Programação Pthreads (Mutexes)

Dr. Osmar Marchi dos Santos

Semáforos

- Lembrando:
  - wait(semáforo) ou P(semáforo) - proberen (testar)
    - Enquanto semáforo for igual a 0, processo bloqueia
    - Caso contrário, decrementa o semáforo
  - signal(semáforo) ou V(semáforo) - verhogen (incrementar)
    - Se existir processo bloqueado, desbloqueia um processo
    - Caso contrário, incrementa o semáforo

Mutex

- Tem operações para lock e unlock (assim como para um semáforo) de uma variável do tipo mutex (mutual exclusion)

- Porém, diferentemente do semáforo, o mutex pertence a uma thread (*ownership*)
  - Uma vez que uma thread A executa um lock e obtém o mutex X, outra thread B não pode liberar (unlock) o mutex X, pois o mesmo já pertence a thread A
  - Já um semáforo pode ter suas operações executadas por diferentes threads de execução

Mutex

- Para usar mutex em pthreads temos:

#include <pthread.h>

pthread_mutex_t *mutex; // Cria uma variável do tipo mutex*

*int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr); // Inicializa um mutex*

*int pthread_mutex_destroy(pthread_mutex_t *mutex); // Destrói um mutex*

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // Macro para inicializar um mutex

Mutex

- Para usar mutex em pthreads temos:

#include <pthread.h>

*int pthread_mutex_lock(pthread_mutex_t *mutex); // Obtém o mutex para uso da thread*

*int pthread_mutex_trylock(pthread_mutex_t *mutex); // Funciona pthread_mutex_lock, mas se o mutex estiver sendo ocupado, retorna imediatamente indicando que o recurso está em uso*

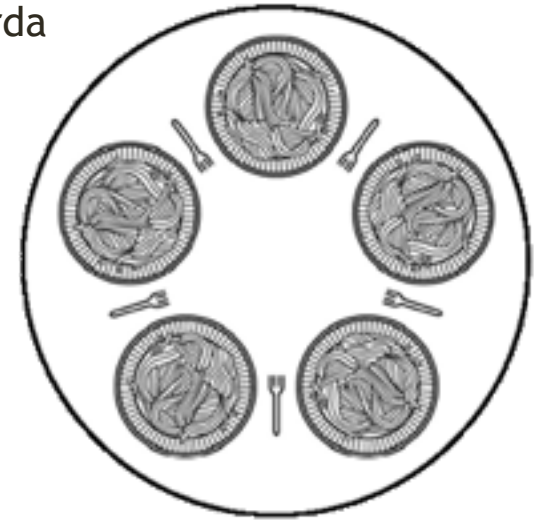
*int pthread_mutex_unlock(pthread_mutex_t *mutex);   // Libera o mutex para uso*

Exemplo:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define NUM_THREADS     5

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *PrintHello(void *threadid) {
   long tid;
   tid = (long)threadid;
   printf("Thread %ld tentando obter mutex!\n", tid);
   pthread_mutex_lock(&mutex);
   printf("Thread %ld executando no mutex!\n", tid);
   usleep(1000000);
   printf("Thread %ld saindo do mutex!\n", tid);
   pthread_mutex_unlock(&mutex);
   printf("Thread %ld fora do mutex!\n", tid);
   pthread_exit(NULL);
}

int main (int argc, char *argv[]) {
   pthread_t threads[NUM_THREADS];
   int rc;
   long t;
   for(t=0; t<NUM_THREADS; t++){
      printf("In main: creating thread %ld\n", t);
      rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
      if (rc){
         printf("ERROR; return code from pthread_create() is %d\n", rc);
         exit(-1);
      }
   }
   for (t = 0; t < NUM_THREADS; t++)
      pthread_join(threads[t],NULL);
}
```

Exercícios - Criar uma solução para o problema dos Filósofos Glutões

- Cinco filósofos estão sentados em uma mesa redonda
- Cada filósofo possui um prato à sua frente
- Cada filósofo possui um garfo à sua direita e à sua esquerda
- Um filósofo precisa de dois garfos para conseguir comer
- Cada garfo é um recurso compartilhado
- Os filósofos tem um comportamento cíclico:
  - Pensa
  - Pega garfo à Esquerda
  - Pega garfo à Direita
  - Come
  - Libera garfo à Direita
  - Libera garfo à Esquerda



- * Um filósofo pega um garfo por vez, nunca ambos os garfos de uma vez
- ** Fazer uma solução em que todos os filósofos começam pegando os garfos à Esquerda (execução tem deadlock?)
- *** Fazer uma solução em que um dos filósofos começa pegando os garfos à Direita (execução tem deadlock?)

Referência

- API para POSIX publicada em https://www2.opengroup.org/