Threads

- pthread_create(3) - create a new thread
- pthread_detach(3) - detach a thread
- pthread_cond_wait(3) - wait on a condition variable
- pthread_cond_signal(3) - unblock a thread waiting for a condition variable
- pthread_cond_broadcast(3) - unblock all threads waiting for a condition variable

Sincronização de Threads

- Atenção às diferenças entre a sincronização à custa de semáforos e variáveis de condição:
  - Semáforos têm "memória", variáveis de condição não têm.
    - Um pthread_cond_signal sem nenhum thread à espera "perde-se"
    - Um V() sem nenhum processo à espera incrementa o valor do semáforo
  - Com semáforos, só se liberta um processo de cada vez
    - Para libertar todos os processos bloqueados, tem de se executar um ciclo de V().
    - Para libertar todos os threads bloqueados numa variável de condição, deve fazer-se um broadcast

Exercícios

- Barbeiro
- Filósofos
- Parque de estacionamento
- Eco-aventura (barco + corda)
- Lockf
- Implementação de monitores e variáveis de condição
- Spooler
- Escalonamento de pedidos de transferência de disco
- Readers & writers (a.k.a "ponte do porto")
- ...

Dining Philosophers

- 5 filósofos, que repartem a sua vida entre 2 estados:
  - Pensar
  - Comer
- Para comer, sentam-se a uma mesa com 5 garfos,
  - Pegam no garfo esquerdo, se possível
  - Pegam no garfo direito, se possível
  - Comem o esparguete
  - Pousam os garfos e vão pensar mais um bocado

Dining Philosophers

- Os garfos são recursos críticos, e um filósofo só come se tiver em seu poder os 2 garfos (esquerdo e direito)
- Aplicando a "receita" da exclusão mútua

```
Come (f){
    P (ESQUERDO(f))
    P (DIREITO(f))
        <come_mesmo>
    V (ESQUERDO(f))
    V (DIREITO(f))
}
```

Nota:

ESQUERDO(x) e DIREITO (x) são macros que indicam os garfos correspondentes ao filósofo X

Dining Philosophers

- Mas também podiamos ter pegado no problema pelo lado da sincronização, obrigando os filósofos a parar...
- Aplicando a "receita" da sincronização

```
Come (f){
    if /* Not OK */ ... P (ESPERA(f)) ...
            <come_mesmo>
    while  /* ???  */ {      /* pode ter de libertar 2 */
        X = escolhe_filosofo /* e agora acorda-o*/
        V (ESPERA(X))
    }
}
```

Hummm, complicado...

Dining Philosophers

- Deadlock:
  - Se todos pegarem no garfo esquerdo "ao mesmo tempo", todos param porque não conseguem o direito. Como só devolvem os garfos depois de comer... Ninguém come!
- Starvation:
  - Dois filósofos podem impedir outro de comer

Dining Philosophers

- Soluções?
  - Não deixar entrar na sala mais de 4 filósofos.
  - Ordenar os garfos de modo a que um filósofo comece sempre pelo seu garfo mais baixo. Por exemplo,
    - Filósofo 0: pega no garfo 0 e depois no 1
    - Filósofo 4: pega no garfo 0 e depois no 4

Readers & Writers

- Há um recurso partilhado por 2 classes de utilizadores
  - Leitores
  - Escritores
- Não há necessidade de exigir exclusão mútua entre os leitores, visto que estes não modificam a informação
- Há necessidade de garantir exclusão mútua nos escritores

a.k.a. Ponte do Porto

- 2 classes: automóveis e camiões
- Segurança
  - Quando muito 1 camião na ponte (=> 0 automóveis), ou
  - Qualquer número de automóveis na ponte
- Prioridade
  - Enquanto forem chegando automóveis, o que acontece aos camiões?
  - E vice-versa?

Barco+corda

- Barco tem capacidade N e só avança se estiver cheio
- Na corda só passa um elemento de cada vez
- Barco só regressa depois de todos os elementos terem passado na corda
- Só há um barco...
- Há várias equipas mas para simplificar admita-se que podem ir misturadas (i.e. membros de várias equipas no barco)