Programando com *Threads* em C

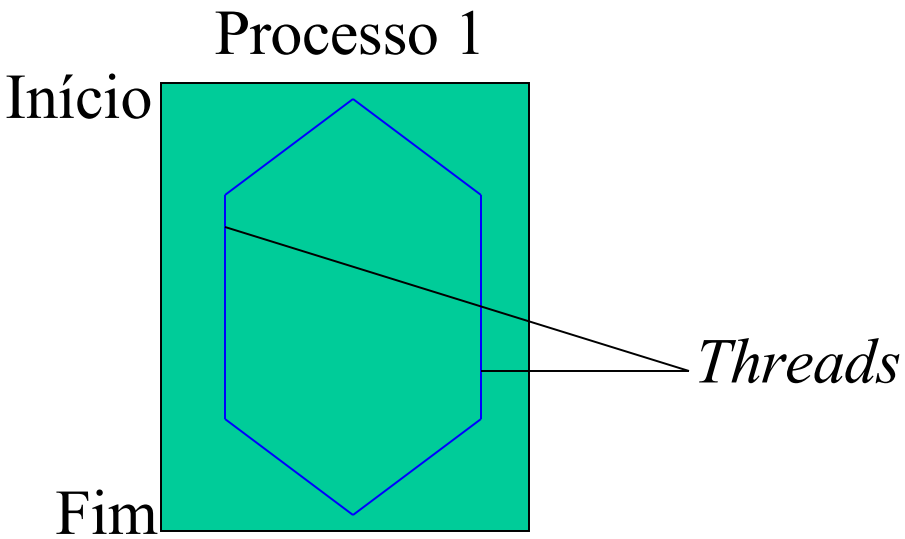
AEDS III

Bruno Diniz de Paula
(diniz@dcc.ufmg.br)

e-Commerce System Performance Evaluation and
Experimental Development Lab

O que são *Threads*?

- Linhas de execução concorrentes
- Memória (pilha) independente
- Podem compartilhar áreas de memória



Problemas

- Sincronização entre elas
  - Condições de corrida (*race conditions*)
  - *Deadlock's*
- Localização de erros
- Difícil garantia de correção dos programas (modelos analíticos e verificação formal)
- Imprevisibilidade

Estruturas e Funções Usadas

Biblioteca pthread.h

- pthread_t (struct)
- pthread_create
- pthread_join
- pthread_kill
- pthread_exit
- outras (man pthreads - turmalina)

Criação de *Threads*

```
pthread_t threads[2];

void *thread_func(void *arg) {
        ...
}

int main(int argc, char **argv) {
        int i;

        for(i=0; i<2; i++) {
                pthread_create(&(threads[i]), NULL, thread_func, NULL);
        }
        for(i=0; i<2; i++) {
                pthread_join(threads[i], NULL);
        }
}
```

Passando Parâmetros

```
pthread_t threads[2];

void *thread_func(void *arg) {
          int *n = (int *)arg;
          ...
}

int main(int argc, char **argv) {
          int i, a = 10;

          for(i=0; i<2; i++) {
                    pthread_create(&(threads[i]), NULL, thread_func, &a);
          }
          for(i=0; i<2; i++) {
                    pthread_join(threads[i], NULL);
          }
}
```

Um Programa Completo (1/2)

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

typedef struct {
          int idx, length;
}thread_arg, *ptr_thread_arg;

pthread_t threads[2];

void *thread_func(void *arg) {
          ptr_thread_arg targ = (ptr_thread_arg)arg;
          int i;

          for(i=targ->idx; i<(targ->idx + targ->length); i++) {
                    printf("Thread %d – value %d\n", pthread_self(), i);
          }
}
```

Um Programa Completo (2/2)

```
int main(int argc, char **argv) {
          thread_arg arguments[2];
          int i;

          for(i=0; i<2; i++) {
                    arguments[i].idx = i * 10;
                    arguments[i].length = 10;
                    pthread_create(&(threads[i]), NULL, thread_func,
&(arguments[i]));
          }
          for(i=0; i<2; i++) {
                    pthread_join(threads[i], NULL);
          }

}
```

Compilando

- Biblioteca de pthreds é dinâmica
- Linha de comando
  - gcc ... -D_REENTRANT -lpthread

Somando Números (1/4)

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

#define NUMTHREADS            2
#define VETSIZE               5000

typedef struct {
          int fromidx, length;
}thread_arg, *ptr_thread_arg;

pthread_t threads[NUMTHREADS];
thread_arg arguments[NUMTHREADS];
int nums[VETSIZE];
int sum;

void *thread_func(void *arg);
```

Somando Números (2/4)

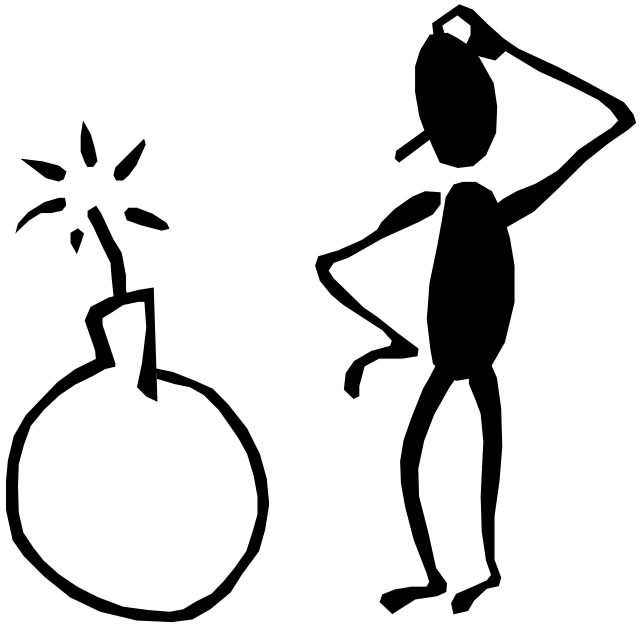
```
int main(int argc, char **argv) {
          int i, length, remainder;

          sum = 0;  length = VETSIZE / NUMTHREADS;
          remainder = VETSIZE % NUMTHREADS;
          for(i=0; i<NUMTHREADS; i++) {
                    arguments[i].fromidx = i * length;
                    arguments[i].length = length;
                    if(i == (NUMTHREADS - 1))
                              arguments[i].length += remainder;
                    pthread_create(&(threads[i]), NULL, thread_func,
&(arguments[i]));
          }
          for(i=0; i<NUMTHREADS; i++) {
                    pthread_join(threads[i], NULL);
          }
          printf("A soma dos numeros do vetor eh %d\n", sum);
}
```

Somando Números (3/4)

```
void *thread_func(void *arg) {
        ptr_thread_arg argument = (ptr_thread_arg)arg;
        int i, localsum = 0, endidx;

        endidx = argument->fromidx + argument->length;
        for(i=argument->fromidx; i<endidx; i++) {
                localsum += nums[i];
        }
        sum += localsum;
}
```

Somando Números (4/4)

Qual é o problema com o programa anterior?



Solução

Sincronização!!!

Alguns Conceitos

- Exclusão mútua
  - uma *thread* está executando sozinha um determinado código, enquanto as outras esperam para poder executar
- Sessão crítica
  - parte do programa que é executada por somente uma *thread* de cada vez (em exclusão mútua)

Primitivas de Sincronização

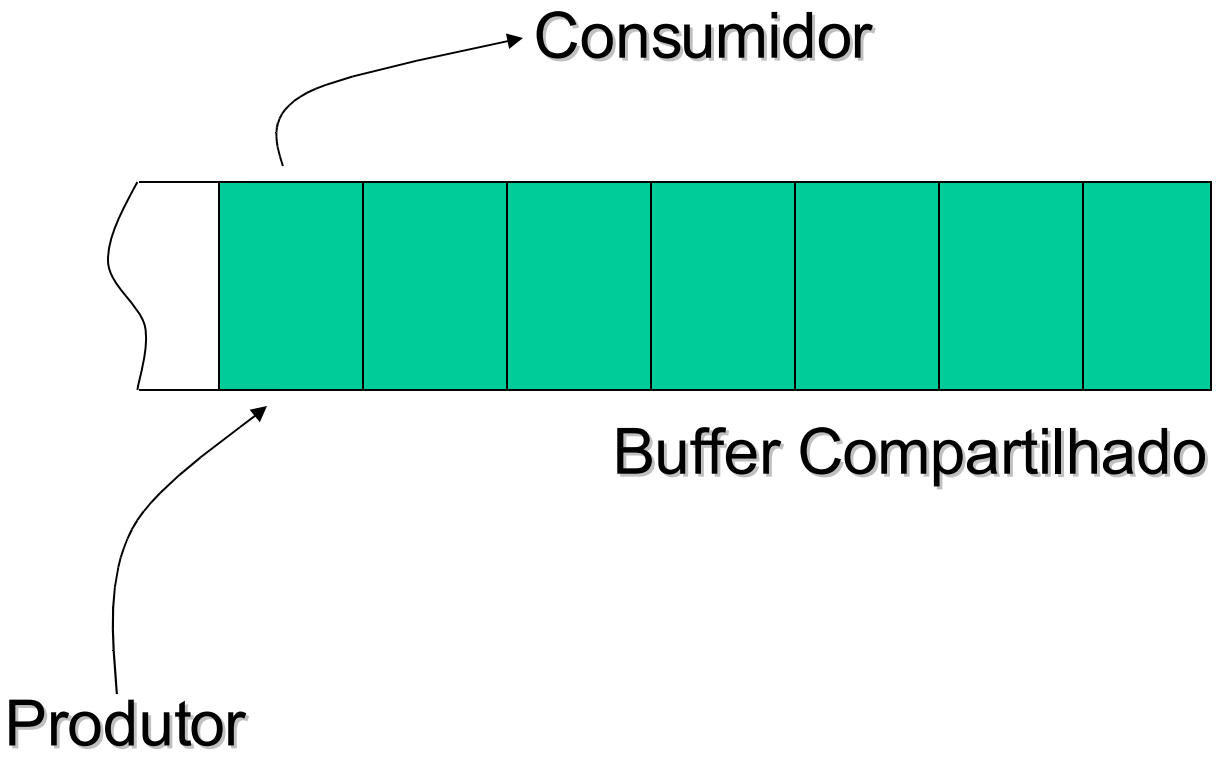
- Mutexes
- Semáforos
- Troca de mensagens

- Monitores (Java)

Estruturas e Funções Usadas

- pthread_mutex_t (struct) – sem. binário
- pthread_mutex_lock
- pthread_mutex_unlock
- sem_t (struct) – sem. não binário
- sem_wait
- sem_post

Produtor / Consumidor (1/4)



Produtor / Consumidor (2/4)

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

#define NUMCONS             2
#define NUMPROD             2
#define BUFFERSIZE          1000

pthread_t cons[NUMCONS];
pthread_t prod[NUMPROD];
int buffer[BUFFERSIZE];
int prod_pos=0, cons_pos=0;

void *consumidor(void *arg);
Void *produtor(void *arg);
```

Produtor / Consumidor (3/4)

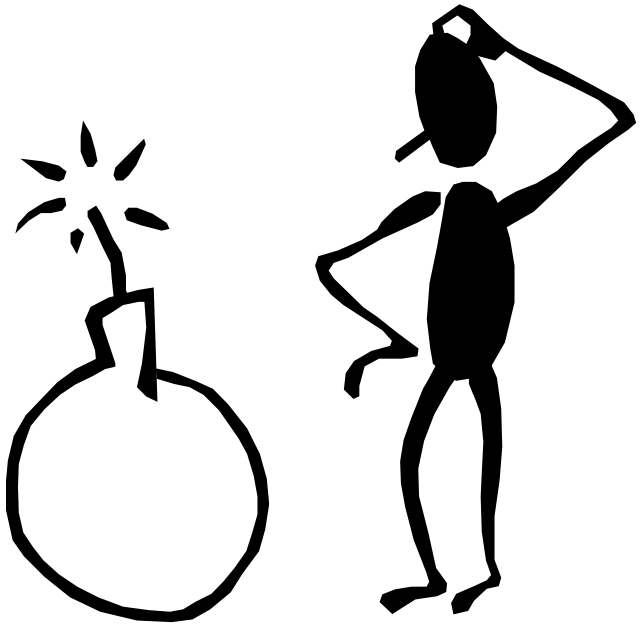
```
int main(int argc, char **argv) {
          int i;

          srand48(time());
          for(i=0; i<NUMCONS; i++) {
                    pthread_create(&(cons[i]), NULL, consumidor, NULL);
          }
          for(i=0; i<NUMPROD; i++) {
                    pthread_create(&(prod[i]), NULL, produtor, NULL);
          }
          for(i=0; i<NUMCONS; i++) {
                    pthread_join(cons[i], NULL);
          }
          for(i=0; i<NUMPROD; i++) {
                    pthread_join(prod[i], NULL);
          }
}
```

Produtor / Consumidor (4/4)

```
void *produtor(void *arg) {
        int n;
        while(1) {
                n = (int)(drand48() * 1000.0);
                buffer[prod_pos] = n;
                prod_pos = (prod_pos+1) % BUFFERSIZE;
                printf("Produzindo numero %d\n", n);
                sleep((int)(drand48() * 4.0));
        }
}
void *consumidor(void *arg) {
        int n;
        while(1) {
                n = buffer[cons_pos];
                cons_pos = (cons_pos+1) % BUFFERSIZE;
                printf("Consumindo numero %d\n", n);
                sleep((int)(drand48() * 4.0));
        }
}
```

E de novo...

Qual é o problema com o programa anterior?



1a. Tentativa de Solução (1/4)

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

#define NUMCONS             2
#define NUMPROD             2
#define BUFFERSIZE          1000

pthread_t cons[NUMCONS];
pthread_t prod[NUMPROD];
pthread_mutex_t buffer_mutex;
int buffer[BUFFERSIZE];
int prod_pos=0, cons_pos=0;

void *consumidor(void *arg);
Void *produtor(void *arg);
```

1a. Tentativa de Solução (2/4)

```
int main(int argc, char **argv) {
        int i;
        srand48(time());
        pthread_mutex_init(&buffer_mutex, NULL);
        for(i=0; i<NUMCONS; i++) {
                pthread_create(&(cons[i]), NULL, consumidor, NULL);
        }
        for(i=0; i<NUMPROD; i++) {
                pthread_create(&(prod[i]), NULL, produtor, NULL);
        }
        for(i=0; i<NUMCONS; i++) {
                pthread_join(cons[i], NULL);
        }
        for(i=0; i<NUMPROD; i++) {
                pthread_join(prod[i], NULL);
        }
}
```

1a. Tentativa de Solução (3/4)

```
void *produtor(void *arg) {
        int n;
        while(1) {
                n = (int)(drand48() * 1000.0);
                pthread_mutex_lock(&buffer_mutex);
                buffer[prod_pos] = n;
                prod_pos = (prod_pos+1) % BUFFERSIZE;
                pthread_mutex_unlock(&buffer_mutex);
                printf("Produzindo numero %d\n", n);
                sleep((int)(drand48() * 4.0));
        }
}
```

1a. Tentativa de Solução (4/4)

```
void *consumidor(void *arg) {
        int n;
        while(1) {
                pthread_mutex_lock(&buffer_mutex);
                n = buffer[cons_pos];
                cons_pos = (cons_pos+1) % BUFFERSIZE;
                pthread_mutex_unlock(&buffer_mutex);
                printf("Consumindo numero %d\n", n);
                sleep((int)(drand48() * 4.0));
        }
}
```

Agora sim...

Ficou correto?

Não!!!! Por quê?

Quem controla a ocupação do buffer?

2a. Tentativa de Solução (1/4)

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

#define NUMCONS              2
#define NUMPROD              2
#define BUFFERSIZE           1000

pthread_t cons[NUMCONS];    pthread_t prod[NUMPROD];
pthread_mutex_t buffer_mutex;
int buffer[BUFFERSIZE];
int prod_pos=0, cons_pos=0;
sem_t free_positions, filled_positions;

void *consumidor(void *arg);
Void *produtor(void *arg);
```

2a. Tentativa de Solução (2/4)

```
int main(int argc, char **argv) {
          int i;
          srand48(time());
          pthread_mutex_init(&buffer_mutex, NULL);
          sem_init(&free_proditions, 0, BUFFERSIZE);
          sem_init(&filled_positions, 0, 0);
          for(i=0; i<NUMCONS; i++) {
                    pthread_create(&(cons[i]), NULL, consumidor, NULL);
          }
          for(i=0; i<NUMPROD; i++) {
                    pthread_create(&(prod[i]), NULL, produtor, NULL);
          }
          ...
}
```

2a. Tentativa de Solução (3/4)

```
void *produtor(void *arg) {
        int n;
        while(1) {
                n = (int)(drand48() * 1000.0);
                sem_wait(&free_positions);
                pthread_mutex_lock(&buffer_mutex);
                buffer[prod_pos] = n;
                prod_pos = (prod_pos+1) % BUFFERSIZE;
                pthread_mutex_unlock(&buffer_mutex);
                sem_post(&filled_positions);
                printf("Produzindo numero %d\n", n);
                sleep((int)(drand48() * 4.0));
        }
}
```

2a. Tentativa de Solução (4/4)

```
void *consumidor(void *arg) {
        int n;
        while(1) {
                sem_wait(&filled_positions);
                pthread_mutex_lock(&buffer_mutex);
                n = buffer[cons_pos];
                cons_pos = (cons_pos+1) % BUFFERSIZE;
                pthread_mutex_unlock(&buffer_mutex);
                sem_post(&free_positions);
                printf("Consumindo numero %d\n", n);
                sleep((int)(drand48() * 4.0));
        }
}
```

Agora...

... é só sair programando usando *threads* loucamente e tirar total no trabalho de AEDS! ☺

Obrigado!