DCOMP/UFSJ - Prof. Rafael Sachetto Oliveira — Sistemas Operacionais

# Primeiro Trabalho Prático

Ciência da Computação — 1º Semestre de 2025

Este trabalho tem por objetivo o melhor entendimento sobre gerenciamento de processos em um sistema operacional.

## 1 Trabalho Prático: Construção de um Shell de Comandos em C

Este trabalho tem como objetivo aprofundar o conhecimento sobre criação de processos, uso de pipes, leitura de entrada e execução de programas no Linux por meio da construção de um shell de comandos.

### Objetivo

Você deve desenvolver, em linguagem C, um shell simples que será executado no terminal Linux. Esse shell será responsável por:

- Ler comandos do usuário utilizando a biblioteca `readline`.
- Executar comandos externos utilizando chamadas `fork()` e `exec()`.
- Implementar comunicação entre processos usando `pipe()`.
- Criar comandos internos (builtins), como `cd`, `exit` e `help`.
- Gerenciar múltiplos comandos conectados via pipes (ex: `ls | grep .c | sort`).
- Permitir execução em background (ex: `./programa &`).
- Tratar erros de chamadas de sistema apropriadamente.

### Restrições

- Não é permitido o uso de `popen()` nem qualquer função de alto nível que encapsule fork/exec/pipe.
- O código deve ser portável e compilável em sistemas Linux usando `gcc`.
- O código deve ser devidamente comentado.

### Entrega

- O código-fonte em C.
- Um Makefile para compilar o shell.
- Um relatório explicando a estrutura do shell, principais funções e decisões de projeto.
- A entrega dos arquivos deverá ser feita via SIGAA e a fórmula para desconto por atraso na entrega é $\frac{2^{d-1}}{0{,}32}\%$, onde $d$ é o atraso em dias. Note que após 6 dias, o trabalho não pode ser mais entregue. Ao final da descrição do trabalho, há outras informações disponíveis sobre a entrega.
- Valor: 10 pontos

## Dica

Veja abaixo um exemplo simples de como usar `fork()` e `pipe()` para comunicação entre processos.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int main() {
    int pipefd[2];
    pid_t pid;
    char buffer[100];

    if (pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    pid = fork();
    if (pid < 0) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        // Processo filho
        close(pipefd[1]); // Fecha a escrita
        read(pipefd[0], buffer, sizeof(buffer));
        printf("Filho recebeu: %s\n", buffer);
        close(pipefd[0]);
    } else {
        // Processo pai
        close(pipefd[0]); // Fecha a leitura
        const char *msg = "Ol do processo pai!";
        write(pipefd[1], msg, strlen(msg) + 1);
        close(pipefd[1]);
        wait(NULL); // Espera o filho terminar
    }

    return 0;
}
```