Capítulo 7

Mecanismos de controle de acesso

A implementação do controle de acesso em um sistema computacional deve ser independente das políticas de controle de acesso adotadas. Como nas demais áreas de um sistema operacional, a separação entre mecanismo e política é importante, por possibilitar trocar a política de controle de acesso sem ter de modificar a implementação do sistema. A infraestrutura de controle de acesso deve ser ao mesmo tempo *inviolável* (impossível de adulterar ou enganar) e *incontornável* (todos os acessos aos recursos do sistema devem passar por ela).

7.1 Infraestrutura básica

A arquitetura básica de uma infraestrutura de controle de acesso típica é composta pelos seguintes elementos (Figura 7.1):

Bases de sujeitos e objetos (*User/Object Bases*): relação dos sujeitos e objetos que compõem o sistema, com seus respectivos atributos;

Base de políticas (*Policy Base*): base de dados contendo as regras que definem como e quando os objetos podem ser acessados pelos sujeitos, ou como/quando os sujeitos podem interagir entre si;

Monitor de referências (*Reference monitor*): elemento que julga a pertinência de cada pedido de acesso. Com base em atributos do sujeito e do objeto (como suas respectivas identidades), nas regras da base de políticas e possivelmente em informações externas (como horário, carga do sistema, etc.), o monitor decide se um acesso deve ser permitido ou negado;

Mediador (impositor ou *Enforcer*): elemento que medeia a interação entre sujeitos e objetos; a cada pedido de acesso a um objeto, o mediador consulta o monitor de referências e permite/nega o acesso, conforme a decisão deste último.

É importante observar que os elementos dessa estrutura são componentes lógicos, que não impõem uma forma de implementação rígida. Por exemplo, em um sistema operacional convencional, o sistema de arquivos possui sua própria estrutura de controle de acesso, com permissões de acesso armazenadas nos próprios arquivos, e um pequeno monitor/mediador associado a algumas chamadas de sistema, como `open` e `mmap`. Outros recursos (como áreas de memória ou semáforos) possuem suas próprias regras e estruturas de controle de acesso, organizadas de forma diversa.

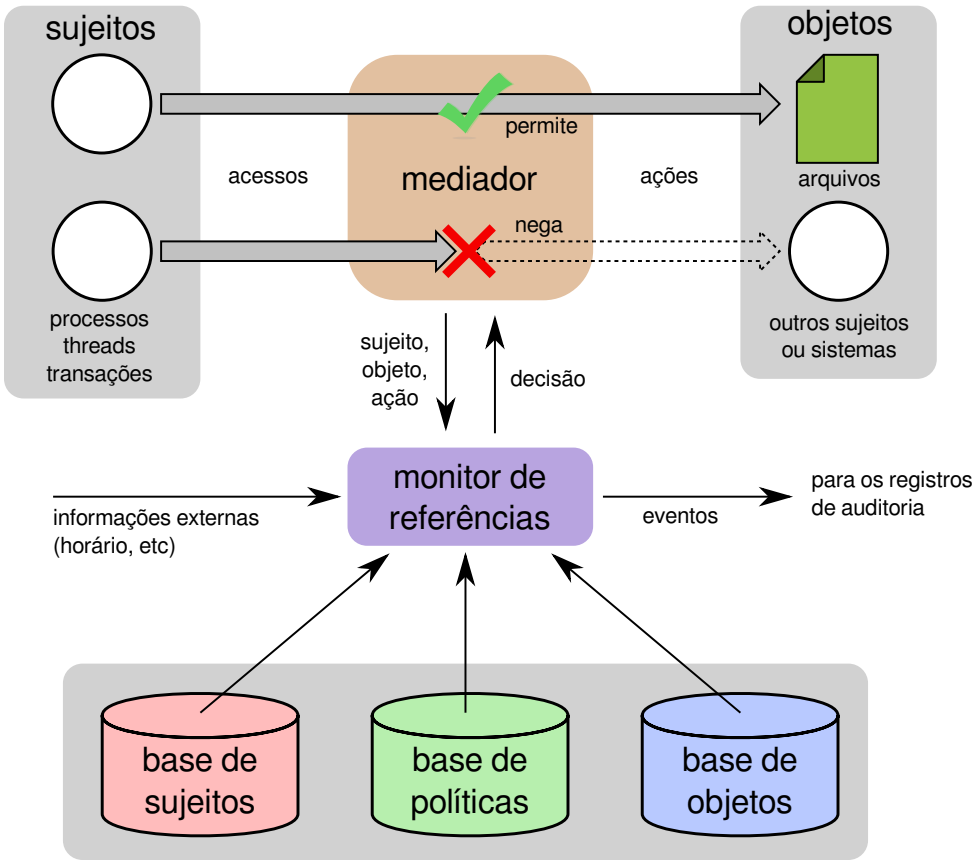


Figura 7.1: Estrutura genérica de uma infraestrutura de controle de acesso.

7.2 Controle de acesso em UNIX

Os sistemas operacionais do mundo UNIX implementam um sistema de ACLs básico bastante rudimentar, no qual existem apenas três sujeitos: *user* (o dono do recurso), *group* (um grupo de usuários ao qual o recurso está associado) e *others* (todos os demais usuários do sistema). Para cada objeto existem três possibilidades de acesso: *read*, *write* e *execute*, cuja semântica depende do tipo de objeto (arquivo, diretório, *socket* de rede, área de memória compartilhada, etc.). Dessa forma, são necessários apenas 9 bits por arquivo para definir suas permissões básicas de acesso.

A Figura 7.2 apresenta uma listagem de diretório típica em UNIX. Nessa listagem, o arquivo `hello-unix.c` pode ser acessado em leitura e escrita por seu proprietário (o usuário `maziero`, com permissões `rw-`), em leitura pelos usuários do grupo `prof` (permissões `r--`) e em leitura pelos demais usuários do sistema (permissões `r--`). Já o arquivo `hello-unix` pode ser acessado em leitura, escrita e execução por seu proprietário (permissões `rwx`), em leitura e execução pelos usuários do grupo `prof` (permissões `r-x`) e não pode ser acessado pelos demais usuários (permissões `---`). No caso de diretórios, a permissão de leitura autoriza a listagem do diretório, a permissão de escrita autoriza sua modificação (criação, remoção ou renomeação de arquivos ou sub-diretórios) e a permissão de execução autoriza usar aquele diretório como diretório de trabalho ou parte de um caminho.

É importante destacar que o controle de acesso é normalmente realizado apenas durante a abertura do arquivo, para a criação de seu descritor em memória. Isso significa

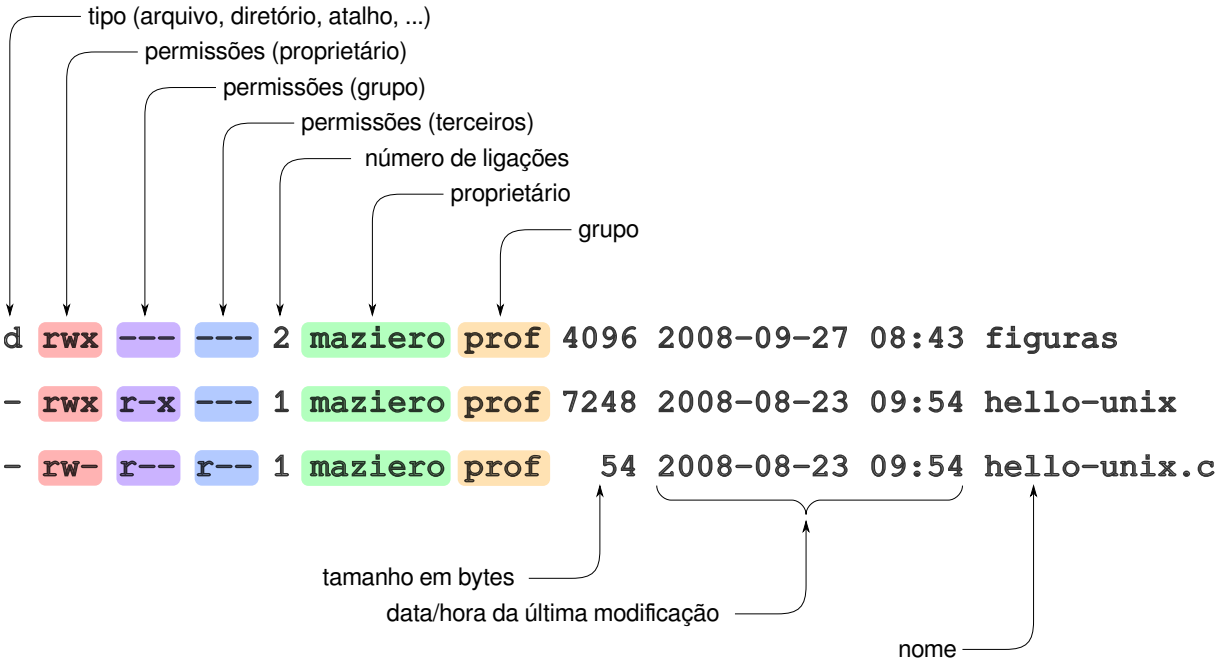


Figura 7.2: Listas de controle de acesso em UNIX.

que, uma vez aberto um arquivo por um processo, este terá acesso ao arquivo enquanto o mantiver aberto, mesmo que as permissões do arquivo sejam modificadas para impedir esse acesso. O controle contínuo de acesso a arquivos é pouco frequentemente implementado em sistemas operacionais, porque verificar as permissões de acesso a cada operação de leitura ou escrita teria um forte impacto negativo sobre o desempenho do sistema.

Dessa forma, um descritor de arquivo aberto pode ser visto como uma capacidade (vide Seção 6.3.4), pois a posse do descritor permite ao processo acessar o arquivo referenciado por ele. O processo recebe esse descritor ao abrir o arquivo e deve apresentá-lo a cada acesso subsequente; o descritor pode ser transferido aos processos filhos ou até mesmo a outros processos, outorgando a eles o acesso ao arquivo aberto. A mesma estratégia é usada em *sockets* de rede, semáforos e outros mecanismos de IPC.

O padrão POSIX 1003.1e definiu ACLs mais detalhadas para o sistema de arquivos, que permitem definir permissões para usuários e grupos específicos além do proprietário do arquivo. Esse padrão é parcialmente implementado em vários sistemas operacionais, como o Linux e o FreeBSD. No Linux, os comandos `getfacl` e `setfacl` permitem manipular essas ACLs, como mostra o exemplo a seguir:

```
1  host:~> ll
2  -rw-r--r-- 1 maziero prof 2450791 2009-06-18 10:47 main.pdf
3
4  host:~> getfacl main.pdf
5  # file: main.pdf
6  # owner: maziero
7  # group: maziero
8  user::rw-
9  group::r--
10 other::r--
11
12 host:~> setfacl -m diogo:rw,rafael:rw main.pdf
13
14 host:~> getfacl main.pdf
15 # file: main.pdf
16 # owner: maziero
17 # group: maziero
18 user::rw-
19 user:diogo:rw-
20 user:rafael:rw-
21 group::r--
22 mask::rw-
23 other::r--
```

No exemplo, o comando da linha 12 define permissões de leitura e escrita específicas para os usuários `diogo` e `rafael` sobre o arquivo `main.pdf`. Essas permissões estendidas são visíveis na linha 19 e 20, junto com as permissões UNIX básicas (nas linhas 18, 21 e 23).

7.3 Controle de acesso em Windows

Os sistemas Windows baseados no núcleo NT (NT, 2000, XP, Vista e sucessores) implementam mecanismos de controle de acesso bastante sofisticados [Brown, 2000; Russinovich et al., 2008]. Em um sistema Windows, cada sujeito (computador, usuário, grupo ou domínio) é unicamente identificado por um *identificador de segurança* (SID - *Security IDentifier*). Cada sujeito do sistema está associado a um *token de acesso*, criado no momento em que o respectivo usuário ou sistema externo se autentica no sistema. A autenticação e o início da sessão do usuário são gerenciados pelo LSASS (*Local Security Authority Subsystem*), que cria os processos iniciais e os associa ao *token* de acesso criado para aquele usuário. Esse *token* normalmente é herdado pelos processos filhos, até o encerramento da sessão do usuário. Ele contém o identificador do usuário (SID), dos grupos aos quais ele pertence, privilégios a ele associados e outras informações. Privilégios são permissões para realizar operações genéricas, que não dependem de um recurso específico, como reiniciar o computador, carregar um *driver* ou depurar um processo.

Por outro lado, cada objeto do sistema está associado a um *descritor de segurança* (SD - *Security Descriptor*). Como objetos, são considerados arquivos e diretórios, processos, impressoras, serviços e chaves de registros, por exemplo. Um descritor de

segurança indica o proprietário e o grupo primário do objeto, uma lista de controle de acesso de sistema (SACL - *System ACL*), uma lista de controle de acesso discricionária (DACL - *Discretionary ACL*) e algumas informações de controle.

A DACL contém uma lista de regras de controle de acesso ao objeto, na forma de ACEs (*Access Control Entries*). Cada ACE contém um identificador de usuário ou grupo, um modo de autorização (positiva ou negativa), um conjunto de permissões (ler, escrever, executar, remover, etc.), sob a forma de um mapa de bits. Quando um sujeito solicita acesso a um recurso, o SRM (*Security Reference Monitor*) compara o *token* de acesso do sujeito com as entradas da DACL do objeto, para permitir ou negar o acesso. Como sujeitos podem pertencer a mais de um grupo e as ACEs podem ser positivas ou negativas, podem ocorrer conflitos entre as ACEs. Por isso, um mecanismo de resolução de conflitos é acionado a cada acesso solicitado ao objeto.

A SACL define que tipo de operações sobre o objeto devem ser registradas pelo sistema, sendo usada basicamente para fins de auditoria (Seção 8). A estrutura das ACEs de auditoria é similar à das ACEs da DACL, embora defina quais ações sobre o objeto em questão devem ser registradas para quais sujeitos. A Figura 7.3 ilustra alguns dos componentes da estrutura de controle de acesso dos sistemas Windows.

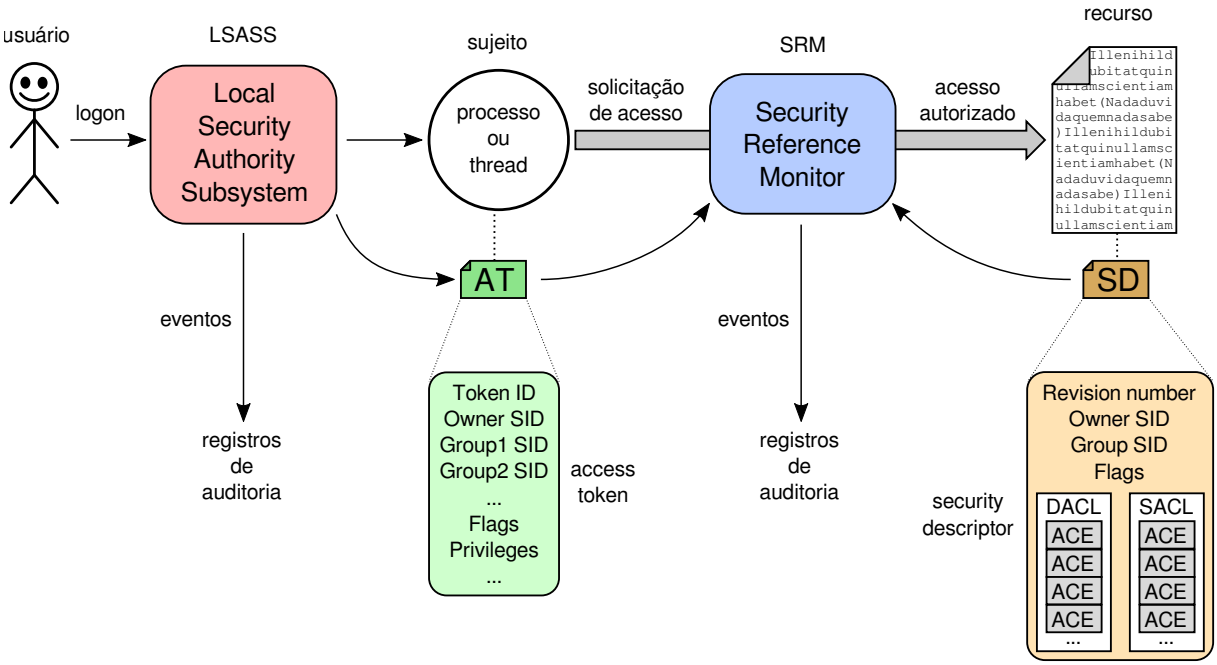


Figura 7.3: Listas de controle de acesso no Windows.

7.4 Outros mecanismos

As políticas de segurança básicas utilizadas na maioria dos sistemas operacionais são discricionárias, baseadas nas identidades dos usuários e em listas de controle de acesso. Entretanto, políticas de segurança mais sofisticadas vêm sendo gradualmente agregadas aos sistemas operacionais mais complexos, visando aumentar sua segurança. Algumas iniciativas dignas de nota são apresentadas a seguir:

- O SELinux é um mecanismo de controle de acesso multipolíticas, desenvolvido pela NSA (*National Security Agency*, USA) [Loscocco and Smalley, 2001] a

partir da arquitetura flexível de segurança *Flask* (*Flux Advanced Security Kernel*) [Spencer et al., 1999]. Ele constitui uma infraestrutura complexa de segurança para o núcleo Linux, capaz de aplicar diversos tipos de políticas obrigatórias aos recursos do sistema operacional. A política default do SELinux é baseada em RBAC e DTE, mas ele também é capaz de implementar políticas de segurança multinível. O SELinux tem sido criticado devido à sua complexidade, que torna difícil sua compreensão e configuração. Em consequência, outros projetos visando adicionar políticas MAC mais simples e fáceis de usar ao núcleo Linux têm sido propostos, como *LIDS*, *SMACK* e *AppArmor*.

- O sistema operacional Windows Vista incorpora uma política denominada *Mandatory Integrity Control* (MIC) que associa aos processos e recursos os níveis de integridade *Low*, *Medium*, *High* e *System* [Microsoft], de forma similar ao modelo de Biba (Seção 6.4.2). Os processos normais dos usuários são classificados como de integridade média, enquanto o navegador Web e executáveis provindos da Internet são classificados como de integridade baixa. Além disso, o Vista conta com o UAC (*User Account Control*) que aplica uma política baseada em RBAC: um usuário com direitos administrativos inicia sua sessão como usuário normal, e somente ativa seu papel administrativo quando necessita efetuar uma ação administrativa.

- O projeto TrustedBSD [Watson, 2001] implementa ACLs no padrão POSIX, capacidades POSIX e o suporte a políticas obrigatórias como Bell LaPadula, Biba, categorias e TE/DTE. Uma versão deste projeto foi portada para o MacOS X, sendo denominada *MacOS X MAC Framework*.

- Desenvolvido nos anos 90, o sistema operacional experimental *EROS* (*Extremely Reliable Operating System*) [Shapiro and Hardy, 2002] implementou um modelo de controle de acesso totalmente baseado em capacidades. Nesse modelo, todas as interfaces dos componentes do sistema só são acessíveis através de capacidades, que são usadas para nomear as interfaces e para controlar seu acesso. O sistema *EROS* deriva de desenvolvimentos anteriores feitos no sistema operacional KeyKOS para a plataforma S/370 [Bomberger et al., 1992].

- Em 2009, o sistema operacional experimental *SeL4*, que estende o sistema micronúcleo L4 [Liedtke, 1996] com um modelo de controle de acesso baseado em capacidades similar ao utilizado no sistema EROS, tornou-se o primeiro sistema operacional cuja segurança foi formalmente verificada [Klein et al., 2009]. A verificação formal é uma técnica de engenharia de software que permite demonstrar matematicamente que a implementação do sistema corresponde à sua especificação, e que a especificação está completa e sem erros.

- O sistema *Trusted Solaris* [Sun Microsystems] implementa várias políticas de segurança: em MLS (*Multi-Level Security*), níveis de segurança são associados aos recursos do sistema e aos usuários. Além disso, a noção de domínios é implementada através de "compartimentos": um recurso associado a um determinado compartimento só pode ser acessado por sujeitos no mesmo compartimento. Para limitar o poder do super-usuário, é usada uma política de tipo RBAC, que divide a administração do sistema em vários papéis que podem ser atribuídos a usuários distintos.

7.5 Mudança de privilégios

Normalmente, os processos em um sistema operacional são sujeitos que representam o usuário que os lançou. Quando um novo processo é criado, ele herda as credenciais de seu processo-pai, ou seja, seus identificadores de usuário e de grupo. Na maioria dos mecanismos de controle de acesso usados em sistemas operacionais, as permissões são atribuídas aos processos em função de suas credenciais. Com isso, normalmente cada novo processo herda as mesmas permissões de seu processo-pai, pois possui as mesmas credenciais dele.

O uso de privilégios fixos é adequado para o uso normal do sistema, pois os processos de cada usuário só devem ter acesso aos recursos autorizados para esse usuário. Entretanto, em algumas situações esse mecanismo se mostra inadequado. Por exemplo, caso um usuário precise executar uma tarefa administrativa, como instalar um novo programa, modificar uma configuração de rede ou atualizar sua senha, alguns de seus processos devem possuir permissões para as ações necessárias, como editar arquivos de configuração do sistema. Os sistemas operacionais atuais oferecem diversas abordagens para resolver esse problema:

Usuários administrativos: são associadas permissões administrativas às sessões de trabalho de alguns usuários específicos, permitindo que seus processos possam efetuar tarefas administrativas, como instalar softwares ou mudar configurações. Esta é a abordagem utilizada em alguns sistemas operacionais de amplo uso. Algumas implementações definem vários tipos de usuários administrativos, com diferentes tipos de privilégios, como acessar dispositivos externos, lançar máquinas virtuais, reiniciar o sistema, etc. Embora simples, essa solução é falha, pois se algum programa com conteúdo malicioso for executado por um usuário administrativo, terá acesso a todas as suas permissões.

Permissões temporárias: conceder sob demanda a certos processos do usuário as permissões de que necessitam para realizar ações administrativas; essas permissões podem ser descartadas pelo processo assim que concluir as ações. Essas permissões podem estar associadas a papéis administrativos (Seção 6.6), ativados quando o usuário tiver necessidade deles. Esta é a abordagem usada pela infraestrutura UAC (*User Access Control*) [Microsoft], na qual um usuário administrativo inicia sua sessão de trabalho como usuário normal, e somente ativa seu papel administrativo quando necessita efetuar uma ação administrativa, desativando-o imediatamente após a conclusão da ação. A ativação do papel administrativo pode impor um procedimento de reautenticação.

Mudança de credenciais: permitir que certos processos do usuário mudem de identidade, assumindo a identidade de algum usuário com permissões suficientes para realizar a ação desejada; pode ser considerada uma variante da atribuição de permissões temporárias. O exemplo mais conhecido de implementação desta abordagem são os flags `setuid` e `setgid` do UNIX, explicados a seguir.

Monitores: definir processos privilegiados, chamados *monitores* ou *supervisores*, recebem pedidos de ações administrativas dos processos não privilegiados, através de uma API pré-definida; os pedidos dos processos normais são validados e atendidos. Esta é a abordagem definida como *separação de privilégios* em [Provos

et al., 2003], e também é usada na infra-estrutura *PolicyKit*, usada para autorizar tarefas administrativas em ambientes *desktop* Linux.

Um mecanismo amplamente usado para mudança de credenciais consiste dos flags `setuid` e `setgid` dos sistemas UNIX. Se um arquivo executável tiver o flag `setuid` habilitado (indicado pelo caractere "s" em suas permissões de usuário), seus processos assumirão as credenciais do proprietário do arquivo. Portanto, se o proprietário de um arquivo executável for o usuário *root*, os processos lançados a partir dele terão todos os privilégios do usuário *root*, independente de quem o tiver lançado. De forma similar, processos lançados a partir de um arquivo executável com o flag `setgid` habilitado terão as credenciais do grupo associado ao arquivo. A Figura 7.4 ilustra esse mecanismo: o primeiro caso representa um executável normal (sem esses flags habilitados); um processo filho lançado a partir do executável possui as mesmas credenciais de seu pai. No segundo caso, o executável pertence ao usuário *root* e tem o flag `setuid` habilitado; assim, o processo filho assume a identidade do usuário *root* e, em consequência, suas permissões de acesso. No último caso, o executável pertence ao usuário *root* e tem o flag `setgid` habilitado; assim, o processo filho pertencerá ao grupo *mail*.

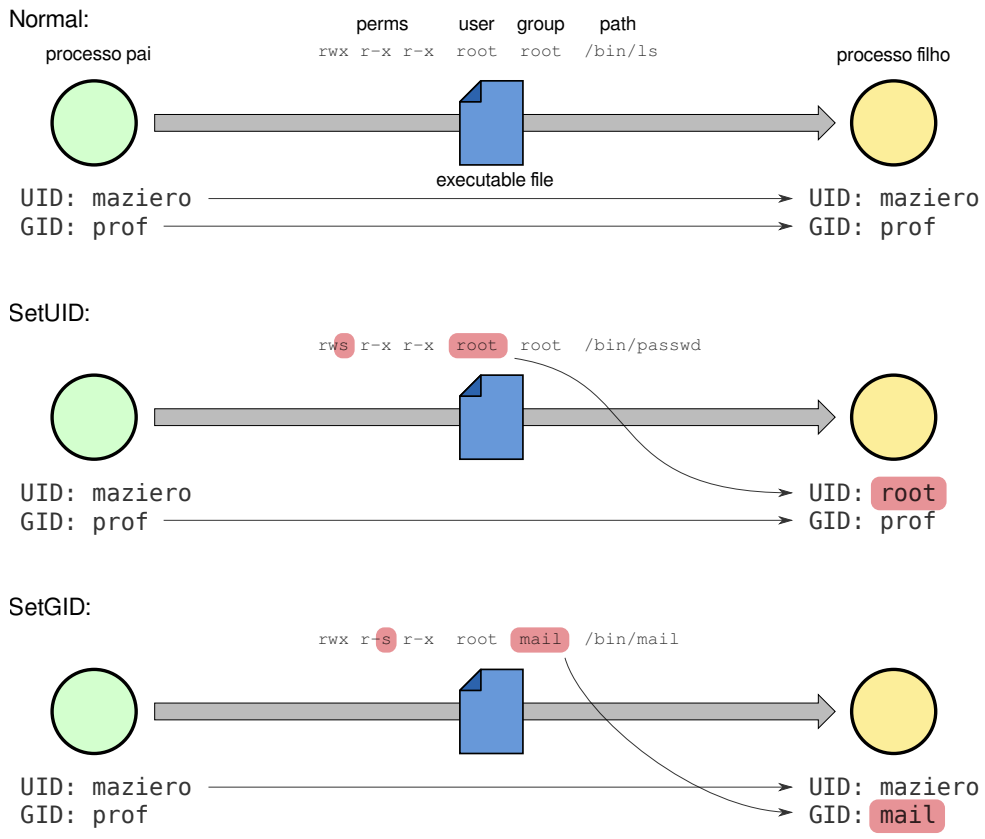


Figura 7.4: Funcionamento dos flags `setuid` e `setgid` do UNIX.

Os flags `setuid` e `setgid` são muito utilizados em programas administrativos no UNIX, como troca de senha e agendamento de tarefas, sempre que for necessário efetuar uma operação inacessível a usuários normais, como modificar o arquivo de senhas. Todavia, esse mecanismo pode ser perigoso, pois o processo filho recebe todos os privilégios do proprietário do arquivo, o que contraria o princípio do privilégio mínimo. Por exemplo, o programa `passwd` deveria somente receber a autorização para

modificar o arquivo de senhas (`/etc/passwd`) e nada mais, pois o superusuário (*root user*) tem acesso a todos os recursos do sistema e pode efetuar todas as operações que desejar. Se o programa `passwd` contiver erros de programação, ele pode ser induzido pelo seu usuário a efetuar ações não previstas, visando comprometer a segurança do sistema (vide Seção 1.3).

Uma alternativa mais segura aos flags `setuid` e `setgid` são os *privilégios POSIX* (*POSIX Capabilities*[1]), definidos no padrão POSIX 1003.1e [Gallmeister, 1994]. Nessa abordagem, o "poder absoluto" do super usuário é dividido em um grande número de pequenos privilégios específicos, que podem ser atribuídos a certos processos do sistema. Como medida adicional de proteção, cada processo pode ativar/desativar os privilégios que possui em função de sua necessidade. Vários sistemas UNIX implementam privilégios POSIX, como é o caso do Linux, que implementa:

- `CAP_CHOWN`: alterar o proprietário de um arquivo qualquer;
- `CAP_USER_DEV`: abrir dispositivos;
- `CAP_USER_FIFO`: usar *pipes* (comunicação);
- `CAP_USER_SOCK`: abrir *sockets* de rede;
- `CAP_NET_BIND_SERVICE`: abrir portas de rede com número abaixo de 1024;
- `CAP_NET_RAW`: abrir *sockets* de baixo nível (*raw sockets*);
- `CAP_KILL`: enviar sinais para processos de outros usuários.
- ... (outros +30 privilégios)

Para cada processo são definidos três conjuntos de privilégios: *Permitidos* (P), *Efetivos* (E) e *Herdáveis* (H). Os privilégios permitidos são aqueles que o processo pode ativar quando desejar, enquanto os efetivos são aqueles ativados no momento (respeitando-se $E \subseteq P$). O conjunto de privilégios herdáveis H é usado no cálculo dos privilégios transmitidos aos processos filhos. Os privilégios POSIX também podem ser atribuídos a programas executáveis em disco, substituindo os tradicionais (e perigosos) flags `setuid` e `setgid`. Assim, quando um executável for lançado, o novo processo recebe um conjunto de privilégios calculado a partir dos privilégios atribuídos ao arquivo executável e aqueles herdados do processo-pai que o criou [Bovet and Cesati, 2005].

Um caso especial de mudança de credenciais ocorre em algumas circunstâncias, quando é necessário **reduzir** as permissões de um processo. Por exemplo, o processo responsável pela autenticação de usuários em um sistema operacional deve criar novos processos para iniciar a sessão de trabalho de cada usuário. O processo autenticador geralmente executa com privilégios elevados, para poder acessar a bases de dados de autenticação dos usuários, enquanto os novos processos devem receber as credenciais do usuário autenticado, que normalmente tem menos privilégios. Em UNIX, um processo pode solicitar a mudança de suas credenciais através da chamada de sistema `setuid()`, entre outras. Em Windows, o mecanismo conhecido como *impersonation* permite a um

[1] O padrão POSIX usou indevidamente o termo "capacidade" para definir o que na verdade são privilégios associados aos processos. O uso indevido do termo *POSIX Capabilities* perdura até hoje em vários sistemas, como é o caso do Linux.

processo ou *thread* abandonar temporariamente seu *token* de acesso e assumir outro, para realizar uma tarefa em nome do sujeito correspondente [Russinovich et al., 2008].

Exercícios

1. Muitas vezes, usuários precisam executar ações que exigem privilégios administrativos, como instalar programas, reconfigurar serviços, etc. Neste contexto, indique quais das seguintes afirmações são incorretas; justifique suas respostas.

   (a) No mecanismo UAC – *User Access Control* – dos sistemas Windows, um usuário administrativo inicia sua seção de trabalho com seus privilégios de usuário normal e recebe mais privilégios somente quando precisa efetuar ações que os requeiram.

   (b) Alguns sistemas operacionais implementam mecanismos de *mudança de credenciais*, através dos quais um processo pode mudar de proprietário.

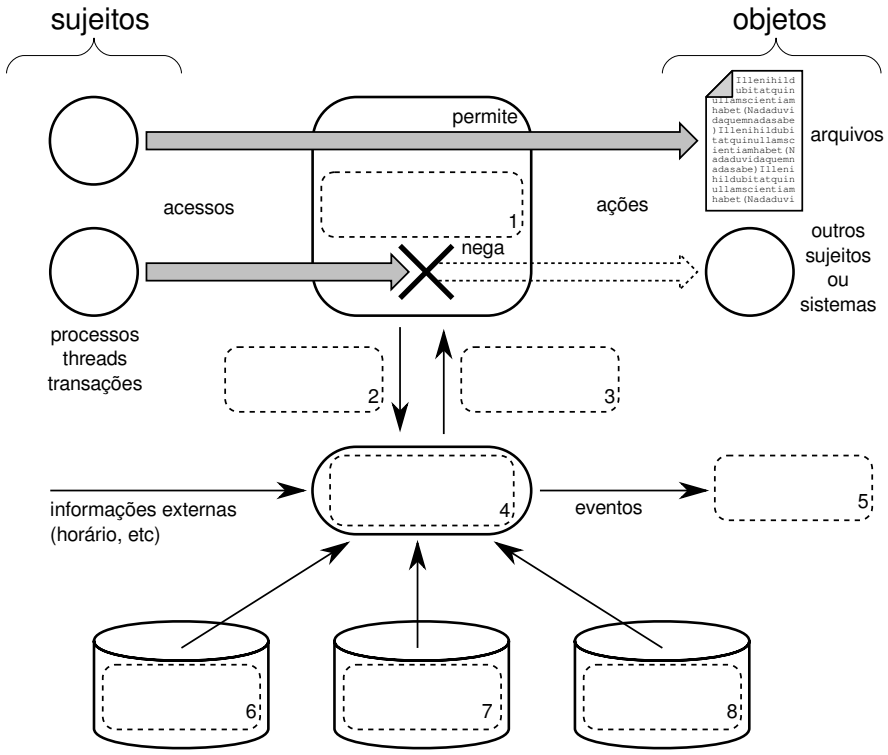
   (c) As "POSIX Capabilities" são uma implementação do mecanismo de *capabilities* para sistemas operacionais que seguem o padrão POSIX.

   (d) Alguns sistemas operacionais separam os usuários em *usuários normais* ou *administrativos*, atribuindo aos últimos permissões para efetuar tarefas administrativas, como instalar programas.

   (e) Alguns sistemas operacionais implementam processos *monitores* que recebem pedidos de ações administrativas vindos de processos com baixo privilégio, que são avaliados e possivelmente atendidos.

   (f) Os flags `setuid` e `setgid` do UNIX implementam um mecanismo de permissões temporárias.

2. O diagrama a seguir representa os principais componentes da infraestrutura de controle de acesso de um sistema operacional típico. Identifique e explique elementos representados pelas caixas tracejadas.



3. A listagem a seguir apresenta alguns programas executáveis e arquivos de dados em um mesmo diretório de um sistema UNIX, com suas respectivas permissões de acesso:

```
- rwx r-s ---  2 marge    family      indent
- rwx r-x --x  2 homer    family      more
- rws r-x --x  2 bart     men         nano
- rwx r-x ---  2 lisa     women       sha1sum

- rw- r-- ---  2 lisa     women       codigo.c
- rw- rw- ---  2 marge    family      dados.csv
- rw- r-- ---  2 bart     men         prova.pdf
- rw- rw- ---  2 homer    family      relatorio.txt
- rw- --- ---  2 bart     men         script.sh
```

Os programas executáveis precisam das seguintes permissões de acesso sobre os arquivos aos quais são aplicados para poder executar:

- `more`, `sha1sum`: leitura
- `nano`, `indent`: leitura e escrita

Considerando os grupos de usuários $men = \{bart, homer, moe\}$, $women = \{marge, lisa, maggie\}$ e $family = \{homer, marge, bart, lisa, maggie\}$, indique quais dos comandos a seguir serão permitidos e quais serão negados. O *prompt* indica qual usuário/grupo está executando o comando:

[ ] `lisa:women> nano codigo.c`

[ ] `lisa:women> more relatorio.txt`

```
[   ] bart:men> nano relatorio.txt
[   ] bart:men> sha1sum prova.pdf
[   ] marge:women> more relatorio.txt
[   ] marge:women> indent codigo.c
[   ] homer:men> sha1sum prova.pdf
[   ] homer:men> nano dados.csv
[   ] moe:men> sha1sum relatorio.txt
[   ] moe:men> nano script.sh
```

Referências

A. Bomberger, A. Frantz, W. Frantz, A. Hardy, N. Hardy, C. Landau, and J. Shapiro. The KeyKOS nanokernel architecture. In *USENIX Workshop on Micro-Kernels and Other Kernel Architectures*, pages 95–112, 1992.

D. Bovet and M. Cesati. *Understanding the Linux Kernel, 3rd edition*. O'Reilly Media, Inc, 2005.

K. Brown. *Programming Windows Security*. Addison-Wesley Professional, 2000.

B. Gallmeister. *POSIX.4: Programming for the Real World*. O'Reilly Media, Inc, 1994.

G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood. SeL4: Formal verification of an OS kernel. In *22nd ACM Symposium on Operating Systems Principles*, Big Sky, MT, USA, Oct 2009.

J. Liedtke. Toward real microkernels. *Communications of the ACM*, 39(9):70–77, 1996.

P. Loscocco and S. Smalley. Integrating flexible support for security policies into the Linux operating system. In *USENIX Annual Technical Conference*, pages 29–42, 2001.

Microsoft. *Security Enhancements in Windows Vista*. Microsoft Corporation, May 2007.

N. Provos, M. Friedl, and P. Honeyman. Preventing privilege escalation. In *12th USENIX Security Symposium*, 2003.

M. Russinovich, D. Solomon, and A. Ionescu. *Microsoft Windows Internals, Fifth Edition*. Microsoft Press, 2008.

J. Shapiro and N. Hardy. Eros: a principle-driven operating system from the ground up. *Software, IEEE*, 19(1):26–33, Jan/Feb 2002. ISSN 0740-7459. doi: 10.1109/52.976938.

R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau. The Flask security architecture: System support for diverse security policies. In *8th USENIX Security Symposium*, pages 123–139, 1999.

Sun Microsystems. *Trusted Solaris User's Guide*. Sun Microsystems, Inc, June 2000.

R. Watson. TrustedBSD: Adding trusted operating system features to FreeBSD. In *USENIX Technical Conference*, 2001.