

Fundamentos de Sistemas de Operação

LEI - 2023/2024

Vitor Duarte
M^a. Cecília Gomes

1

Aula 23

- Máquinas virtuais e hipervisores (cont.)
- Contentores
- OSTEP: cap. B
- Silberschatz, Operating Systems Concepts, 10th Ed, Cap 2.7, 2.8, 18.5

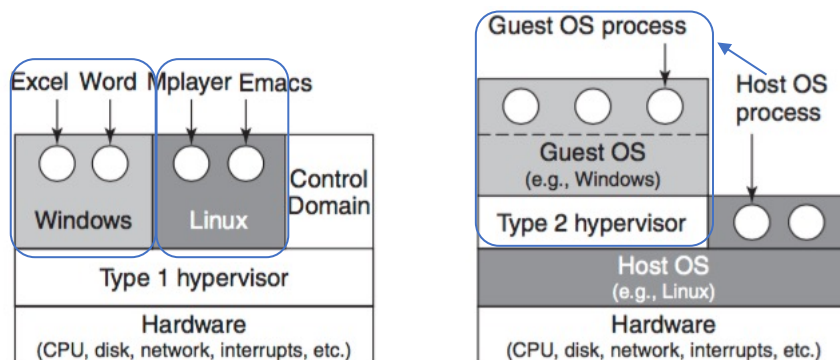
2

Virtualização de hardware

- *Virtual Machine*
 - Oferece a interface de uma máquina “real”
 - Arquitetura real ou semelhante a uma real
 - Pode oferecer uma arquitetura diferente da real em que executa (JVM, emuladores de computadores antigos ou consolas de jogos, etc)
 - Capaz de executar um SO com os seus processos
 - Virtualiza o modo supervisor, periféricos, etc
 - Acedendo aos periféricos reais ou com emulação
- Sobre o hardware real (host) podemos executar várias instâncias de VM
 - Para obter vários SO num computador
 - Uma forma de conseguir multiprogramação
 - IBM CP/CMS (1968), VM/370 (1972)

Tipos de hipervisores

- Hipervisor é responsável pela gestão da virtualização entre VMs
- Type 1: Corre nativo (como um microkernel). Ex: VMware ESXI, Xen
- Type 2: Um processo no SO host. Ex: VMware Workstation, VirtualBox



Implementação

- virtualização do cpu

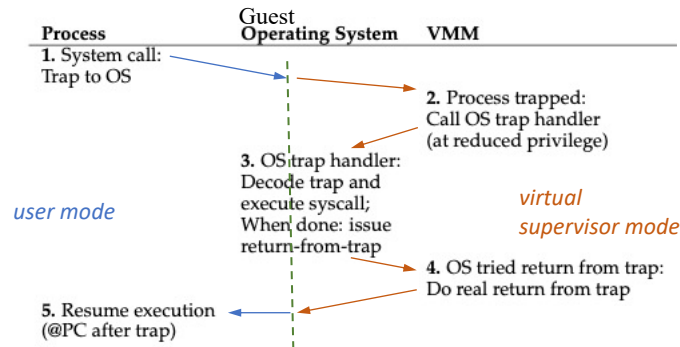
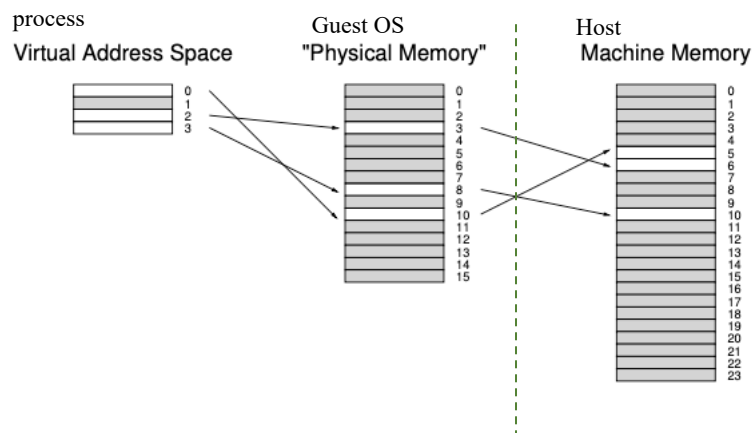


Figure B.3: System Call Flow with Virtualization

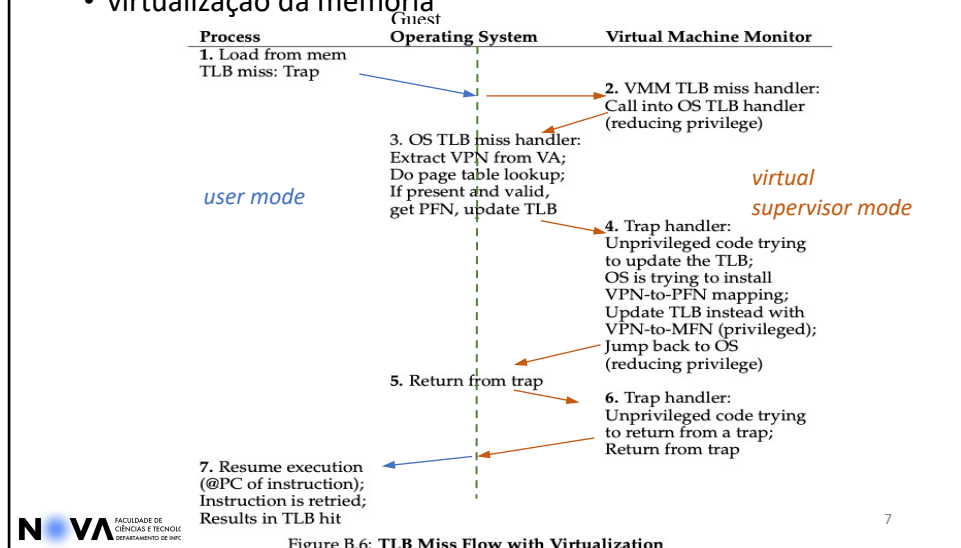
Implementação

- virtualização da memória



Implementação

• virtualização da memória



7

Vantagens / Desvantagens

- Executar vários OS no mesmo computador
 - partilha de recursos para melhor utilização destes
 - facilidades na gestão (migração, clonagem, backup, etc)
- Executar programas de OS diferentes em simultâneo
- Permite desenvolvimento e teste em diferentes ambientes
 - até quando estes ainda não existem
- Está na base das ofertas dos serviços Cloud
- Maior ocupação/duplicação de software entre *guests*
- Pode ter baixo desempenho
 - especialmente se emula todo o hardware e um ISA diferente
- As VMs podem ter melhor o desempenho se...
 - Virtualizam a arquitetura host em vez de emular outra
 - Suporte do hardware: Intel-VT; AMD-V
 - Hipervisores Tipo 1
 - Usando paravirtualização
 - Guest OS com drivers específicos para VM

8

Outras formas de isolamento

- Evitar sobrecarregar com a virtualização do hardware, novo kernel e nova instalação completa do SO
- Quando na mesma arquitectura e SO, pretende-se:
 - Criar ambientes tipo SO, com configurações específicas, para a execução de alguns processos (por variados motivos: segurança; gestão de recursos; dependências de software; etc.)
 - Estes ambientes criados e administrados por utilizadores normais, sem direitos de administrador (root)
 - Manter vários destes ambientes definidos e usar quando necessário, possivelmente vários em simultâneo
 - "empacotar" estes ambientes e definições para fácil distribuição/cópia e execução por qualquer outro utilizador
- Soluções existem com os nomes de zonas, partições, contentores e outros

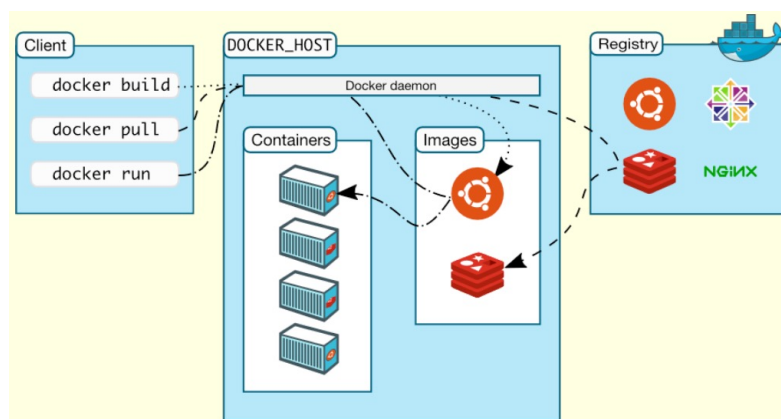
Mecanismos básicos dos SO

- Isolanto o sistema de ficheiros do SO
 - **chroot** – change root directory (syscall e comando)
 - permite definir para o(s) processo(s) um sistema de ficheiros isolado do normal (incluindo bibliotecas, programas, etc)
- Isolando restantes recursos (Linux):
 - Name Spaces – permite criar uma partição do SO para um conjunto de processos só veja/aceda a um espaço de PIDs, UIDs, mounts, networks e cgroups
 - Control groups – permite definir os recursos permitidos e em que limites para um grupo de processos (incluindo cpu, memória, devices, redes/firewall, etc)

Containers (contentores)

- Ambiente virtual para execução de um ou vários processos, executando sobre o kernel e o hardware real, mas em isolamento face aos restantes processos e com recursos limitados
- Suporta espaços separados para PIDs, UIDs, sistema de ficheiros, IP rede, hostname, limites ao cpu e memória, etc
 - funciona como um computador separado, com possibilidade de partilhar diretorias com o sistema nativo e de comunicar via rede com o exterior
- Exemplos de implementações de containers (Linux):
 - Docker, LXC, OpenVZ, etc...
 - no caso do Docker, permite execução de containers Linux sobre MS-Windows e MacOS, para além de Linux

Componentes e ferramentas do Docker



Componentes e ferramentas do Docker

- *Docker Host* – computador que corre o dockerd, contém os containers e os executa
- *Docker Cliente* - as ferramentas (docker) para o utilizador usar os containers (pode estar no mesmo computador que o Host ou não)
- *Image* – imagem base (SF com software, etc) para um contentor poder executar (muitas imagens são definidas com base noutras)
- *Container* – uma instância de imagem e respetiva configuração, em execução, parado ou terminado
- *Registry* - serviço (num servidor) que guarda imagens, podendo ser local ou remoto, privado ou público (ex. Docker Hub)

13

Demo...

- Instalação, partir de docker.com ou usando pacotes da sua distribuição Linux.
- Exemplo (como *root* ou usando *sudo*):

```
apt install docker.io (debian/ubuntu/...)
adduser fso docker (garantir permissões para o utilizador "fso")
```

- Download/execução:

```
docker run hello-world
```

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
719385e32844: Pull complete
Digest: sha256:c79d06dfdfd3d3eb04cafd0dc2bacab0992ebc243e083cabe208bac4dd7
Status: Downloaded newer image for hello-world:latest
```

download da imagem

```
Hello from Docker!
```

```
This message shows that your installation appears to be working correctly.
```

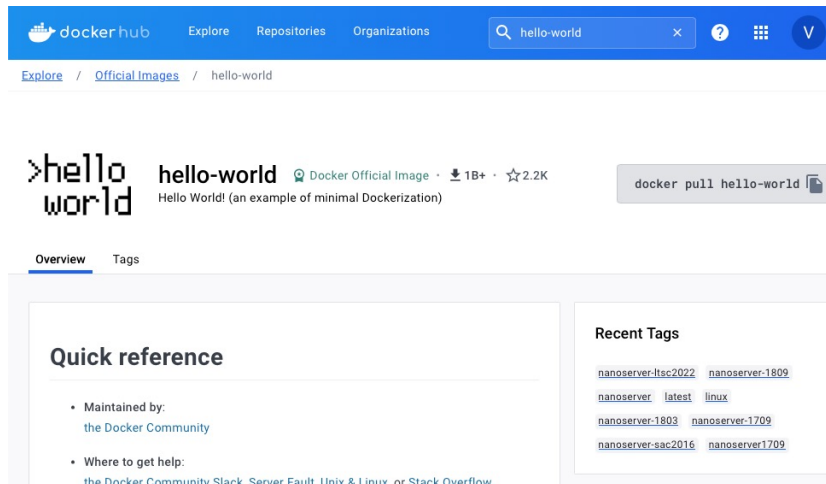
```
To generate this message, Docker took the following steps:
```

1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub. (amd64)
 3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.
- ```
.... etc....
```

output da execução

14

# hub.docker.com



15

## • outros comandos (ver documentação)

```
docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
(nenhum container em execução)
```

```
docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
f9231781c163 hello-world "/hello" 8 minutes ago Exited (0) ...
(container terminado)
```

```
docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
hello-world latest 9c7a54a9a43c 7 months ago 13.3kB
(imagens disponíveis localmente)
```

```
docker (obter help)
Usage: docker [OPTIONS] COMMAND
```

A self-sufficient runtime for containers

```
Options:
 --config string Location of client config files (default
"/home/fso/.docker")
 -c, --context string Name of the context to use to connect to the da
 -D, --debug Enable debug mode
 -H, --host list Daemon socket(s) to connect to
 -l, --log-level string Set the logging level
("debug"|"info"|"warn"|"error"|"fatal")
etc.....
```

16



- mais

`docker images`

`docker run --rm -ti busybox sh`

`ps aux` dentro vs `ps aux` fora

`telnetd` dentro e `telnet` de fora para dentro

partilhando "volumes"/diretorias em modo `rw`

`docker run --rm -ti -v $PWD:/mnt:rw busybox sh`

etc

17

## Vantagens / Desvantagens

- Empacotar aplicações com todas as suas dependências – independente do SO onde executa, fácil de migrar, copiar, executar, suspender, continuar, etc
- Isolamentar entre SO real e contentor, e vice-versa
- Controlar e monitorizar dos recursos atribuídos a cada contentor
- Facilita e automatiza a gestão e orquestração de sistemas complexos (p.e. responder a aumento de clientes Web, lançando mais instâncias do container com o servidor Web)
- Alguns overheads, mas menores que na virtualização completa do hardware
- Um container só pode executar sobre SO para que foi desenvolvido e usar recursos existentes (não contém um OS completo, nem hardware virtual)

18