# Relatório do Laboratório de CES-33 Compiladores

18 de março de 2018

## Disciplina: CES-33

Estudante: Felipe Guimarães

Turma 19.3

**Instituto Tecnológico de Aeronáutica**

# I. Introdução

Nesse laboratório, foram estudados os processos e suas interações via pipe. Na primeira experiência, foi visto como criar um processo zumbi a partir do fork de um processo mãe. Tal processo ocupa memória na tabela de processos mas não consegue se finalizar sozinho pois está esperando o processo pai que está em sleep. No segundo experimento, foi implementado um programa onde há a comunicação entre um processo mãe e um processo filho via pipe, mostrando como os processos podem trocar informação entre eles.

# II. Parte 1

Para criarmos o processo zumbi, criamos um fork de um processo pai. Identificamos os processos pai e filhos a partir do valor retornado pelo método fork em cada processo. No processo pai, chamamos um sleep de 20 segundos, e o processo filho termina normalmente. Porém, o processo filho fica esperando o processo pai como um processo zumbi, já que o processo pai está em sleep e não lê o o status de saída do processo filho para tirá-lo da tabela de processos.

Para testar o programa, o mesmo foi executado e verificou-se na tabela de processos a partir do comando `ps -aux` que existia um processo pai e um processo filho zumbi num intervalo de 20 segundos. Após esse intervalo de tempo, o processo pai saiu do sleep e ambos foram finalizados e removidos da tabela de processos.

O código do programa para essa questão se encontra em anexo.

# III. Parte 2

A segunda parte consiste em criar um código que cria dos processos que conseguem se comunicar via pipe. Para isso, criamos um fork de um processo e o processo pai escreve no primeiro pipe a mensagem a ser enviada. Ele fecha o `READ_END` e o `WRITE_END` do primeiro pipe e espera o processo filho finalizar. O processo filho então lê o que está escrito no primeiro pipe, chama a função de inverter a caixa da string e escreve o resultado no segundo pipe. Logo, ele fecha os end’s do segundo pipe e termina sua execução.

Finalmente, o processo pai continua sua execução e lê do segundo pipe a mensagem enviada pelo processo filho. A mensagem é printada na tela e o processo pai é encerrado.

Para testar a robustez do programa, foram adicionados sleeps para checar se algum erro ocorreria. O primeiro foi antes do pai escrever a mensagem no primeiro pipe e o segundo antes do filho escrever no segundo pipe. Mesmo com ambos os sleeps, o programa funcionou corretamente com o output esperado.

O código do programa para essa questão se encontra em anexo.

# IV. Conclusão

Todos os programas funcionaram corretamente e com eles foi possível entender melhor os conceitos que regem o comportamento dos processos em um sistema operacional.