Relatório escalonador

Membros do grupo: Gabriel Madeira e Juliana Pinheiro

A respeito do funcionamento do código:

Aparentemente, de acordo com todos os testes feitos, o interpretador/escalonador está com seu funcionamento completo onde é possível escalonar de acordo com todas as especificações dadas no enunciado, Round-Robin, Real Time, CPU Bound, I/O Bound, lembrando da prioridade de Real Time.

A respeito de como funciona o código:

Interpretador:

Como requisitado na explicação do trabalho, o interpretador abre o arquivo exec.txt contendo os programas a serem escalonados, um por linha, e os envia um por um para o escalonador. Além de enviá-los, em relação aos processos Real Time, também é feita uma verificação a respeito dos tempos.

Essa verificação é feita de tal forma que os aloca em uma fila temporária para que seja possível analisar se os tempos se sobrepõem e, se não existir nenhum conflito, são enviados normalmente. Também é verificado se o tempo de início somado com o tempo inicial resulta em um tempo maior que 60 segundos, o que não é permitido tendo em vista que é executado um programa por minuto.

Como a mesma struct de queue_node é utilizada para os dois tipos de processos, a maneira de diferenciá-los é no valor do tempo de início e tempo de duração onde os processos Round-Robin possuem -1 nesses atributos enquanto os processos Real Time possuem o valor indicado no arquivo exec.txt

A forma de comunicação entre os processos do interpretador e escalonador escolhida foi a memória compartilhada.

Escalonador:

O escalonador consiste de um loop infinito que segue verificando algumas condições para decidir quem escalonar.

A organização dos processos foi feita em três filas, uma para processos Real Time, uma para processos Round-Robin e uma fila de espera para aqueles que forem I/O Bound. As filas de Round-Robin e Real Time foram organizadas de acordo com a ordem que foram recebidos os processos, então encontram-se todos na ordem correta do arquivo.

Para processo Round-Robin: É removido o primeiro da fila e executado durante a fatia de tempo definida, 1 segundo, e então ele pode ter 2 destinos, se for CPU Bound vai direto para o final da fila de prontos (que no código é a fila de processos Round-Robin) se for I/O Bound (definido através de um sinal enviado pelo programa em execução para o escalonador) é enviado para a fila de espera e fica nela até que a sua operação I/O Bound termine.
Ressaltando que como o tempo de uma operação I/O pode variar muito em processos em cenários reais e para o trabalho foi explicado que os processos de teste apenas enviariam um sinal para o escalonador para sinalizar que são I/O Bound e depois seguiriam como um loop infinito, no código foi utilizado um valor de 3 segundos para simular a operação I/O de programas Round-Robin. Então, após 3 segundos, os processos Round-Robin I/O Bound retornariam para o final da fila de prontos.

Para processos Real Time: Como estes possuem uma prioridade acima dos processos Round-Robin, o escalonador segue sempre verificando se o tempo de início do primeiro processo da fila de prontos de Real Time foi atingido para que caso este momento chegue, ele seja escalonado antes que um processo Round-Robin. Após o escalonamento de um processo Real Time, este possui apenas um destino já que apenas processos Round-Robin podem ser I/O Bound, como ele é CPU Bound será executado durante a sua duração definida e ao chegar no fim desta, é enviado para o final da sua fila de prontos. Vale lembrar que, para ter 1 processo Real Time por minuto sendo executado, como especificado nos requisitos do trabalho, o tempo de início do escalonador é reiniciado ao fim da execução de um processo Real Time para que as contas em relação ao tempo de início dos outros processos Real Time esteja correta.

Uma importante decisão a respeito do escalonamento entre Real Time e I/O Bound: Surgiu uma dúvida ao longo do desenvolvimento do código onde o que aconteceria caso o tempo de um processo na fila de espera acabasse durante a execução de um processo Real Time, uma vez que este conflito seria inevitável.
As únicas duas abordagens seriam manter o processo Real Time executando até o final e após sua conclusão retornar com o processo da fila de espera para sua respectiva fila de pronto ou, durante a execução de um processo ao invés de apenas verificar se o seu tempo de duração chegou ao fim, também verificar se o tempo de algum processo na fila de

espera chegou ao fim para que ao longo da execução do processo Real Time, o processo na fila de espera pudesse retornar para sua fila de pronto. Tendo em vista que, para este trabalho, os processos Real Time possuem a maior prioridade, foi escolhido que o processo em espera apenas retorne para sua fila de pronto assim que o processo Real Time terminasse a execução, uma vez que sua prioridade é maior então não deveria ser atrapalhada. Entretanto é possível implementar a outra abordagem também.

Processos I/O Bound:

A forma como foi desenvolvido o tratamento destes processos foi utilizado uma variável global e o handler do sinal enviado pelo processo sendo executado.
A variável global pode ser 1 ou 0, onde 1 representa que o programa é I/O Bound e que este receberá o tratamento explicado anteriormente da fila de espera

A respeito dos testes realizados:

Foram utilizados 5 arquivos para realizar os testes: 3 Round-Robin, onde dentre eles 2 são CPU Bound e 1 I/O Bound e 2 Real Time, onde os dois são CPU Bound

As especificações dos arquivos são:

test1.c - Round-Robin CPU Bound
test2.c - Round-Robin I/O Bound
test3.c - Real Time I=0 D=5 CPU Bound
test4.c - Real Time I=1 D=10 CPU Bound
test5.c - Round-Robin CPU Bound

Escalonamento:

O escalonamento dos processos descritos anteriormente está explícito ao longo da execução do código, porém o esperado é que: Os programas teste1, test2 e test5, nesta ordem, são posicionados na fila de prontos de Round-Robin e os programas test4 e test5 na fila de prontos de Real Time, nesta ordem também. Até que o tempo do primeiro processo Real Time (test3) seja atingido, seguirá ocorrendo o escalonamento dos 3 Round-Robin existentes onde test1 e test5 ficarão sendo alternados e o test2 também será alternado porém por ser I/O Bound terá um intervalo de 3 segundos ao invés de apenas 1.

No momento que o tempo de test3 for atingido, o mesmo será iniciado e finalizado após seus 5 segundos de duração. Vale lembrar que como o test3 inicia no exato minuto (1:00), o test2 (Round-Robin I/O Bound) estará em espera e terminaria seu tempo nesta fila durante a execução de test3 e, de acordo com o que foi explicado anteriormente, só retornará para sua fila de prontos assim que o test3 terminar sua execução. A partir do momento que o test3 terminar, começará a ser contado o tempo até o início do test4 que como também é CPU Bound será apenas executado pelo seu tempo de duração definido e retornar ao final de sua fila de pronto. E é dessa forma que o escalonador seguirá.

A respeito de como compilar os programas:

Para o programa principal contendo o interpretador e escalonador basta o comando pelo terminal utilizando o gcc: **gcc -Wall -o main_out main.c**.
Para os programas de teste, todos foram nomeados com o mesmo nome de seus arquivos porém apenas removendo o .c, então **test1.c** vira **test1** apenas, tendo em vista essa simplicidade foi feito um simples script em bash para que compilasse todos de uma vez com o comando **gcc -o test# test#.c** onde seria apenas substituído o # pelo número correto do programa. O script também será enviado junto de todos os outros arquivos caso deseje utilizar, o nome é **compile_test_programs** e para utilizá-lo basta dar permissão de execução com **chmod +x compile_test_programs** e executá-lo com **./compile_test_programs**

Além dos arquivos de teste ditos anteriormente também foram testados alguns tempos de início e duração que colidissem para testar as exceções desejadas.
Alguns destes tempos foram I=20 D=50 -> Não pode ser executado já que I + D >= 60;
Programa1 com I=0 e D=5, programa2 com I=10 e D=10 e programa3 com I=7 e D=20 -> programa3 não poderia ser escalonado já que seu tempo colide com o programa2.

Observação: Foi sugerido que o programa parasse após 2 minutos de execução, porém como existe mais de um processo real time que, pela restrição de um processo por minuto, não teria sua execução completa com apenas 2 minutos, foi programado para terminar o programa com 3 minutos de execução.