COPPE UFRJ

GTA / UFRJ

Conceitos

Pedro Cruz

EEL770 – Sistemas Operacionais

Tipos de sistemas operacionais

GTA / UFRJ

- Computadores de grande porte
- Servidores
- Multiprocessadores
- Computadores pessoais
- Computadores portáteis
- Sistemas embarcados
- Internet das coisas (*Internet of Things* – IoT)
  - Nós sensores
  - Smart cards
- Tempo real

Processo

GTA / UFRJ

- Programa em execução
  - Espaço de endereçamento
    - Executável
      - Código
    - Dados
      - Informações usadas
    - Pilha
      - Armazenamento local
      - Passagem de parâmetros

Processo

GTA / UFRJ

- Dados necessários e suficientes para a execução de um programa
  - Espaço de endereçamento
    - Imagem do núcleo
  - Tabela de processos

Processos

GTA / UFRJ

- Processos pai e processos filho
- Comunicação entre processos

Pilha – *call stack*

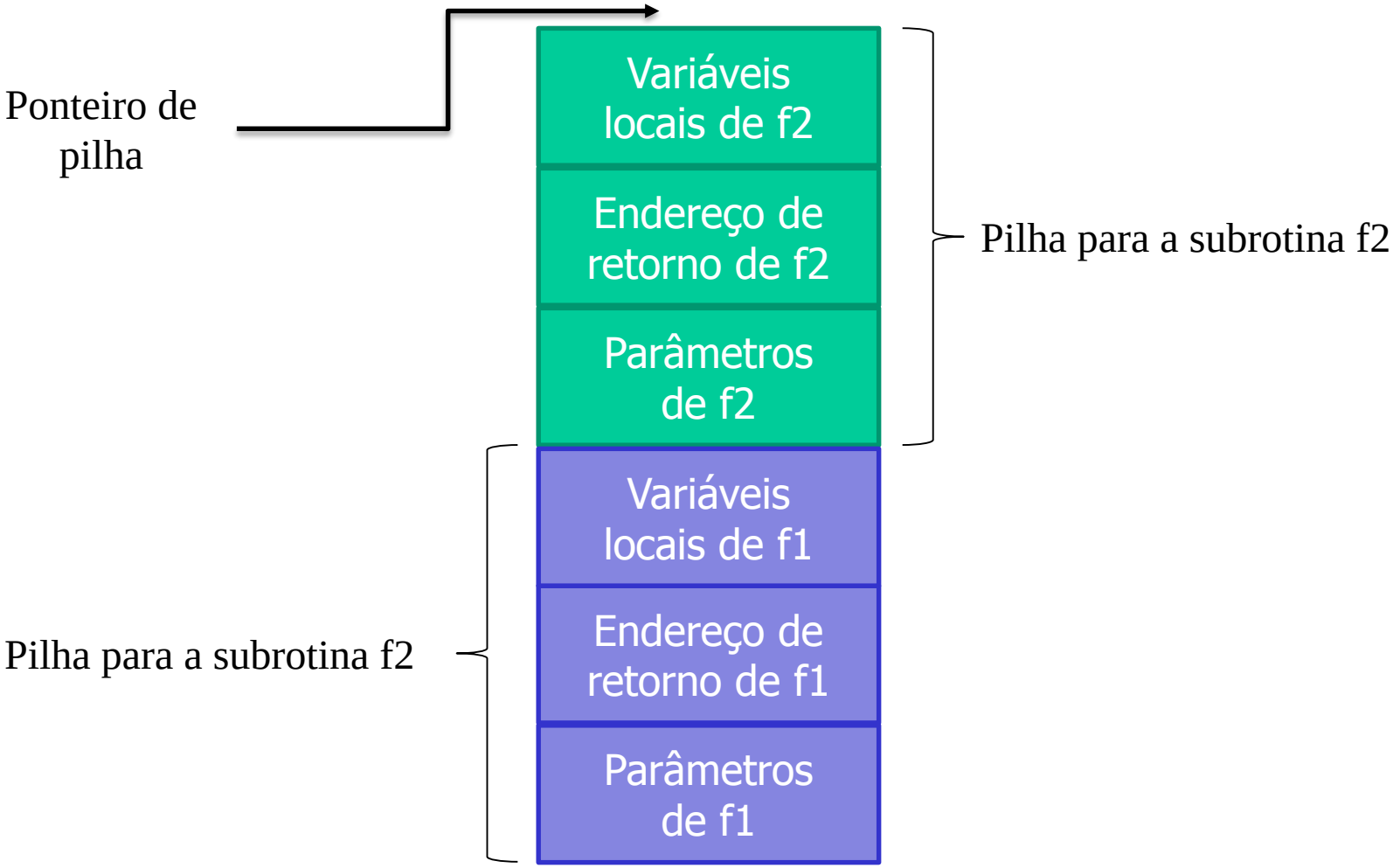
GTA / UFRJ

- Estrutura de dados LIFO (*last in first out)*
  - Armazenamento de dados locais
  - Armazenamento do fluxo de chamadas
    - Chamadas de subrotinas são colocadas na pilha
  - Passagem de parâmetros
  - Retorno de valores

- Estouro de pilha – *Stack Overflow*
  - Tamanho da pilha excede o máximo

Pilha

GTA / UFRJ



Heap

GTA / UFRJ

- Pedaço de memória livre para alocação dinâmica
  - Rotinas podem pedir memória durante execução
  - Memória alocada no *heap*

Chamadas de sistema

GTA / UFRJ

- Chamadas reais ao *hardware* são dependentes do *hardware*
  - Muitas vezes, necessário linguagem de máquina
- Chamadas reais ao *hardware* criam problemas de gerenciamento e segurança
  - Modo núcleo

- Sistema operacional fornece uma biblioteca de chamadas de sistema
  - Programadores possuem interface única com o *hardware*
  - Sistema operacional pode gerenciar o acesso ao *hardware*

A chamada *read* - UNIX

GTA / UFRJ

- counter = read(fd, buffer, nbytes)
  - *fd* é o arquivo a ser lido
  - *buffer* é o endereço onde colocar o valor da leitura
  - *nbytes* é o número de bytes a ser lido
  - *counter* é o número de bytes que foi de fato lido
    - Menor que *nbytes* se arquivo chega ao fim
    - -1, se erro de leitura

Execução da chamada *read*

GTA / UFRJ

- Programa do usuário realiza chamada *read*
- Parâmetros são empilhados
- Chamada *read* é empilhada
  - Parâmetros vão para registradores
- Instrução *trap* transfere para o núcleo
- Operacional lê os dados e armazena em *buffer*
  - Retorna para *read*
- Chamada *read* retorna para usuário

Instrução *trap*

GTA / UFRJ

- Troca do modo usuário para modo núcleo
- Chama uma subrotina do operacional
- Não pode ir para qualquer endereço de memória
  - Por razões de segurança, existe um ou poucos endereços que podem ser executados após a instrução *trap*

Chamadas de sistema - exemplos

- Gerenciamento de processos
- Gerenciamento de arquivos
- Gerenciamento de diretórios e arquivos
- Outras

Gerenciamento de processos - exemplos

- *pid* = *fork*()
  - Cria processo filho
- *pid* = *waitpid*(*pid*, &*statloc*, *options*)
  - Espera processo terminar
- *s* = *execve*(*name*, *argv*, *encironp*)
  - Troca a imagem do núcleo de um processo
    - Executa um executável de nome *name*
- *exit*(*status*)
  - Conclui um processo e retorna *status*

Exemplo

GTA / UFRJ

```
While(true){
        type_prompt();
        read_command(command, parameters);

        if(fork() != 0){
                waitpid(-1, &status, 0);
        }else{
                execve(command, parameters, 0);
        }
```

Gerenciamento de arquivos - exemplos

- *fd = open(file, mode)*
  - Abre um arquivo para leitura, escrita ou ambos
- *s = close(fd)*
  - Fecha um arquivo
- *n = read(fd, buffer, nbytes)*
  - Lê um arquivo aberto
- *n = write(fd, buffer, nbytes)*
  - Escreve em um arquivo aberto
- *position = lseek(fd, offset, whence)*
  - Move o ponteiro do arquivo

Gerenciamento de diretórios - exemplos

GTA / UFRJ

- *s = mkdir(name, mode)*
  - Cria um diretório
- *s = rmdir(name)*
  - Remove um diretório vazio
- *s = link(name1, name2)*
  - Cria uma entrada name2, apontando para name1
- *s = unlink(name)*
  - Remove uma entrada de diretório
- *s = mount(special, name, flag)*
  - Monta um sistema de arquivos

Chamadas diversas - exemplos

GTA / UFRJ

- *s = chdir(dirname)*
  - Altera o diretório de trabalho
- *s = chmod*
  - Altera a proteção de um arquivo
- *s = kill (pid, signal)*
  - Envia um sinal para um processo
- *seconds = time(&seconds)*
  - Obtém o tempo desde 1º de janeiro de 1970

Arquitetura de sistemas

GTA / UFRJ

- Módulos

- Relacionamentos entre módulos

Modelos comuns de arquitetura

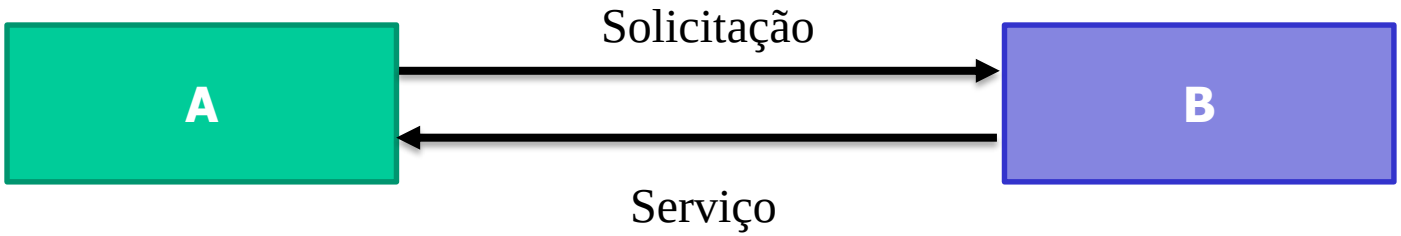
GTA / UFRJ

- Orientada a serviço
- Cliente-servidor
- Camadas
- Mestre-escravo
- .....

Orientada a serviço

GTA / UFRJ

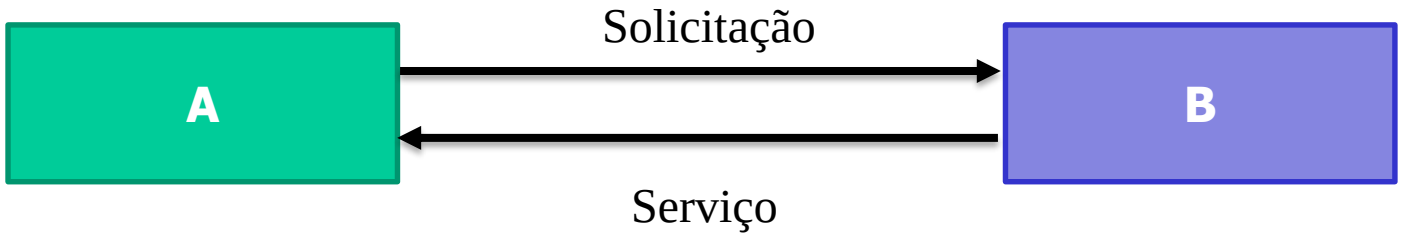
- Coringa de modelos de arquitetura
  - Módulo **A** solicita um serviço
    - Entrega dados necessários para a execução do serviço
  - Módulo **B** entrega serviço



Orientada a serviço

GTA / UFRJ

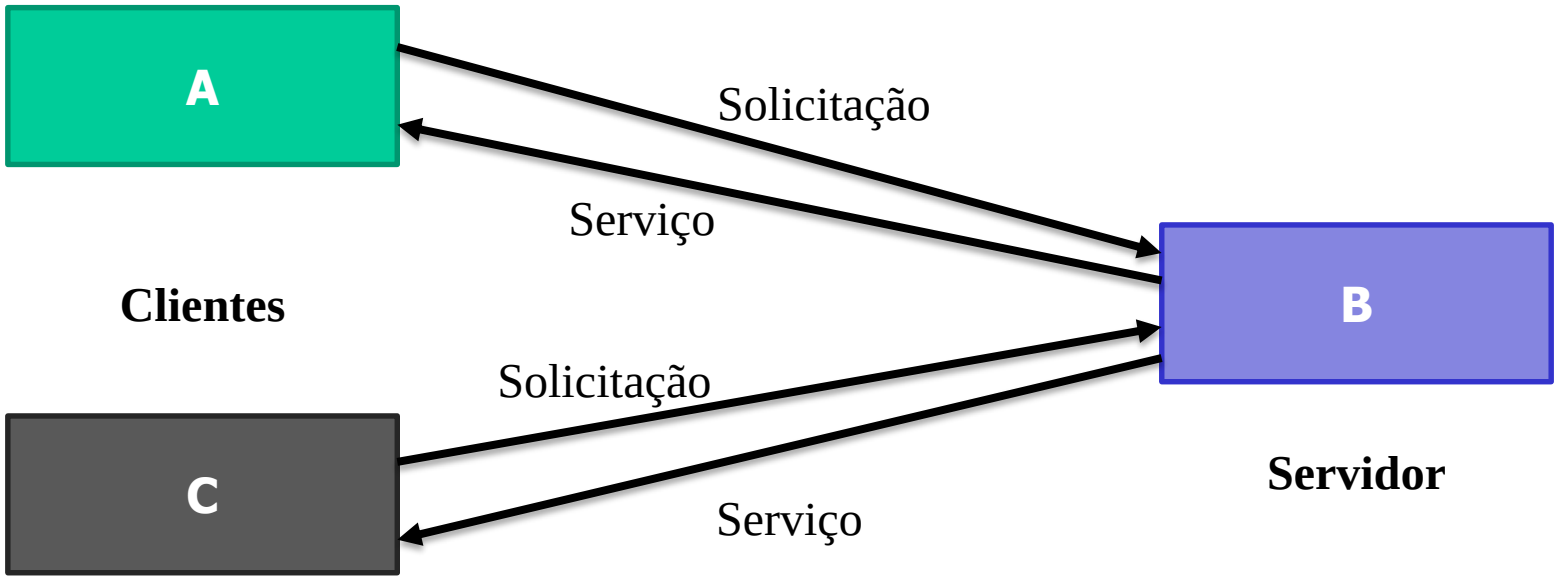
- Divisão de tarefas
  - **A** não precisa executar todas as tarefas
  - **B** não precisa executar todas as tarefas
- Encapsulamento
  - **A** só precisa saber o necessário para realizar solicitação
  - **B** só precisa saber o necessário para atender solicitação



Cliente-servidor

GTA / UFRJ

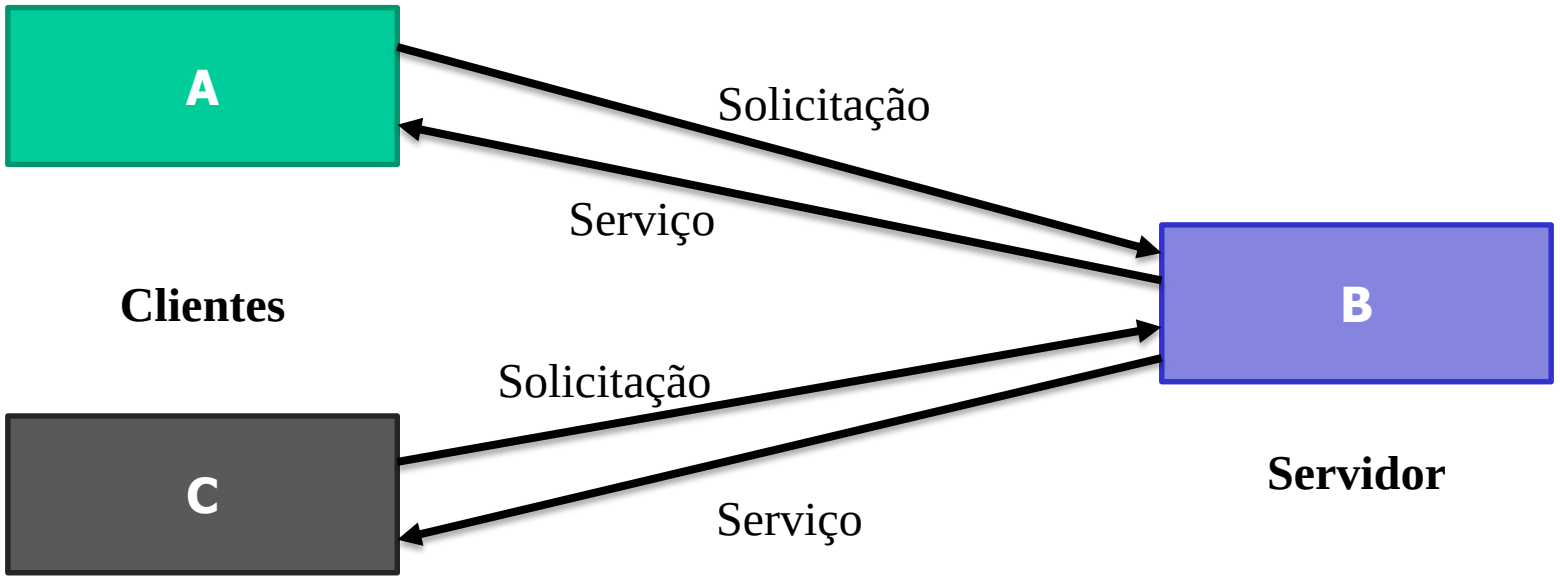
- Modelo básico de serviço
  - Módulo **A** solicita um serviço
    - Entrega dados necessários para a execução do serviço
  - Módulo **B** entrega serviço



Cliente-servidor

GTA / UFRJ

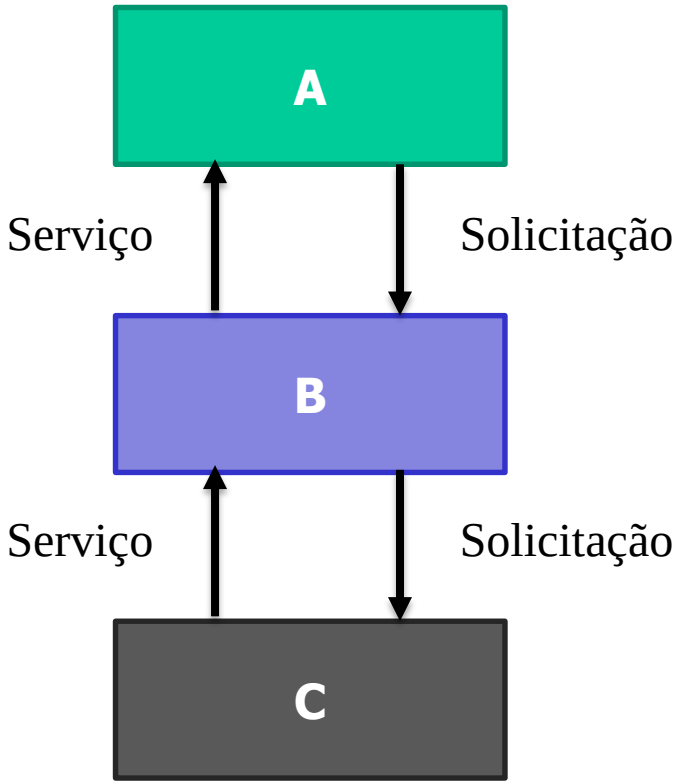
- Divisão de tarefas
  - Clientes e servidor executam tarefas especializadas
- Encapsulamento
  - Compartilhamento apenas do necessário



Modelo em camadas

GTA / UFRJ

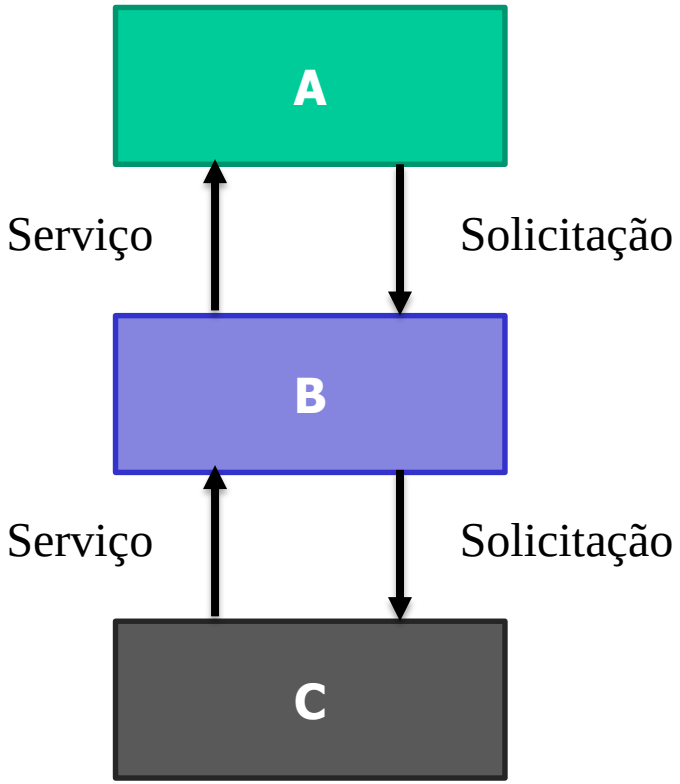
- Camadas divididas de maneira hierárquica
- Camadas 'de cima' mais próximas do usuário
- Camada 'de cima' pede serviço
- Camada 'de baixo' executa serviço
- Camadas são isoladas



Modelo em camadas

GTA / UFRJ

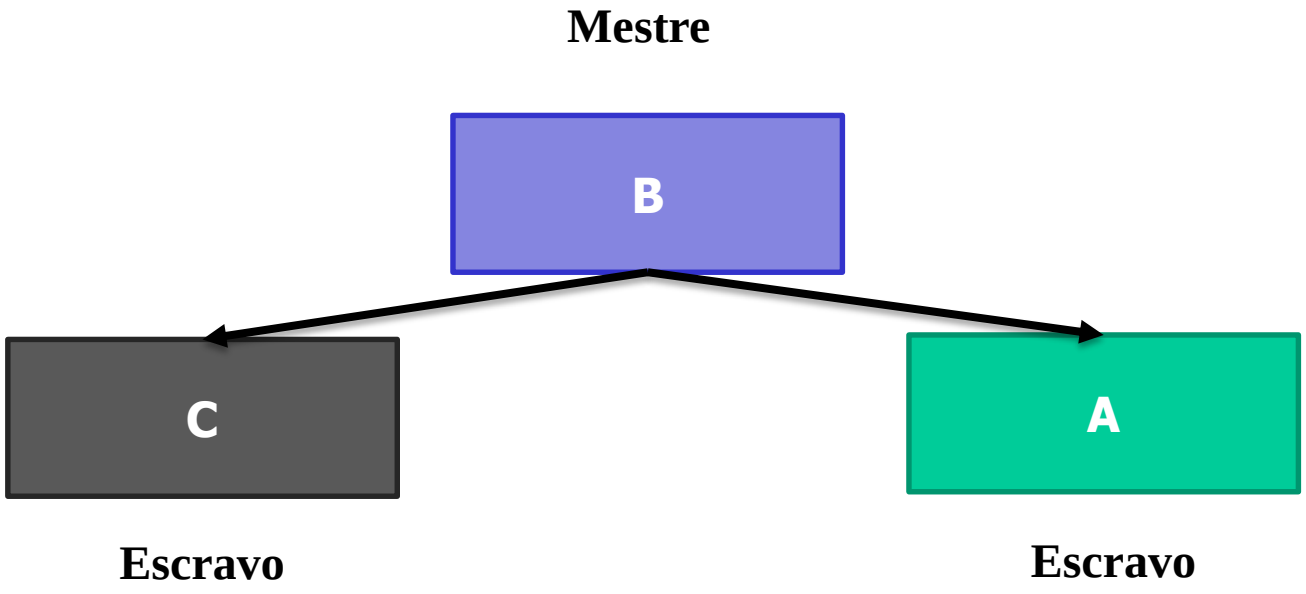
- Divisão de tarefas
  - Cada camada executa uma parte do serviço
- Encapsulamento
  - Informações são compartilhadas apenas com as camadas que precisam da informação
    - (Meia verdade)



Mestre-escravo

GTA / UFRJ

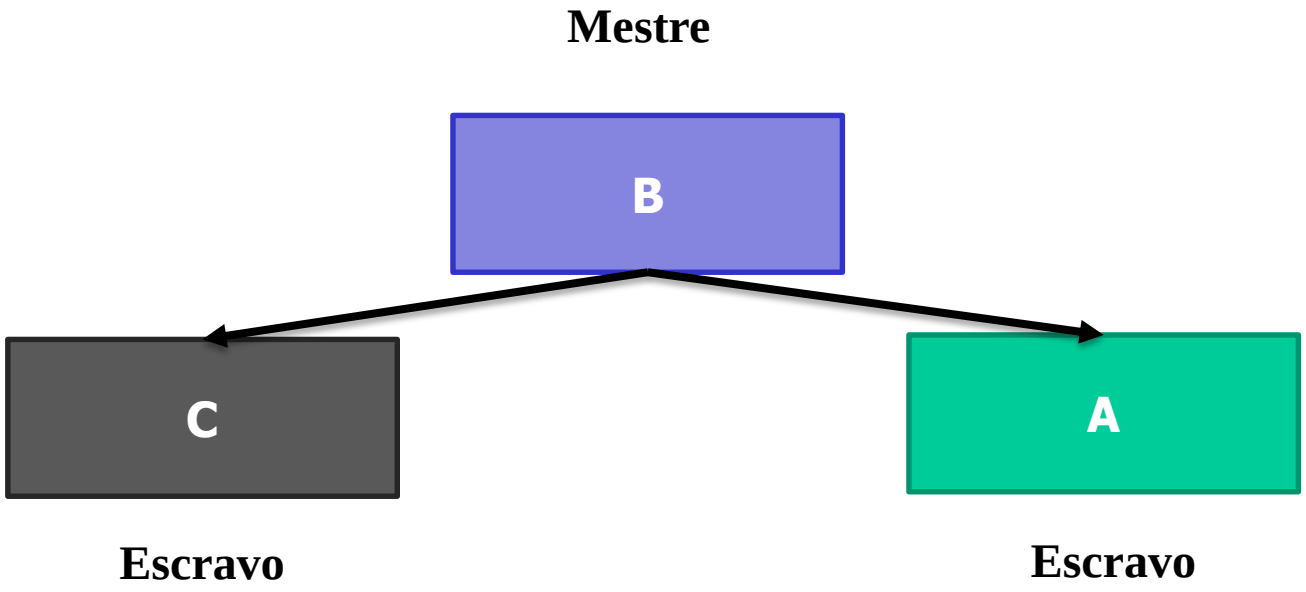
- Módulo **mestre** coordena todos os escravos
- **Escravos** obedecem e reportam ao mestre



Mestre-escravo

GTA / UFRJ

- Divisão de tarefas
  - Cada módulo executa tarefas específicas
- Encapsulamento
  - Apenas o mestre precisa saber toda informação



Estrutura de sistemas operacionais

- Sistemas monolíticos
- Sistemas em camadas
- Sistemas de Micronúcleos
- Sistemas de Cliente-servidor
- Sistemas de Máquinas virtuais
- Sistemas de Exonúcleos

Sistemas monolíticos

GTA / UFRJ

- Programa único
  - Chamadas de sistema são hierarquizadas
    - Programa principal
    - Rotinas de serviço
    - Rotinas utilitárias

Sistemas de camadas

GTA / UFRJ

- Cada camada executa uma função
  - Estudo de caso: *Technische Hogeschool Eindhoven* - THE

Camada	Função
5	O operador
4	Programas de usuário
3	Gerenciamento de entrada/saída
2	Comunicação entre processos
1	Memória e gerenciamento de hardware
0	Alocação de processador e multiprogramação

Sistemas de micronúcleo

- O máximo de tarefas é executado em modo usuário
  - Redução de erros fatais

- Desacoplamento entre mecanismo e política
  - Implementação de uma tarefa pontual é no núcleo
  - Implementação da decisão sobre a tarefa é fora do núcleo

Modelo cliente-servidor

GTA / UFRJ

- Processos executam serviços
  - Servidores
- Processos usam serviços
  - Clientes

Máquinas virtuais

GTA / UFRJ

- Hipervisor simula *hardware*
  - Sistema operacional hóspede
    - Sistema operacional hospedeiro

Exonúcleo

GTA / UFRJ

- *Hardware* é dividido entre usuários
  - Operacional exonúcleo executa a divisão

COPPE UFRJ

GTA / UFRJ

Conceitos

Pedro Cruz

EEL770 – Sistemas Operacionais