Deadlock

CIn UFPE

STATEMENT | SUBMIT | SUBMISSIONS | STATISTICS

Time Limit
8 seconds

Memory Limit
768 MB

Output Limit
4 MB

A principal função de um certo sistema operacional é gerir o acesso aos recursos do sistema (memória RAM, disco, teclado, mouse, etc.). Nesse sistema, os recursos são usados por vários processos (programas em execução). Se um processo necessita utilizar um determinado recurso, ele deve:

1. Solicitar o recurso
2. Usar o recurso
3. Liberar o recurso

Um mesmo recurso não pode ser utilizado por dois processos diferentes ao mesmo tempo. Caso um recurso `R` já esteja sendo utilizado por um processo `P1`, e seja solicitado por um processo `P2`, este último será forçado a esperar **numa fila** até que o primeiro libere o recurso espontaneamente, ou que o processo seja encerrado de modo forçado pelo sistema operacional a terminar a execução.

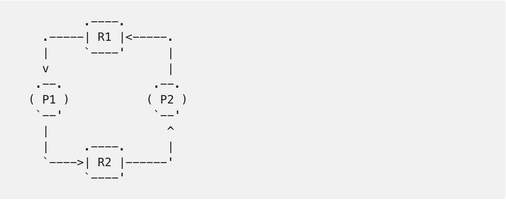
Podemos representar isto através de um grafo dirigido de alocação de recursos do seguinte modo:

1. Processos e recursos são vértices
2. Se um processo `P1` solicita um recurso `R1` e este está disponível, haverá uma aresta saindo de `R1` para `P1`
3. Se um processo `P2` solicita um recurso `R1` e este não está disponível, haverá uma aresta saindo de `P2` para `R1`

Além disso, para que um processo termine a execução de sua tarefa e libere os recursos ultilizados, ele necessitará de todos os recursos solicitados. Porém pode haver problemas nesta alocação de recursos, por exemplo na seguinte situação.

1. O processo `P1` solicita o recurso `R1`, que está livre.
2. O processo `P2` solicita o recurso `R2`, que está livre.
3. O processo `P1` solicita o recurso `R2`, que está ocupado por `P2`.
4. O processo `P2` solicita o recurso `R1`, que está ocupado por `P1`.

Essa situação é ilustrada pelo diagrama a seguir.



Nenhum dos processos consegue terminar a execução pois dependem de outro recurso que apenas outro processo do mesmo grupo pode fornecer, ficando em uma situação de dependencia circular chamada **deadlock**. O sistema operacional é responsável por verificar se um processo `P` está envolvido num deadlock usando o seguinte método.

É realizado um percurso em profundidade a partir do vértice `P`, durante o qual cada vértice recebe uma cor BRANCA, CINZA, ou PRETA. Inicialmente, todos os vértices têm cor BRANCA. Ao "iniciarmos" a visita de um vértice, atribuímos-lhe a cor CINZA. Ao encerrarmos a visita do vértice, isto é, após todos dos seus vizinhos terem sido visitados, atribuímos-lhe a cor PRETA. Durante a visita de um vértice, consideramos cada vizinho: se o vizinho tiver cor BRANCA, continuamos o percurso a partir dele; se o vizinho tiver cor PRETA, ignoramos e passamos para o próximo; e se o vizinho tiver cor CINZA, então o percurso encerra pois um deadlock foi encontrado.

Caso o SO detecte que o processo `P` está envolvido num deadlock, então ele deverá forçar o seu término, liberando todos os recursos que ele utiliza e retirando-o das filas de solicitação de todos os recursos. Cada recurso liberado pelo encerramento do processo `P` deve ser imediatamente alocado ao próximo processo da sua fila de solicitação, se houver algum.

Input Specification

A entrada descreve um cenário de alocação de recursos do SO e é composta por várias linhas, cada uma com um evento numa das formas a seguir.

- `REQ Pi Rj` - o processo com ID `Pi` solicita um recurso com ID `Rj`
- `FRE Pi` - encerra normalmente e libera os recursos alocados pelo processo com um ID `Pi`
- `DLK Pi` - verifica a ocorrência de *deadlock* a partir do processo `Pi` e, caso detecte algum, força o seu término.
- `END` - fim da entrada

Cada processo tem um identificador único (ID) na forma `Pi`, onde `i` é um número inteiro. De forma similar, cada recurso tem um identificador único na forma `Rj`, onde `j` é um inteiro.

Output Specification

Para cada evento da entrada, deverá ser impressa uma saída como descrito a seguir.

- `REQ Pi Rj` - imprime uma linha
  - `AVAIL` , caso o recurso `Rj` esteja disponível e possa ser alocado ao processo `Pi`
  - `WAIT W`, caso o recurso `Rj` esteja em uso e `Pi` seja forçado a esperar. `W` indica o número de processos da fila por `Rj`, incluindo o `Pi`.
- `FRE Pi` - imprime uma linha `TERM F` indicado o número de recursos liberados pelo processo `Pi`
- `DLK Pi` - imprime uma linha
  - `NONE`, caso não seja detectado deadlock
  - `KILL F W`, case seja detectado um deadlock. `F` é um inteiro correspondente ao número de recursos detidos por `Pi` e liberados de forma forçado; e `W` é um inteiro com o número de filas de espera dos quais `Pi` foi removido.

Sample Input #1

```
REQ P0 R3
REQ P1 R5
REQ P2 R3
REQ P3 R7
REQ P4 R1
REQ P2 R8
```

Sample Output #1

```
AVAIL
AVAIL
WAIT 1
AVAIL
AVAIL
AVAIL
```

Sample Input #2

```
REQ P0 R4
REQ P1 R0
REQ P2 R3
REQ P3 R2
REQ P4 R5
REQ P4 R3
```

Sample Output #2

```
AVAIL
AVAIL
AVAIL
AVAIL
AVAIL
WAIT 1
```

Sample Input #3

```
REQ P0 R8
REQ P1 R1
REQ P2 R6
REQ P3 R8
REQ P4 R1
REQ P5 R6
```

Sample Output #3

```
AVAIL
AVAIL
AVAIL
WAIT 1
WAIT 1
WAIT 1
```