INE5410 – Programação Concorrente

Unidade II – Fundamentos de Programação Concorrente

Prof. Frank Siqueira
frank.siqueira@ufsc.br

Conteúdo

- Processos
- Threads
- Exclusão Mútua
- **Semáforos**
- Deadlocks

Semáforos

- Os Semáforos são outro tipo de mecanismo de controle de concorrência
- Criado por Dijkstra nos anos 60 para permitir a sincronização de tarefas
- Um semáforo é um tipo de dados que possui:
  - Um contador (valor inteiro sem sinal)
  - Uma fila de tarefas (threads) em espera
- Operações de um semáforo:
  - P(S): *Prolagen = proberen te verlagen* (tenta reduzir)
  - V(S): *Verhogen* (aumentar)

Semáforos

- Operação P(S)

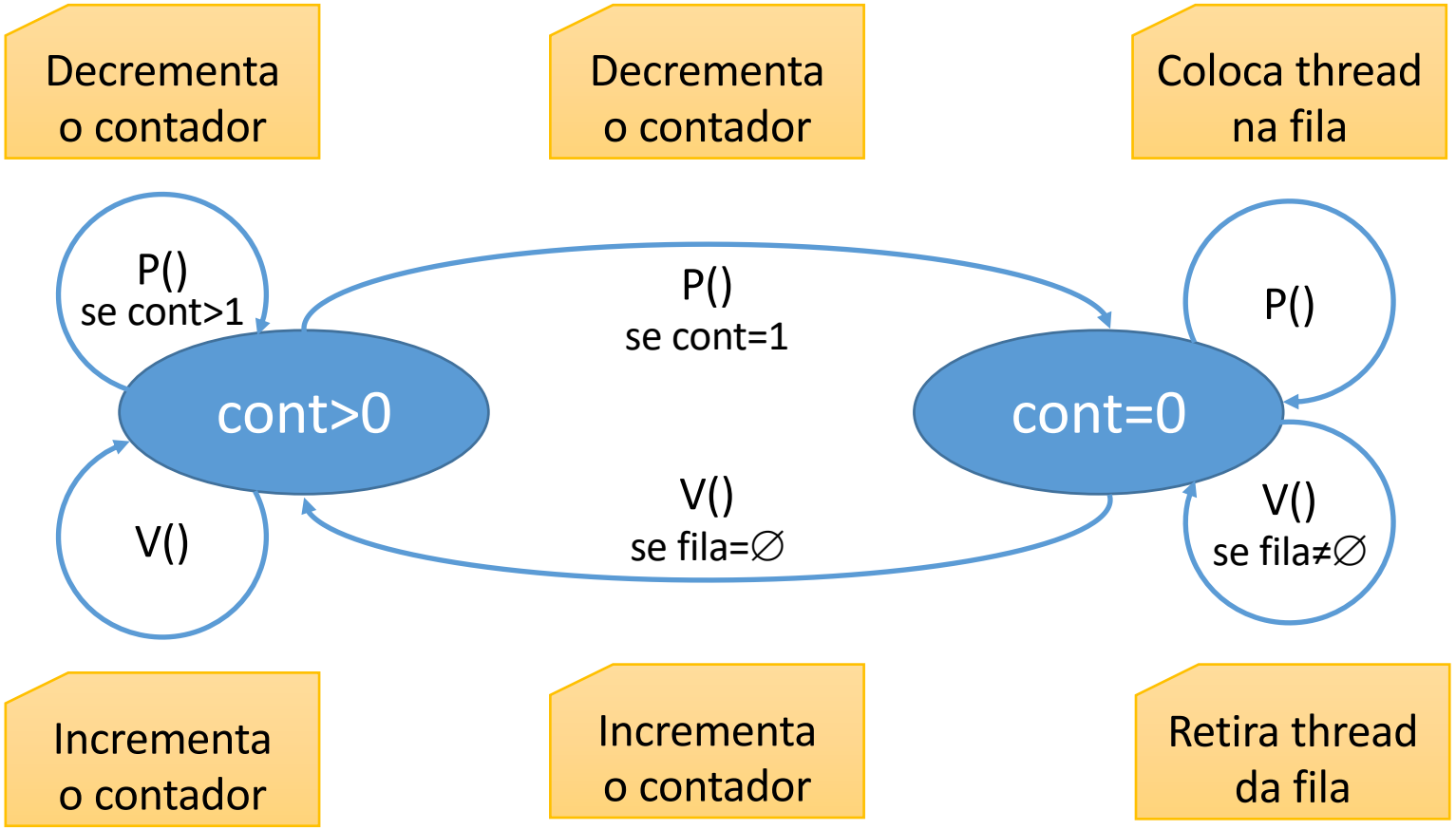
```
P(S):
   SE o contador > 0 ENTÃO
         Decrementa o contador do semáforo S
   SENÃO
         Bloqueia a thread que fez a chamada
         Insere a thread no fim da fila do semáforo S
```

- Operação V(S)

```
V(S):
   SE a fila do semáforo não está vazia ENTÃO
         Libera a thread do início da fila do semáforo S
   SENÃO
         Incrementa o contador do semáforo S
```

Semáforos

- Estados de um Semáforo



Semáforos

- Um Semáforo com valor inicial 1 é idêntico a um Mutex?
  - Não. Há algumas diferenças sutis.
  - O Mutex só assume dois valores: 0 (bloqueado) e 1 (livre); já o Semáforo pode ter o valor do contador incrementado mesmo que inicie em 1
  - O Mutex só deve ser liberado pela thread que o travou (em tese, pois algumas implementações não obrigam isso); já o Semáforo pode ser incrementado/decrementado por várias threads
  - Apesar disso, na maioria dos casos um semáforo com valor 1 pode ser usado como um Mutex

Semáforos

- Semáforos POSIX: <semaphore.h>
  - O semáforo deve ser declarado como variável global:
    sem_t semaforo;
  - O semáforo deve ser inicializado antes de ser usado:
    sem_init(&semaforo, <compart_filhos>,<val_inicial>);
  - Tenta obter o semáforo (entrando ou não na fila):
    sem_wait(&semaforo); *// entra na fila se ñ conseguir*
    sem_trywait(&semaforo); *// não entra na fila*
  - Retorna o valor do contador do semáforo:
    sem_getvalue(&semáforo, &valor);
  - Libera uma posição no semáforo:
    sem_post(&semaforo);
  - Destrói o semáforo quando ele deixar de ser usado:
    sem_destroy(&semaforo);

Semáforos

- Exemplo: suponha que temos um número limitado de licenças para usar um programa

```
...
#define NUM_THREADS     10
#define NUM_LICENCAS    2
#define USA_LICENCA()   sleep(4)

sem_t licencas;                 //

void * func_thread(void *arg) {
   int id = *(int *)arg;
   sem_wait(&licencas);
   printf("Thread %d pegou a licença\n", id);
   USA_LICENCA();
   printf("Thread %d liberou a licença\n", id);
   sem_post(&licencas);
   pthread_exit(NULL);
}
...
```

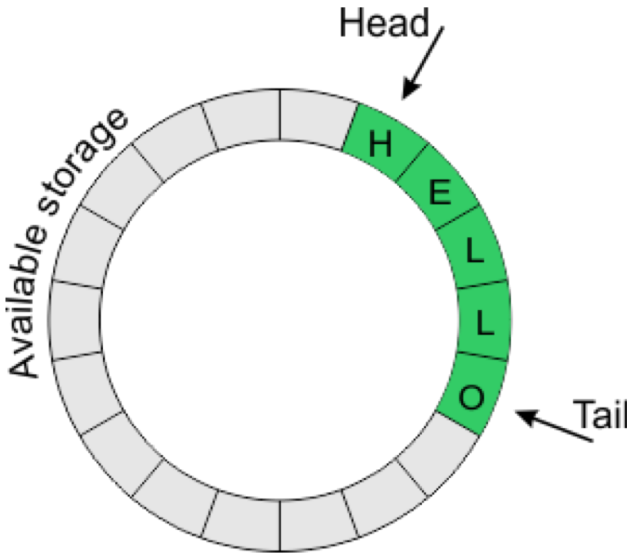
Semáforo

- Exemplo: função *main*() do programa que controla o uso das licenças com semáforo

```
...
int main(int argc, char** argv) {
        pthread_t threads[NUM_THREADS];
        int ids[NUM_THREADS];

        sem_init(&licencas, 0 , NUM_LICENCAS);

        for(int i=0; i < NUM_THREADS; i++) {
                ids[i] = i;
                pthread_create(&threads[i], NULL, func_thread, (void *) &ids[i]);
        }

        for(int i=0; i < NUM_THREADS; i++)
                        pthread_join(threads[i], NULL);

        sem_destroy(&licencas);
        return 0;
}
```

Semáforos

- Exercício: Produtor/Consumidor
  - Duas threads compartilham um buffer circular de tamanho fixo
    - Produtor: insere produtos no buffer
    - Consumidor: retira produtos do buffer
  - Cuidados a serem tomados:
    - Produtor quer armazenar dados, mas o buffer está cheio
    - Consumidor quer retirar dados, mas o buffer está vazio



Semáforos

- Solução: Produtor/Consumidor
  - Crie dois semáforos:
    - Espaços: controla a quantidade de espaços livres (inicia com o tamanho do buffer)
    - Produtos: controla a quantidade de produtos armazenados no buffer (inicia em zero)
  - Comportamentos:
    - Produtor:
      - Decrementa o semáforo de espaços livres;
      - Insere produto no buffer;
      - Incrementa o semáforo de produtos no buffer.
    - Consumidor:
      - Decrementa o semáforo de produtos no buffer;
      - Remove produto do buffer;
      - Incrementa o semáforo de espaços livres.