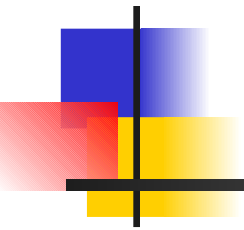
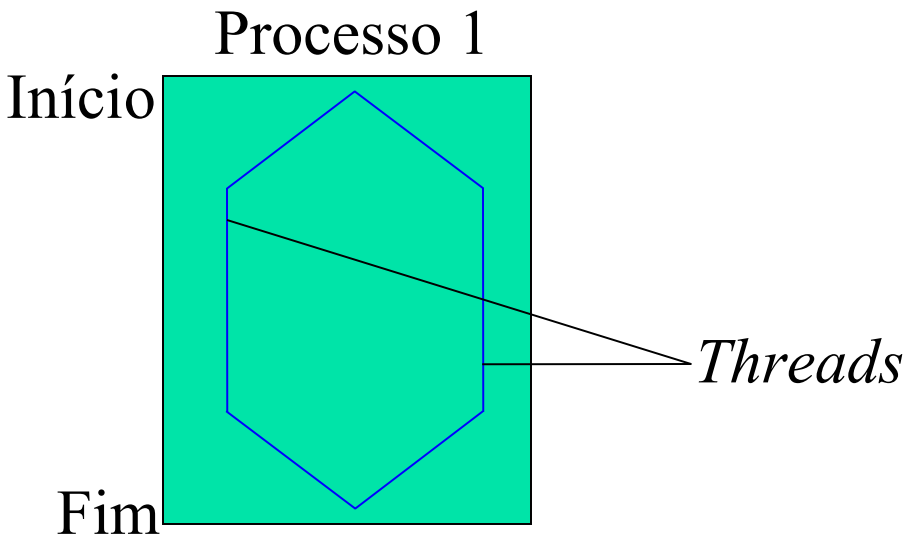
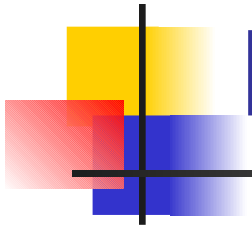


Programando com *Threads* em C

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

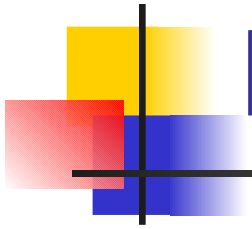
Estruturas e Funções Usadas

Biblioteca pthread.h

- pthread_t (struct)
- pthread_create
- pthread_join
- pthread_kill
- pthread_exit

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

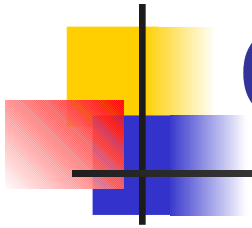
```
pthread_t threads[2];

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 ... -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)

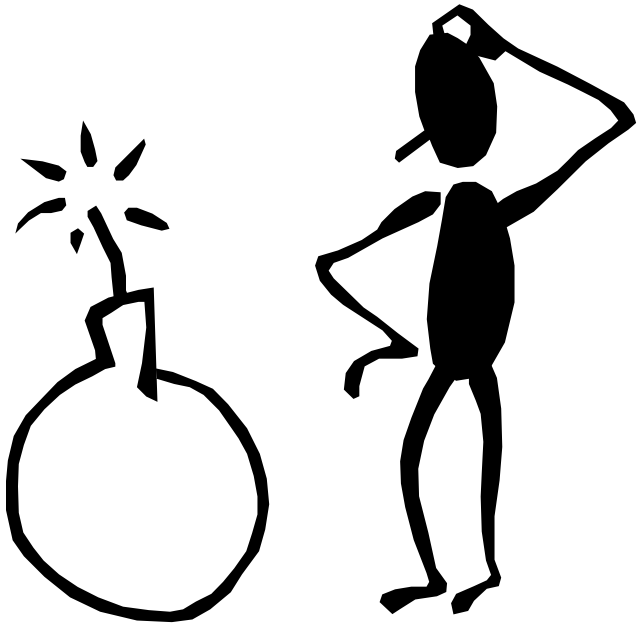
```
int main(int argc, char **argv) {
    int i, length, remainder;

    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;
}
```

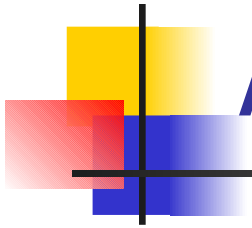
Somando Números (4/4)

Qual é o problema com o programa anterior?



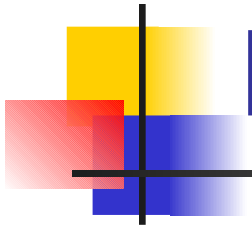
Solução

Sincronização!!!



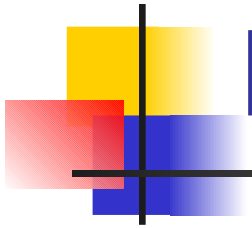
Alguns Conceitos

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



Primitivas de Sincronização

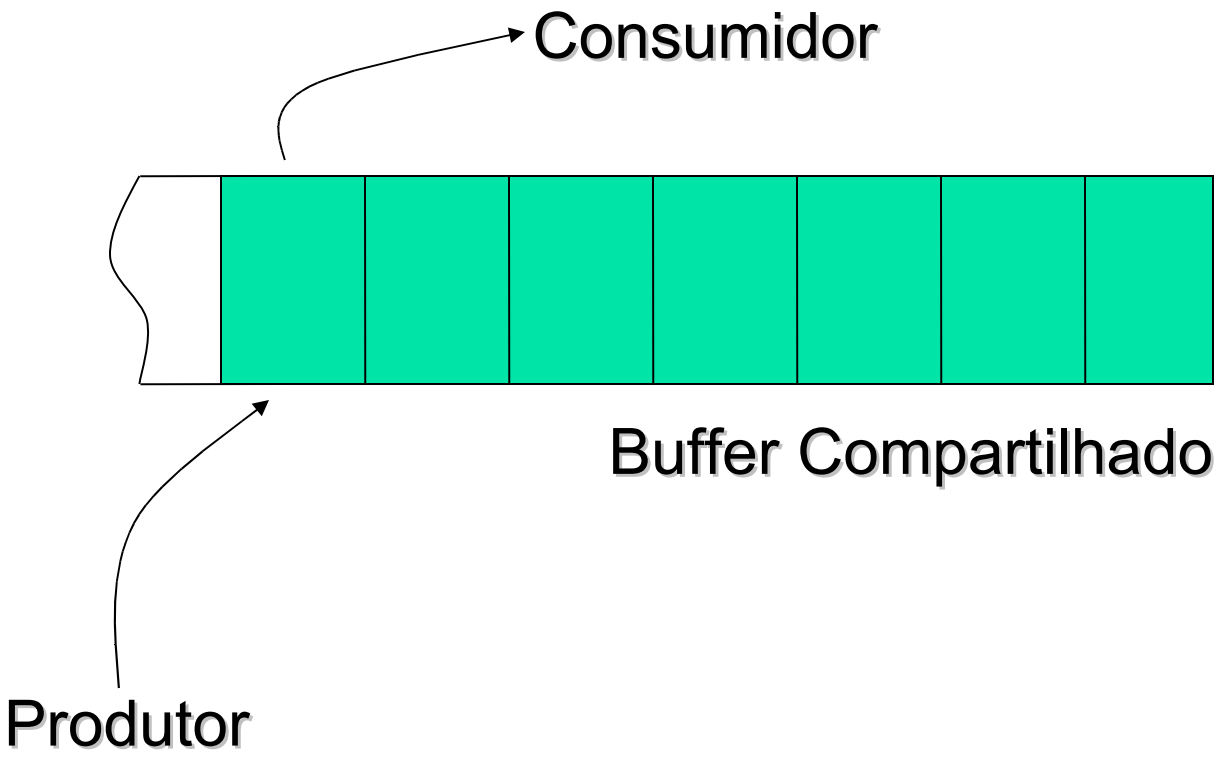
- Semáforos
- Monitores
- Troca de mensagens



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 currentidx;

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

Produtor / Consumidor (3/4)

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

    srand48(time());  currentidx = 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);
    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[currentidx++] = n;
     printf("Produzindo numero %d\n", n);
     sleep((int)(drand48() * 4.0));
    }
}
```

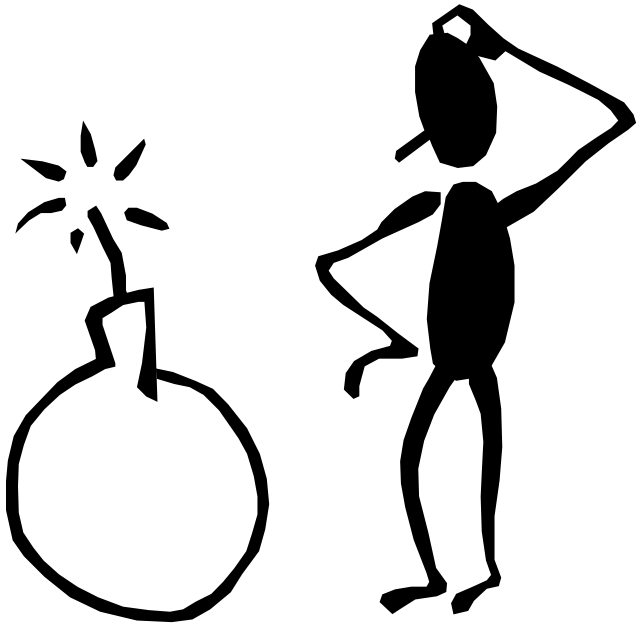
Produtor / Consumidor (4/4)

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

```
void *consumidor(void *arg) {
int n;
   while(1) {
      n = buffer[--currentidx];
      printf("Consumindo numero %d\n", n);
      sleep((int)(drand48() * 4.0));
   }
}
```

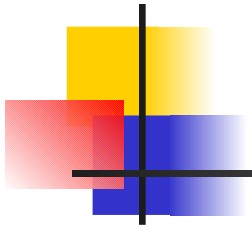
E de novo...

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 currentidx;

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



1a. Tentativa de Solução (2/4)

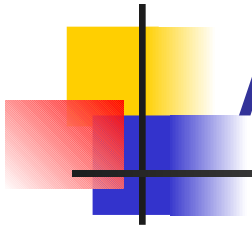
```
int main(int argc, char **argv) {
    int i;
    srand48(time());  currentidx = 0;
    pthread_mutex_init(&buffer_mutex, NULL);
    for(i=0; i<NUMCONS; i++)
        pthread_create(&(cons[i]), NULL, consumidor, NULL);
    for(i=0; i<NUMPROD; i++)
        pthread_create(&(prod[i]), NULL, produtor, 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[currentidx++] = n;
        pthread_mutex_unlock(&buffer_mutex);
        printf("Produzindo numero %d\n", n);
        sleep((int)(drand48() * 4.0));
    }
}
```

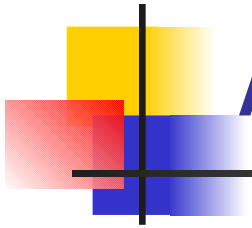
1a. Tentativa de Solução (4/4)

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



Agora sim...

Ficou correto?

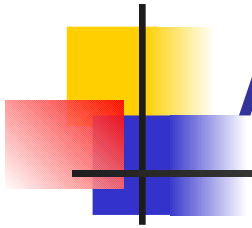


Agora sim...

Ficou correto?

Não!!!! Por quê?

Quem controla a ocupação do buffer?



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 currentidx;
sem_t buffer_full, buffer_empty;

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

2a. Tentativa de Solução (2/4)

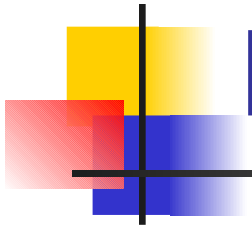
```
int main(int argc, char **argv) {
    int i;
    srand48(time());  currentidx = 0;
    pthread_mutex_init(&buffer_mutex, NULL);
    sem_init(&buffer_full, 0, BUFFERSIZE);
    sem_init(&buffer_empty, 0, 0);
    for(i=0; i<NUMCONS; i++) {
        pthread_create(&(cons[i]), NULL, consumidor, NULL);
    }
    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(&buffer_full);
        pthread_mutex_lock(&buffer_mutex);
        buffer[currentidx++] = n;
        pthread_mutex_unlock(&buffer_mutex);
        sem_post(&buffer_empty);
        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(&buffer_empty);
        pthread_mutex_lock(&buffer_mutex);
        n = buffer[--currentidx];
        pthread_mutex_unlock(&buffer_mutex);
        sem_post(&buffer_full);
        printf("Consumindo numero %d\n", n);
        sleep((int)(drand48() * 4.0));
    }
}
```



Barreira (1/3)

Arquivo barrier.h

```
typedef struct {
    pthread_mutex_t mutex;
    sem_t waitsem;
    int nthreads, current;
}barrier_t, *ptr_barrier_t;


void barrier_init(ptr_barrier_t, int);
void barrier(ptr_barrier_t);
```

Barreira (2/3)

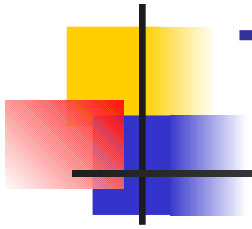
Arquivo barrier.c

```
void barrier_init(ptr_barrier_t pbarrier, int nt) {
    pbarrier->nthreads = nt;
        pbarrier->current = 0;
    pthread_mutex_init(&(pbarrier->mutex), NULL);
    sem_init(&(pbarrier->waitsem), 0, 0);
}
```

Barreira (3/3)

Arquivo barrier.c

```
void barrier(ptr_barrier_t pbarrier) {
    int i;
    pthread_mutex_lock(&(pbarrier->mutex));
    pbarrier->current++;
    if(pbarrier->current < pbarrier->nthreads) {
        pthread_mutex_unlock(&(pbarrier->mutex));
        sem_wait(&(pbarrier->waitsem));
    }else{
        for(i=0; i<(pbarrier->nthreads - 1); i++)
            sem_post(&(pbarrier->waitsem));
        pbarrier->current = 0;
        pthread_mutex_unlock(&(pbarrier->mutex));
    }
}
```



Tutorial de PThreads

https://computing.llnl.gov/tutorials/pthreads/