #define PROC\_QUANTUM 10 |
| --- |

Com isso, agora temos que começar a atribuição de tickets.

No arquivo *pm.c*, inicializamos o número total de tickets como 0:

| PUBLIC unsigned ntickets = 0; |
| --- |

Os processos filho são criados no arquivo *fork.c*, onde é feita a procura por um “espaço” livre (um processo não usado) na tabela de processos, e atribuido à ele as informaçoes necessária para que possa ser escalonado. Aqui, adicionamos a atribuição de 1 ticket à esse processo:

| proc->tickets = 1; |
| --- |

logo abaixo, incrementamos o número de tickets no sistema:

| ntickets++; |
| --- |

Com isso, já seria possível realizarmos a loteria. Seria necessário apenas selecionar um número entre 1 e ntickets no escalonamento e escolher o processo que estivesse segurando esse ticket. Porém o Lottery Scheduling conta com artimanhas para tornar esse processo mais justo, principalmente para aqueles processos que já começaram a rodar mais ainda não terminaram.

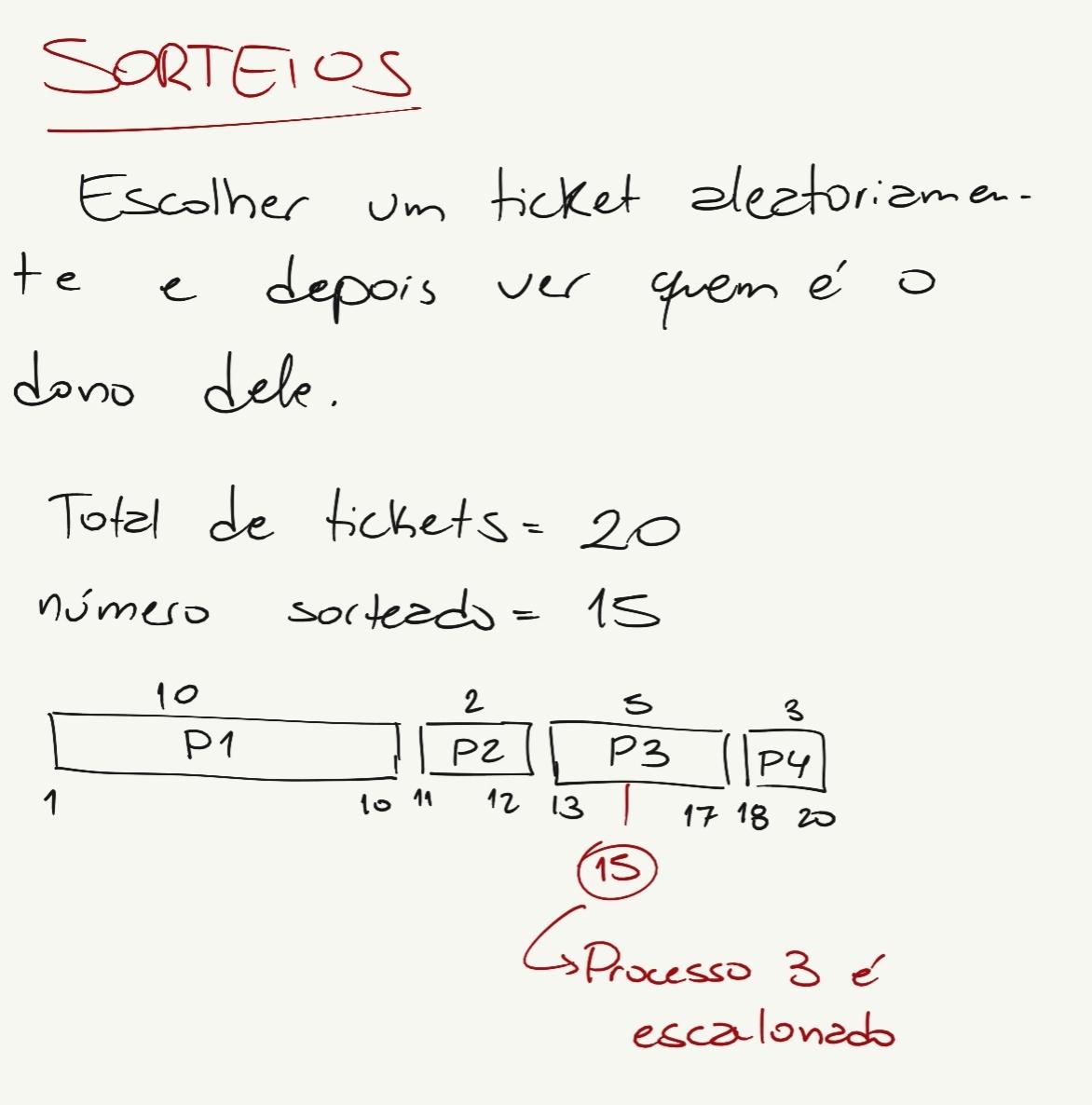
observação: como todos os processos ja começam com um ticket, nesse ponto já foi solucionado o problema de starvation.

Agora podemos começar com o escalonamento de fato.

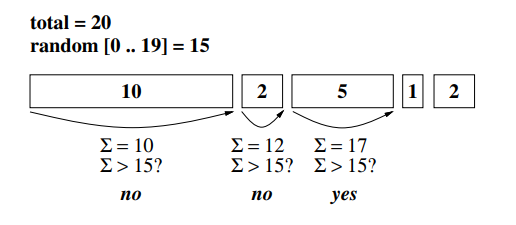
No arquivo *sched.c*, a função *yeld()* é responsável por escolher o próximo processo a ocupar a CPU. Então é nela que realizaremos o sorteio do ticket.

Primeiro, devemos escolher um número entre os possíveis para o ticket. Esse número é um número pseudo-aleatório gerado no mesmo arquivo, pela função newrand(), que usa um gerador linear congruente [2], e com esse número faremos o módulo com ntickets, assim garantimos que o número sorteado existe dentre os tickets.

O sorteio é feito de maneira simples. Percorremos a lista de todos os processos do sistema, pulando aqueles que não estejam em estado pronto (PROC\_READY). Fazemos então um somatório de quantos tickets já “colhemos” até agora, e continuamos até que esse somatório seja maior que número sorteado. Com isso, sabemos que o processo que acabamos de colher foi o selecionado pelo sorteio. A imagem abaixo torna essa explicação mais visual:

****

Essa lógica foi implementada seguindo a indicação do artigo original:



Agora faltam apenas duas coisas: retirar todos os tickets do total quando um processo acaba, e realizar a compensação de tickets quando ele para sem ter usado todo seu quantum.

Para a compensação, alteraremos a função *sleep()* em *sleep.c*.

Na estrutura do processo, temos o atributo *counter*, que mantém uma contagem do runtime faltante do processo, ou seja, quanto tempo de quantum ele tem **sobrando**.

É feito o cálculo da compensação usando o *counter* do processo e o PROC\_QUANTUM, que é o “todo” que o processo pode usar. É adicionada essa compensação aos tickets do processo e também aos tickets totais do sistema.

Traduzimos isso para:

| int f = (int)(PROC\_QUANTUM/(curr\_proc->counter)); int comp = (f-1)\*(curr\_proc->tickets); ntickets += comp; curr\_proc->tickets += comp; |
| --- |

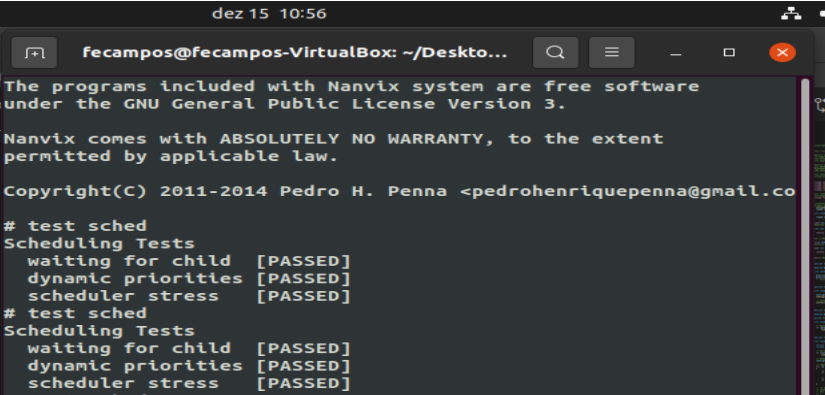
Já a subtração do total de tickets do sistema é feita no arquivo *die.c*.

Nesse caso, apenas realizamos a subtração do número de tickets que esse processo tinha do número de tickets totais do sistema:

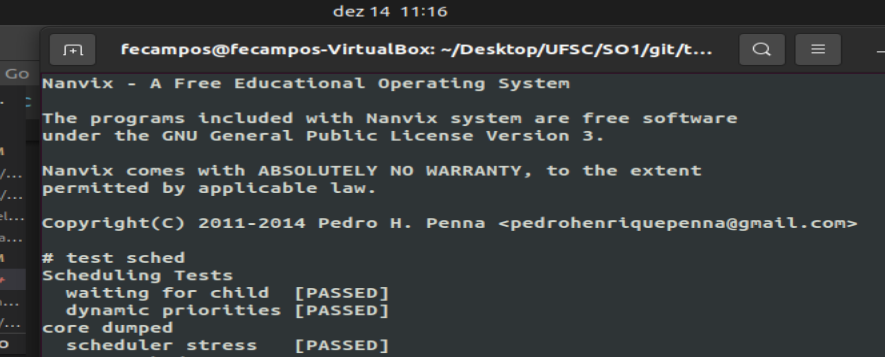
| ntickets -= curr\_proc->tickets; |
| --- |

* O quantum escolhido para esse sistema foi de 10ms, pois foi o proposto no trabalho original e optamos por manter.
* O gerador de números pseudo-aleatórios foi retirado do site rosettacode.org. Optamos por essa solução por não arriscarmos uma implementação problemática, e pois existirem diversas implementações boas do LCG na internet, como essa que aproveitamos. Para esse gerador, foram mantidas as constantes multiplicador, incrementado e modulo.
* Link do vídeo: <https://youtu.be/NTyKWUY0GH4>

Abaixo, um print do sucesso nos testes de scheduling do nanvix:



Obs: na nossa primeira implementação, nosso sorteio era realizado escolhendo um processo p assim que sum >**=** r. Isso nos trazia o problema de *core dumped* nos testes. Após rever o artigo e consertar, fazendo com que p seja escolhido quando sum **>** r, o problema foi resolvido. Abaixo, um print do que acontecia antes de arrumarmos:



[1] Lottery Scheduling: Flexible Proportional-Share Resource Management <https://www.usenix.org/legacy/publications/library/proceedings/osdi/full\_papers/waldspurger.pdf>. Acesso em 06 de dezembro de 2021.

[2] Linear Congruential Generator <<https://rosettacode.org/wiki/Linear_congruential_generator#C>>. Acesso em 06 de dezembro de 2021.