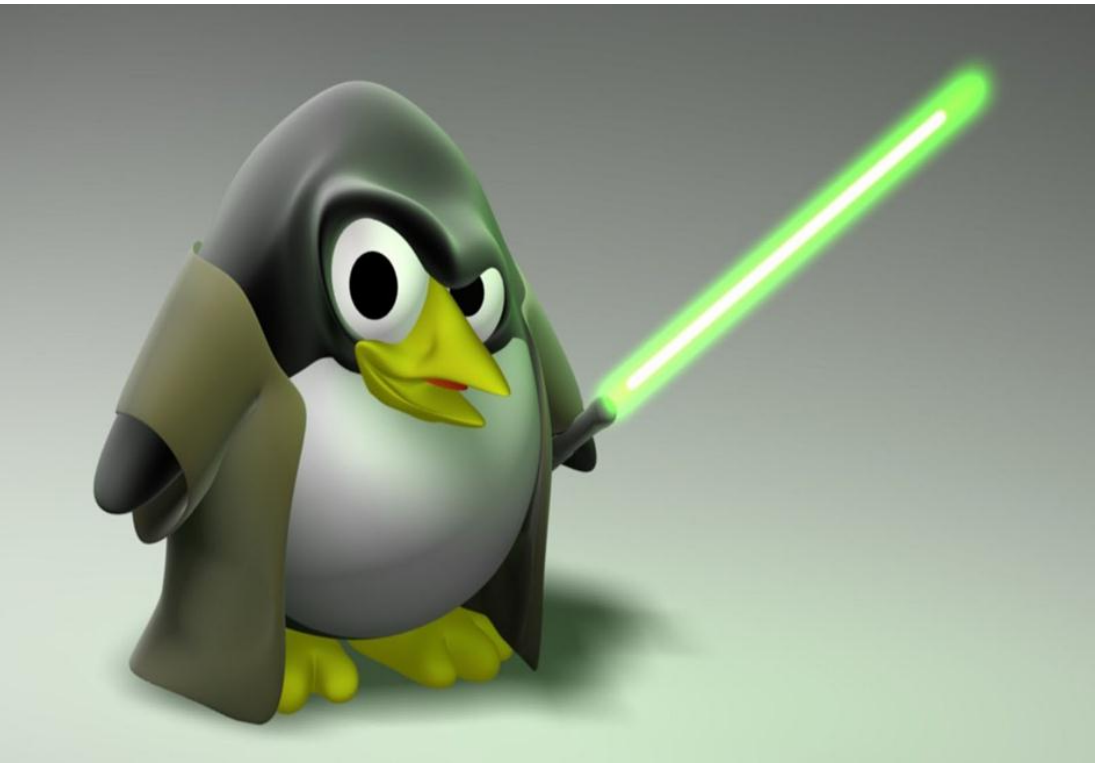
Departamento de Computação

SISTEMAS OPERACIONAIS 1
21270 A

Lab 1: Shell



Departamento de Computação
Prof. Kelen Cristiane Teixeira Vivaldini

Shell

- Shell é um programa que conecta e interpreta os comandos digitados por um usuário, ou seja é um interpretador de linhas de comandos que faz a interface entre o usuário e o sistema operacional (kernel)
- Também pode servir como linguagem de script (shell script) para realizar tarefas mais complicadas (p. ex. agrupar arquivos de texto ou baixar dados automaticamente da web)

Departamento de Computação

Shell

- Existem duas categorias de shells: linha de comando e gráfico. Intuitivamente, os shells de linha de comando fornecem a interface de usuário da linha de comando comumente usada em ambientes Unix/Linux, enquanto os shells gráficos fornecem interface gráfica do usuário (GUI), como o MS Windows.
- Nesta pratica, nos focaremos em shells de linha de comando.

Departamento de Computação

Shell

- O shell é apenas outro programa de usuário.
- Os arquivos / bin / sh, / bin / bash e / bin / tcsh são todos os arquivos executáveis para shells.
- A única coisa especial sobre o seu shell de login é que ele está listado em seu registro de login para que / bin / login (o programa que o solicite para sua senha) sabe o programa a ser iniciado quando você logar.
- Se você executar “cat / Etc / passwd”, você verá os registros de login da máquina e o programa de logon de login listado como o último campo.

Departamento de Computação

Comandos básicos do shell

- **pwd** – identifica o diretório atual (Present Work Directory)
- **ls** – lista o conteúdo do diretório atual (LiSt)
- **cd** <diretorio_destino> – muda para o diretório de destino (Change Directory)
- Dica: Se a pasta de destino estiver dentro do diretório atual não é necessário digitar o caminho completo (../../../diretorio_destino) basta digitar somente cd e o nome da pasta desejada. Para retornar ao diretório anterior existe o comando cd –
- **mkdir** <nome> - Cria um diretório com o nome especificado (MaKe Directory)
- **rmdir** <nome> - Remove um diretório com o nome especificado (ReMove Directory)
- **cp** <file1> <file2> - Copia arquivo ou diretório.
- Se o arquivo ou diretório desejado reside dentro do diretório atual utilizamos o comando: cp picture.jpg picture-02.jpg
- Se o arquivo ou diretório reside em outro diretório devemos especificar o caminho completo: cp /home/chuck/pictures/picture.jpg /home/chuck/backup/picture.jpg
- **mv** <file1> <file2> - Pode ser usado para renomear um arquivo ou para movê-lo (sem copiar) entre diretórios, a utilização se dá da mesma maneira que o comando cp

Basic Shell Structure

- Muitas system calls:
  write, read, fork, exec, wait

```
while (1) {
  write(1, "$ ", 2);
  readcmd (cmd, argc, argv); // parse user input
  if ((pid = fork()) == 0) // child
    exec(cmd, args);
  else if (pid > 0) // parent
    wait(0);
  else
    perror("fork");
}
```

Departamento de Computação

Terminal I/O

- Terminal tem padrão in e out bem como error
  - fd(standard in) = 0, keyboard
  - fd(standard out) = 1, terminal
  - fd(standard error) = 2, terminal
- Substitua o padrão out:

```
close(1);
fd = open(...);
assert(fd == 1
```

Departamento de Computação

Redirection

- Um shell em sistemas semelhantes a Unix geralmente suporta dois recursos interessantes:
  - **Redirecionamento de input/out**
- Quando inicializado o comando, existe três arquivos padrões abertos:
  - **stdin** (geralmente mapeia para entrar do teclado)
  - **stdout** (geralmente mapeia para a saída normal para a tela)
  - **stderr** (geralmente mapeia para mensagens de erro para a tela).

Departamento de Computação

Redirection

- Abaixo estão alguns exemplos de redirecionamento:

$ cmd1 < in.txt

executa cmd1, usando in.txt como fonte de entrada, em vez do teclado.

$ cmd2 > out.txt

executa cmd2 e coloca a saída para arquivar out.txt.

$ cmd3 > out.txt 2> err.txt

executa cmd3 e coloca a saída normal para arquivar out.txt e as mensagens de erro para o arquivo err.txt.

Departamento de Computação

Redirection

ls > out

Mais informações sobre redirecionamento podem ser encontrada em:

https://ava.ead.ufscar.br/mod/page/view.php?id=391030

Departamento de Computação

Pipes

- O comando abaixo conecta a saída padrão de cmd1 à entrada padrão de cmd2 e novamente conecta a saída padrão de cmd2 à entrada padrão de cmd3, usando o operador de pipeline '|'

$ cmd1 | cmd2 | cmd3

$ sort <file.txt | uniq | ufscar

- que conta o número de linhas exclusivas em file.txt.
- Sem pipe, você precisaria usar três comandos e dois arquivos intermediários para contar as linhas exclusivas em um arquivo.

Pipes

- Considere o conjunto de operacoes sobre dados em "in":

```
$ sort < in > out
$ uniq out > out
$ wc out2
$ rm out out2
```

Isto pode ser realizado usando somente pipes:

```
$ sort < in | uniq | wc
```

Departamento de Computação

Outros comando

sort - ordena arquivos.
last - mostra as últimas N linhas de um arquivo.
head - mostra as primeiras N linhas de um arquivo.
grep - mostra linhas que satisfaçam determinado padrão.
wc - conta número de linhas e palavras.
cut - mostra apenas determinadas colunas da entrada na saída.
tr - substitui, elimina ou reduz a quantidade de caracteres da entrada.
grep - mostra linhas que satisfaçam determinado padrão.

Background Jobs

- Use &: $ sleep & 10

```
while (1) {
  write(1, "$ ", 2);
  readcmd (cmd, argc, argv); // parse user input
  if ((pid = fork()) == 0) // child
    exec(cmd, argv, argc);
  else if (pid > 0) // parent
    if (argv[argc - 1][0] != '&') // wait unless sent to bg
      wait(0);
  else
    perror("fork");
}
```

Kernel

- As rotinas do sistema são executadas concorrentemente (ao mesmo tempo) sem uma ordem pré-definida, com base em eventos (acontecimentos) dissociados do tempo (eventos assíncronos)

Funções do Kernel

São funções do Kernel:

- Tratamento de interrupções e exceções;
- Criação, eliminação, sincronização,
- Escalonamento e controle de processos
- Gerência da memória, do sistemas de arquivos, das operações de entrada e saída;
- Suporte a redes locais e distribuídas
- Contabilização, auditoria e segurança do sistema

Departamento de Computação

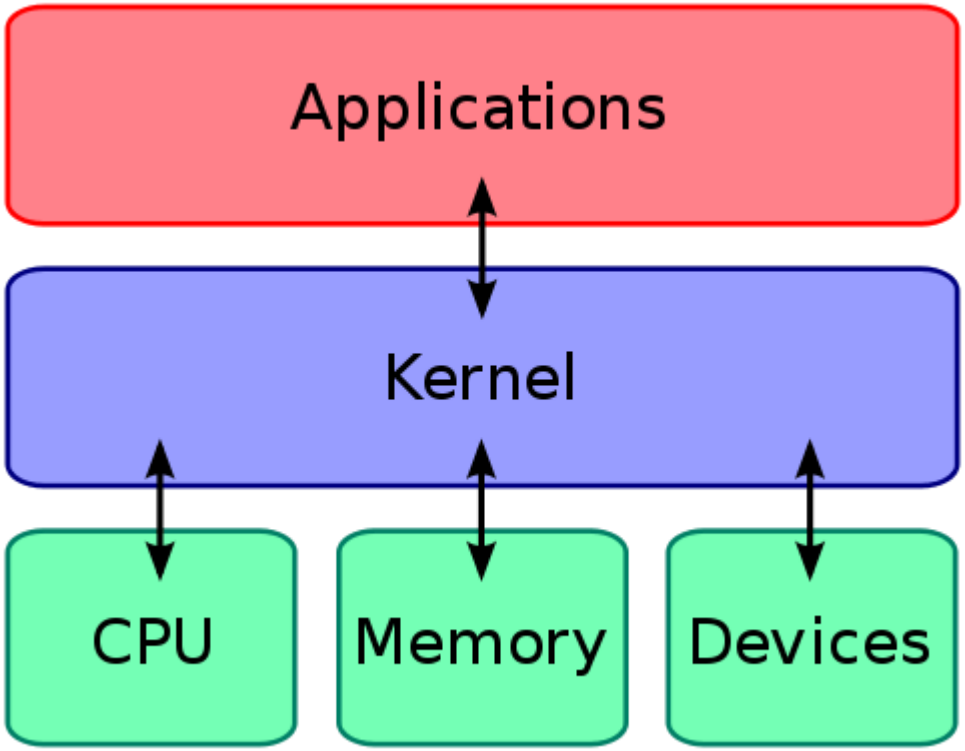
Modos de acesso

O modos de acesso dos processadores é um mecanismo presente no hardware dos processadores:

- No **MODO USUÁRIO** uma aplicação só pode executar instruções não privilegiadas, ou seja, instruções que não oferecem riscos ao sistema.
- No **MODO KERNEL** uma aplicação pode executar instruções não privilegiadas e privilegiadas, ou seja, instrução que oferece risco ao sistema (exemplo: instruções que
- acessam dados no disco)

Departamento de Computação

Modos de acesso



Departamento de Computação

Espaço do Usuário (EU)

- Este é o espaço no qual os aplicativos de usuário são executados.
- **GNU C Library = glibc**
- **EU = Aplicações do Usuário + GNU C Library**

GNU C Library (glibc)

- Fornece a **interface de chamada do sistema** que se conecta ao kernel e fornece o mecanismo para transição entre o aplicativo de espaço de usuário e o kernel.

- Isso é importante, pois **o kernel e o aplicativo do usuário ocupam espaços de endereços diferentes** e protegidos.

Departamento de Computação

Espaços de Endereçamento

- Embora **cada processo no espaço de usuário** ocupe seu **próprio espaço de endereçamento**, o **kernel ocupa um único espaço** de endereço.

Departamento de Computação

Interfaces Externas ao Kernel Linux

- C Library ou GNU C Library
- ***System Call Interface*** (Chamadas do Sistema)

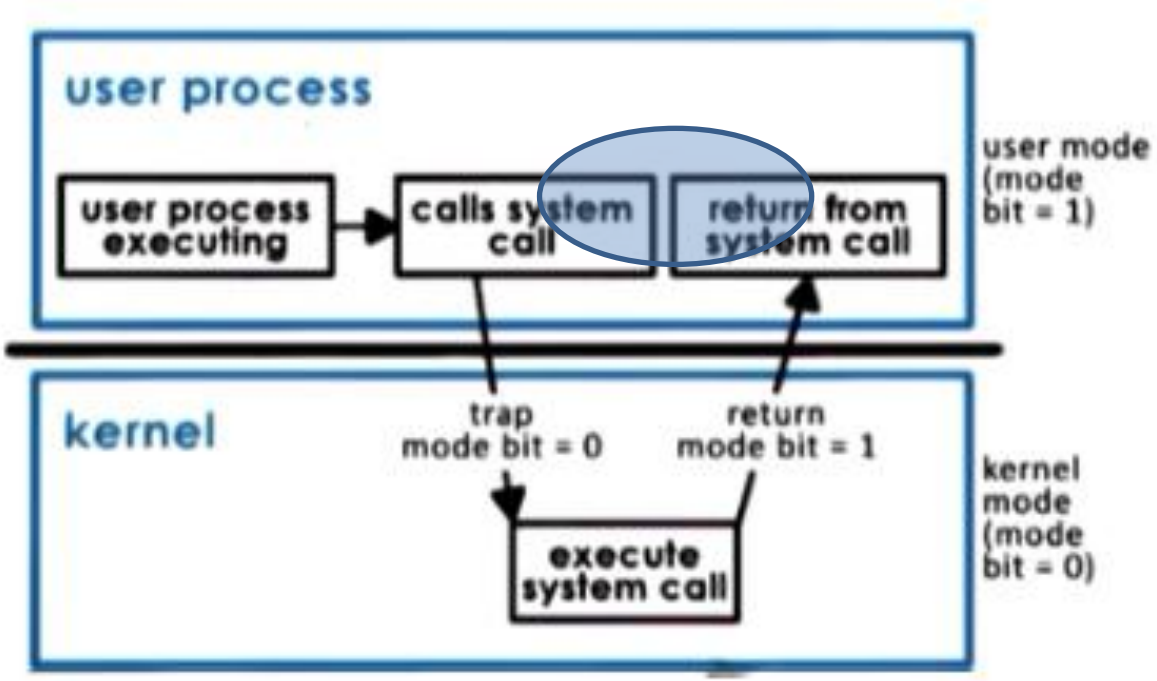
Departamento de Computação

System Call

- Interface de programação aos serviços fornecidos pelo SO.
- Tipicamente escritos em uma linguagem de alto nível (C or C++).
- Geralmente acessadas por programas via uma API (Application
- Program Interface) do que diretamente pelo uso de chamadas de sistema.
- Três APIs mais comuns são :
  - Win32 API para Windows.
  - POSIX API para sistemas baseados em POSIX (incluindo todas as versões de UNIX, Linux, e Mac OS X).
  - Java API para a máquina virtual Java (JVM).

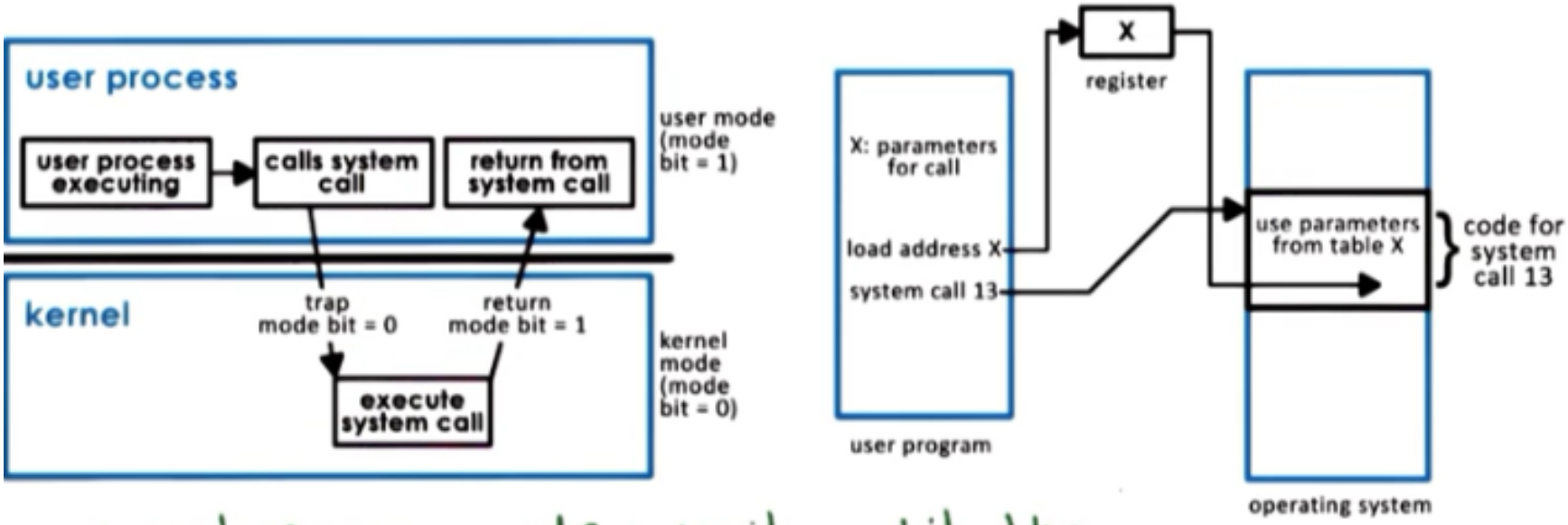
System Call

System Call Flowchart



System Call

System Call Flowchart



synchronous mode: wait until the sys call completes

Departamento de Computação

System Call

- fork
  - Creates a new process
  - Child gets a copy of process state: user memory, kernel state
  - Child gets a different, unique PID
  - Parent learns child's PID
- Exec
  - Replaces user memory from file
  - Retains same PID and most other kernel state
- wait
  - Waits for a child to exit
  - Zombie process -- child dies before parent calls wait
  - Returns immediately if child already exited, zombie leave

Departamento de Computação

System Call Return Value

- 0 superior após o sucesso
- erro -1 ou algum outro valor <0
- No último, esse é o código de erro
- Use perror () para imprimir uma mensagem de erro descritiva

Departamento de Computação

Exercícios

1- programa bash

- Utilize um editor de texto;
- Crie um arquivo , e digite o seguinte dentro dele:

```
#! / Bin / bash
echo “Hello DC - SO"
```

- A primeira linha diz ao Linux usar o interpretador bash para executar este script. Nomei o arquivo de lab_dc.sh. Em seguida, faça o script executável:

```
$ chmod 700 lab_dc.sh
$ ./lab_dc.sh
Hello DC - SO
```

Departamento de Computação

Exercícios

2 - Programa bash

- Escreva um programa que copia todos os arquivos em um diretório e, em seguida, exclui o diretório junto com seu conteúdo. Isso pode ser feito com os seguintes comandos:

```
$ mkdir trash
$ cp * trash
$ rm -rf trash
$ mkdir trash
```

Departamento de Computação

Exercícios

2 - Programa bash

```
$ mkdir trash
$ cp * trash
$ rm -rf trash
$ mkdir trash
```

- Em vez de ter que digitar tudo de forma interativa no shell, escreva um programa de shell em vez disso:

```
$ cat trash.sh
#!/bin/bash
# this script deletes some files
  cp * trash
  rm -rf trash
  mkdir trash
  echo “Deleted all files!”
```

Exec()

Exemplo:

- /* arquivo test_exec.c */

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    execl("/bin/ls","ls","test_exec.c",NULL) ;
    printf ("Eu ainda nao estou morto\n") ;
    exit(0);
}
```

Departamento de Computação

Exec()

Resultado da execução:

test_exec.c

- O comando ls é executado, mas o printf não. Isto mostra que o processo não retorna após a execução do execl.
- O exemplo seguinte mostra a utilidade do fork neste caso.

Departamento de Computação

fork()

```
/* arquivo test_exec_fork.c */
#include <stdio.h>
 #include <unistd.h>
 int main()
 {
        if ( fork()==0 )
        execl( "/bin/ls","ls","test_exec.c",NULL) ;
         else {
                sleep(2) ;  /* espera o fim de ls para executar o printf() */
        printf ("Eu sou o pai e finalmente posso continuar\n") ;
        }
         exit(0);
}
```

Departamento de Computação

Referências

- Khaled N. Khasawneh (2016) Advanced Operating Systems (CS 202).
- CS422/52(2014) Introduction to Operating Systems.

fork()

Resultado da execução:

test_exec.c

Eu sou o pai e finalmente posso continuar

- Neste caso, o filho morre após a execução do ls, e o pai continuará a viver, executando então o printf.