UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

SEL0604 – Sistemas Operacionais I
Exercício sobre o problema do Jantar dos Filósofos

Prof. Vanderlei Bonato

Lucas Alves Roris (11913771)

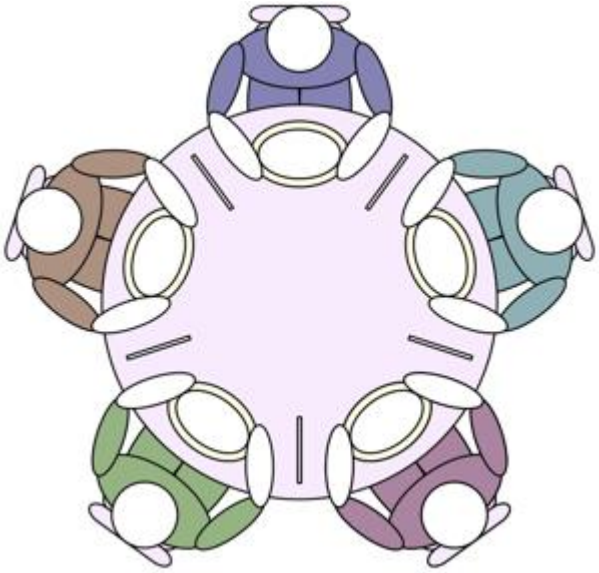
São Carlos
2022

Sumário

Introdução ao problema

O Jantar dos filósofos é um problema criado por Dijkstra em 1965 para exemplificar a dificuldade de sincronização de processos dos programas. Portanto, foram criados diversos algoritmos com diversas abordagens para solucionar esse problema.

O problema é definido a seguir: Cinco filósofos estão sentados em uma mesa redonda para um jantar. Cada filósofo tem um prato com comida à sua frente. Para começar a comer, o filósofo deve pegar dois hashis um de cada lado do prato, porém os hashis da direita são compartilhados com o filósofo a direita, e vice-versa. Após conseguir os talheres, começa a comer por um tempo e volta a pensar, devolvendo os hashis a mesa, possibilitando os filósofos adjacentes a comer.



A premissa do problema é bem simples, porém ao implementar uma solução, é possível que todos os filósofos fiquem travados com um talher esperando o outro terminar de usar o hashi. Ou até que um filósofo fique muito tempo esperando os talheres e acabe “morrendo de fome”.

Podemos traçar um paralelo desse problema com o desafio de sincronização de threads/processos, onde os problemas que podem a ocorrer são de Deadlock e Starvation.

Deadlock é quando um processo aloca algum recurso do computador, mas também está tentando acessar outro recurso que outro processo está utilizando, e caso esse segundo processo estiver esperando o primeiro para continuar sua execução, temos um loop que nunca se quebrará.

Starvation (morrer de fome) é quando o escalonador de processos possui algum tipo de fila de prioridade e um processo pode acabar nunca sendo executado por conta de sua baixa prioridade.

Esses dois problemas podem ser resolvidos de diversas maneiras, com vantagens e desvantagens, porém para esse problema foi implementado o conceito de semáforos e monitores.

O que é um monitor/semáforo?

Um monitor é um mecanismo de alto nível com objetivo de impedir o acesso concorrente inadequado entre duas threads ou processos a um recurso do sistema. Para isso, ele faz com que a segunda thread seja obrigada a esperar o monitor dizer que aquele recurso não está mais sendo utilizado.

Para implementar um monitor, existem diversas maneiras, porém a mais usada é usando semáforos. Um semáforo é basicamente uma informação que indica se um objeto pode ou não ser acessado por um contador atômico.

Quando um thread tenta acessar o recuso, o semáforo é decrementado em 1 e quando termina de usar, incrementa em 1. Porém caso seja igual a zero quando a thread quiser acessar, terá que esperar até o semáforo voltar a sua condição inicial.

Código Explicado

O código pode ser encontrado na mesma pasta desse relatório, ou no link do GitHub: https://github.com/LucasRorisCube/Dining_Philosopher.git .

O código possui uma lógica bem simples. A main começa criando 5 threads, representando 5 filósofos, e espera todos eles terminarem seus processos.

Cada filósofo vai continuar trocando entre os estados de THINKING, HUNGRY e EATING indefinidamente dentro de um loop eterno, portanto para fechar o programa deve-se aplicar o comando Ctrl + c, encerrando todos eles.

```
while(1){
  monitor_StartEat(*i);

  int waiting = -1;
  while(monitor_data.p[*i] != EATING) {

    waiting += 1;
    if(waiting > 5){

      sem_post(&(monitor_data.semParaComer));
      while(!((monitor_data.p[(*i + 4) % 5] != EATING) && (monitor_data.p[*i] == HUNGRY) && (monitor_data.p[(*i + 1) % 5] != EATING)));
      monitor_data.p[*i] = EATING;

    } else {

      sem_wait(&(monitor_data.semParaComer));
      monitor_Verify(*i);

    }
  }

  printf("Philosopher %d is eating!\n", *i + 1);
  delay(rand()%5+1);

  monitor_EndEat(*i);

  printf("Philosopher %d finished eating!\n", *i + 1);
  sem_post(&(monitor_data.semParaComer));
  delay(rand()%11+1);
}

pthread_exit(0);
```

A função principal dos filósofos (função monitor_StartEat()) começa fazendo eles tentarem comer, que define o estado como HUNGRY e tenta comer. Como são 5 filósofos, apenas dois vão conseguir comer no começo, e os outros vão entrar em um loop esperando eles terminarem.

Caso o filósofo esteja esperando a sua vez, ele entra na fila do semáforo esperando os filósofos adjacentes avisarem que terminaram de comer.

Quando um filósofo acaba de comer, envia um sinal para os processos voltarem a ativa e tentarem comer. Enquanto isso o filósofo volta por um estado THINKING por um tempo aleatório.

O fato desse programa não possuir deadlock é o uso do monitor que faz com que caso o filósofo não consiga comer, ao invés de segurar um talher e esperar o outro, ele não pega nenhum e espera um aviso do semáforo.

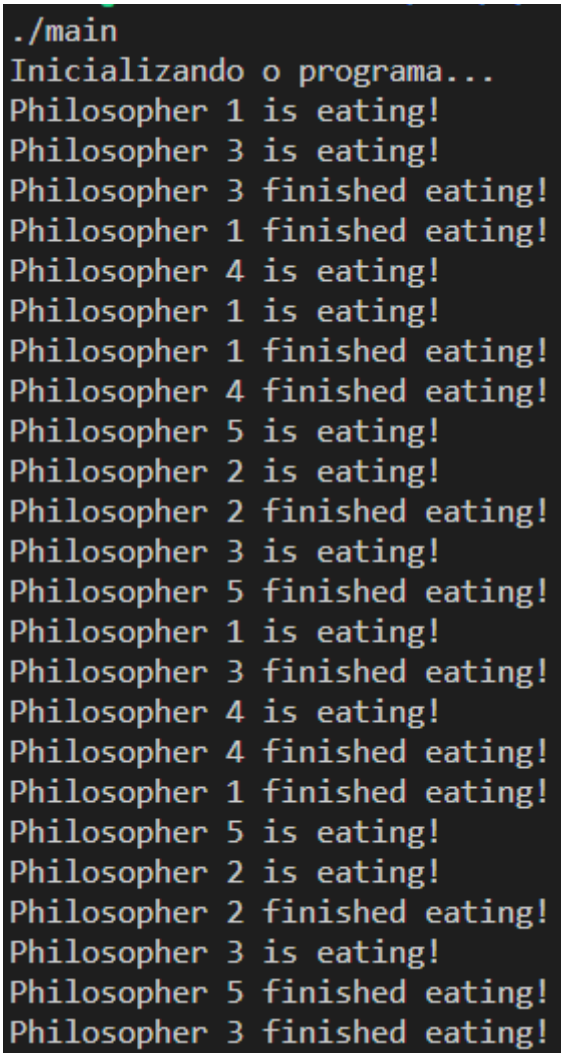
Mesmo que o aviso do semáforo não possibilite ele a comer naquele momento, ele nunca espera mais do que 5 vezes (número total de filósofos) para comer já que está numa fila.

O problema de Starvation teoricamente já está resolvido pelo que foi explicado acima, porém para deixar mais seguro, na função principal dos filósofos, caso ele saia da fila e não consiga comer, uma variável “waiting” é

incrementada, e quando passar de 5, ele força a ser o próxima na fila do semáforo.

Com essa implementação a mais, mesmo que os dois palitos ao seu lado estiverem ocupados, no momento em que um deles é disponibilizado, o filósofo já entra na fila do semáforo para o outro, o que impede de algum outro tentar comer também.

Esse código foi executado por uma noite inteira sem travar em nenhum momento e os filósofos foram escalonados bem uniforme.



Conclusão

O jantar dos filósofos é um problema muito relacionado com o problema de sincronização de processos e threads, portanto resolve-lo acaba implicando em aprender como o sistema operacional usa seus recursos da maneira mais eficiente e segura possível.

Apesar de um problema simples, sua solução não é trivial, tendo em conta que a solução requer diversos componentes do sistema operacional como semáforo, variáveis atômicas, etc.

A solução é bem explicada e simples de entender, e para sintetiza-la foi necessário aprender e aplicar diversos assuntos ensinados na disciplina como monitor, semáforo, threads, processos, variáveis atômicas, etc.

Ao final o código nunca irá travar e nenhum filósofo será privado de comer por muito tempo.