Aula 07

Variáveis de Condição e Semáforos

Por que Locks não são Suficientes?

Locks garantem exclusão mútua.

Mas não resolvem a necessidade de esperar uma condição.

Exemplo: Consumidor espera buffer não vazio; Produtor espera buffer não cheio.

Apenas um lock não basta.

A Ineficiência do Spinning

- Exemplo:

```
while (done == 0) ; // spin
```

- Problema: consome 100% da CPU sem trabalho útil.
- Questão: Como fazer uma thread dormir até a condição ser atendida?

```c
volatile int done = 0;

void *child(void *arg) {
    printf("child\n");
    done = 1;
    return NULL;
}

int main(int argc, char *argv[]) {
    printf("parent: begin\n");
    pthread_t c;
    Pthread_create(&c, NULL, child, NULL);

    while (done == 0)
    ; // spin

    printf("parent: end\n");
    return 0;
}
```

Variáveis de Condição (CV)

- Fila explícita onde threads podem esperar por condições.
- `wait(cond, lock)`: libera o lock e suspende a thread.
- `signal(cond)`: acorda uma thread esperando.
- Sempre usadas em conjunto com mutex.

```
// thr_exit() - FILHA
void thr_exit() {
        done = 1;
        Pthread_cond_signal(&c);
}

// thr_join() - PAI (VERSÃO COM FALHA)
void thr_join() {
    if (done == 0) // 1. Pai checa a condição
        // <-- A INTERRUPÇÃO PODE OCORRER AQUI
        Pthread_cond_wait(&c); // 2. Pai tenta dormir
}
```

Exemplo 1: join() com Variável de Condição

```c
int done = 0;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c = PTHREAD_COND_INITIALIZER;

// Chamado pela thread filha para sinalizar que terminou
void thr_exit() {
    Pthread_mutex_lock(&m);
    done = 1;
    Pthread_cond_signal(&c);
    Pthread_mutex_unlock(&m);
}

// Chamado pela thread pai para esperar pela filha
void thr_join() {
    Pthread_mutex_lock(&m);
    while (done == 0)
        Pthread_cond_wait(&c, &m); // Libera o lock e dorme
    Pthread_mutex_unlock(&m);
}
```

```c
// Função principal e da filha (simplificada)
void *child(void *arg) {
    printf("child\n");
    thr_exit();
    return NULL;
}

int main(int argc, char *argv[]) {
    printf("parent: begin\n");
    pthread_t p;
    Pthread_create(&p, NULL, child, NULL);
    thr_join(); // Pai espera aqui
    printf("parent: end\n");
}
```

Por que while e não if? Semântica Mesa

- `signal()` é apenas uma dica, não garante condição verdadeira.
- Outro consumidor pode 'roubar' o item antes.
- while garante re-verificação após acordar.
- Semântica Mesa é usada em sistemas modernos.

Problema Clássico: Produtor/Consumidor

Produtores geram itens para um buffer limitado.

Consumidores retiram e processam itens.

Produtores esperam buffer cheio; Consumidores esperam buffer vazio.

Aplicações: filas de servidores web, pipes em UNIX.

Produtor/Consumidor - Solução com 1 CV (Errada)

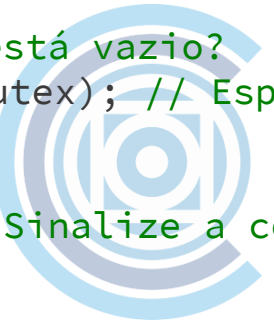
- Cenário: P, C1, C2.
- C1 e C2 dormem (buffer vazio).
- P insere item, acorda C1.
- C1 consome e sinaliza C2.
- C2 acorda, vê buffer vazio, volta a dormir.
- Todos dormem → Deadlock.

Produtor/Consumidor - Solução Final (Correta)

- Usar duas CVs:
  - produtores esperam buffer cheio.
  - consumidores esperam buffer vazio.
- Produtor sinaliza fill, consumidor sinaliza empty.
- Evita deadlock.

```
// --- PRODUCER (Versão Correta) ---
void *producer(void *arg) {
    // ... loop ...
    Pthread_mutex_lock(&mutex);
    while (count == MAX) // O buffer está cheio?
        Pthread_cond_wait(&empty, &mutex); // Espere na
    condição 'empty'
    put(i);
    Pthread_cond_signal(&fill); // Sinalize a condição
    'fill'
    Pthread_mutex_unlock(&mutex);
}

// --- CONSUMER (Versão Correta) ---
void *consumer(void *arg) {
    // ... loop ...
    Pthread_mutex_lock(&mutex);
    while (count == 0) // O buffer está vazio?
        Pthread_cond_wait(&fill, &mutex); // Espere na
    condição 'fill'
    int tmp = get();
    Pthread_cond_signal(&empty); // Sinalize a condição
    'empty'
    Pthread_mutex_unlock(&mutex);
}
```

Semáforos

- Inventado por Dijkstra.
- `sem_wait(s)`: decrementa, bloqueia se <0.
- `sem_post(s)`: incrementa, acorda se houver threads.
- Generalização de lock e CV.

Usos do Semáforo

- Como Lock: semáforo binário inicial=1.
- Como Ordenação (join): semáforo inicial=0, pai espera, filha libera.

Produtor/Consumidor com Semáforos

- 3 Semáforos:
  - empty: conta slots vazios, inicial=MAX.
  - full: conta slots cheios, inicial=0.
  - mutex: exclusão mútua, inicial=1.
- Ordem de wait é crucial para evitar deadlocks!