

Programação Concorrente

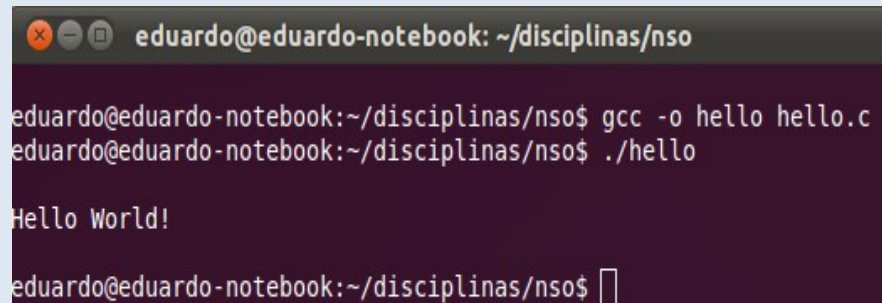
Pthreads

Prof. Eduardo Alchieri

Ferramentas

- Linguagem de programação C (Pthreads)
 - gcc – GNU Compiler Collection
- Qualquer editor de texto

```
hello.c  
  
#include <stdio.h>  
  
int main(){  
    printf ("\nHello World!\n\n");  
    return 1;  
}
```



A terminal window with a dark background and light text. The title bar shows 'eduardo@eduardo-notebook: ~/disciplinas/nso'. The terminal shows the following commands and output:

```
eduardo@eduardo-notebook:~/disciplinas/nso$ gcc -o hello hello.c  
eduardo@eduardo-notebook:~/disciplinas/nso$ ./hello  
  
Hello World!  
  
eduardo@eduardo-notebook:~/disciplinas/nso$
```

O Modelo POSIX Threads (Pthreads)

- Modelo que suporta a criação e manipulação de tarefas cuja execução possa ser intercalada ou executada concorrentemente.
- O modelo Pthreads pertence à família POSIX (*Portable Operating System Interface*) e define um conjunto de rotinas (biblioteca) para manipulação de threads.
- As definições da biblioteca Pthreads encontram-se em 'pthread.h' e sua implementação em 'libpthread.so'.
 - Para compilar um programa com threads é necessário incluir o cabeçalho '#include <pthread.h>' no início do programa e compilá-lo com a opção '-lpthread'.

Estruturas e Funções Usadas

- Biblioteca pthread.h
 - pthread_t (struct)
 - pthread_create
 - pthread_join
 - pthread_kill
 - pthread_exit
 - Outras: man pthreads

Criação de Threads

- Quando o programa inicia, uma thread (main thread) é criada. Após isso, outras threads podem ser criadas através da função:

```
int pthread_create(pthread_t *th, pthread_attr_t *attr, void *  
                  (*start_routine)          , void *arg);
```

- `pthread_create()` cria um novo *thread* que inicia a sua execução na função indicada por `start_routine` com o argumento indicado em `arg`. Em caso de sucesso instancia `th` com o identificador do novo *thread* e retorna 0, senão retorna um código de erro.

Criação de Threads

- Exemplo:

```
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);
    }
}
```

Criação de Threads

- Exemplo (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);
    }
}
```

Sincronização

- Como sincronizar as threads ?
 - Mutexes: `pthread_mutex_t` (struct) – sem. binário
`pthread_mutex_lock`
`pthread_mutex_unlock`
 - Operações:

```
int pthread_mutex_init(pthread_mutex_t *mutex,  
pthread_mutexattr_t *mutexattr);
```

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

```
int pthread_mutex_trylock(pthread_mutex_t *mutex);
```


Sincronização

- Como sincronizar as threads ?
 - Variáveis condição:
 - `pthread_cond_t` (estrutura)
 - Operações:
 - `pthread_cond_wait(pthread_cond_t * cond, pthread_cond_t * mutex)`
 - `pthread_cond_signal(pthread_cond_t * cond)`
 - `pthread_cond_broadcast(pthread_cond_t * cond)`

Sincronização

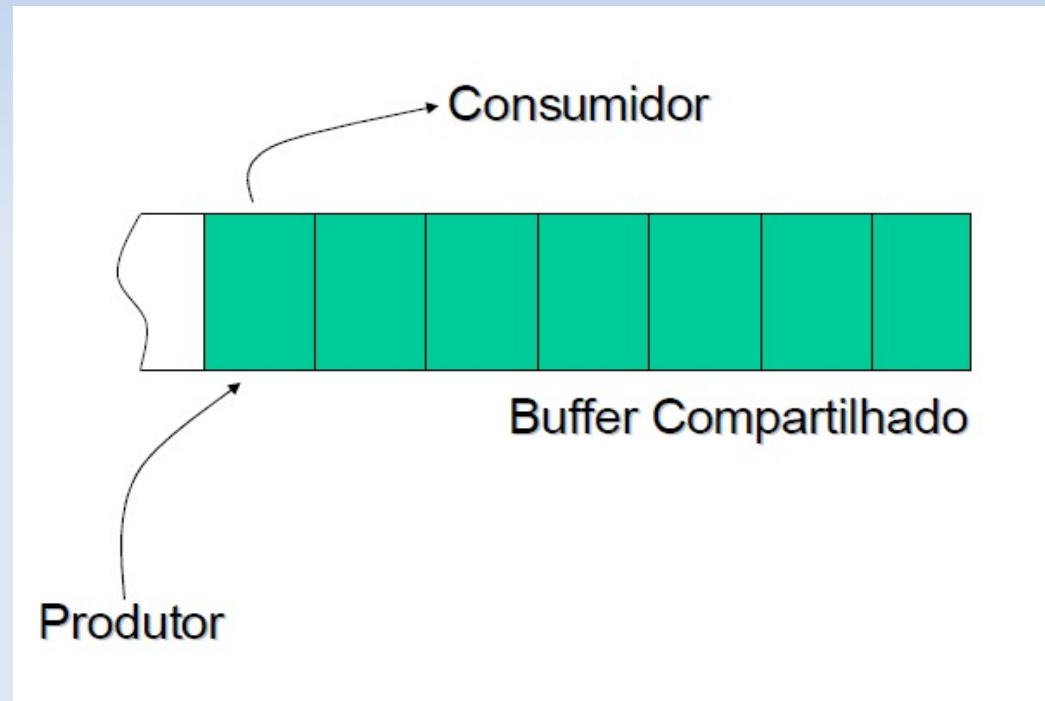
- Como sincronizar as threads ?
 - Semáforos ('semaphore.h'):

```
sem_t (struct) – sem. não binário  
sem_wait  
sem_post
```

- Operações:

```
int sem_init(sem_t *sem, int pshared, unsigned int value);  
int sem_wait(sem_t * sem);  
int sem_trywait(sem_t * sem);  
int sem_post(sem_t * sem);  
int sem_destroy(sem_t * sem);
```

Produtor/Consumidor



1ª Tentativa de solução

```
#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);
```

1ª Tentativa de solução

```
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);
    }
}
```

1ª Tentativa de solução

```
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));
    }
}
```

1ª Tentativa de solução

Qual é o problema com o programa anterior ?

2ª Tentativa de solução

```
#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);
```


2ª Tentativa de solução

```
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);
    }
}
```

2ª Tentativa de solução

```
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));
    }
}

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));
    }
}
```

2ª Tentativa de solução

Qual é o problema com o programa anterior ?

3ª Tentativa de solução

```
#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);
```

3ª Tentativa de solução

```
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);
    }
    ...
}
```

3ª Tentativa de solução

```
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));
    }
}

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));
    }
}
```

3ª Tentativa de solução

Qual é o problema com o programa anterior ?

- NENHUM! Agora está correto!!!