

Threads: Conceitos e Implementação

Renê de Souza Pinto – rene@renesp.com.br

8 de Abril de 2011

Licença



Threads: Conceitos e Implementação, por Renê de Souza Pinto – rene@renesp.com.br, é licenciado sob a Atribuição-Uso não-comercial-Compartilhamento pela mesma licença 3.0 Unported - http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt_BR

Índice

- 1 **Conceitos**
 - Introdução
 - Tipos de Threads

- 2 **Implementação**
 - Pthreads
 - Semáforos
 - Passagem de Mensagens
 - Monitores em Java

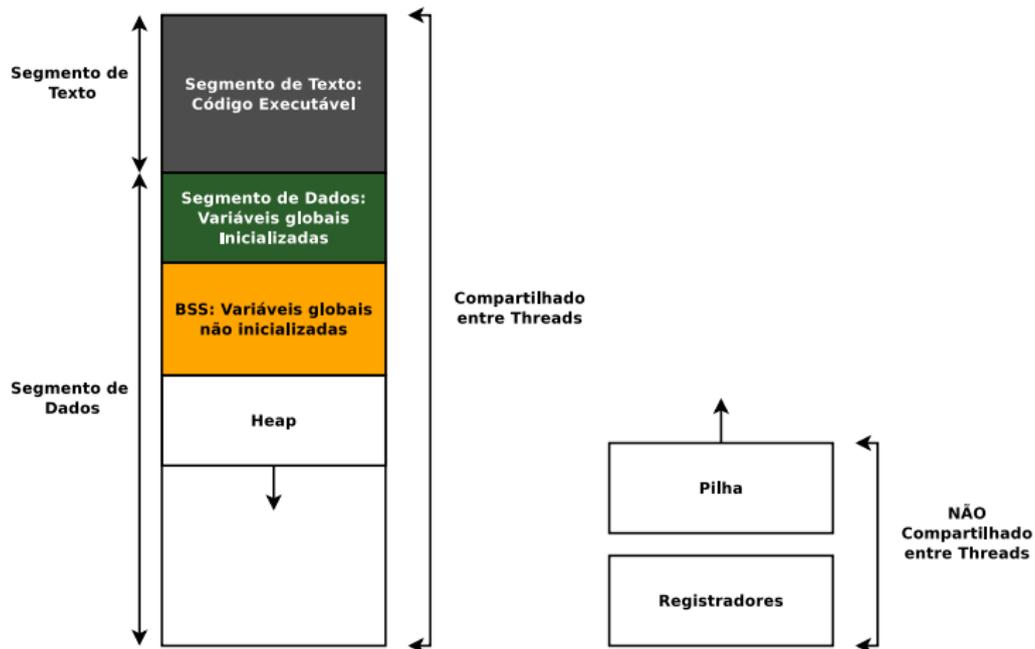
O que são Threads?

- Regiões de um processo:
 - **Segmento (ou seção) de texto:** Contém o código executável (instruções) do programa.
 - **Segmento (ou seção) de dados:** Contém variáveis globais inicializadas.
 - **BSS:** Contém variáveis globais não inicializadas.
 - **Heap:** Região utilizada para alocar memória ao processo (pode ser expandida ou contraída através das chamadas ao sistema `brk()` e `sbrk()`).
 - Algumas vezes **Seg. Dados + BSS + Heap** são simplesmente chamados de **Segmento de Dados**.
- **Pilha:** Região da memória utilizada para guardar dados dinâmicos (parâmetros de funções, variáveis locais, valores de retorno).

O que são Threads?

- **Threads:** Também chamado de processo leve.
 - Compartilha Regiões e Recursos[4]:
 - Variáveis globais, arquivos abertos, processos filhos, alarmes pendentes, sinais e manipuladores de sinais, informações de conta.
 - Itens **NÃO** compartilhados: Pilha, Contador de Programa, Registradores, Estado do processo (Thread)
- **Multithreading:** Sistemas suportam múltiplas Threads.

O que são Threads?



Tipos de Threads

- Threads podem ser implementadas no **Espaço do kernel** ou no **Espaço do Usuário**.

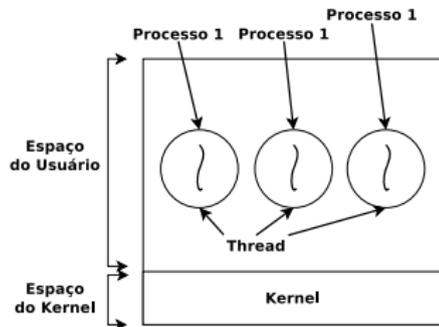


Figura : Threads em Espaço do Kernel

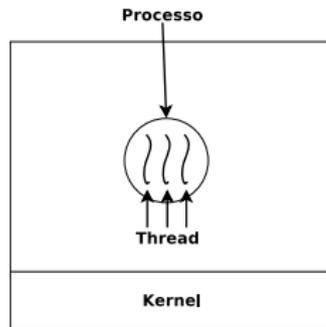


Figura : Threads em Espaço do Usuário

Adaptado de [4]

Tipos de Threads

- Threads no Espaço de Kernel:
 - São enxergadas pelo kernel, que tem controle sobre cada Thread.
 - São escalonadas pelo kernel da mesma maneira que os processos.
 - Podem bloquear através de chamadas ao sistema sem atrapalhar outras Threads do mesmo pai.
 - Efetuar uma chamada ao sistema é substancialmente custoso, assim se operações de Threads (criação, término, etc) são comuns, o overhead pode ser grande[4].

Tipos de Threads

- Threads no Espaço de Usuário:
 - Somente o processo o processo pai (*Run-time system*) é enxergado pelo kernel.
 - Podem ser implementadas em SOs que não suportam Threads.
 - Permitem que as Threads sejam escalonada por diferentes algoritmos.
 - Chavear entre as Threads em espaço do usuário é mais rápido (menos custoso).
 - Se uma Thread efetua uma chamada ao sistema e é bloqueada pelo kernel (devido a uma espera de I/O por exemplo), todas as outras Threads irão bloquear (Kernel só enxerga 1 processo).

Tipos de Threads

- *Run-time system* deve guardar tabela de Threads (registradores, PC, etc).
- Chaveamento entre as Threads:
 - Voluntário: A própria Thread sai de execução (eficiente, porém uma Thread pode ficar executando indefinidamente).
 - Feito pelo *Run-time system*, deve atuar periodicamente (custoso).

Implementação

Como Implementar Threads?

Implementação

- Em Sistemas que suportam Threads (como o Linux), podemos criá-las diretamente através de chamadas ao sistema (no Linux, a chamada ao sistema *clone()* pode ser utilizada para se criar Threads).
- Threads em Espaço do Usuário: Através do uso de bibliotecas (Por exemplo, **GNU Pth**¹).
- Independente do tipo, é comum o uso de bibliotecas para facilitar a criação, manuseio e manipulação de Threads.

¹<http://www.gnu.org/software/pth>

Pthreads

POSIX threads: Padrão *POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995)*:

- Define uma API para criação e manipulação de Threads.
- Bibliotecas que implementam a API do padrão **POSIX threads** são chamadas de *Pthreads*.
- A **GLIBC** (biblioteca C da GNU) implementa Pthreads.
- Para Windows, existe a biblioteca **Pthreads-w32**.

Pthreads

Estrutura e funções básicas:

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

Pthreads

pthread_create : Cria uma nova Thread

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

- **thread**: Estrutura para a Thread criada.
- **attr**: Atributos da nova Thread (NULL para atributos padrão).
- **start_routine**: Rotina que será executada pela Thread quando iniciada.
- **arg**: Argumento a ser passado para a rotina de início (*start_routine*).
- Retorna 0 se houve sucesso, ou o código do erro, caso contrário.

Pthreads

pthread_join : Aguarda o término de alguma Thread

```
int pthread_join(pthread_t thread, void **retval);
```

- Esta função aguarda pelo término da Thread especificada no argumento **thread**. Se a mesma já terminou sua execução quando a função foi chamada, então a função retorna imediatamente.
- **retval**: Se não for NULL, este argumento conterà o código de retorno da Thread que estava sendo aguardada.
- Retorna 0 se houve sucesso, ou o código do erro, caso contrário.

Pthreads

pthread_kill : Envia um sinal a alguma Thread

```
int pthread_kill(pthread_t thread, int sig);
```

- **thread**: Thread a receber o sinal.
- **sig**: Sinal a ser enviado. Por exemplo, **SIGSTOP**.
- Retorna 0 se houve sucesso, ou o código do erro, caso contrário.

Pthreads

pthread_exit : Finaliza a Thread

```
void pthread_exit(void *retval);
```

- **retval**: Código de retorno da Thread.

Pthreads - Como compilar?

Como compilar um programa que usa Pthreads?

- Nos fontes:

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

- Para compilar (use o parâmetro `-pthread`):

```
gcc foo.c -o foo -pthread
```

Pthreads - Exemplo

```
void *diz_quem_eh(void *nome);

int main(int argc, const char *argv[])
{
    pthread_t ths[4];
    int i;
    char *nomes[] = {"A", "B", "C", "D"};

    for (i = 0; i < 4; i++) {
        pthread_create(&ths[i], NULL, diz_quem_eh, (void*) nomes[i]);
    }

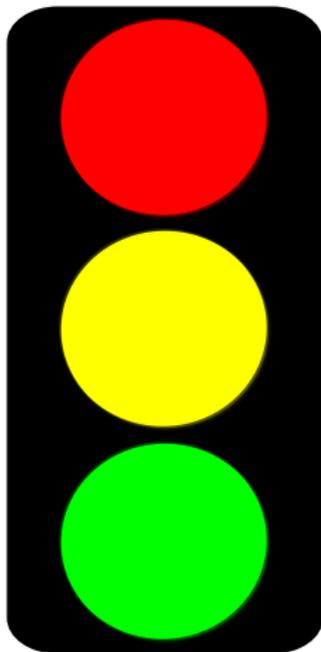
    /* Faz um join em todas as Threads para aguardar
     * o término de todas elas */
    for (i = 0; i < 4; i++) {
        pthread_join(ths[i], NULL);
    }

    printf("Todas as threads terminaram!\n");
    return 0;
}
```

Pthreads - Exemplo

```
/**
 * Função que será executada pelas Threads
 */
void *diz_quem_eh(void *nome)
{
    /* Diz quem é */
    printf("Eu sou o %s\n", (char*)nome);
}
```

Semáforos



Semáforos

Como Implementar Semáforos?

Pthreads

Estrutura e Funções básicas:

- **sem_t** (struct)
- **sem_init**
- **sem_wait**
- **sem_post**
- **sem_destroy**

Semáforos

sem_init : Inicializa um semáforo sem nome

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

- **sem**: Estrutura para o semáforo a ser criado.
- **pshared**: 0 indica que o semáforo é compartilhado somente entre o processo e suas threads. Qualquer valor diferente de zero indica que o semáforo é compartilhado entre processos.
- **value**: Valor inicial do semáforo.
- Retorna 0 se houve sucesso, ou -1 caso contrário, setando a variável *errno* para indicar o erro.

Semáforos

sem_wait : Efetua um *down()* no semáforo

```
int sem_wait(sem_t *sem);
```

```
int sem_trywait(sem_t *sem);
```

```
int sem_timedwait(sem_t *sem, const struct timespec *abs_timeout
```

- **sem**: Semáforo.
- **sem_wait**: Se o semáforo estiver em 0 bloqueia até que o valor seja incrementado.
- **sem_trywait**: Idem a **sem_wait**, porém se o semáforo estiver em 0, retorna um erro ao invés de bloquear.
- **sem_timedwait**: Idem a **sem_wait**, porém especifica um tempo máximo para ficar bloqueado aguardando o semáforo ser liberado.
- Todas retornam 0 se houve sucesso, ou -1 caso contrário, setando a variável *errno* para indicar o erro e deixando o valor do semáforo intocado.

Semáforos

sem_post : Efetua um *up()* no semáforo

```
int sem_post(sem_t *sem);
```

- **sem**: Semáforo.
- Retorna 0 se houve sucesso, ou -1 caso contrário, setando a variável *errno* para indicar o erro e deixando o valor do semáforo intocado.

Semáforos

sem_destroy : Finaliza um semáforo criado com *sem_init*

```
int sem_destroy(sem_t *sem);
```

- **sem**: Semáforo a ser finalizado.
- Retorna 0 se houve sucesso, ou -1 caso contrário, setando a variável *errno* para indicar o erro.

Semáforos - Como compilar?

Como compilar um programa que usa semáforos?

- Nos fontes:

```
#include <semaphore.h>
```

- Para compilar (use o parâmetro `-lrt` ou `-pthread`):

```
gcc foo.c -o foo -lrt
```

```
gcc foo.c -o foo -pthread
```

Semáforos - Exemplo

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

#define TAM 5      /* Tamanho do Buffer */
#define N_PROD 2  /* Quantidade de produtores */
#define N_CONS 45 /* Quantidade de consumidores */

/* Protótipos */
void *producer(void *arg);
void *consumer(void *arg);
void insert_item(char item);
char remove_item();

/* Semáforos */
sem_t mutex, empty, full;

/* Buffer */
char buffer[TAM];
int buffer_pos = 0;
```

Semáforos - Exemplo

```
int main(int argc, const char *argv[])
{
    pthread_t prods[N_PROD], cons[N_CONS];
    int i;

    sem_init(&mutex, 0, 1);
    sem_init(&empty, 0, TAM);
    sem_init(&full, 0, 0);

    /* Cria as Threads de produtores */
    for (i = 0; i < N_PROD; i++) {
        pthread_create(&prods[i], NULL, producer, NULL);
    }
    /* Cria as Threads de consumidores */
    for (i = 0; i < N_CONS; i++) {
        pthread_create(&cons[i], NULL, consumer, NULL);
    }

    /* Faz o join em todas as threads */
    for (i = 0; i < N_PROD; i++) {
        pthread_join(prods[i], NULL);
    }
    for (i = 0; i < N_CONS; i++) {
        pthread_join(cons[i], NULL);
    }

    return 0;
}
```

Semáforos - Exemplo

```
/** Produtor */
void *producer(void *arg)
{
    char item;

    while(1) {
        item = 'A';
        sem_wait(&empty);
        sem_wait(&mutex);
        insert_item(item);
        sem_post(&mutex);
        sem_post(&full);
    }
}

/** Consumidor */
void *consumer(void *arg)
{
    char item;

    while(1) {
        sem_wait(&full);
        sem_wait(&mutex);
        item = remove_item();
        sem_post(&mutex);
        sem_post(&empty);
        printf("Consumi: %c\n", item); /* consome item */
    }
}
```

Semáforos - Exemplo

```
/* Insere no buffer */
void insert_item(char item)
{
    printf("Produzi: %c\n", item);
    buffer[buffer_pos++] = item;
}

/* Remove do buffer */
char remove_item()
{
    return buffer[buffer_pos--];
}
```

Passagem de Mensagens



Mecanismos de IPC no Unix

Mecanismos de Comunicação Interprocesso no Unix

- Filas de mensagens
- Semáforos
- Memória compartilhada

Dois Padrões:

- System V IPC: Interface original do *Unix System V*.
- POSIX.1-2001: Padrão mais moderno. Todas as funções abordadas anteriormente (manipulação de threads, semáforos), fazem parte deste padrão.

Passagem de Mensagens

Como Implementar Passagem de Mensagens?

Passagem de Mensagens

Estrutura e Funções básicas (Padrão POSIX.1-2001):

- **mqd_t** (struct)
- **mq_open**
- **mq_send**
- **mq_receive**
- **mq_close**

Passagem de Mensagens

mq_open : Cria uma fila de mensagens ou abre uma existente

```
mqd_t mq_open(const char *name, int oflag);  
mqd_t mq_open(const char *name, int oflag, mode_t mode,  
              struct mq_attr *attr);
```

- **name**: Nome da fila. Deve estar no formato: /<nome>
- **oflag**: Flags de abertura (O_RDONLY, O_WRONLY, etc).
- **mode**: Se O_CREAT foi especificado em *oflag*, então *mode* especifica as permissões a serem estabelecidas para a nova fila.
- **attr**: Atributos da fila.
- Retorna o novo descritor se houve sucesso, ou (mqd_t)-1 caso contrário, setando a variável *errno* para indicar o erro.

Passagem de Mensagens

mq_send : Envia uma mensagem para a fila

```
int mq_send(mqd_t mqdes, const char *msg_ptr,  
            size_t msg_len, unsigned msg_prio);
```

```
int mq_timedsend(mqd_t mqdes, const char *msg_ptr,  
                 size_t msg_len, unsigned msg_prio,  
                 const struct timespec *abs_timeout);
```

- **mqdes**: Descritor da fila.
- **msg_ptr** e **msg_len**: Buffer e tamanho do buffer, respectivamente.
- **msg_prio**: Inteiro não negativo que indica a prioridade da mensagem. As mensagens são colocadas na fila em ordem decrescente de prioridade.
- Retorna 0 se houve sucesso, ou -1 caso contrário, setando a variável *errno* para indicar o erro.

Passagem de Mensagens

Notas importantes:

- Se a fila de mensagens estiver cheia, **mq_send()** bloqueia até que haja espaço suficiente na fila. Se a fila foi aberta com a flag *O_NONBLOCK*, então a chamada não bloqueia, retornando apenas um erro.
- **mq_timedsend()** comporta-se como **mq_send()**, exceto quando a fila está cheia e a flag *O_NONBLOCK* não foi especificada. Neste caso, a chamada será bloqueada somente pelo tempo especificado por *abs_timeout*.

Passagem de Mensagens

mq_receive : Recebe uma mensagem da fila de mensagens

```
ssize_t mq_receive(mqd_t mqdes, char *msg_ptr,  
                  size_t msg_len, unsigned *msg_prio);
```

```
ssize_t mq_timedreceive(mqd_t mqdes, char *msg_ptr,  
                       size_t msg_len, unsigned *msg_prio,  
                       const struct timespec *abs_timeout);
```

- **mqdes**: Descritor da fila.
- **msg_ptr** e **msg_len**: Buffer e tamanho do buffer, respectivamente.
- **msg_prio**: Se não é NULL, recebe o valor de prioridade da mensagem recebida.
- Retorna o número de bytes recebidos. Em caso de erro, retorna -1, setando a variável *errno* para indicar o erro.

Passagem de Mensagens

Notas importantes:

- Se a fila de mensagens estiver vazia, **mq_receive()** bloqueia até que alguma mensagem chegue na fila. Se a fila foi aberta com a flag *O_NONBLOCK*, então a chamada não bloqueia, retornando apenas um erro.
- **mq_timedreceive()** comporta-se como **mq_receive()**, exceto quando a fila está vazia e a flag *O_NONBLOCK* não foi especificada. Neste caso, a chamada será bloqueada somente pelo tempo especificado por *abs_timeout*.

Passagem de Mensagens

mq_close : Finaliza uma fila de mensagens aberta

```
int mq_close(mqd_t mqdes);
```

- **mqdes**: Descritor da fila a ser finalizada.
- Retorna 0 se houve sucesso, ou -1 caso contrário, setando a variável *errno* para indicar o erro.

Passagem de Mensagens

mq_unlink : Remove uma fila de mensagens criada

```
int mq_unlink(const char *name);
```

- **name**: Nome da fila a ser removida.
- Retorna 0 se houve sucesso, ou -1 caso contrário, setando a variável *errno* para indicar o erro.

Passagem de Mensagens - Como compilar?

Como compilar um programa que usa fila de mensagens?

- Nos fontes:

- Para **mq_timedsend** e **mq_timedreceive**:

```
#include <time.h>
#include <mqueue.h>
```

- Para **mq_open**:

```
#include <fcntl.h>      /* For O_* constants */
#include <sys/stat.h>   /* For mode constants */
#include <mqueue.h>
```

- Para **mq_send**, **mq_receive** e **mq_close**:

```
#include <mqueue.h>
```

- Para compilar (use o parâmetro **-lrt**):

```
gcc foo.c -o foo -lrt
```

Passagem de Mensagens - Exemplo

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <pthread.h>
#include <fcntl.h>          /* For O_* constants */
#include <sys/stat.h>      /* For mode constants */
#include <mqueue.h>
#include <unistd.h>

#include <errno.h>

#define MQ_PROD_NAME "/produtor" /* Nome da fila de mensagens do produtor */
#define MQ_CONS_NAME "/consumidor" /* Nome da fila de mensagens do consumidor */

#define TAM 5 /* Tamanho do Buffer */
#define N_PROD 2 /* Quantidade de produtores */
#define N_CONS 5 /* Quantidade de consumidores */

#define MSG_VAZIA "VAZIO" /* Mensagem vazia */
#define MSG_CHEIA "ITEM" /* Mensagem cheia */
#define MSG_PRIO 0 /* Prioridade da mensagem */

/* Protótipos */
void *producer(void *arg);
void *consumer(void *arg);
void exit_prog(int signum);
```

Passagem de Mensagens - Exemplo

```
int main(int argc, const char *argv[])
{
    pthread_t prods[N_PROD];
    pthread_t cons[N_CONS];
    int i;

    signal(SIGINT, exit_prog);

    /* Inicia filas de mensagens */
    if ((msgq_produzidor = mq_open(MQ_PROD_NAME, O_RDWR | O_CREAT, S_IRWXU, NULL)) < 0 ||
        (msgq_consumidor = mq_open(MQ_CONS_NAME, O_RDWR | O_CREAT, S_IRWXU, NULL)) < 0) {
        perror("Erro ao iniciar fila(s) de mensagens!\n");
        return(EXIT_FAILURE);
    } else {
        /* Pega atributos padrão */
        mq_getattr(msgq_produzidor, &attr_prod);
        mq_getattr(msgq_consumidor, &attr_cons);
    }

    /* Cria as Threads de produtores */
    for (i = 0; i < N_PROD; i++) {
        pthread_create(&prods[i], NULL, producer, NULL);
    }

    /* Cria as Threads de consumidores */
    for (i = 0; i < N_CONS; i++) {
        pthread_create(&cons[i], NULL, consumer, NULL);
    }
}
```

Passagem de Mensagens - Exemplo

```
/* Faz o join em todas as threads */
for (i = 0; i < N_PROD; i++) {
    pthread_join(prods[i], NULL);
}
for (i = 0; i < N_CONS; i++) {
    pthread_join(cons[i], NULL);
}

return 0;
}

/* Finaliza o programa */
void exit_prog(int signum)
{
    if (signum == SIGINT) {
        mq_close(msgq_produzidor);
        mq_close(msgq_consumidor);
        mq_unlink(MQ_PROD_NAME);
        mq_unlink(MQ_CONS_NAME);
        printf("Filas de mensagens finalizadas!\n");
        kill(0, SIGTERM);
    }
}
```

Passagem de Mensagens - Exemplo

```

/** Produtor */
void *producer(void *arg)
{
    char *item;
    char *msg;
    int prio;

    if ((msg=calloc(attr_prod.mq_msgsize, sizeof(char))) == NULL)
        return NULL;

    while(1) {
        /* Produz item */
        item = MSG_CHEIA;
        printf("Produzi: %s\n", MSG_CHEIA);

        /* Recebe mensagem vazia enviada pelo consumidor */
        mq_receive(msgq_produzidor, msg, attr_prod.mq_msgsize, &prio);

        /* Constrói a mensagem e envia o item para o consumidor */
        strcpy(msg, MSG_CHEIA);
        mq_send(msgq_consumidor, msg, attr_cons.mq_msgsize, MSG_PRIO);
    }

    return NULL;
}

```

Passagem de Mensagens - Exemplo

```
/** Consumidor */
void *consumer(void *arg)
{
    int i, prio;
    char *msg, *item;

    if ((msg=calloc(attr_cons.mq_msgsize, sizeof(char))) == NULL)
        return NULL;

    /* Envia TAM vazios para o produtor */
    strcpy(msg, MSG_VAZIA);
    for (i = 0; i < TAM; i++)
        mq_send(msgq_produto, msg, attr_prod.mq_msgsize, MSG_PRIO);

    while(1) {
        /* Recebe item enviado pelo produtor */
        mq_receive(msgq_consumidor, msg, attr_cons.mq_msgsize, &prio);

        /* Extraí o item */
        item = msg;
        printf("Consumi: %s\n", item);

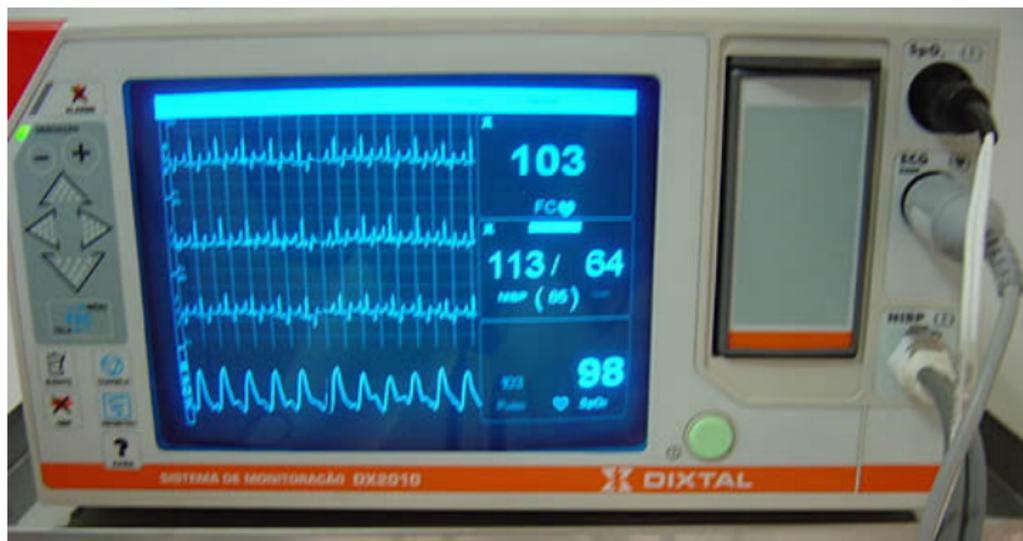
        /* Envia mensagem vazia de volta ao produtor */
        strcpy(msg, MSG_VAZIA);
        mq_send(msgq_produto, msg, attr_prod.mq_msgsize, MSG_PRIO);
    }

    return NULL;
}
```

Passagem de Mensagens - Dicas

- **mq_receive**: Utilizar *msg_len* diferente do tamanho máximo da mensagem (estipulado pelo atributo *mq_msgsize*) pode gerar perdas de mensagens.
- Os atributos da fila de mensagens pode ser lidos e alterados com **mq_getattr()** e **mq_setattr()**, respectivamente.
- Maiores informações, consulte o manual: **man mq_overview**
- Alguns códigos mostrados anteriormente não incluíam código para limpeza (finalização de semáforos, etc). Lembre-se de limpar a *bagunça* antes de finalizar seu programa!

Monitores



Monitores em Java

- No Java todo objeto possui um monitor associado.
- Fácil utilização.
- Através da palavra chave **synchronized**
 - Pode ser utilizada na declaração de métodos ou no início de blocos.

Monitores em Java - Exemplo

```
// Exemplo tirado do livro Sistemas Operacionais - 2a edicao - Tanenbaum
import java.util.Random;

public class ProducerConsumer
{
    static final int N = 10; //tamanho do buffer
    static producer p = new producer(); //instância de um novo thread produtor
    static consumer c = new consumer(); //instância de um novo thread consumidor
    static our_monitor mon = new our_monitor(); //instância de um novo monitor
    static Random rn = new Random(500);

    static class producer extends Thread //thread produtor
    {
        private int produce_item() //gera numero randomico que sera inserido no buffer
        {
            int val;
            val = rn.nextInt() % 100;
            System.out.println("Produzindo o item: " + val);
            return val;
        }

        public void run() //método run contém o código do thread produtor
        {
            int item;
            while(true)
            {
                item = produce_item(); //produz item q vai ser inserido no buffer
                mon.insert(item); //chama metodo insert do monitor
            }
        }
    }
}
```

Monitores em Java - Exemplo

```

        try {
            sleep(1);
        } catch (InterruptedException e) {
            e.printStackTrace(); // TODO Auto-generated catch block
        }
    }
}

static class consumer extends Thread //Thread consumidor
{
    private void consume_item(int item) //Consome item do buffer
    {
        System.out.println("Consumido o item: " + item);
    }

    public void run() //método run contém código do thread consumidor
    {
        int item;
        while(true)
        {
            item = mon.remove(); //chama metodo remove do monitor
            consume_item(item); //consome item
            try {
                sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace(); // TODO Auto-generated catch block
            }
        }
    }
}

```

Monitores em Java - Exemplo

```

static class our_monitor
{
    private int buffer[] = new int[N];
    private int count = 0, lo = 0, hi =0; //count->contador de itens no buffer, lo->posição que conter

    private void go_to_sleep()
    {
        try {
            wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    //synchronized garante que somente uma tread terá acesso a região crítica (método) por vez
    public synchronized void insert(int val)
    {
        if(count == N) //se buffer estiver cheio, vá dormir
        {
            go_to_sleep();
            System.out.println("Produtor dormindo");
        }
        buffer[hi] = val; //insere item na posicao hi do buffer
        hi = (hi + 1) % N; //posição onde sera inserido o proximo item
        count = count + 1; //mais um item no buffer
        if(count == 1) //se consumidor estava dormindo, acorde-o
        {

```

Monitores em Java - Exemplo

```

        System.out.println("Consumidor acordado");
        notify();
    }
}

public synchronized int remove()
{
    int val;
    if(count == 0) //se buffer estiver vazio, vá dormir
    {
        go_to_sleep();
        System.out.println("Consumidor dormindo");
    }
    val = buffer[lo]; //busca valor na posição lo do buffer
    lo = (lo + 1) % N; //posição onde será buscado o próximo item
    count = count - 1; //menos um item no buffer
    if(count == N - 1) //se produtor estava dormindo, acorde-o
    {
        System.out.println("Produtor acordado");
        notify();
    }
    return val;
}
}

```

Monitores em Java - Exemplo

```
public static void main(String args[])  
{  
    p.start(); //inicia o thread produtor  
    c.start(); //inicia o thread consumidor  
}
```

Finalizando...

- Para informações sobre funções C, consulte as páginas de manual:
 - standards, sem_overview, mq_overview
- Não deixe de consultar as referências: [4], [3], [2], [1]

Ufa! Por hoje é só!

Referências I

-  [Linux Manual Pages.](#)
-  [Maurice J. Bach.](#)
The design of the Unix Operating System.
Prentice-Hall, 1986.
-  [Andrew S. Tanenbaum.](#)
Sistemas Operacionais: projeto e implementação.
Bookman, 2000.
-  [A.S. Tanenbaum, A.L.G. Ronaldo, and L.A. Consularo.](#)
Sistemas operacionais modernos.
Pearson Prentice Hall, 2006.