Threads, Mutex e Semáforos

. . .

MAC0219/5742 Introdução à Computação Concorrente, Paralela e Distribuída (2019)

Alfredo Goldman
Giuliano Belinassi
Matheus Tavares

Slides baseados no material “Notas de Aula — MAC0431” do professor Marcos Gubitoso, de 2016

Conceitos Básicos

- Threads vs Processos
- Concorrência e Paralelismo
- Operação Atômica
- Taxonomia de Flynn
  - SISD
  - SIMD
  - MISD
  - MIMD

Condições de Corrida

```
int x = 0;

async thread_one() {
    x++;
}

async thread_two() {
    x--;
}
thread_one( );
thread_two( );
printf(“%d\n”, x);
```

Qual o valor de x?

- -1
- 1
- 0
- n.d.a.

Condições de Corrida

```
x   dw  0

thread_one:
    mov eax, DWORD [x]
    inc eax
    mov DWORD [x], eax
    ret

thread_two:
    mov eax, [x]
    dec eax
    mov [x], eax
    ret
```

Qual o valor de x?

- -1
- 1
- 0
- n.d.a.

Sessão Crítica

Trecho de código que deve ser executado inteiramente por **uma thread**, antes que qualquer outra thread o comece a executar.

Exemplo:

```
{
    mov eax, DWORD [x]
    inc eax
    mov DWORD [x], eax
}
```

Nota: A sessão crítica se torna um trecho sequencial dentro do código paralelo, portanto, use com moderação :)

Como Implementar?

```
int turn = 1;

void thread_one() {
    turn = 1; // diz que é a sua vez
    while (turn != 1)
        usleep (1); // aguarda

    x++;

    turn = 2; // passa a vez
}
```

```
void thread_two() {
    turn = 2; // diz que é a sua vez
    while (turn != 2)
        usleep (1); // aguarda

    x++;

    turn = 1; // passa a vez
}
```

Esse código funciona?

Solução Simples: “Busy Waiting”

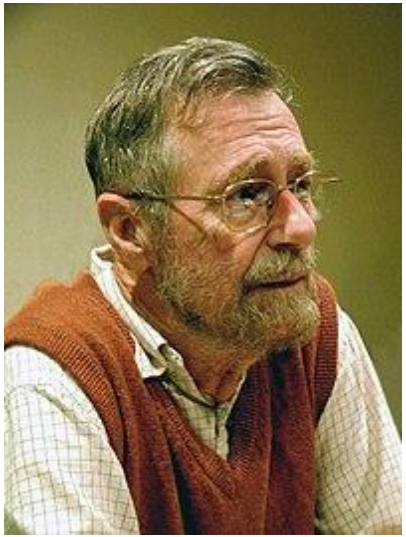
```
bool wants_to_enter_one = 0;
bool wants_to_enter_two = 0;
int turn = 1;

void thread_one() {
    wants_to_enter_one = true; // sinaliza que quer entrar
    turn = 2; // passa a prioridade
    while (wants_to_enter_two && turn == 2)
        usleep (1); // aguarda

    x++;

    wants_to_enter_one = false; // não quero mais entrar
}
```

Mas “Busy Waiting” não é legal...

- Busy waiting, como o nome diz, ocupa a CPU...
- Solução é um recurso do SO chamado: Mutex

Semáforos

- Criado por Dijkstra
- Controle de acesso à um recurso
- Conta o número de processos/threads que podem usar um recurso ao mesmo tempo sem que nenhum outro deixe de usar
- Operações:
  - **P**: decrementa o contador, se > 0. Se estiver em 0, aguarda.
  - **V**: incrementa o contador.



Mutex: Semáforos Binários

- Semáforos que só permitem os valores **0 e 1.**
- Usados como *locks*, i.e., travas que só permitem o acesso a **uma thread por vez.**

Exemplo:

```
mutex_lock(&mutex);
{
    /* critical section work */
}
mutex_unlock(&mutex);
```

Variáveis de Condição

- Variáveis que permitem a threads, esperarem até que uma condição seja satisfeita.
- Úteis para sincronização

Exemplo:

```
void one_thread()
    mutex_lock(&mutex);
    while (x < 10)
        condition_wait(&cond,
                       &mutex);
    /* work on x */
    mutex_unlock(&mutex);
}
```

```
void another_thread()
    mutex_lock(&mutex);
    x += 8; / arbitrary work on x */
    if (x >= 10)
        condition_broadcast(&cond);
    mutex_unlock(&mutex);
}
```

Barreiras

- Quando alguma thread entra em uma barreira, ela não pode continuar até que todas as threads tenham entrado. Então, todas são liberadas.
- Úteis para sincronização

Exemplo:

```
void thread_work(int thread_id, int A[N][M], int B[N][M]) // There are N
    for (int j = 0; j < M; ++j)                           // threads
        A[thread_id][j] += sin(j);
    barrier_wait();
    for (int j = 0; j < M; ++j)
        B[thread_id][j] += A[(thread_id + 1) % N][j];
}
```

Dependência de Dados

- Só se pode paralelizar tarefas independentes.

Exemplo: O que pode ser paralelizado no código abaixo?

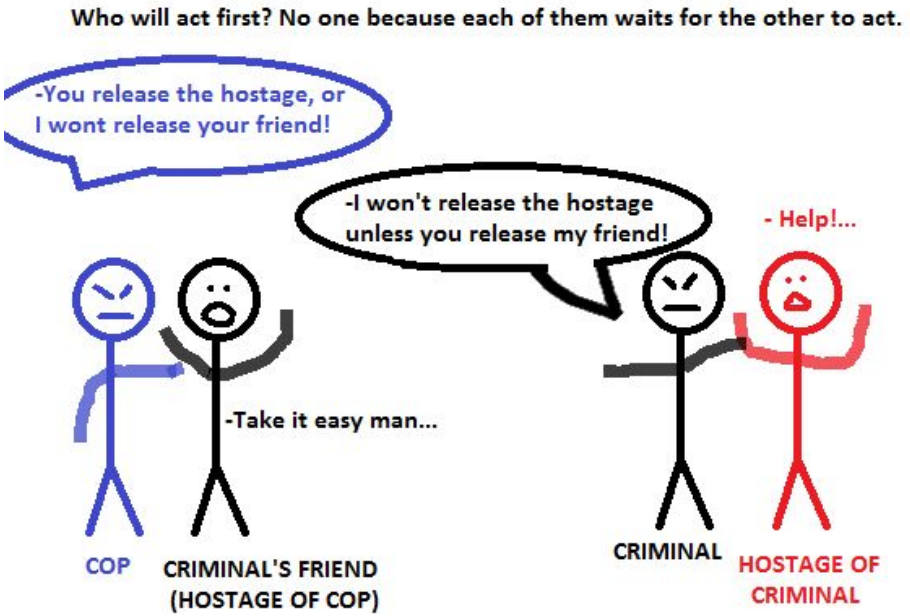
```
void f (int A[N][N]) {
    for (int i = 0; i < N; ++i) {
        for (int j = 1; j < N; ++j) {
            A[i][j] += A[i][j-1];
        }
    }
}
```

DeadLock

- Múltiplos processos/threads bloqueados, esperando uns pelos outros.

Fonte: https://www.oficinadanet.com.br/post/12786-sistemas-operacionais-o-que-e-deadlock

DeadLock



De https://stackoverflow.com/questions/34512/what-is-a-deadlock#35640575

A API *pthread.h* e *semaphore.h*

- `pthread_create`
- `pthread_join`
- `pthread_mutex_lock`
- `pthread_mutex_unlock`
- `pthread_cond_wait`
- `pthread_cond_broadcast`
- `pthread_barrier_wait`
- `sem_post`
- `sem_wait`
- `...`

Vejamos na prática

Recomendações

- Foundations of Multithreaded, Parallel and Distributed Programming, Gregory R. Andrews (Tem um capítulo só de pthreads)
- Modern Operating Systems, A.S. Tanenbaum (Tem um capítulo conceitual sobre threads)
- Introduction to High Performance Scientific Computing, Eijkhout, Victor & van de Geijn, Robert & Chow
- Man pages de pthreads