**UNIVERSIDADE FEDERAL DE UBERLÂNDIA**

**FACULDADE DE ENGENHARIA MECÂNICA**

**Curso de Graduação em Engenharia Mecatrônica**

# RELATÓRIO III DE SISTEMAS DIGITAIS(FEELT49081)

## LINUX COMO AMBIENTE DE PROGRAMAÇÃO

Prof. Éder Alves de Moura

11321EMT

Guilherme Vitor dos Santos Rodrigues 009

**Uberlândia, Janeiro de 2022.**

# Sumário

# 1. Resultados de vídeos propostos

Para o teste do Bash no Linux, temos como resultados, o programa feito na extensão *.sh* como seu resultado no terminal:

```
#!/usr/bin/bash

echo "Name = $1";
echo "Age = $2";
echo "Sex = $3";
echo "my first bash program!!!"
GREET="HelooDOG"
echo $GREET

while true; do
        read -p "Do you wish to drink a beer? " yn
        case $yn in
                [Yy]* ) break;;
                [Nn]* ) break;;
                * ) echo "Please answer yes or no.";;
        esac

done

if [$2 -lt 21]; then
        echo "You are too young to take this beer!! "
else
        echo "Enjoy your beer $1 !!"
fi

long_process &
slow_stuff &
```

```
guilherme@guilherme-VirtualBox:~$ ./beerme.sh name 32 sex
Name = name
Age = 32
Sex = sex
my first bash program!!!
HelooDOG
Do you wish to drink a beer? y
./beerme.sh: line 20: [32: command not found
Enjoy your beer name !!
./beerme.sh: line 26: long_process: command not found
./beerme.sh: line 28: async_job: command not found
./beerme.sh: line 27: slow_stuff: command not found
```

# 2. Resumo do Livro

## 2.1 IDs de processo

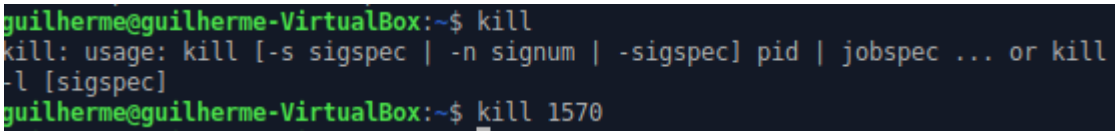
Cada processo no Linux é identificado pelo seu ID chamado de *pid*, sendo que cada um tem um processo pai. O processo pai tem um ID identificado pelo seu *ppid*. Com a chamada *getpid()* consegue-se o *pid* do processo, e com *getppid()* o *pid* do processo pai.

## 2.2 Visualizando processos ativos

Utilizando o comando *ps* pode se observar cada processo executado no sistema.

## 2.3 Matando um processo

Para essa funcionalidade é necessário o comando *kil*:




## 2.4 Criando processos

Duas técnicas são utilizadas para criar processos, uma simples mas que gera problemas de segurança e outra mais complexa mas que gera uma grande flexibilidade. A primeira técnica é utilizar o *system* que invoca um programa com o privilégio do *root* e possui diferentes resultados em diferentes aplicações UNIX. Utilizando o *fork* o processo cria outro que é chamado de processo filho e com a utilização do *exec* é possível transformar este processo novo em instancias de novos programas. Não há uma garantia de qual processo rodará antes, pode ser o programa pai ou o filho, assim como por quanto tempo ele vai rodar. Assim é possível definir qual o programa mais importante devido à prioridade de cada processo, pela definição de um fator chamado *niceness*. E utilizando o comando *nice* é possível definir a lista dos valores de prioridade de um processo.

## 2.5 Sinais

Sinais são utilizados para a comunicação e manipulação dos processos em Linux. Estes são mensagens que são processadas pelos programas em qualquer fase do debug. A usabilidade destes sinais é definida por condições específicas, para terminar processos, definir a disposição dos sinais, para determinar erros, violação de segmentação. Geralmente se utilizam para uma configuração melhor dos sinais, que são assíncronos, um programa chamado *signal handler* e ele funciona retornando o controle do programa principal após performar o trabalho mínimo para responder o sinal.

## 2.6 Matar um processo e processos zumbis

Para definir o término de um processo, o necessário a se fazer é utilizar o comando *kill* no *pid* do processo pai. Porém se um processo filho termina enquanto o seu processo pai está chamando uma função *wait*, este processo filho desaparece e seu status de finalização é passado para o seu pai por via da chamada *wait.*

Um processo zumbi é um ptocesso que termina mas não termina completamente vazio. O processo pai limpa este processo zumbi utilizando a função *wait.* Para resolver este

problema ao terminar sua execução, o processo filho tem de fazer a chamada *wait*, porém se o processo filho não termina nesse ponto, o processo pai bloqueia este *wait* até o processo filho terminar.

## 3. Resumo do Vídeo

- **Sobre Linus e a criação do Linux**: Sendo criado em 1991, o Linux teve como protótipo um simples emulador de terminal e hoje ele é um sistema operacional completo que roda em pequenas e funcionais grandes maquinas cujo objetivo era ser um sistema de livre acesso.
- **Como o kernel do Linux é o mesmo e ao mesmo tempo diferente dos outros quando foi criado:** A diferença que o Linux tinha dos outros era que ele trabalhava com um design monolítico enquanto os outros tinham o design de microkernel, além do fato do kernel do Linux ser preemptivo e grátis e livre para o uso.
- **Como programar no kernel é diferente de outro espaço:** O GNU C de outros espaços pode ser instalado de outra maneira e o Kernel carece de proteção de memória. Além disso operações de ponto flutuante funcionam de forma diferente no Kernel e utilizam uma pilha de pequeno tamanho fixado pré-processado.
- **Como Processos são monitorados e geridos no kernel:** Os processos podem ser monitorados pelo PID de cada um utilizando no terminal o comando *ps.* Para a geração de processos no kernel, primeiramente a tarefa tem que fazer uma chamada com o fork() e criar um processo novo. Quando a tarefa está pronta mas não rodando é necessário que o Scheduler despache a tarefa e logo após com a tarefa rodando há uma estados em que há a finalização da task ou a espera do processo para um evento específico
- **Threads no Linux:** Threads são abstrações de programação modernas que permitem programação concorrente. O Linux tem uma implementação de threads única em que não há este conceito de threads pré-definida. Neste caso, o sistema operacional implementa todas as threads como processos padrão em que nenhuma delas possui espaço de endereçamento.
- **Cronograma de processos e de algoritmos:** Os processos são programas ativos em que o cronograma de processos decide qual processo roda e por quanto tempo. E isso é a base do sistema operando em multitasking.
- **O que é uma chamada de sistema e como chama-la:** Uma chamada de sistema é a forma programática em qual um programa requisita um serviço de um kernel do sistema operacional que é executada e é uma forma de interação com este sistema. Geralmente uma chamada de sistema é tipicamente acessada via chamada de função sendo que no Linux cada chamada é classificada com um número.
- **Implementação da chamada de sistema no Kernel:** Primeiramente para cada chamada de sistema se deve definir um proposito e assim definido, o comportamento dessa chamada não

pode mudar por existir aplicações que podem contar com essa chamada. Então para funcionar, cada chamada de sistema tem de verificar seus parâmetros para garantir que elas são validas e legais.

- **O que é uma interrupção e como elas são manipulada no Kernel:** Interrupções habilitam o hardware a sinalizar ao processador e cada valor de interrupção são frequentemente chamadas em linhas IRQ em que cada uma tem um valor numérico.
- **Como proteger regiões criticas e condições de corrida:** Uma seção critica é um segmento de código que acessa variáveis compartilhadas e tem de ser executada como uma ação atômica. Isso significa que em um grupo de processos cooperativos em um dado ponto, apenas um processo deve ser executado é sua região critica. Uma condição de corrida é uma situação indesejada que ocorre quando um dispositivo ou sistema tenta performar duas ou mais operações no mesmo tempo, então devido a natureza do dispositivo ou sistema, as operações devem ser uma sequencia própria para ser terminadas corretamente.
- **Entendendo a noção de tempo do Kernel:** Um grande numero de funções são dependentes do tempo e não dos eventos. Certamente, o conceito de tempo em um computador é meio obscuro então o kernel deve trabalhar com o hardware do sistema e compreender e gerenciar o tempo, logo um manipulador de interrupções é importante para isso.
- **Teoria de Gerenciamento de memória do Kernel:** A alocação de memória dentro do Kernel não é tão acessível quanto a alocação fora, então em sistemas operacionais de computadores paginar é um esquema de gerenciamento de memória pelo qual o computador guarda e recupera dados por um armazenamento secundário para usar em memória principal.
- **Camada de abstração do sistema de arquivo:** Uma camada de abstração é uma forma de esconder os detalhes de trabalho de um subsistema permitindo a separação de problemas e facilitar inoperabilidade e independência de plataforma. Uma interface genérica para qualquer tipo de sistema de arquivo é factível apenas porque os implementos do kernel são uma camada de abstração em volta da interface de sistema de arquivos low-level.

**Referências**

[1] - “Kurt Wall. Linux Programming Unleashed". SAMS, 2007.capitulo 3