Especiaria do deserto*

*Relatório do trabalho final de Programação Concorrente, 2021/1, Eduardo A. P. Alchieri

1st Lucas de Almeida Bandeira Macedo
Departamento de Ciência da Computação (CiC)
Universidade de Brasília (UnB)
Brasília, Brasil
lucasabmacedo@hotmail.com

***Resumo*—Este é o relatório para o projeto final da matéria Programação Concorrente, 2021/1, Universidade de Brasília. Aqui, serão explicados o problema idealizado e suas dificuldades, assim como a solução encontrada, sua execução e os conhecimentos por ela exigidos.**

I. Introdução

Este projeto foi realizado com o objetivo de estudar o conteúdo ministrado nas aulas de Programação Concorrente.

O projeto, assim como a maioria dos problemas envolvidos na área, envolve sincronização de processos e memória compartilhada. A proposta do projeto é o desenvolvimento de um problema, contextualizado de forma livre, que possa ser resolvido com programação concorrente, e que exija (ao menos) dois mecanismos de sincronização e uma memória compartilhada entre processos.

Este relatório será dividido em três seções:

1) Seção II: A formalização do problema desenvolvido, apresentando sua contextualização, assim como uma análise técnica dos problemas de sincronização e memória compartilhada que a solução desenvolvida deve resolver.
2) Seção III: A solução, citando o ambiente desenvolvido, explicando os mecanismos de sincronização utilizados e a explicação detalhada de cada função codificada, assim como o fluxo de execução do algoritmo proposto.
3) Seção IV Por fim, concluindo com os aprendizados adquiridos tanto pelo curso quanto pelo projeto final, e uma última análise sobre possíveis aplicações do problema proposto.

II. O problema

A contextualização do problema é a seguinte:

”No planeta deserto de Arrakis, a Especiaria é a substância mais preciosa, não há uma pessoa que não a deseje. Por isso, o contrabando é forte e organizado. Para extrair a Especiaria da areia, é necessário um contrabandista e um equipamento de extração. Chamaremos o contrabandista encarregado da extração de 'extrator'. Cada equipamento extrai da areia determinado volume de Especiaria e armazena em si mesmo, como se fosse uma seringa. Enquanto um equipamento está cheio, não é possível usá-lo para extrair mais Especiaria.

O processo de esvaziar os equipamentos também exige mão de obra. É necessário que cada pessoa, que esteja encarregada dessa tarefa, pegue um equipamento de extração cheio e o esvazie dentro de um grande recipiente, que será lacrado quando estiver cheio. Chamaremos o contrabandista encarregado de esvaziar os equipamentos de manutentor.

De tempos em tempos alguém virá buscar o recipiente, e só o levará quando ele estiver lacrado. Tempos depois da saída, após algum tempo, essa pessoa voltará com um novo recipiente vazio e irá embora, retornando após determinado período de tempo. Só pode existir um recipiente por vez.

Como Arrakis é um planeta muito quente e seco, de tempos em tempos os trabalhadores precisarão parar para descansar e beber água, para não desidratar ou desmaiar.”

O problema apresentado, ambientado no planeta desértico de Duna [1], ilustra uma fila de produção, em que a ”especiaria” é extraída pelos ”extratores”, armazenada pelos ”manutentores”, e despachada pelos ”viajantes”.

Todos os ”contrabandistas” aqui dependem diretamente uns dos outros. Se um grupo parar ou atrasar, ninguém mais conseguirá continuar o trabalho. Isso apenas reforça a extrema necessidade de organização que eles terão que assumir, ou, em termos de concorrência, isso apenas reforça a delicada sincronização que terá que ser codificada.

Para melhorar o entendimento técnico do problema, aqui estão algumas regras a serem seguidas:

1) Caso um extrator fique sem equipamento, ele esperará até que um equipamento seja liberado.
2) Caso um manutentor não tenha mais equipamentos para esvaziar, ele esperará até que mais um equipamento necessite de sua mão de obra.
3) Caso não haja recipientes disponíveis para o manutentor, ele esperará até que mais um recipiente apareça.
4) O critério de desidratação de um contrabandista é baseado no número de ações que ele toma. Após x ações (ou seja, encher ou esvaziar o equipamento x vezes), ele parará. O viajante não precisa descansar.

A. Produtor, produtor&consumidor e consumidor

Como já definido anteriormente, o problema contextualiza uma fila de produção, podemos até recontextualizá-lo em uma esteira de fábrica, sem muitos problemas ou perdas. Assim,

como em todas as filas de produção, podemos classificar os agentes em produtores e consumidores - que ”coincidentemente” são termos muito conhecidos na área da paralelização computacional.

Os *extratores* podem ser classificados como produtores pois, o ato de extrair a especiaria da areia, e armazená-la no equipamento, pode ser traduzida no ato de produzir um dado e armazená-lo no buffer, para que os consumidores leiam.

Os *viajantes* podem ser vistos como consumidores, já que sua única tarefa é esperar que o ”recipiente” encha, para que possam levá-lo embora. Podemos relacionar com o ato de esperar um buffer encher, para consumir todos os dados de uma vez.

Os *manutentores* não ficaram por último à toa. Eles são o grupo mais interessante deste problema, pois agem tanto como produtores como consumidores. Podemos enxergar isso estudando as duas analises feitas anteriormente: para o extrator, o manutentor consome os dados que ele produz. E, no ponto de vista do viajante, o manutentor está produzindo dados para que ele os consuma. O manutentor se torna o nó que une as duas pontas, sendo possivelmente o grupo / o processo mais importante presente.

B. Exclusão mútua

No meio de tudo, há a necessidade de exclusão mútua no acesso a uma memória compartilhada, ou seja, dois processos não podem acessar a mesma área de memória ao mesmo tempo.

A exclusão mútua é necessária em dois pontos específicos do problema: entre o extrator e o manutentor, e entre o viajante e o manutentor.

Entre o extrator e o manutentor, essa necessidade mora junto dos equipamentos, que são compartilhados entre os dois grupos. Como existe um número limitado de equipamentos, e ambos podem tentar acessar estes equipamentos simultaneamente, é necessário implementar uma região de exclusão mútua.

Entre o viajante e o manutentor a lógica é bem parecida. Os manutentores acessam o recipiente para enche-lo, enquanto os viajantes esperam que ele encha para leva-lo embora. Não podemos correr o risco de um encher ao mesmo tempo que o outro leve embora, então também temos que garantir que acessarão o recipiente um de cada vez.

III. Solução e Algoritmo

O algoritmo foi realizado na linguagem de programação C [4], com as bibliotecas pthread [2] e semaphores [3]. Todo o algoritmo realizado aqui foi pensado com as ferramentas e mecanismos presentes nessas bibliotecas, portanto, é possível que uma tradução para outro ambiente exija algumas mudanças.

A. Mecanismos de Sincronização

No algoritmo desenvolvido para este problema, foram usados locks, variáveis de condição e semáforos.

Temos dois semáforos, um que indica o número de equipamentos cheios, e um que indica o número de equipamentos vazios. Eles funcionam em conjunto. O semáforo de equipamentos vazios é decrementado pelo grupo dos extratores, para indicar que pegaram um equipamento vazio, e incrementado pelo grupo dos manutentores, para indicar que esvaziaram um equipamento cheio. O semáforo de equipamentos cheios é exatamente a mesma coisa, mas invertido: decrementado pelos manutentores, para indicar que esvaziaram um equipamento cheio, e incrementado pelos extratores, para indicar que acabaram de encher um equipamento antes vazio.

Temos, também, duas variáveis de condição: um para o viajante, e o outro para os manutentores. Ambas orbitam o mesmo propósito: caso o recipiente esteja indisponível para este grupo, então eles dormem. Como toda variável de condição, ela funciona junto com um lock, que é o lock de acesso ao recipiente (sua lógica foi explicada na seção II-B). Assim, no ponto de vista do viajante, se o recipiente ainda não está cheio, ele dorme e espera que os manutentores o encham, e, quando encher, o acordem. Com os manutentores é similar. Enquanto o recipiente estiver cheio ou indisponível, eles dormem e esperam que o viajante os acorde, quando ele disponibilizar um recipiente vazio novamente.

A thread principal, ou a função main, servirá apenas para criar as threads correspondentes aos contrabandistas e inicializar o que for necessário. Assim, o semáforo de equipamentos vazios será inicializado com o número de permissões equivalente ao número total de equipamentos disponíveis para o problema, enquanto o semáforo de equipamentos cheios vai ser inicializado com 0 permissões, pois ninguém trabalhou ainda, então nenhuma especiaria foi extraída. Após as inicializações, a main irá criar todas as threads necessarias e aguardar que encerrem (com um join), mas nenhuma encerrará, pois rodarão em loops infinitos.

Além da main, temos três funções neste projeto, correspondente a cada um dos grupos de contrabandistas: ”extrair”, correspondente aos extratores, ”manutenir”, correspondente aos manutentores, e ”recipiente”, correspondente aos viajantes (que ficam a cargo de transportar o recipiente).

B. Extrair

A função correspondente ao grupo dos extratores é a mais simples de todo o algoritmo. Primeira coisa que ela faz é decrementar o semáforo de equipamentos vazios. Depois de simular a extração da especiaria, ou melhor, da produção de dados para o buffer, a função incrementa o semáforo de equipamentos cheios.

Posteriormente, ela incrementará o contador de ações tomadas, que é inicializado antes da função verdadeiramente começar. É, então, feito um teste se já está na hora deste trabalhador descansar (relembrando que, na contextualização, ele deve descansar a cada X ações tomadas, para se rehidratar). Caso ele já tenha cansado, então simula um descanso e zera este contador. Vale ressaltar que não há necessidade de exclusão mútua neste contador, já que cada trabalhador conta suas ações tomadas independentemente.

Por fim, a função volta para o semáforo de equipamentos vazios, pois ela roda em um loop infinito, para que nunca termine.

C. Manutenir

A função de manutenir é bem semelhante a de extrair, mas com uma mudança na ação que o trabalhador toma. Primeira coisa que ela faz é decrementar o semáforo de equipamentos cheios. Em seguida, o manutentor tomará posse do lock do recipiente, que cria uma região de exclusão mútua para a capacidade atual do recipiente. Em seguida, ele checará se o recipiente está cheio: caso esteja, ele mandará um sinal para acordar o viajante (que acordará caso esteja esperando que o recipiente encha) e dormirá (esperando que o viajante o acorde); caso não esteja cheio, ele continuará o fluxo normal de trabalho, esvaziando o equipamento no recipiente (incrementando sua capacidade), liberando o lock de capacidade e incrementando o semáforo de equipamentos vazios.

Em seguida, da mesma forma que os extratores, a função checará se o manutentor está cansado. Caso esteja, ele descansará. A função nunca finalizará, pois também está encapsulada por um loop infinito, assim como a função do extrator (seção III-B).

D. Recipiente

A terceira e última função implementada para este algoritmo diz respeito ao viajante, responsável por levar recipientes cheios e trazer recipientes vazios. Primeira ação que ela vai fazer é tomar posse do lock do recipiente, para poder checar corretamente se o recipiente já está cheio. Não estando cheio, o viajante dormirá, enquanto espera por um manutentor que o acorde quando estiver cheio. Quando o recipiente estiver vazio, ele acordará, liberando o lock do recipiente e prosseguindo para o resto da função.

Em seguida, a função simulará sua viagem, pegando o recipiente cheio e trazendo um novo. Finalmente, quando o viajante chegar com um novo recipiente vazio, ele novamente assumirá o controle do lock, reiniciará o contador de capacidade do recipiente, acordará todos os manutentores que estão dormindo e liberará o lock novamente. Em seguida, irá embora, apenas para retornar posteriormente, no começo da função (pois, como de costume, está em um loop infinito).

E. Fluxo de execução

Aqui faremos uma análise do fluxo de execução do algoritmo, assim como suas possibilidades de comportamento, dada a natureza imprevisível de um programa concorrente.

Vale ressaltar que o fato de os trabalhadores ”pararem para descansar e se hidratar” não impacta nessa análise, já que eles não exigem nenhum mecanismo de sincronização. Portanto, não será considerado na análise.

1) Inicio: Assim como indicado na contextualização do problema, a fila de produção começa pelo extrator. Todos os equipamentos começam vazios, assim como o recipiente, então ele é o único que pode fazer alguma ação. Neste momento, o viajante ainda não chegou e, se chegar antes de qualquer ação tomada, ele vê que o recipiente ainda não está cheio e dorme, e os manutentores aguardam no semáforo de equipamentos cheios, que foi inicializado com 0 permissões.

Assim, os extratores começam suas ações, pegando equipamentos vazios e enchendo-os. O número máximo de extratores que podem extrair especiaria simultaneamente é o número mínimo entre o número de equipamentos disponíveis e o número de extratores disponíveis.

2) Pós-extração: Com um equipamento cheio disponível, o leque de caminhos possíveis se expande. O manutentor agora consegue agir, pegando um equipamento cheio e colocando-o no recipiente. É impossível que dois manutentores tentem encher o recipiente ao mesmo tempo, já que a ação está encapsulada por um lock, garantido a exclusão mútua. Também é impossível, pelo mesmo motivo, que o manutentor tente colocar mais especiaria em um recipiente cheio. Caso ele encontre, com um equipamento em mãos, um recipiente cheio, então ele dormirá e liberará o lock.

Assim como com os extratores, o número máximo de manutentores trabalhando simultaneamente é o mínimo entre o número de equipamentos cheios disponíveis e o número de manutentores disponíveis.

Com o extrator, temos apenas duas opções: há ou não há equipamentos disponíveis. Se houver, ele simplesmente continuará seu trabalho. Caso não haja, ele esperará no semáforo de equipamentos vazios, e continuará seu trabalho assim que mais um manutentor terminar o dele.

3) Pós-armazenamento: Com ao menos um manutentor terminando a sua ação, mais uma vez os caminhos se bifurcam. Para o caso do recipiente estar cheio, temos duas opções: o viajante ainda não chegou, então nada acontecerá ao recipiente até que ele chegue (e ele não dormirá, pois só dorme se ele ver o recipiente não-cheio, portanto, não há risco de deadlock aqui), ou o viajante já tinha chegado e estava dormindo, e neste caso ele será acordado pelo primeiro manutentor que ver o recipiente cheio.

Independentemente do que acontecer com o viajante, o manutentor que ver o recipiente cheio sempre mandará um sinal para acordar o viajante e dormirá.

Caso o recipiente não esteja cheio, o manutentor simplesmente esvaziará o equipamento no recipiente e continuará seu trabalho.

4) O retorno do recipiente: Finalmente, a última etapa do algoritmo é quando o viajante devolve um recipiente vazio. Durante a devolução, ele toma posse do lock do recipiente, reseta o contador de capacidade do recipiente e acorda todos os manutentores dormindo (broadcast). Como ele tem posse do mesmo lock que todos os manutentores estavam em posse antes de dormir, então não há chance de um manutentor acidentalmente dormir enquanto o viajante executa o broadcast. Caso não tenha nenhum manutentor dormindo, eles não dormirão após a volta do recipiente vazio, pois eles não encontrarão um recipiente cheio (que é a condição usada para que eles durmam).

IV. Conclusões

Existe claramente uma divisão na mente de um estudante de ciência da computação: antes de aprender programação concorrente e depois. É estranho imaginar como era possível programar sem esse conhecimento, depois de perceber que isso está em literalmente todo o lugar. Essa simples fila de produção, proposta neste relatório, é o menor dos exemplos. Sistemas operacionais, aplicativos, sistemas gerenciadores de banco de dados (SGBD's)... são inúmeros os exemplos.

O problema proposto parece ser apenas um pequeno exercício, quase nos moldes de um problema de programação competitiva, mas esse mesmo conceito pode ser recontextualizado em tantos outros problemas: podemos dizer que os extratores são inputs dos usuários, os manutentores um arquivo de texto e o viajante a rede e pronto, temos uma abstração da execução do office online: o "viajante" espera até que determinada quantidade de dados/palavras seja armazenada no buffer, para enviar a atualização para o servidor sem sobrecarrega-lo com atualizações a cada nova letra digitada.

Esta área de estudo é incrivelmente flexível em suas abstrações, e uma mesma lógica pode ser reaplicada em vários outros problemas conhecidos. Essas contextualizações com histórias corriqueiras são um ótimo jeito de entender os conceitos. Afinal, programamos para o programa fazer algo que nós não façamos nós mesmos.

References

[1] Duna, https://en.wikipedia.org/wiki/Dune_(franchise).

[2] pthread.h, https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html.

[3] semaphore.h, https://pubs.opengroup.org/onlinepubs/7908799/xsh/semaphore.h.html.

[4] C language, https://en.wikipedia.org/wiki/C_(programming_language).