Fundamentos de Segurança Informática (FSI)

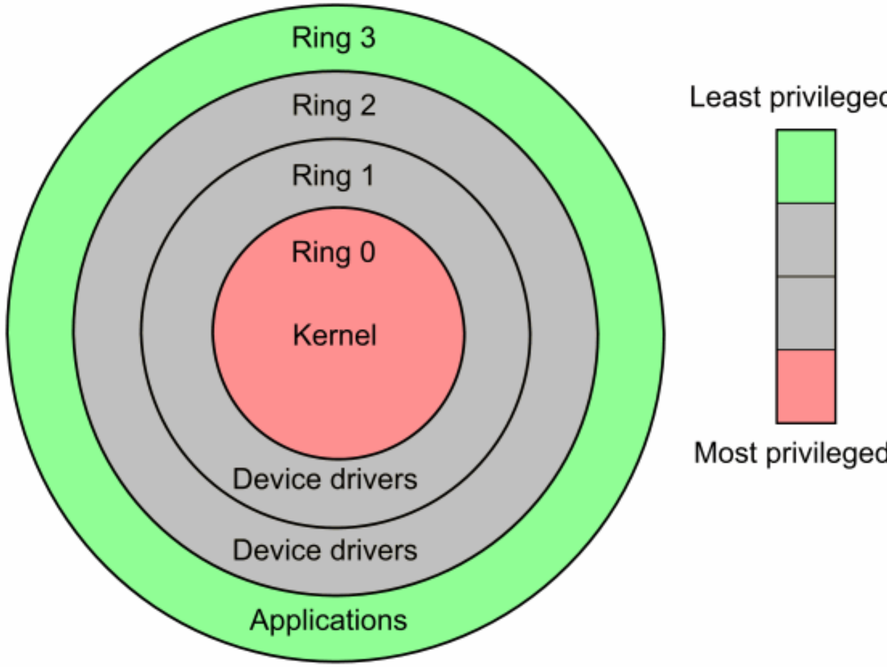
2021/2022 - LEIC

Manuel Barbosa
mbb@fc.up.pt

Aula 8
Segurança de Sistemas 2

Kernel

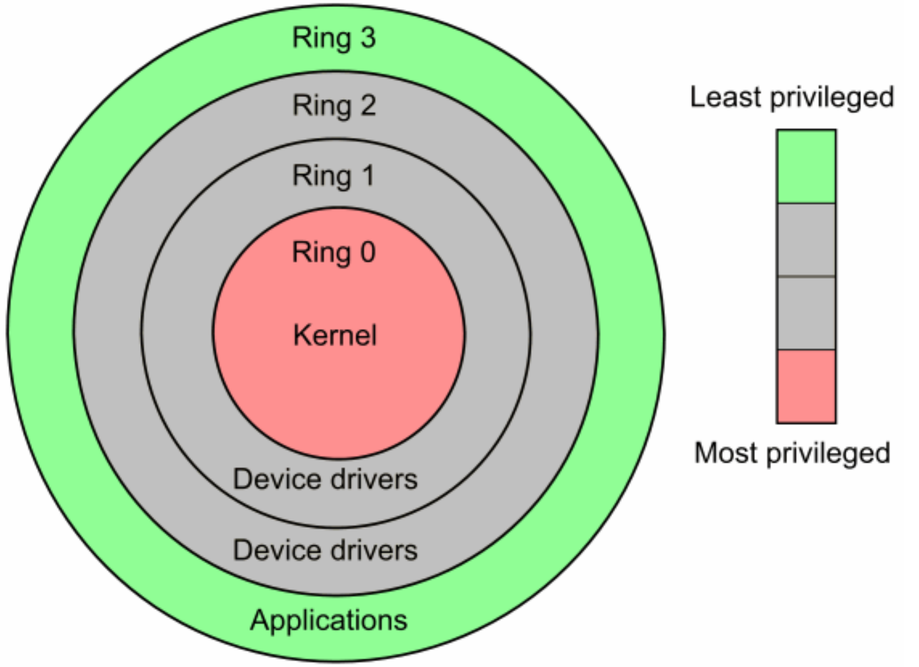
- O Kernel é a parte do sistema operativo que desempenha as operações mais críticas:
  - o processador está em *Kernel mode* e todas as operações são permitidas a esse código
  - os processadores permitem geralmente definir vários níveis de privilégio (ring em Intel)
  - em muitos casos usam-se apenas dois: *kernel mode* e *user mode*
  - o código em *user mode* não tem acesso directo aos recursos do sistema
  - qualquer troca de informação entre os dois níveis faz parte de uma superfície de ataque



http://programming.realworldrobotics.com/system-kernel/local-kernel-debugging/understanding-user-and-kernel-mode

Kernel

- O Kernel está protegido dos processos em modo utilizador:
  - tem um espaço de memória gerido de forma independente (pelo próprio kernel)
  - o processador garante que apenas código que corre em modo kernel pode executar um conjunto de instruções privilegiadas
  - qualquer processo em user mode (incluindo device drivers) tem de aceder aos recursos do sistema usando *system calls*
  - parte do código das system calls executa em kernel mode!



http://programming.realworldrobotics.com/system-kernel/local-kernel-debugging/understanding-user-and-kernel-mode

Confinamento (veremos mais à frente)

- Os pontos de entrada em system calls são críticos:
  - para causar danos, um processo em modo utilizador tem de o fazer através de uma system call!
  - Isto implica implementar sistemas de monitorização e controlo de chamadas ao sistema
  - Reference monitor:
    - sempre presente (se terminar, têm de ser terminados os processos monitorizados)
    - tem de ser simples para poder ser analisado e validado mais facilmente que o sistema todo

System Calls

- Controlo de processos (e.g., fork, load, execute, wait, alloc, free)
- Acesso a ficheiros (criar, ler, escrever, etc.)
- Gestão de dispositivos (obter acesso, escrever/ler, etc.)
- Configuração do sistema (hora, data, características, estado, etc.)
- Comunicações (estabelecer ligação, enviar/receber mensagens, etc.)
- Proteção (alterar/obter pemissões de acesso a recursos)

System Calls

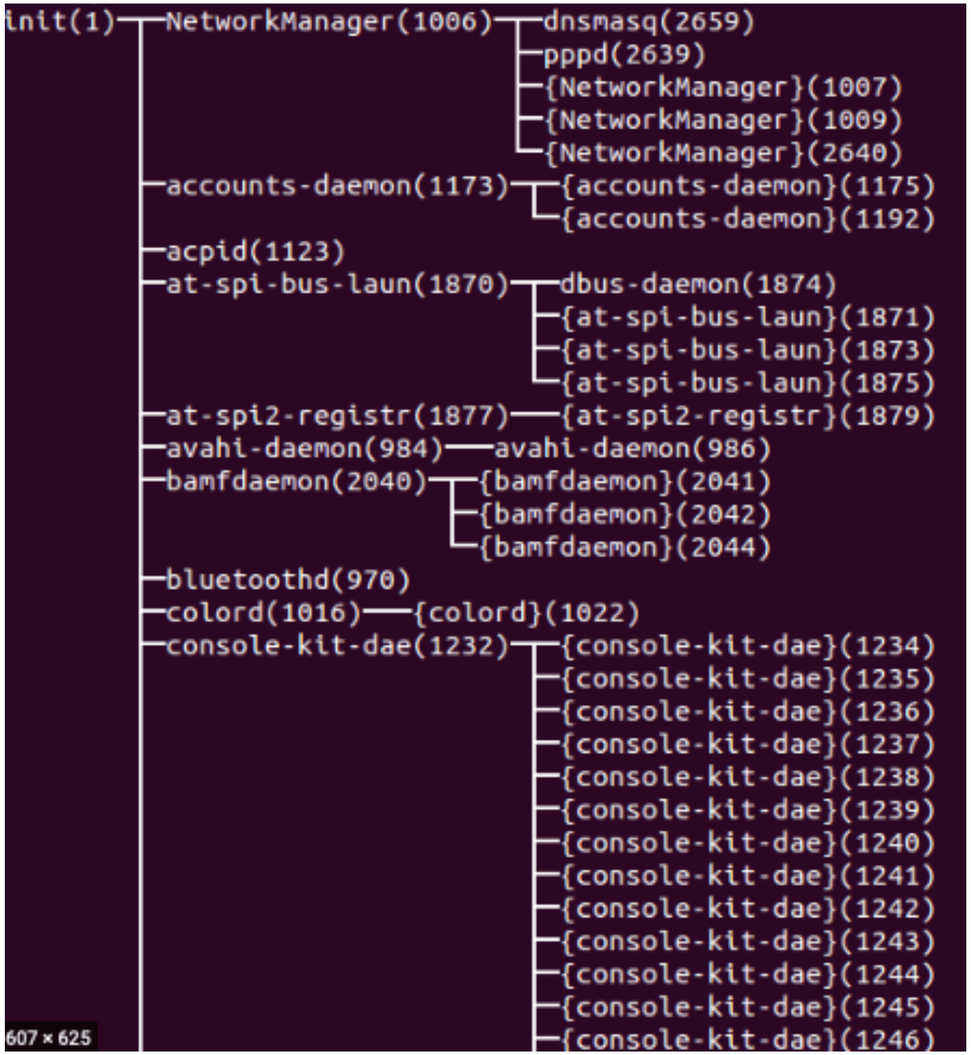
- Para entrar num modo de funcionamento com mais privilégios, o código user mode deve:
  - preparar argumentos, e identificar um ponto permitido para acesso a kernel mode
  - executar uma instrução especial que passa o controlo para o kernel
- Existe um número limitado deste tipo de pontos de entrada:
  - registos específicos para parâmetros, que são tipicamente apontadores para memória de processos em user mode
  - o processamento dessa informação é da total responsabilidade do kernel
- Um número limitado de pontos de entrada => superfície de ataque bem definida

Processos

- O Kernel define a noção de processo: uma instância de um programa que está a executar
- Os programas começam por estar guardados em armazenamento não volátil (e.g., código no disco)
- Para serem executados têm de ser carregados para memória e receber um identificador como processo
- Cada processo deve executar num contexto em que tem acesso a um conjunto de recursos, que devem estar disponíveis independentemente de outros processos
- A fronteira entre processos é uma fronteira de confiança: os processos têm de estar confinados/isolados entre si

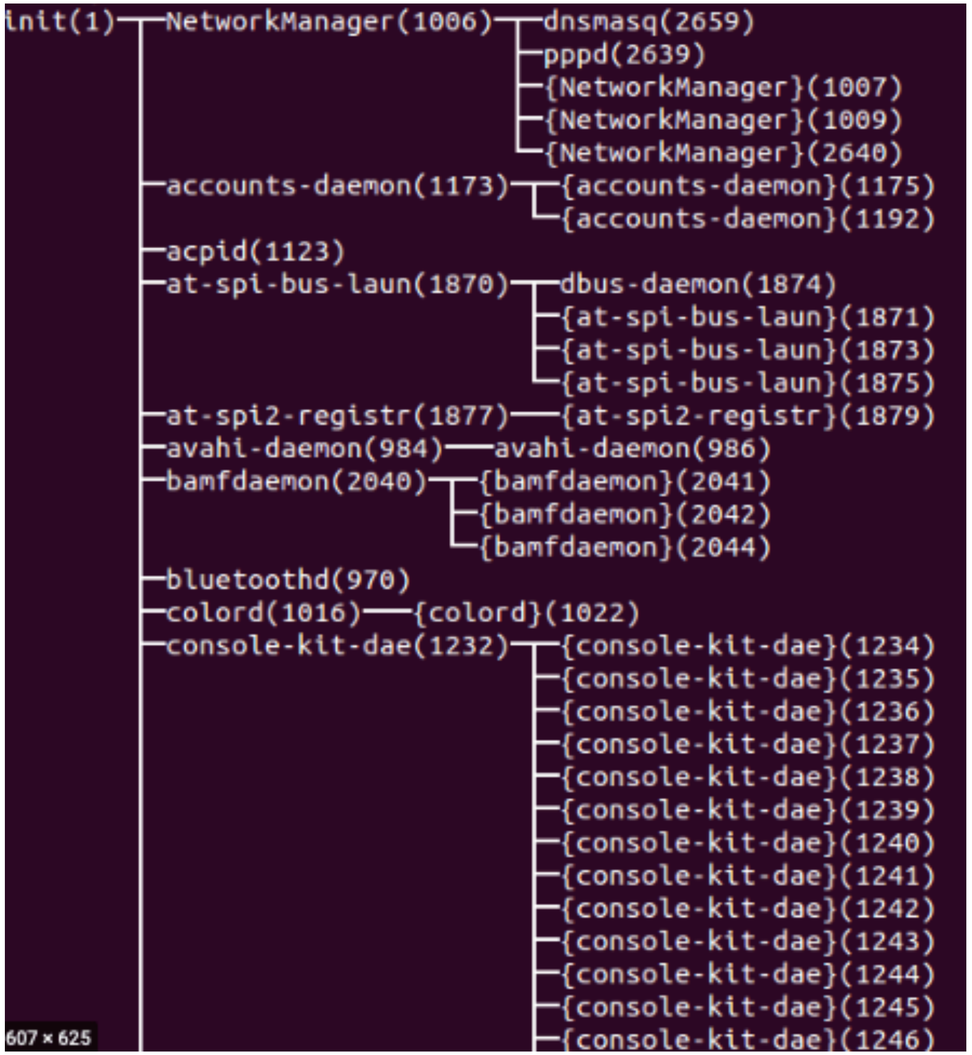
Processos

- O Kernel garante acesso a recursos:
  - atribui uma fração razoável de tempo de processador a cada processo (time slicing)
  - atribui um espaço do memória sobre o qual o processo pode trabalhar
  - concede acesso a outros recursos através de system calls
- Nos SO multi-utilizador, existe um conjunto de processos base que interagem com o utilizador humano:
  - quando o utilizador lança uma aplicação, o SO vê um dos processos que interagem com o utilizador (e.g., shell, GUI, etc.) a criar um novo processo => *forking*
  - Os SO gerem uma hierarquia de processos, em que tipicamente os descendentes herdam os privilégios dos seus criadores



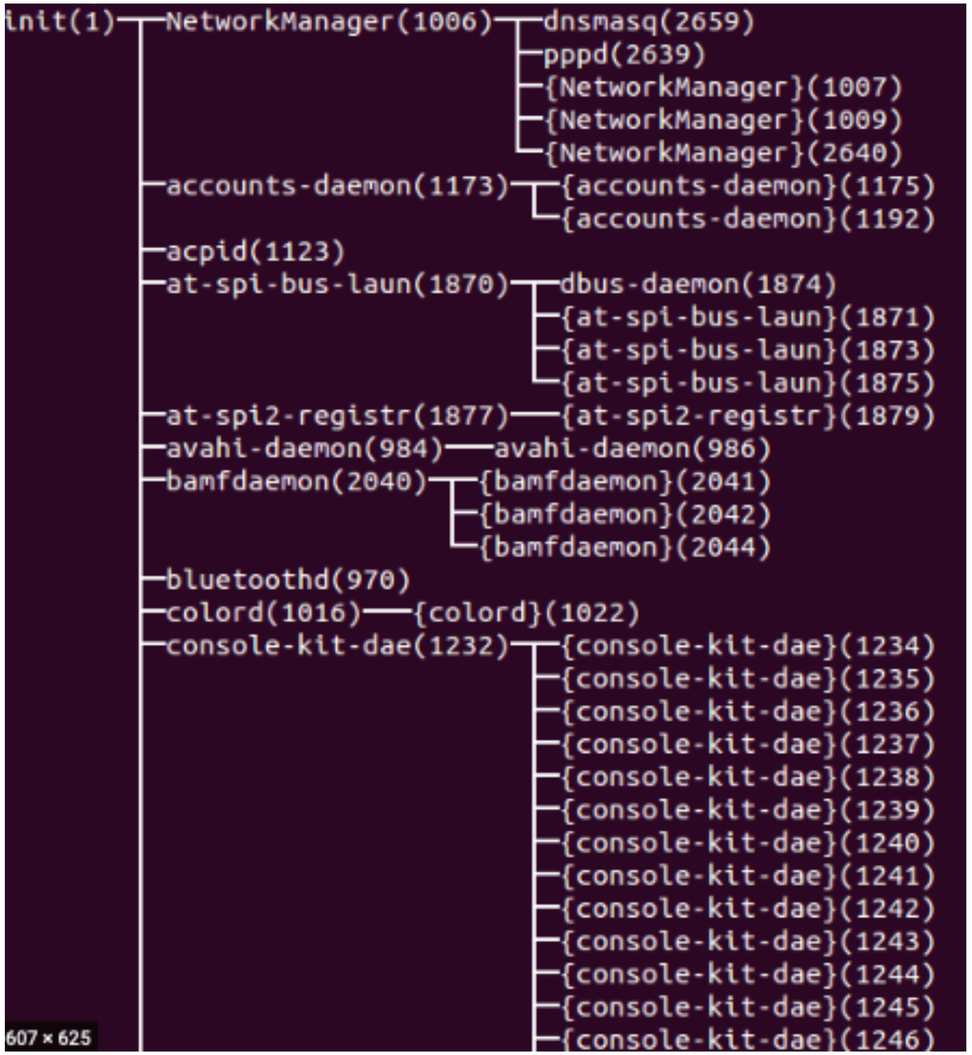
Processos

- Em Linux pode ver-se a árvore com `pstree`
- O processo raiz é o `init` (PID 0)
- PID é um identificador único de processo
- As permissões atribuídas a cada processo dependem do utilizador que o cria
  - cada utilizador tem um UID (único para o utilizador) e um GID (único para o grupo)
  - tipicamente o 0 é reservado para o super-user (root)
  - o processo é associado aos mesmos UID, GID



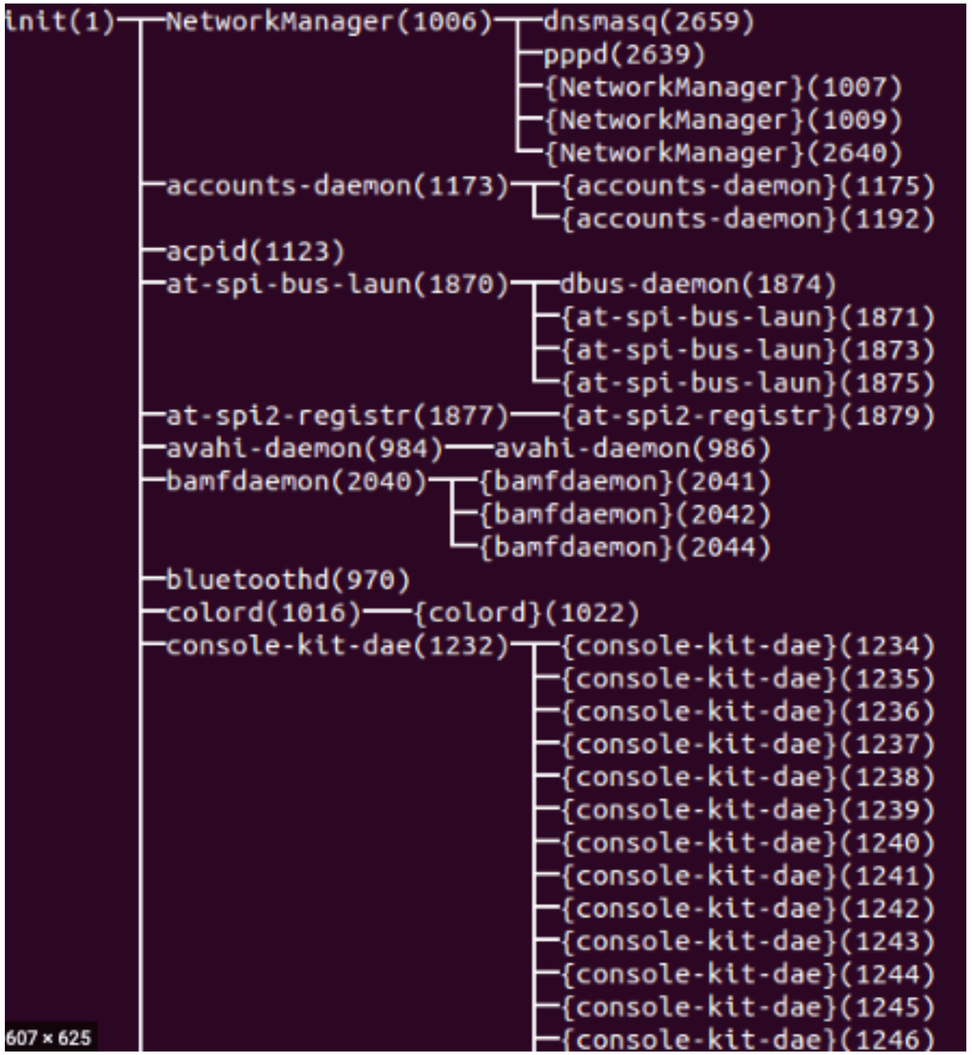
Processos

- Muitas vezes é necessário comunicar entre processos (IPC) => system calls
  - através do sistema de ficheiros
  - memória partilhada
  - mensagens síncronas: pipes, sockets
  - mensagens assíncronas: signals



Processos

- Daemons, services:
  - processos que não são visíveis pelo utilizador (diretamente)
  - E.g., indexação, login remoto, impressoras, sincronização de ficheiros, etc.
  - geralmente arrancados antes dos próprios processos que interagem com o utilizador
  - executam tipicamente com privilégios superiores aos dos utilizadores e sobrevivem às suas sessões



Modelo de Confiança

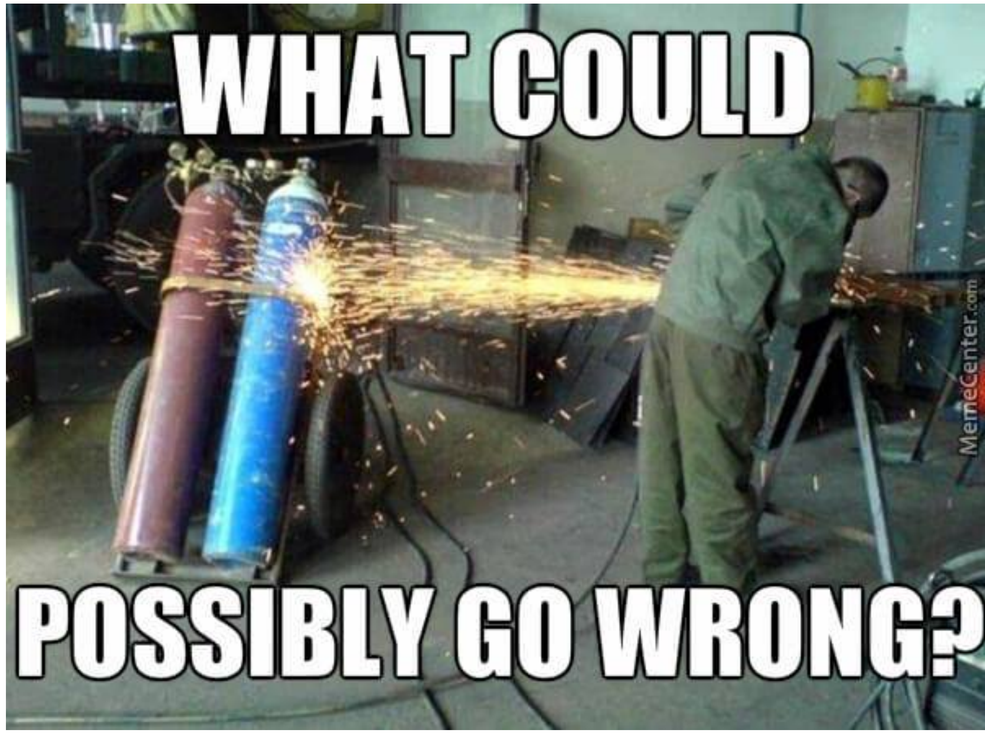
- A confiança depositada nos processos lançados é indutiva:
  - o código armazenado no computador (nomeadamente a BIOS e o kernel) após uma instalação é "confiável"
  - o processo de *boot* utiliza este código para colocar o *kernel* em memória e passar-lhe o controlo, criando um estado "confiável"
  - o kernel lança processos com permissões que garantem que nenhum novo processo pode alterar o estado de confiança
  - os processos de hibernação preservam o estado de confiança
  - os administradores podem alterar o software instalado no sistema e o sistema de permissões, mas garantem que qualquer actualização preserva o estado de confiança

Modelo de Confiança

- O que significa "confiável":
  - que o sistema faz exatamente (e apenas) aquilo que foi especificado
  - exemplo: não transmite a nossa informação sensível para o exterior sem autorização
  - exemplo: garante que as nossas comunicações são estabelecidas com as entidades com quem queremos comunicar (e.g., servidores Google)
  - exemplo: cifra toda a informação em disco e limpa a memória quando fazemos shutdown

Modelo de Ameaças

- Ataques em todos os níveis do boot:
  - BIOS corrompida
  - ficheiros de hibernação corrompidos/roubados
  - *bootloader* corrompido
  - cold boot attacks



- Vulnerabilidades que afetam a implementação dos mecanismos de arranque fazem um bypass completo ao modelo de confiança (o arranque é a âncora)

Medidas de Mitigação

- A maioria dos problemas de segurança surgem através de erros de administração
- Veremos exemplos de processos maliciosos (malware) e das formas que utilizam para corromper o modelo de confiança, bem como medidas de mitigação
- A monitorização é uma forma de mitigação comum para detectar quebras neste modelo:
  - logs de eventos permitem detectar comportamentos suspeitos, como o crash repetido de um processo que está a tentar explorar uma vulnerabilidade
  - aplicações de monitorização de processos permitem visualizar os processos que estão a executar, os recursos que utilizam, e os ficheiros de código que os originam
- a mediação de instalação/execução de código com base em assinaturas digitais fornece também um entrave à introdução de código malicioso num sistema

Medidas de Mitigação

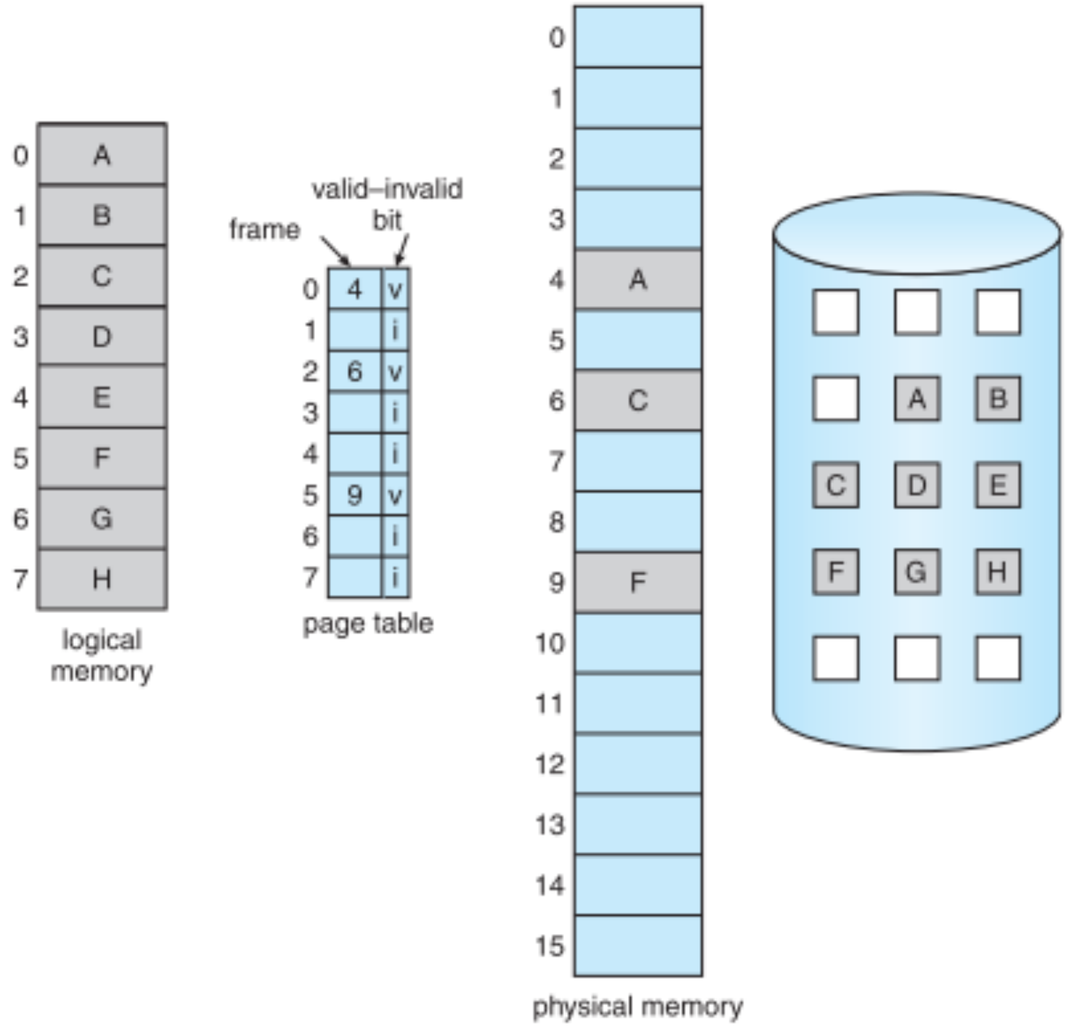
- Para já vamos estudar os mecanismos de segurança que são utilizados nos sistemas operativos para garantir isolamento entre processos
  - impedem que um utilizador com poucos privilégios (e portanto fora do círculo de confiança) possa utilizar o sistema para além do que lhe é permitido
  - impedem que um processo que contenha uma vulnerabilidade não abra uma porta que corrompa todo o sistema (defesa em profundidade)
  - vamos focar-nos na gestão da memória, processos e sistema de ficheiros, que são aspectos fundamentais comuns a todos os SO

Memória

- A regra fundamental da gestão de memória diz que:
  - um processo não pode aceder ao espaço de memória de outro processo
  - a confidencialidade, integridade e controlo de fluxo do kernel tem de ser protegida de todos os processos que executam em modo utilizador
- Como se garante?
  - em run-time os acessos são mediados por um conjunto de mecanismos de hardware e software geridos pelo kernel
  - as partes da memória virtual que estão em disco podem ser alvo de ataque *off-line* se um adversário puder aceder a essa informação => disco cifrado

Memória

- O espaço de memória gerido por um SO é muito maior do que o espaço físico:
  - o processador dá suporte a mecanismos de memória virtual
  - o espaço de endereçamento de um processo está dividido em páginas
  - algumas estão em memória outras em armazenamento não volátil
  - quando é necessário trocar, diz-se que ocorreu um "page fault"



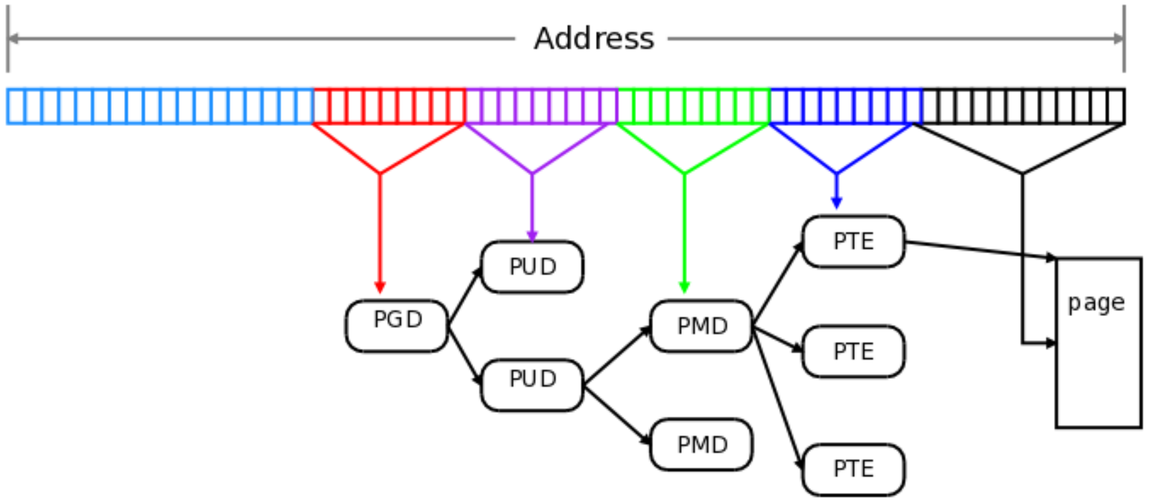
https://unix.stackexchange.com/questions/474605/does-a-pager-move-or-copy-a-page-when-handling-a-page-fault

Tradução de Endereços

- A tradução de endereços necessária à implementação de mecanismos de memória virtual cumpre dois propósitos:
  - isolamento: cada processo acede a uma zona de memória que não existe na realidade, e que dá visão/acesso limitados aos recursos
  - eficiência: esconde mecanismos de optimização (caching, speculative access, paging, etc.)
- Nos últimos anos ficou claro que algumas optimizações criam, de facto, novos pontos de ataque através de canais subliminares (Specter, Meltdown)

Tradução de Endereços

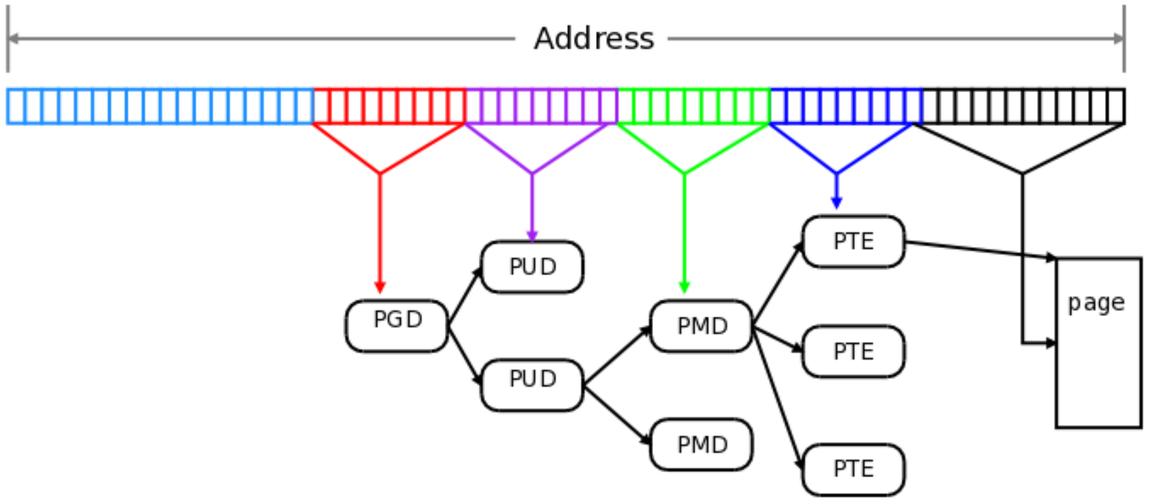
- A memória virtual está dividida em páginas, e.g., 4KB
- O sistema tem de armazenar, para cada página (se utilizada) a sua localização física
- Page-table: árvore esparsa com informação nas folhas
- Processador oferece suporte para gerir estas estruturas



https://lwn.net/Articles/717293/

Tradução de Endereços

- Aceder a uma page table (que está em memória) é penalizador
- Translation Lookaside Buffer (TLB):
  - cache de páginas traduzidas recentemente
  - informação para controlo de acessos em cada página
  - Read/Write/eXecute



https://lwn.net/Articles/717293/

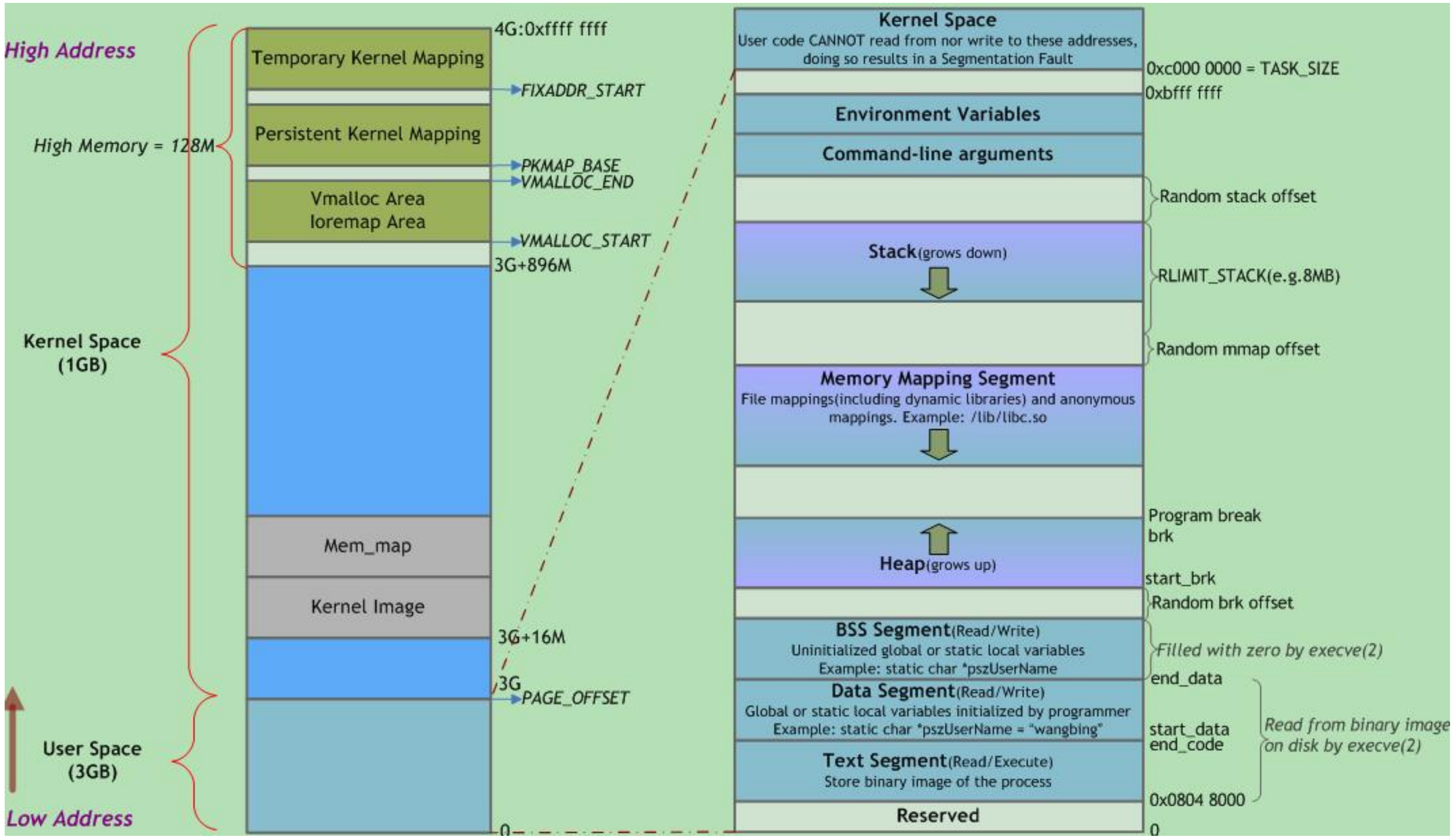
Tradução de Endereços

- Como lidar com chamadas ao sistema?
  - uma gestão totalmente independente dos espaços de endereçamento tornaria as mudanças de contexto muito ineficientes
- Kernel mapping:
  - parte da memória virtual do kernel está mapeada diretamente na memória virtual de cada processo, mas com permissões diferentes
  - User mode (UR,UW,UX), Kernel/privileged mode (PR,PW,PX)

Memória Virtual

Parte do espaço de memória de um processo é ocupado/ gerido pelo Kernel, por questões de eficiência.

O processo não tem acesso a esse espaço, mas pode interagir com ele via system calls: e.g.: memory map.



https://www.programmersought.com/article/1093948310/

Kernel Mapping

- Quando um processo faz uma system call não é necessário alterar o sistema de mapeamento de páginas:
  - a memória relevante para o kernel já está mapeada
  - mais importante: a parte da memória do processo relevante à system call coexiste no mesmo espaço de endereçamento
- Quando se muda de processo de utilizador, as tabelas de páginas são alteradas, mas as que dizem respeito à memória do kernel são as mesmas

Defesa em profundidade

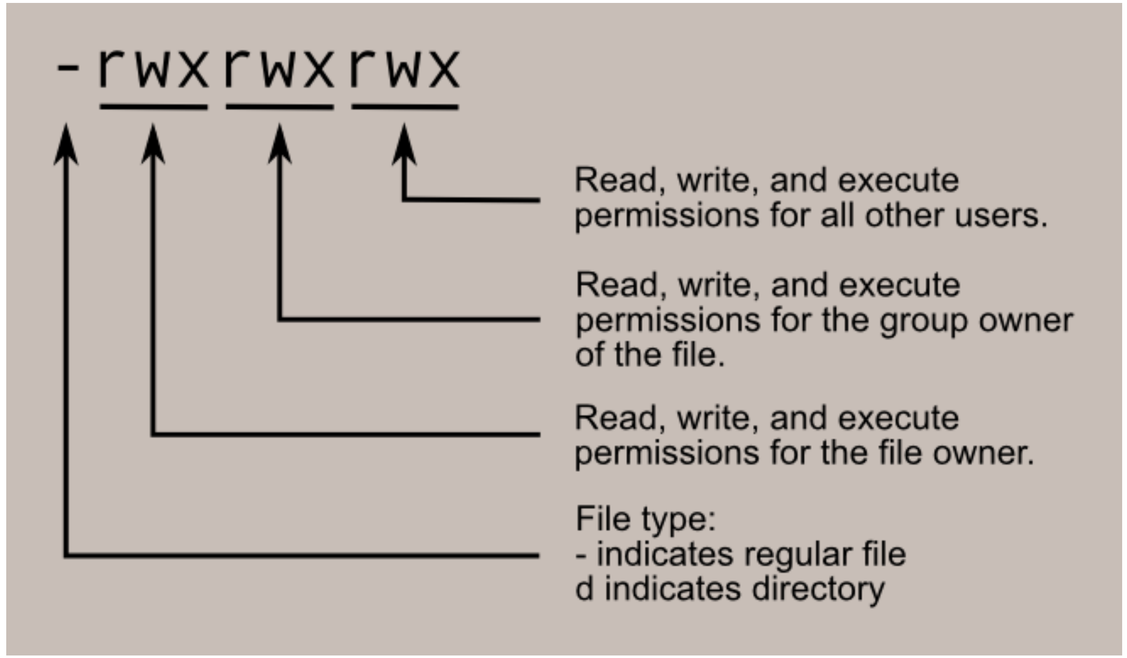
- Que permissões deve ter o kernel sobre a memória dos outros processos:
  - todas as permissões => ERRO!
- Defesa em profundidade:
  - nem o kernel deve ser capaz de violar a regra W^X
  - impedir o kernel de escrever em partes da memória do utilizador é uma forma de impedir fugas de informação/código malicioso no caso de kernel corrompido

Sistema de Ficheiros

- Veremos como exemplo os sistemas *nix:
  - atores: utilizadores
  - recursos: ficheiros e pastas
  - ações/acessos:
    - read/write/execute: evidente para ficheiros
    - read/write/execute: listar conteúdo, criar conteúdo adicional, "entrar" na pasta
    - alterar as próprias permissões?

Sistema de Ficheiros

- Cada utilizador pertence a um grupo: permite uma forma de RBAC
- Cada recurso tem um *owner* e um *grupo*
  - as permissões são atribuídas de forma independente a
    - owner (ACL)
    - membros do grupo associado ao recurso (batch ACL/RBAC rígido?)
    - todos os outros utilizadores (?)



Sistema de Ficheiros

- Superuser:
  - antigamente um utilizador especial (*root)*
  - hoje em dia um papel/role: `sudo`
  - `uid = 0` utilizado para identificar esse utilizador/papel
  - boas práticas: utilização mínima

Sistema de Ficheiros

- Alteração de permissões:
  - sempre permitido ao *superuser*
  - permissões podem ser alteradas pelo *owner* (`chmod`)
  - *owner* pode ser alterado pelo *superuser* (`chown`)
  - grupo poder ser alterado por *owner* e *superuser* (`chgrp`)
- *owner* altera permissões => *Discretionary Access Control*
- Mandatory Access Control => apenas administrador (e.g., SELinux)

Sistema de Ficheiros

- Como funciona o login?
  - o sistema executa um processo login como root
  - esse processo autentica o utilizador (tem acesso às credenciais no sistema)
  - altera o seu próprio `uid` e `gid` para os associados ao utilizador
  - lança o processo de `shell`
- Crítico: o login executa *drop privileges*
- O reverso (*elevate privileges*) deve ser impossível (e o `passwd`?)

Sistema de Ficheiros

- O bit `setuid` associado a um ficheiro:
  - Permite fixar o utilizador associado um processo ao owner do executável (e não ao utilizador que executa)
  - Pode ser ativado pelo *superuser* e pelo *owner* do ficheiro
  - Implicações:
    - se o *owner* tiver muitos privilégios
    - permite elevação de privilégios!
  - No caso do `passwd` o *owner* é o utilizador *root*.

Sistema de Ficheiros

- Tudo é um ficheiro:
  - como minimizar o número de system calls/superfície de ataque?
  - utilizar a mesma interface construída para o sistema de ficheiros para outros recursos
  - Em *nix: sockets, pipes, dispositivos de I/O, objetos do kernel, etc.
  - O sistema de controlo de acessos é sempre o mesmo!

Exemplo de utilização: Android

- Os sistemas Android executam sobre um sub-sistema Linux
- Problema:
  - restringir o acesso de aplicações a recursos
  - solução: cada aplicação tem o seu próprio utilizador
  - problema: múltiplos utilizadores?
    - solução ad-hox: `u1_a23`

Privilégios de Processos

- Quando executamos um processo, tipicamente executa com o UID do utilizador que o lançou
  - pode aceder aos mesmos recursos
- Alguns processos são executados com o UID do owner do ficheiro executável (bit setuid = 1)
- Os processos do kernel arrancam com UID = 0 (root)
  - acesso a todos os recursos => privilégio máximo!

Privilégios de Processos

- A transição de privilégios é mais complexa do que parece à partida
- Um processo tem, de facto, três UIDs:
  - Effective User ID (EUID): determina as permissões
  - Real User ID (RUID): utilizador que lançou o processo
  - Saved User ID (SUID): utilizado em transições, lembra o anterior

Privilégios de Processos

- O que é possível fazer em tempo de execução?
- O utilizador *root* pode usar a system call `setuid(x)` para alterar estes valores para UIDs arbitrários:
  - `EUID => x`
  - `RUID => x`
  - `SUID => x`
- Isto permite a um processo reduzir os próprios privilégios:
  - quando o Apache cria um processo para atender um utilizador reduz os privilégios do processo descendente

Sistema de Ficheiros

- Permissões de processos:
  - os utilizadores interagem com o sistema através de processos
  - cada processo tem associado um *effective user id*
    - determina as permissões do processo
    - em geral: uid do utilizador que lançou o processo
    - existem exceções: e.g., mudar a password usando `passwd`

Privilégios de Processos

- É possível fazer uma redução temporária de privilégios:
- A system call `seteuid(x)` altera apenas o EUID e preserva o RUID e o SUID
- A system call `setuid(x)` para não root permite restaurar EUID ao valor RUID ou SUID!
- Utilização típica:
  - baixar privilégios => executar código de risco => restaurar privilégios
- Perigo: usar `seteuid` quando se pretende alteração permanente (porquê?)

Privilégios de Processos

- Complexidade:
  - mesmo com um sistema tão simples
  - existe um sistema de transições entre estados de confiança
  - onde é muito fácil cometer erros

Conclusão

- O sistema de controlo de acessos em *nix é essencialmente uma implementação de Access Control Lists, com algum *batching*
- Vantagem => simples e funciona na prática
- Desvantagem => pouco robusto e pouco flexível
  - uma falha num processo tipo *passwd* ou *ssh* (`euid = 0`) tem consequências catastróficas
  - *root* utilizado para muita coisa => erros de administração