

# Sistemas Operacionais II

## *Aula 2 - Processos e shell*

Autor: Renê de Souza Pinto

Orientação: Prof. Dr. Francisco José Monaco

`rene@grad.icmc.usp.br`, `monaco@icmc.usp.br`

Universidade de São Paulo

Instituto de Ciências Matemáticas e Computação - ICMC

Escola de Engenharia de São Carlos - EESC

# Sumario

- Processos
- Shell
- Prática 2: minishell

# Processos

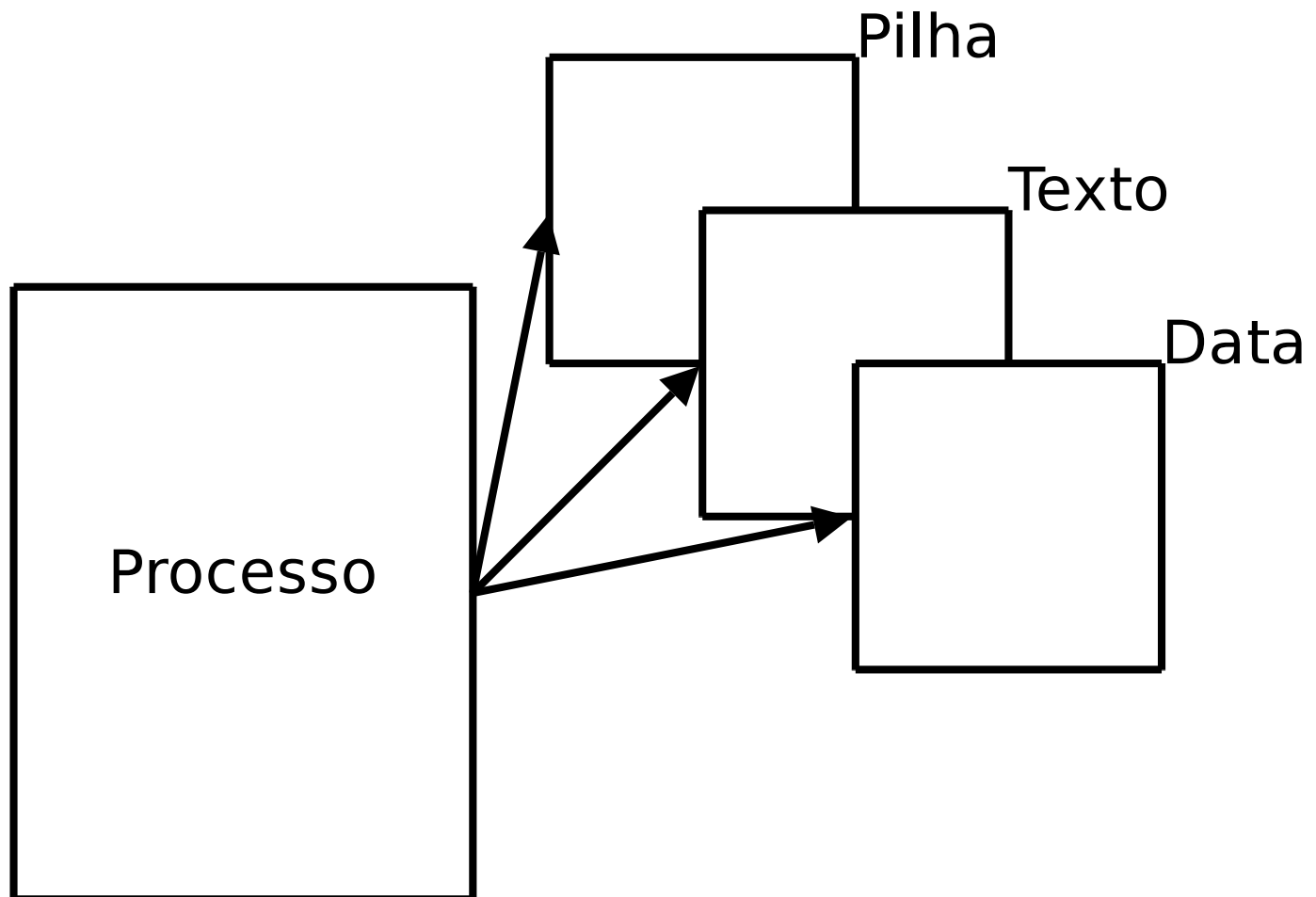
- Processos:

# Processos

- Processos:
  - Instância de um aplicativo
  - Dividido em três segmentos (regiões):
    - Pilha
    - Texto
    - Data

# Processos

- Processos:

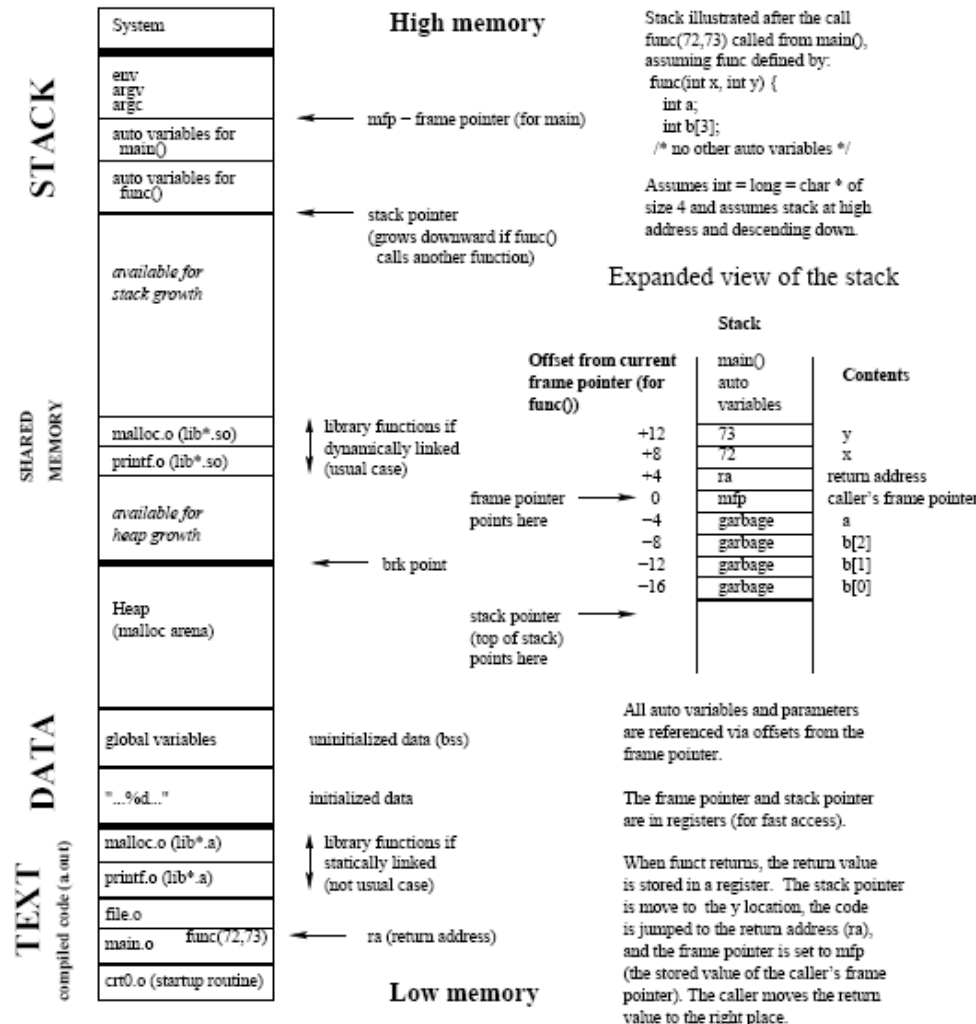


# Regioes

- **Pilha:** É a região de armazenamento dinâmico do programa. Passagem de parâmetros para funções, salvamento de registradores, funções recursivas, todos dependem da Pilha.
- **Texto:** É o código executável do programa (que contém as instruções)
- **Data:** Região aonde ficam as variáveis inicializadas
  - **BSS:** Área para as variáveis não inicializadas

# Layout de um processo C

## Memory Layout (Virtual address space of a C process)



# Area de Heap

- Heap: Área utilizada para expansão da memória do processo (através da *syscall brk()*)
  - *malloc* da biblioteca C: Utiliza *brk*

# Multitarefa

- Multitarefa preemptivo: Processadores modernos permitem chaveamento de processos, o SO pode fazer cada processo ser executado durante um breve período de tempo (*quantum*) dando a impressão de que todos processos estão sendo executados ao mesmo tempo!

# Estados de um processo

- Um processo pode assumir vários estados:
  - Executando: O processo está sendo executado
  - Dormindo: O processo está aguardando por algo
  - Zumbi: O processo terminou mas seu pai ainda não executou a chamada *wait*
    - Este estado é necessário pois o pai pode querer acessar o filho a qualquer momento, por isso o processo filho não pode ser destruído simplesmente quando termina, mas deve aguardar pela chamada *wait* ou pelo final da execução do processo pai.

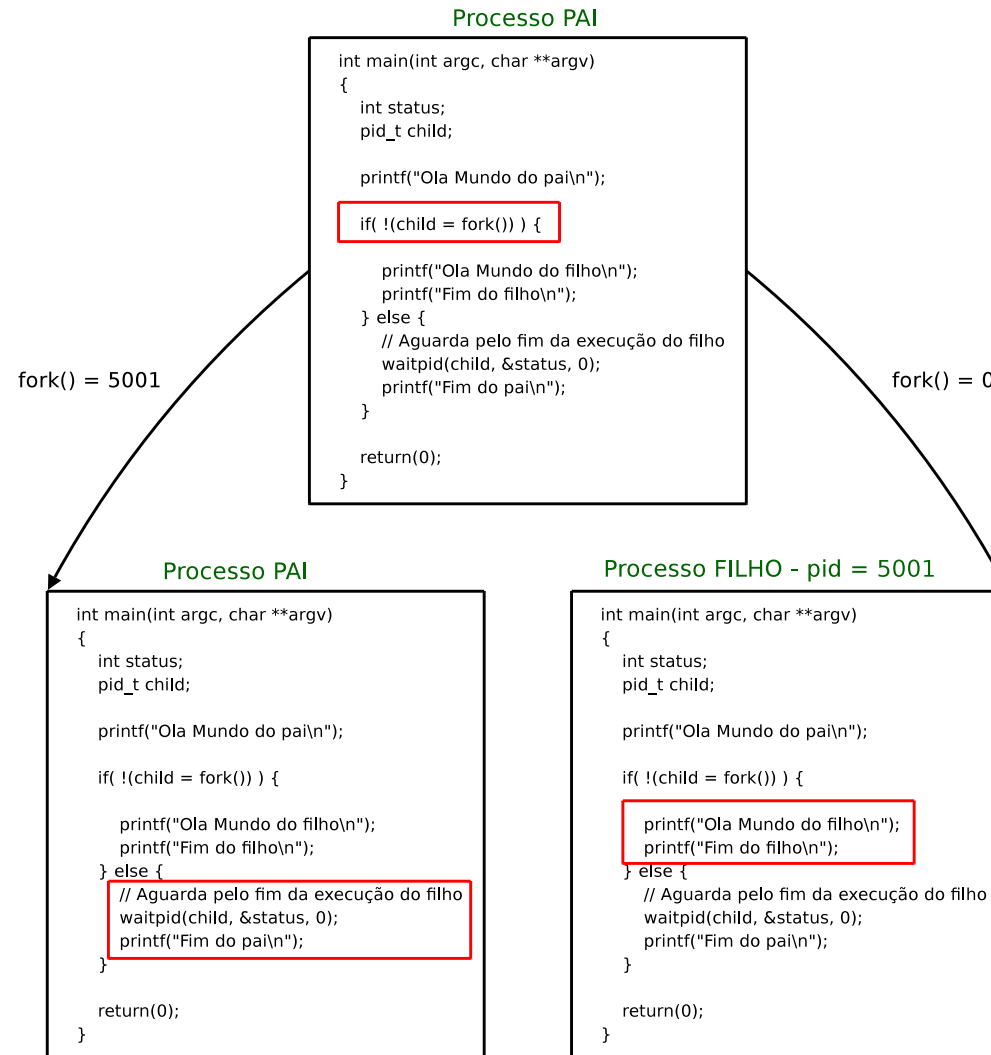
# Syscalls para processos

- Principais chamadas de sistemas para processos são:
  - *fork*: Cria um processo filho duplicando o processo atual, na maioria dos sistemas é a única maneira de se criar um novo processo
  - *execve*: Substitui o espaço do processo atual com um binário executável que é carregado

# fork

- `pid_t fork(void);`
- Retorno:
  - Para o processo pai retorna o PID (process ID) do processo filho
  - Para o filho retorna 0

# fork - Exemplo



# execve

- `int execve(const char *filename, char *const argv[], char *const envp[]);`
- Entrada:
  - *filename*: Arquivo que será carregado
  - *argv*: Parâmetros a serem passados para o programa
  - *envp*: Variáveis de ambiente

# execve

- `int execve(const char *filename, char *const argv[], char *const envp[]);`
- Retorno:
  - Quando executada com sucesso, `execve` nunca retorna pois o processo já foi substituído pelo arquivo carregado
  - -1 se ocorreu erro, e a variável `errno` é setada apropriadamente

# execve - Exemplo

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

extern char **environ;

int main(int argc, char **argv)
{
    printf("Vou executar /bin/ls\n");

    execve("/bin/ls", NULL, environ);

    // Se tudo deu certo, nao executa mais a partir daqui
    printf("Oooooooooops... nao consegui executar /bin/ls\n\n");

    return(0);
}
```

# waitpid

- `pid_t waitpid(pid_t pid, int *status, int options);`
- Entrada:
  - *pid*: PID do processo filho a ser aguardado
  - *options*: Opções de espera (aguarda pelo final do processo, pelo recebimento de um sinal, etc)
- Retorno:
  - *status*: Informações de status (se terminou normalmente, por um sinal, etc)
  - PID do processo filho ou -1 em caso de erro



# Shell

# Shell

- É a camada entre o usuário e os aplicativos
- Ex: BASH, Korn, CSH, etc...
- Os mais robustos possuem uma linguagem interpretada
- Comandos internos e controle de processos

# Shell

- Funções básicas:
  - Ler a linha de comando do usuário
  - Separar o comando e parâmetros passados
  - Executar o comando (built-in ou não)
  - Esperar que o processo filho seja finalizado
  - Retornar o controle para o usuário

# Pratica 2

- Iniciar o desenvolvimento de um shell
  - Comece com um shell bem simples
  - Depois faça um *parser* para os parâmetros digitados pelo usuário
  - Por enquanto é só, mas como desafio tente implementar comandos internos ou até mesmo *pipes* (para redirecionamento de saída)

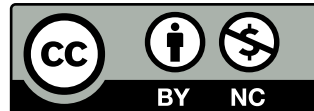


**Bom Trabalho!**

## Referências

- [1] Tanenbaum, A. S./ Woodhull A. S.; Sistemas Operacionais: projeto e implementação
- [2] Bach, M. J.: The design of the Unix operating system. Prentice-Hall, 1986.
- [3] Love, Robert: Desenvolvimento do Kernel do Linux. Rio de Janeiro: Editora Ciência Moderna, 2004.

# Licença



Este documento é licenciado sob a  
Creative Commons Atribuição-Usó Não-Comercial 2.5  
Brasil License.