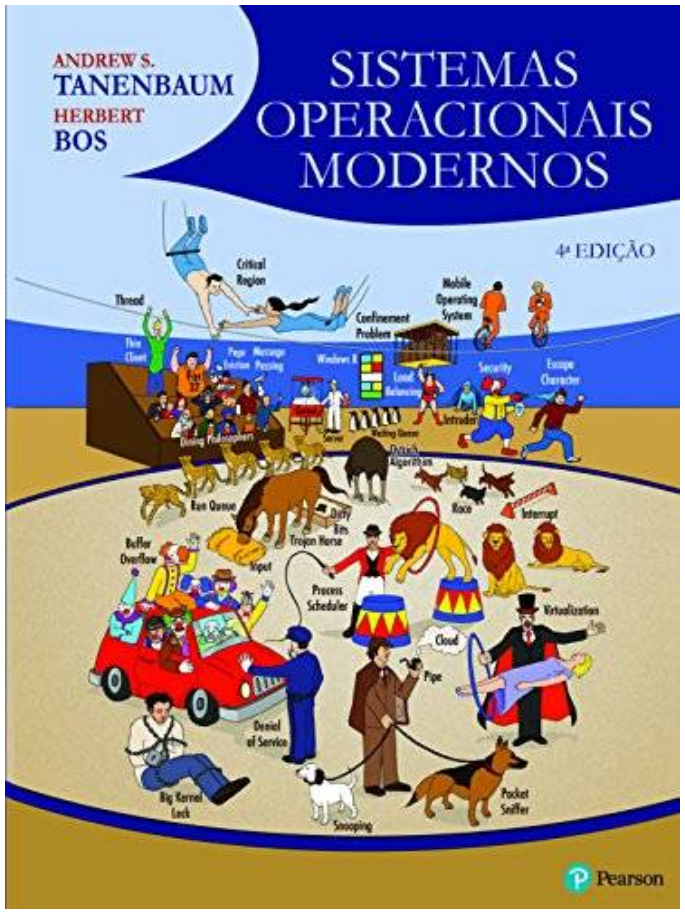
SISTEMAS OPERACIONAIS

Eduardo Ono

eduardo.ono@unisal.br

Bibliografia



TANENBAUM, Andrew S.; BOS, Herbert. Sistemas operacionais modernos. 4. ed. São Paulo: Pearson Education do Brasil, 2016.

- https://bv4.digitalpages.com.br/#/legacy/36876
- https://archive.org/details/SistemasOperacionaisModernosTanenbaum4Edio/page/n3

Sistemas: Conceitos

Definição: Um sistema é um conjunto de elementos interdependentes de modo a formar um todo organizado.

Todo sistema possui um objetivo geral a ser atingido.

Ex.: Sistema digestivo, GPS (Sistema de Posicionamento Global), freio ABS (Sistema "Anti-Blocagem"), etc.

Sistemas Computacionais

Definição: Um Sistema Computacional é um conjunto de dispositivos eletrônicos (hardware) capaz de processar informações de acordo com um programa (software).

Sistema Operacional

Definição: Um Sistema Operacional é um conjunto de gerenciadores (elementos) interdependentes de modo a formar um gerenciador organizado para sistemas computacionais.

Um dos objetivos de um Sistema Operacional é fornecer uma interface para que o usuário tenha acesso aos recursos de hardware de um sistema computacional.

Ex.: Microsoft Windows, Ubuntu (Linux), Android (smartphones).

Terminologias: Recurso

Recurso do Sistema

- Em computação, um recurso do sistema, ou simplesmente recurso, é qualquer componente físico ou virtual de disponibilidade limitada em um sistema computacional. Todo componente interno ao sistema computacional é um recurso.

Terminologias: Handle

- Handle
  - Def.: Em computação/programação, um *handle* é uma referência abstrata para um recurso.
  - As *handles* são utilizadas quando aplicativos referenciam blocos de memória ou objetos gerenciados por um outro sistema, por exemplo, banco de dados ou sistema operacional.

Exemplo (1) de “Handle”

```
#include <stdio.h>
#include <string.h>

int main( )
{
  FILE *handle;
  char buffer[1024];

  handle = fopen( "ARQUIVO.TXT", "r" );
  while ( strcmp( buffer, "\n" )!=1 )
  {
    fgets( buffer, 1024, handle );
    printf( handle, "%s\n", buffer);
  }
  fclose( handle );
}
```

Exemplo (2) de Handle

```
#include <stdio.h>

int main( )
{
     FILE *handle;
     char buffer[128];

     handle = fopen("ARQUIVO.TXT", "w");
     strcpy(buffer, "Exemplo de uma linha.");
     fprintf(handle, "%s\n", buffer);
     fclose(handle);
}
```

Argumentos de Linha de Comando

```
#include <stdio.h>

int main( int argc, char* argv[] )
{
      int i;
      printf( "Quantidade de argumentos: %d\n",
                                          argc );
      printf( "Executavel: %s\n", argv[0] );
      printf( "Argumentos: " );
      for ( i=1; i<argc; i++ )
            printf( "%s ", argv[i] );
      return 0;
}
```

S.O.: Processos

- No contexto de S.O., *processo* é uma instância de um programa em execução. É um módulo executável único, que corre concorrentemente com outros módulos executáveis.
- *Processos* são módulos separados e carregáveis (ao contrário de *threads*).

Processos

Em geral, processos são formados pelos seguintes recursos:

- Uma imagem do código de máquina executável associado a um programa;
- Memória, que inclui o código executável;
- Descritores de S.O. que são alocados ao processo, como descritores de arquivos (“handles”, conforme a terminologia do Windows).
- Atributos de segurança.
- Etc.

Processos: Criação

Um processo “pai” cria processos “filhos”, que, por sua vez, podem criar outros processos, formando uma árvore de processos.

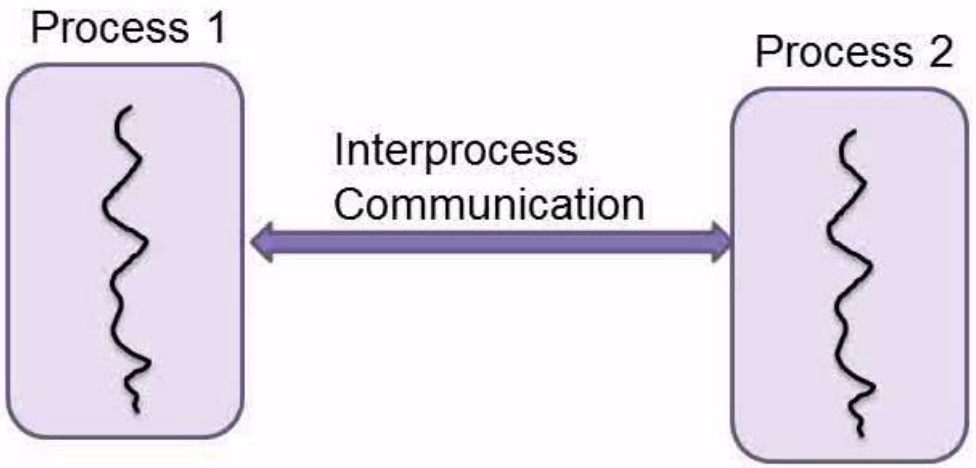
- Geralmente, processos são gerenciados através de um identificador de processo (PID).

Execução

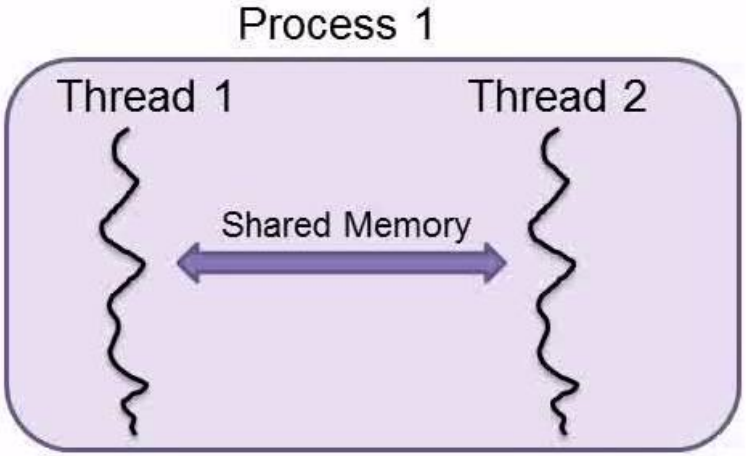
- Pai e filho executam concorrentemente.
- Pai aguarda a conclusão da execução do filho.

Processos vs. Threads

Multiprocessing:



Multithreading:



Pros:
- Fast to start
- Low overhead

Cons:
- Difficult to implement
- Can't run on distributed system

Threads

- *Threads*
  - Def.: Linha de execução ou Encadeamento de execução é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrencialmente.

  - O suporte à thread é fornecido pelo próprio S.O., no caso da linha de execução ao nível do núcleo, ou implementada através de uma biblioteca de uma determinada linguagem.

Threads do Usuário

Gerenciamento de threads feita por bibliotecas ("libraries") carregadas a nível do usuário.

Exemplos de três bibliotecas de threads:

- POSIX pthreads
- Win32 threads
- Java threads

Threads: Padrão POSIX

Portable Operating System Interface (POSIX) é uma família de padrões especificadas pela IEEE Computer Society.

Objetivo

- Manter a compatibilidade entre sistemas operacionais. O padrão POSIX define uma API, bem como comandos (*shell*) para compatibilidade entre variantes do Unix e outros sistemas operacionais.

Passando parâmetros para a função *pthread_create*

Algoritmo de Peterson

```
#define FALSE  0
#define TRUE   1
#define N      2                        /* número de processos */

int turn;                               /* de quem é a vez? */
int interested[N];                      /* todos os valores 0 (FALSE) */

void enter_region(int process);         /* processo é 0 ou 1 */
{
     int other;                         /* número de outro processo */

     other = 1 - process;               /* o oposto do processo */
     interested[process] = TRUE;        /* mostra que você está interessado */
     turn = process;                    /* altera o valor de turn */
     while (turn == process && interested[other] == TRUE) /* comando nulo */ ;
}

void leave_region(int process)          /* processo: quem está saindo */
{
     interested[process] = FALSE;       /* indica a saída da região crítica */
}
```

Fonte: Tanenbaum, Sistemas Operacionais Modernos, 2009.

Algoritmo de Peterson

```
bool t1_quer_entrar, t2_quer_entrar;
int thread_preferencial;

void thread1()
{
        while ( 1 )
        {
                t1_quer_entrar = true;
                thread_preferencial = 2;
                while ( t2_quer_entrar && thread_preferencial==2 );
                // Região crítica
                // ...
                // Região normal
                t1_quer_entrar = false;
                // Outras tarefas
        }
}
```

Segurança (Cap. 9)

Segurança: Metas e Ameaças

Meta	Ameaça
Confidencialidade de dados	Exposição de dados
Integridade de dados	Manipulação de dados * Corrupção de dados
Disponibilidade do sistema	Recusa de serviços
Exclusão de invasores	Controle do sistema por vírus ** Sequestro do sistema

** Info Exame, 51% das empresas no Brasil já sofreram sequestro de sistemas, 13.03.2017.

Ameaças (malwares)

Malwares:

- Cavalos de Tróia
- Vírus
- Vermes (Worms)
- Spywares
- Rootkits

Defesas

- Listar os recursos específicos anti-malwares que o anti-malware que você utiliza.

Segurança: Criptografia

Princípio de Kerckhoffs (Auguste Kerckhoffs)

- Todos os algoritmos deveriam ser públicos e o segredo deveria ser reservado às chaves.

Segurança: Criptografia básica

$$T_p = A_D(T_c, K)$$

Tp: Texto puro (original)

AD: Algoritmo de decriptação

Tc: Texto codificado (encriptado)

K: Chave