MC514 - Lab3
Sushi Bar

Raphael Kubo da Costa

O problema

Not remotely classical...

Um *sushi bar* tem 5 lugares. Se um deles estiver vazio, você pode chegar, sentar e comer.

Se os 5 lugares estiverem ocupados, todas as pessoas estão comendo juntas, portanto você precisa esperar todos terminarem e saírem para comer.

Ready... Spank, spank, spank.

A solução

Na verdade, há duas soluções sugeridas. A segunda é mais legal.

Usamos uma solução de "passe o bastão":
Uma thread não precisa necessariamente liberar o mutex que usou. Ele vai sendo passado até a última thread. Basta tomar cuidado...

Vamos ao algoritmo.

O algoritmo

Copiado^H^H^H^H^H^H^H Baseado fortemente na sugestão do livro.

Usamos as seguintes variáveis para ajudar:

```
Contadores: eating, waiting
Semáforos: mutex, block
Flag: must_wait
```

Vamos ao resto do algoritmo (em Python pra ficar conciso ;)

O algoritmo

```
mutex.wait()
if must_wait:
   waiting += 1
   mutex.signal()
   block.wait()
   waiting -= 1

eating += 1
must_wait = (eating == 5)
if waiting and not must_wait:
   block.signal()
else:
   mutex.signal()

# Eat! Insert code here.
```

```
mutex.wait()
eating -= 1
if eating == 0: must_wait = 0

if waiting and not must_wait:
   block.signal()
else:
   mutex.signal()
```

E como fica em C?
Cadê o seu código?!

A implementação

Open source. Licença BSD. Hospedado no github: **<http://github.com/rakuco/sushibar-mt>**

O algoritmo está na função **sushibar_run** em **sushibar.c**

Diferenças:

- Os asserts são para fazer verificações de consistência
- Os sleeps são para ajudar a coisa a parecer mais multithreaded

`mutex` e `block` foram implementados como semáforos (`sem_t`), que, dependendo da libc, usam mutexes e variáveis de condição por baixo