ADMINISTRAÇÃO DE SISTEMAS
UNIX

(nota: este manual foi produzido no início dos anos 90 e poderá estar algo desactualizado)

Título: UNIX Administração
Autores: Fernando Fernández
Pedro Ferreira

3ª edição: Lisboa, Janeiro 1994

N O T A I N T R O D U T Ó R I A

Este manual (e o curso a que se refere) é dirigido às pessoas sobre quem caiu a responsabilidade de supervisionar uma instalação UNIX. Essas pessoas devem estar já familiarizadas com a utilização da máquina através da interacção directa com o Sistema Operativo via Shell. Devem ter conhecimentos razoáveis no campo da informática, e será melhor se tiverem bases de programação de computadores (embora isto não seja fundamental). A MoreData exige aos formandos que frequentam este curso conhecimentos equivalentes aos ministrados no curso introdução à informática, UNIX Operação e Edição de texto em UNIX. Opcionalmente, será útil conhecer os conceitos dados nos cursos UNIX - Programação em Shell e Introdução à lógica de programação.

Neste manual focam-se assuntos que interessam a estas pessoas: segurança, gestão de utilizadores, gestão de discos, controle de utilização, gestão de terminais e impressoras. Procura-se não só explicar a forma de gerir o sistema mas também o modo como este funciona, de maneira a que o futuro administrador possa compreender melhor os problemas que lhe surgirão no dia-a-dia. Toda a informação necessária à administração de um sistema UNIX está, como não podia deixar de ser, nos manuais de máquina. Mas nem sempre aí se encontra da forma mais explicita ou acessivel. Mais do que ensinar comandos, pretendemos ensinar métodos e estes não se encontram no manual. No entanto, uma das boas qualidades do bom administrador de sistemas é conhecer os manuais como a palma da sua mão. Os manuais são insubstituiveis. Quase todas as implementações do UNIX têm as suas particularidades (embora felizmente isto tenda a acabar com a imposição de standards) e estas só estão descritos nos manuais fornecidos com a máquina. O administrador de sistema não é um utilizador normal. Para além de conhecer profundamente a utilização do sistema ele vai ser responsável por manter a segurança deste, melhorar o seu funcionamento, restaurar ficheiros apagados por engano, apoiar os outros utilizadores, etc. O trabalho que o aguarda vai ser simultaneamente mais fácil e mais dificil do que seria de esperar. Mais fácil porque depois de compreender o que os manuais tentam dizer e o funcionamento do computador o trabalho torna-se extremamente simples. Mais dificil, porque nada nem ninguém pode preparar o administrador de sistema para as estranhas ocorrências da realidade.

Boa sorte.

Pedro Ferreira
Fernando Fernández

ÍNDICE

1. ELEMENTOS SOBRE SISTEMAS OPERATIVOS

1.1. ORIGENS

Num computador elementar, como o mais simples dos computadores pessoais, ou aqueles desenhados para controlar equipamento numa fábrica, existe sempre algum software, normalmente designado "monitor" (ou "programa monitor"). Consiste no código para realizar algumas rotinas, normalmente residente em ROM, de modo a que o utilizador não possa destrui-lo acidentalmente. As sub-rotinas que o compõe servirão talvez para enviar caracteres para um monitor video (CRT), ou ler informação de um dispositivo como um teclado. Estas sub-rotinas fazem com que o trabalho do programador seja muito mais simples e podem talvez ser melhorados pelo fabricante do hardware se este tiver evoluções do género de passar a poder ser ligado a outros periféricos, que necessitem de tratamento diferente.

Os programas para estes pequenos sistemas são geralmente escritos em assembler e as sub-rotinas do monitor chamadas através de instrucções do género "salta para a rotina no endereço tal". Todas as sub-rotinas têm de ter o seu endereço fixo e conhecido do programador.

O monitor, ou "programa supervisor" como também é chamado, oferece portanto rotinas de software que se encarregam dos detalhes do Input/Output com periféricos, alguma gestão interna, etc. Desta forma, embora o utilizador esteja a comunicar com a aplicação criada pelo programador através do terminal, esta comunicação pode muito bem estar a ser feita através das rotinas de I/O do monitor.

Todos os modernos sistemas operativos descendem dos humildes "supervisores" e "monitores" deste tipo. O monitor liberta o programador/utilizador da escrita de software para usar terminais e impressoras. Em vez disso, o programador faz uma ***chamada ao sistema*** pedindo para executar uma determinada tarefa de I/O e o monitor toma conta da tarefa daí em diante, realizando as operações no hardware que fazem com que o pedido seja satisfeito. É importante notar que embora existam duas áreas diferentes de software em memória (o programa aplicação e o monitor) 0 CPU só pode executar uma de cada vez.

1.2. SISTEMAS MONO UTILIZADOR

O desenvolvimento dos monitores levou aos sistemas operativos dos micro-computadores que incluem sistemas de ficheiros em disco. Os exemplos mais famosos são o CP/M e o MS-DOS. O software inclui agora chamadas ao sistema para acesso a ficheiros no sistema de ficheiros (para leitura de um ficheiro por nome, por exemplo), de tal forma que os programas só têm que chamar o sistema para realizar qualquer operação. Enquanto que no caso do monitor em ROM o endereço de cada rotina tinha de ser conhecido, neste caso duas diferenças devem ser realçadas. Primeiramente: o programador/utilizador não precisa de saber os endereços de todas as rotinas dos serviços do sistema. Cada um dos serviços é numerado e portanto o utilizador apenas "pede" o serviço pelo número. Para uma leitura (read) poderá pedir o "o serviço 26" ou algo assim. Em segundo lugar: qualquer nova implementação do sistema, em que as rotinas associadas ao serviço mudem de endereço, não implica uma mudança no programa/aplicação, já que as chamadas ao sistema deixaram de ser feitas por endereço.

Como além disto, o sistema operativo reside em disco (sendo carregado para memória na fase de arranque da máquina) a substituição de uma versão por outra mais actual é muito mais económica e simples do que quando este residia em ROM.

O código residente do sistema operativo - ***Kernel***, na nomenclatura UNIX - é colocado na memória nos endereços mais baixos. O programa aplicação, que este seja um editor, um jogo ou uma folha de cálculo, utiliza o que resta da memória. O hardware simula o início da memória para o programa aplicação, de modo que este não tem que ser alterado para poder ser executado a partir de outro endereço.

De uma forma geral, os CPU's actuais contêm 2 registos especiais nos quais se colocam os endereços inferior e superior de um programa aplicação. Chamaremos a estes registos BASE e TOPO (embora sejam apenas nomes genéricos, que diferem de um fabricante para outro).

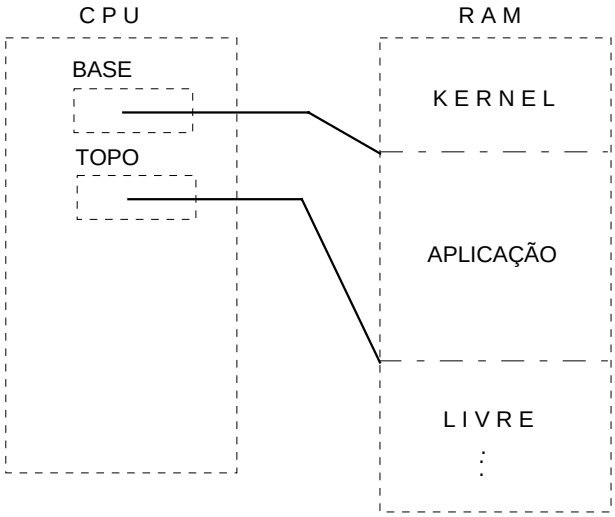


Figura 49. Registos BASE e TOPO

Se um programa é carregado para o endereço X e necessita de K bytes de memória para correr, então no registo BASE é colocado o endereço X e no registo TOPO é colocado o endereço X+K. O hardware está muitas vezes preparado para impedir que o programa aplicação não aceda a endereços de memória fora dos limites definidos por estes dois registos, cabendo ao sistema operativo decidir ou não ligar este dispositivo.

Os acessos à memória pelo programa aplicação são feitos do seguinte modo: sempre que este pretende aceder ao endereço Y, o CPU acede ao endereço Y+BASE. Se o programa aplicação pretende um serviço do sistema operativo, este funcionamento é desligado. Convém realçar o facto de este pormenor ser implementado por hardware. este hardware opera em dois modos: o modo executivo (ou "kernel mode" em unixês), em que os registos BASE e TOPO não têm qualquer efeito, e todos os endereços são acessiveis, e o modo utilizador ("user mode"), em que o endereço contido no registo BASE é somado a todos os endereços referenciados pelo programa e este não pode aceder a endereços fora do intervalo BASE/TOPO.

Esta filosofia e este hardware são a base do conceito de ***Gestão de Memória*** dos sistema operativos actuais, permitindo até criar sistemas de multi-processamento (com mais do que um programa aplicação em memória simultaneamente) dos quais falaremos mais adiante.

Este sistema operativo simples que acabámos de descrever, confia no facto de existir apenas mais um programa/aplicação na máquina para além dele mesmo. Para além disto este processo está localizado num endereço fixo,

conhecido do sistema. O processador vai estar activo sobre o programa aplicação excepto quando:

a) quando foi feita uma chamada ao sistema
b) quando o programa aplicação termina

Quando o programa aplicação termina, o controlo da máquina é devolvido ao código do sistema responsável pelo diálogo com o utilizador, e que oferece uma quantidade de comandos que permitem requerer tarefas tais como apagar ficheiros do disco, modificar-lhes os nomes, ou carregar um programa novo. Uma vez carregado outro programa da disquete para a memória, não são necessários mais acessos a esta, já que o código de execução das chamadas ao sistema reside em memória desde que a máquina arrancou. Isto só é válido, claro, se o próprio programa não faz acesso a disquete.

O kernel liberta o utilizador dos detalhes dificultosos de carregar um programa a partir de um disco[1] (disco pode significar também disquete) ou das actividades realizadas na máquina quando se lê ficheiros, mas existem situações em que a sua actuação não é perfeita. Acontece, por exemplo, que um programa que só necessita de interacção no seu início bloqueia o terminal até ao fim da sua execução, quando a vontade do utilizador seria continuar a trabalhar noutra coisa qualquer. Parece bastante razoável pedir que esta segunda fase da tarefa seja realizada em background. Nestes pequenos sistemas, isto só é possivel para certas tarefas especifícas como o envio de ficheiros para a impressora, através de alguns truques que diferem ainda bastante de uma verdadeira capacidade de multi-tarefa.

1.3. SISTEMAS MULTI-TAREFA E MULTI-UTILIZADOR

Convém distinguir aqui os conceitos de multi-tarefa e multi-processamento. Embora a nossa definição seja discutivel, já que a confusão entre os diversos autores é grande, consideramos um sistema multi-tarefa (multi-tasking) aquele que simula a execução simultânea de diversos programas através da distribuição do tempo do CPU por cada um. Chamamos sistema multi-processamento aquele que ***contém no seu hardware mais do que um CPU***[1]. Convém também notar que ao longo de todo o texto temos vido a assumir que a máquina em que o UNIX corre não tem mais do que um processador, o que é verdade para a maioria dos casos.

[1] *Na maior parte dos casos, os sistemas de multi-processamento são também multi-tarefa.*

Vamos pôr desde já o UNIX no seu contexto correcto. O kernel do UNIX é contruido a volta da ideia suportar um ambiente multi-tarefa em que a memória, o processador e os periféricos sejam partilhados por um conjunto de processos que competem entre si.
O kernel do UNIX realiza duas tarefas básicas. Uma é aceder aos desejos dos diversos processos activos, que querem aceder ao disco, imprimir dados, ler teclados, etc. Para realizar estas tarefas contém diversas rotinas, chamadas "device drivers", que sabem exactamente as operações a realizar no hardware para que a tarefa seja concluida. Todo o hardware contido num sistema tem que ter o seu device driver associado no kernel. Os periféricos estão assinalados no sistema de ficheiros do UNIX como ***ficheiros especiais*** e estes têm sempre um major number e minor number, que são a forma de o kernel saber qual o device driver que deve chamar, no caso de alguém pretender aceder a esses periféricos. A outra tarefa básica do kernel do UNIX é organizar a segurança (garantindo que determinado utilizador não tem acesso aos ficheiros ou a memória associada a um processo de outro utilizador sem permissões para tal) e a partilha do processador entre os diversos processos activos (garantindo que um utilizador nunca possa monopolizar o processador). Para tal, o sistema mantém uma tabela com informação sobre todos os processos activos, com a sua localização na memória, o seu tamanho, o seu dono, o seu estado, etc. Cada processo tem um número que o identifica, designado por PID (process identification).

A partir do momento em que os mecanismos para a multi-programação existem, o ambiente multi-utilizador segue-se como uma extensão natural. A capacidade principal a adicionar a um sistema multi-utilizador é a segurança, garantindo privacidade aos diversos utilizadores.

1.4. ASPECTOS DOS SISTEMAS MULTI-UTILIZADOR

Vimos até agora uma imagem bastante estática de como o UNIX poderia correr mais do que um processo. Garantindo que os processos activos ficam sempre nos mesmos endereços de memória, então parecerá bastante simples fazer com que o kernel divida o CPU durante uns milisegundos de cada vez, entre os diversos processos. Eventualmente, um dos processos há-de terminar, e a sua memória pode ser dada a outro processo que precise de correr. Seria bom que as coisas fossem assim tão simples, e nas próximas secções vamos examinar algumas das coisas que podem causar problemas.

1.4.1. PARTILHA DE PROCESSADOR

A partir do momento em que o UNIX passou o controlo do processador, quando é que este controlo volta para o kernel? Tal como muitos dos sistema operativos multi-programação, o UNIX assume que a maior parte dos processos tem que realizar frequentemente chamadas ao sistema, normalmente para operações de I/O. Estas chamadas ao sistema são feitas não através do modo normal de chamadas de rotinas ou sub-rotinas, mas através de um dispositivo misto hardware/software normalmente designado por ***software interrupt*** ou TRAP. O TRAP vai não só passar o controlo do processador para o kernel, mas também desinibir o dispositivo de controlo de acesso a memória que utiliza os registos BASE e TOPO, referidos anteriormente. O processador deixa de correr no modo utilizador e passa a correr no modo executivo, que em UNIX de designa modo kernel (kernel mode).
O kernel do UNIX, acordado pelo TRAP, descobre que foi chamado por um programa/aplicação e vai dar inicio àtarefa correspondente, se necessário chamando o respectivo device driver. Se o serviço requerido fôr demorado (uma leitura de disco, por exemplo), o kernel, muito provavelmente, irá estudar a hipótese de pôr outro processo a correr, aproveitando assim a oportunidade para não deixar parar o processador. Consultando a tabela de processos, vai pesar as prioridades de cada um dos processos que está a espera de correr, dando maior importância aos que estão parados há mais tempo. O kernel sabe onde reside o código do processo a lançar (faz parte da tabela) e após colocar cuidadosamente os novos valores nos registo BASE e TOPO passa o controlo do processador para o processo passando a modo utilizador. Quando este necessita de um serviço do sistema, faz um TRAP, e o ciclo recomeça. Cada chamada ao sistema fornece ao kernel uma possibilidade de decidir qual o processo que deve correr.

Este esquema, tal como o descrevemos, não poderá controlar processos com utilização intensiva do CPU, isto é, programas que não necessitem de serviços do sistema (tais como programas de cálculo intensivo). Estes programas fecham-se sobre si mesmos durante períodos longos, e não dão oportunidade ao kernel de lançar outros processos. Para estes casos, o próprio hardware tem que interferir, interrompendo o CPU através de um hardware interrupt, passando automáticamente o controlo do CPU para o kernel.

1.4.2. GESTÃO DE MEMÓRIA

1.4.2.1. Memória virtual por SWAPPING

Suponhamos agora que a memória está cheia de processos e que apesar disto, um utilizador requer a criação de mais um processo. A solução para estes casos é uma solução de compromisso entre velocidade e quantidade de processos. O kernel tem reservada para seu uso uma área em disco chamada a ***swap area*** em que guarda temporariamente processos que não estejam activos e não tenham grande probabilidade de ser activados nos tempos mais próximos (lembrem-se de queestamos a falar de milisegundos) podendo assim libertar a memória associada a um processo ainda activo.

Ao colocar em disco o processo, o UNIX perde bastante tempo, já que para reactivar o processo será necessário lê-lo do disco para a memória, mas isto evita a necessidade de limitar o número de processos a quantidade de memória. Assim, teóricamente, a quantidade de memória apenas limita o tamanho máximo dos processos a correr, embora na pratica demasiada quantidade de swapping seja insuportável.

Num sistema muito sobracarregado, em que o tamanho e número dos processos activos ultrapassa de longe a quantidade de memória presente no hardware, a quantidade de swapping e a sua administração degradarão de tal forma a velocidade do sistema que o computador passará mais tempo a fazer swapping do que a correr os processos. Com a adição de memória, existirá menos swapping e o sistema parecerá muito mais potente. Casos pode haver em que a velocidade de resposta normal ao utilizador pode ser multiplicada por um factor de 2 ou 3.

A politica de escolha de processos a retirar temporariamente da memória varia com as implementações do UNIX, mas os processos que estão a aguardar input de um terminal são as escolhas mais óbvias, já que a resposta de um ser humano pode demorar vários segundos, ou mesmo minutos.

1.4.2.2. Fragmentação

Quando um programa termina, a sua memória fica disponivel para que outro processo (novo ou swapped) aí seja colocado. Se o novo processo fôr maior do que o que terminou não poderá ser colocado no seu lugar, já que não cabe. Se o novo processo é mais pequeno, uma pequena área de memória ficará por utilizar, e gradualmente, a memória ficará com muitos pequenos "buracos" não utilizados, cuja área total é grande, embora nenhum deles seja suficientemente grande para conter um novo processo. Este problema designa-se por ***fragmentação*** e para ser evitado, o sistema tem que ter a capacidade de deslocar os processos de modo a tapar o buracos que se vão criando com o tempo, deixando a memória livre agrupada numa única área agora utilizável. O sistema tem que controlar cuidadosamente as deslocações feitas de forma a poder mais tarde lançá-los correctamente.

1.4.2.3. Memória virtual por PAGING

Nos sistemas de memória integralmente virtual qualquer programa deve poder ser executado, mesmo que não caiba inteiramente na memória. O conceito de swapping alarga-se para bocados de processos. Estes bocados, designados páginas, podem ser tratados independentemente pelo kernel, que move paro o disco as páginas com menor probabilidade de serem necessárias, afim de obter memória. Desde que a página activa de um processo esteja em memória, este processo pode continuar. Isto reduz as actividades de swapping consideravelmente e reduz também os problemas resultantes da fragmentação. As páginas de um processo nem sequer têm que estar contiguas na memória, de tal forma que podem ser usados pequenos buracos dispersos pela memória para correr um programa. O programa necessita a partir de agora mais registos BASE e TOPO, dois para cada página, de modo que isto também implica mais hardware especial.

1.4.2.4. Código partilhado

Em qualquer sistema, duas activações simultâneas do mesmo programa (dois processos com o mesmo programa mas não os mesmos dados), como por exemplo dois editores ou dois compiladores, conterão uma enorme quantidade de informação duplicada. O seu código será o mesmo, mas os seus dados (que incluem os valores das variáveis) vão ser diferentes. Os sistemas mais sofisticados permitem que o código seja guardado num único local da memória, e ambos os processos tenham autorização de acesso a esta memória, tendo os seus segmentos de dados separados. Isto é realizado permitindo que os dois processos tenham alguns dos seus registos BASE (aqueles que se referem ao código) com valores idênticos, apontando para a mesma área de memória.

1.4.3. GESTÃO DE I/O

1.4.3.1. Organização de pedidos de acesso ao disco

Uns quantos processos podem estar suspensos à espera de receber dados do disco. O sistema tem que manter uma fila de espera para acesso a determinado disco, e quando uma transferência termina o kernel pode tentar satisfazer qualquer um dos outros pedidos. Estes podem ser ordenados pela ordem em que foram feitos ou pela ordem que melhor velocidade de acesso permita em média. Se esta última politica fôr escolhida um processo pode sofrer com isso, vendo outro passar-lhe a frente, mas a média de tempo de acesso melhorará certamente. Mais uma vez, estas particularidades de funcionamento dependem da implementação do UNIX com que se está a trabalhar.

1.4.3.2. BUFFER CACHE

.Não se deve confundir a buffer cache, que é um dispositivo de software que existe em ***todas*** as máquinas UNIX com as memórias cache, dispositivos de hardware constituidos por memórias extremamente rápidas, que se colocam entre o CPU e a memória principal, de forma a acelerar os acesso à memória. O kernel usa a buffer cache sempre que um programa acede a um dispositivo (como discos ou tapes) que trabalha com transferência de informação por blocos. Se um programa pretende ler 16 bytes de um ficheiro, o kernel lê mais (512, por exemplo), e se o programa quiser mais tarde ler mais 16 bytes o kernel não tem que ir novamente ao disco, bastando consultar o buffer que tinha guardado. O UNIX mantém uma certa quantidade de buffers deste género (a qual chama, como já viram, buffer cache), que não só serve para acelerar as leituras mas também as escritas, já que nem sempre o disco é actualizado com a informação que os programas põem nos ficheiros. Na maior parte do tempo, os ficheiros em disco estão desactualizados, já que as modificações estão a ser feitas em memória. Se mais de um processo acede a um ficheiro, estes acessos são todos concentrados sobre o mesmo buffer, e o disco é actualizado a intervalos de tempo fixos através da operação ***.sync***. Esta operação é muito importante, já que como se viu, nem sempre os ficheiros estão actualizados com o que está na buffer cache, e embora o kernel se encarregue de a realizar de tempo a tempo o super-user pode requerê-la através do comando com o mesmo nome - sync.

A utilização da buffer cache pelo UNIX é também a razão pela qual não se podem desligar as máquinas UNIX sem fazer shutdown, já que uma das tarefas deste comando é realizar o sync.

Na figura que se segue encontra-se resumido o funcionamento do kernel do UNIX através do seu esquema de blocos.

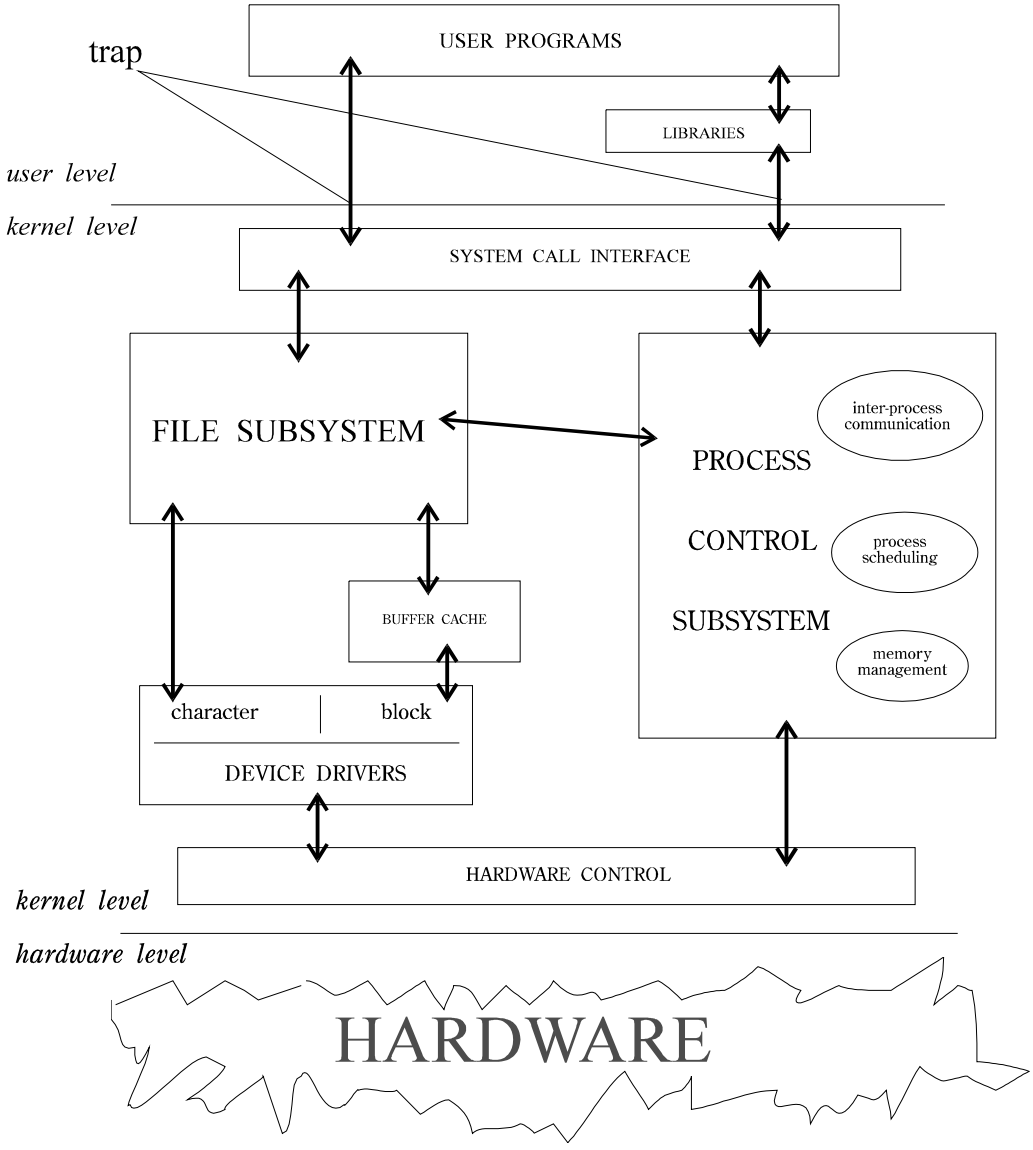


Figura 50. Esquema de blocos do Kernel do UNIX

2. SEGURANÇA DE DADOS E GESTÃO DE UTILIZADORES

2.1. INDIVÍDUOS E GRUPOS

Um sistema UNIX, sendo multiposto pode ser utilizado por vários individuos mantendo cada um deles a segurança da sua informação. Para tal torna-se necessário dotar cada potencial utilizador de uma identificação chamada ***logname*** ou nome de login. Os logname são palavras com um número variável de caracteres,em que geralmente se usam apenas letras minusculas. As limitações aos logname,quer no número de caracteres, quer nos caracteres que se podem usar varia com a implementação do UNIX, no entanto é aconselhável não usar metacaracteres do shell nos logname devido às dificuldades de manipulação que isso pode trazer.

A cada logname o sistema associa um número que é utilizado em todas as representações internas, tal número chama-se ***uid*** ou ***user-id***.

Os utilizadores com user-id 0 recebem um tratamento especial, tendo a possibilidadede superar todas as seguranças do sistema, por exemplo: podem escrever em ficheiros para os quais não têm permissões de escrita. Por isto são conhecidos como super-utilizadores (ou super-user's). O número de super-utilizadores num sistema UNIX deve ser limitado ao máximo, de preferência devem existir apenas o utilizador root e um super-utilizador com acesso a menus de administração de sistema (ex: sa ou sysadm). Não se deve remover do sistema o utilizador root pois alguns comandos que assumem a sua existência podem deixar de funcionar.

Os utilizadores de uma máquina UNIX são divididos em ***grupos*** podendo cada utilizador fazer parte de um ou mais grupos. Tal como os utilizadores os grupos são identificados por um nome conhecido por ***group-name***. A cada group-name o sistema associa também um número que utiliza em todas as representações internas de grupos, tal número chama-se ***gid*** ou ***group-id***.

Para garantir a segurança da informação dos utilizadores torna-se necessário impedir que seja possível alguém identificar-se perante o sistema operativo como outro utilizador. Para tal cada utilizador pode criar uma palavra chave que fica associada ao seu uid e que o sistema armazena numa forma codificada sendo impossivel conhecê-la recorrendo ao computador. As palavras chave são habitualmente conhecidas por ***password***.

A password de um utilizador é alterada através do comando ***passwd***, que tem o seguinte formato:

passwd [uname]

o argumento opcional (uname) representa um user-name válido do sistema e só um super-utilizador o pode usar para mudar a password doutro utilizador, se se omitir o argumento, o comando altera a passwd do utilizador que invocou o comando.

O administrador de sistema tem a possibilidade de escolher para cada utilizador o directório em que fica posicionado quando inicia a sessão de trabalho, e o programa que este executa, podendo existir utilizadores que apenas interagem com o computador através de programas aplicacionais sem acesso a toda a potencia do Sistema Operativo.

Cada processo tem associado uma uid e uma gid que são inicializadas sempre que se inicia uma sessão de trabalho. Alem do uid cada utilizador tem também associado um uid efectivo que em situação normal é igual ao uid mas que pode ser mudado de forma a assumir provisóriamente a identificação de outro utilizador. O comando para substituir a identificação de um utilizador de shell ou c-shell é o ***su*** (substitute user-id) que tem o seguinte formato:

su [-] [user-name [arg ...]]

onde o argumento opcional user-name é o nome do utilizador de que se quer assumir a identificação. Quando se corre este comando é executado o programa de login associado ao utilizador user-name ao qual são passados os argumentos arg. O comando su valida a possibilidade de um utilizador mudar a sua identificação de login pedindo a palavra chave (password) associada ao novo user-name. Se se usar a opção - o ambiente (directório, variáveis, etc.) é inicializado como se se tratasse de uma sessão de trabalho normal, caso contrário é usado o ambiente do programa chamador. Quando se omite o user-name é usado o nome root (ou seja o super-user). Sempre que este comando é executado o sistema cria um registo no ficheiro ***/usr/adm/sulog***. Note-se que este comando não inicia uma nova sessão de trabalho mas executa um programa que quando termina deixa o sistema na sessão de trabalho que executou o comando su.

Em qualquer momento um utilizador pode pedir ao sistema que o inclua noutro grupo do qual faça parte; o sistema verifica se o utilizador faz parte desse grupo e inicia uma nova sessão de trabalho com a nova identificação de

grupo. O comando que realiza essa função chama-se ***newgrp*** e tem o seguinte formato:

newgrp group-name

onde group-name é o nome do grupo de que se deseja ficar a fazer parte.

2.2. A GESTÃO DOS UTILIZADORES E FICHEIROS ASSOCIADOS

Os dados necessários para gerir os utilizadores de um sistema UNIX encontram-se sempre num ficheiro de texto chamado ***/etc/passwd***, tal ficheiro pode ser lido por qualquer utilizador, mas só pode ser escrito por um super-utilizador. O ficheiro /etc/passwd é constituido por 1 linha para cada utilizador contendo cada linha 7 campos separados por ":".

Os campos do ficheiro /etc/passwd contêm a seguinte informação:

logname	o nome com o qual o utilizador se identifica perante o sistema.
password	a palavra chave do utilizador, codificada.
uid	o número de idenficação usado internamente pelo sistema.
gid	o número do grupo a que o utilizador pertence por defeito.
comentário	um campo cuja utilização depende da implementação do UNIX.
directório de login	o nome do diretório que se torna corrente logo após o inicio de uma sessão de trabalho.
programa a executar	o nome do programa que se executa sempre que se inicia uma sessão de trabalho.

Um exemplo de um ficheiro /etc/passwd é o seguinte:

```
root:pfit.ZqZNsAfA:0:0:Super user:/:/bin/sh
rtcsh:pfit.ZqZNsAfA:0:0:Super user:/:/bin/csh
shutdown::0:0:Shutdown script:/:/usr/desp/bin/scripts/shutdown
cron:NOLOGIN:1:1:Cron daemon for periodic tasks:/:
bin:NOLOGIN:3:3:System file administration:/:
uucp::4:4:Uucp
administration:/usr/spool/uucppublic:/usr/lib/uucp/uucico
asg:NOLOGIN:6:6:Assignable device administration:/:
sysinfo:NOLOGIN:10:10:Access to system information:/:
network:NOLOGIN:12:12:Mail and Network
administration:/usr/spool/micnet:
lp:NOLOGIN:14:3:Print spooler administration:/usr/spool/lp:
dos:NOLOGIN:16:10:Access to Dos devices:/:
informix::100:52::/usr/informix:/bin/sh
Pedro:NZAv3HwDIqgkg:202:53:22:/usr/pedro:/bin/csh
Sergio::208:53:0:/usr/sergio:/bin/csh
Xandra:Pe3sGbJzAz0js:209:53:0:/usr/xandra:/bin/csh
Miguel:wolP29VXN1Y8M:211:53::/u/miguel:/bin/csh
dsp::212:50::/usr/dsp:/bin/csh
Fernando::213:53:0:/u/Fernando:/bin/csh
Ana::213:53:0:/u/Ana:/bin/csh
```

Figura 1. Exemplo do ficheiro /etc/passwd

note-se a existência de vários utilizadores com uid 0 no incio do ficheiro, tais utilizadores são super-utilizadores e conseguem tornear todas as seguranças do sistema operativo.

Sendo o ficheiro /etc/passwd um ficheiro de texto, o administrador de sistema pode fazer a sua manutenção usando qualquer editor de texto, no entanto a maioria dos fabricantes do sistema UNIX fornecem programas que fazem essa manutenção, através de um programa interactivo ou através de menus. A utilização do editor de texto para manipular o ficheiro /etc/passwd não garante a validação dos dados deste ficheiro. Um administrador
de sistema deve lembrar-se sempre que este ficheiro é uma das peças mais sensíveis do sistema operativo UNIX, já que controla todo o processo de login, a sua destruição impede que qualquer utilizador trabalhe com o sistema obrigando a procedimentos de socorro complexos ou, em casos extremos, á reinstalação de todo o sistema operativo com a consequente perda de informação.

Existe um comando que permite aos administradores de sistema validar a carência do ficheiro /etc/passwd em qualquer altura, tal comando chama-se *pwck* e tem o seguinte formato:

pwck [ficheiro]

Este comando valida a coerência da informação contida no ficheiro de passwords cujo nome lhe é passado como argumento (se o argumento for

omitido será o ficheiro /etc/passwd). A validação processa-se sobre: Número de campos por registo, logname, uid, gid e a existência do programa a executar. Se existirem incoerências serão escritas mensagens na saída de dados standard, caso contrário o comando será silencioso. É de toda a conveniência a utilização deste comando logo após qualquer alteração ao ficheiro /etc/passwd.

Algumas implementações de UNIX utilizam o programa cron para periodicamente copiar o ficheiro /etc/passwd de forma a possibilitar a recuperação de eventuais destruições acidentais do mesmo. É de toda a conveniência que o administrador de sistema faça também a sua cópia de segurança do ficheiro sempre que tenha que o manipular.
Outro aspecto importante da manipulação do ficheiro /etc/passwd é o controlo das permissões do directório /etc, já que se se der a possibilidade a qualquer utilizador de criar ficheiros nesse directório ele poderá sempre criar um ficheiro /etc/passwd com permissões de escrita para si, subvertendo assim a segurança do sistema operativo.

Os grupos de utilizadores num sistema UNIX são controlados através do ficheiro */etc/group* que é um ficheiro de texto com uma linha por cada grupo. Cada linha é constituida por quatro campos separados por ":" com o seguinte conteúdo:

group-name	o nome do grupo a que a linha diz respeito.
password	uma palavra chave codificada associada ao grupo. Note-seque não existe no standard UNIX nenhum comando para actualizar este campo (do tipo do comando passwd), o que faz com que a manutenção de password para grupos seja um trabalho fastidioso com o editor de texto, o comando passwd e o ficheiro /etc/passwd. Quando este campo está preenchido o comando newgrp valida a password de grupo.
gid	contém a identificação numérica do grupo que é usada em todas as representações internas de grupos.
utilizadores	contém a lista, separada por virgulas, de todos os utilizadores que fazem parte de um grupo. O comando newgrp verifica se um determinado utilizador pode mudar o seu grupo através deste campo.

Um exemplo do ficheiro /etc/group é o seguinte:

```
rootgrp::1:root,va
sys::2:root,bin,sys,adm
bin::3:bin,Pedro
adm::4:root,adm,daemon
ncrm::5:ncrm
uucp::6:uucp,nuucp
mail::7:root
daemon::9:root,daemon
lp::71:lp
usr::100:Pedro
informix::101:informix
```

Figura 2. Exemplo do ficheiro /etc/group

Tal como o ficheiro /etc/passwd também o ficheiro /etc/group é uma parte bastante sensível de um sistema Unix, por isso aconselha-se os utilizadores de sistema a usar as mesmas precauções com o ficheiro /etc/group que com o ficheiro /etc/passwd. Existe também um comando que permite validar a coerência da informação contida no ficheiro /etc/group ou noutro ficheiro com a mesma estrutura. Tal comando chama-se *grpck* e tem o seguinte formato:

grpck

O campo valida o número de campos, o nome dos grupos,os gid e a existência dos logname no ficheiro /etc/passwd. As mensagens de erro são escritas saída de erros standard e caso não sejam detectadas incoerências a saída de erros standard será vazia.

2.3. UTILIZADORES DE shell

No ponto anterior foi visto que um utilizador, ao iniciar uma sessão de trabalho, pode executar qualquer programa. Vamos debruçar-nos sobre os utilizadores da linguagem de controlo de sistema do UNIX.

2.3.1. FICHEIROS DE EXECUÇÃO AUTOMÁTICA

Os utilizadores da linguagem shell, ou seja aqueles cujo campo 7 do ficheiro /etc/passwd contém "/bin/sh", trabalham com o interpretador de comandos usualmente conhecido como **Bourne-shell**. O Bourne-shell permite a execução de tarefas de inicialização automáticamente após o login, para tal existem dois ficheiros nos quais se colocam comandos que são sempre executados antes de qualquer sessão de trabalho:

/etc/profile	contém instruções que são executadas por todos os utilizadores de shell no inicio de uma sessão de trabalho, usa-se habitualmente para tarefas de inicialização comuns a todos os utilizadores (ex: verificação do correio, inicialização da variável PATH, comando umask).
$HOME/.profile	contém instruções a ser executadas no inicio de cada sessão de trabalho, especificas de cada utilizador (ex: alteração da prompt, inicialização da variável EXINIT, etc.).

Quaisquer instruções contidas nestes ficheiros são executadas dentro do processo de shell que se está a iniciar, ou seja, o resultado não é o mesmo que executar os comandos /etc/profile ou $HOME/.profile no inicio da sessão de trabalho, pois se assim fosse seria impossivel alterar o ambiente da sessão de shell dessa forma.

A figura seguinte contém um exemplo de um ficheiro "/etc/profile":

```
if [ -f /etc/motd ]
then
      cat /etc/motd
else
      > /etc/motd
fi

# check mail
if [ -f /usr/bin/mailx ]
then
      if /usr/bin/mailx -e
      then
          echo You have mail.
      fi
fi

# check news
if [ -r /usr/bin/news ]
then
     news -n
fi

# initialize informix variables
INFORMIXDIR=/usr/informix
PATH=$INFORMIXDIR/bin$PATH
export INFORMIXDIR PATH
```

Figura 3. Exemplo de um ficheiro /etc/profile

Neste exemplo testa-se a existência de um ficheiro /etc/motd cujo conteúdo será mostrado no écran, verifica-se a existência de correio e de noticias, e inicializam-se as variáveis próprias da base de dados INFORMIX.

De seguida pode ver-se um exemplo de um ficheiro "**.profile**":

```
# Variáveis
PATH=.:$HOME/bin:$PATH
export PATH
clear
echo "login em \c" >> .logtime date >> .logtime
```

Figura 4. Exemplo de um ficheiro .profile

Pode ver-se neste exemplo a inicialização da variável PATH para possibilitar a execução de comandos dos directórios "." e "$HOME/bin", a limpeza do ecran e o registo da hora de login num ficheiro .logtime.

2.3.2. O SHELL RESTRITO

O sistema UNIX contém um programa conhecido por ***shell restrito*** (restricted shell), contido no ficheiro /bin/rsh e que corresponde a executar o programa "/bin/sh -r". Este interpretador de comandos tem exactamente a mesma sintaxe do Bourne-shell com limitações que visam impedir os utilizadores de destruir informação. Os utilizadores do shell restrito não podem:

- Mudar de directório.
- Alterar as variáveis PATH e SHELL.
- Executar comandos que contenham / no nome, ou seja executar comandos que se encontrem fora dos directórios da variável PATH.
- Redireccionar o output de comandos ou seja usar os caracteres > e >>.
- Usar o comando exec.

O administrador de sistema deve dar apenas shell restrito aos utilizadores pouco experientes ou dignos de pouca confiança pois dessa forma pode poupar muitos aborrecimentos com acidentes ou boicotes. Quando se configuram utilizadores com shell restrito é conveniente não incluir na sua variável PATH os directórios de sistema habituais, mas criar um directório no qual se colocam comandos só para os utilizadores de shell restrito.

2.4. UTILIZADORES DE c-shell

Outra linguagem de controlo de sistema habitualmente usada é o C-shell, um interpretador de comandos com uma sintaxe próxima da linguagem C. Os utilizadores de C-shell terão no campo 7 do ficheiro /etc/passwd um programa com o nome csh, que se pode encontrar em diferentes directórios (ex: /usr/bin/csh, /bin/csh, /usr/ucb/csh). Tal como no shell é possivel executar comandos automáticamente no inicio das sessões de c-shell. Podem também executar-se programas automáticamente sempre que se lança um programa do interpretador C-shell, contrariamente ao Bourne-shell onde os ficheiros .profile e profile só são executados no inicio da sessão de trabalho (login), e pode ainda executar-se comandos no fim das sessões de tradalho.
Existem três ficheiros de execução automática em C-shell sendo todos especificos de cada utilizador, não existe nenhum ficheiro com instruções de execução comum a todos os utilizadores.

$HOME/.login	ficheiro que contém instruções, especificas de um utilizador, a executar na inicio de uma sessão de trabalho (login). Serve o mesmo propósito que o ficheiro $HOME/.profile no Bourne-shell.
$HOME/.cshrc	ficheiro que contém instruções a serem executadas no inicio de cada execução do C-shell, independentemente de se tratar ou não do inicio da sessão de trabalho (login).
$HOME/.logout	ficheiro que contém instruções a serem executadas no fim de cada sessão de trabalho.

Tal como no Bourne-shell as instruções contidas nestes ficheiros são executadas sem criar um novo processo, o que permite utilizá-los para inicializar variáveis.

Segue-se um exemplo de um ficheiro "$HOME/.login":

```
setenv EXINIT "set redraw ai showmatch ts=4 sw=4"
setenv INFORMIXDIR "/usr/informix"
setenv DBDATE "Y2MD/"
setenv TERMCAP "$INFORMIXDIR/etc/termcap"
set ignoreeof
set path = (/usr/ucb /bin /usr/bin  . $INFORMIXDIR/bin)
tput clear
```

Figura 5. Exemplo do ficheiro .login

No exemplo pode ver-se a inicialização de diversas variáveis e a limpeza do terminal. As variáveis a inicializar são:

EXINIT	A variável de inicialização dos editores vi e ex.
INFORMIXDIR	A variável de inicialização da base de dados INFORMIX.
DBDATE	A variável da base de dados INFORMIX que especifica o formato das datas.
TERMCAP	A variável com o nome do ficheiro onde se encontra a base de dados de terminais termcap.
ignoreeof	A variável c-shell que impede o caracter ^D (EOF) de terminar uma sessão de trabalho.

path	A variável c-shell com a lista de directórios que contêm os programas.

Na figura seguinte pode ver-se um exemplo de um ficheiro .cshrc:

```
set history=20
set prompt="\! (`pwd`)  % "
alias h          history
alias m          "more -d"
alias adeus      exit
alias rh         rehash
alias ps         "ps -f"
alias RM   /bin/rm
alias cd         'cd \!* ; set prompt="\! (`pwd`)  % "'
alias md   mkdir
```

Figura 6. Exemplo de um ficheiro .cshrc

Pode ver-se a inicialização de duas variáveis de c-shell: history que especifica o número de comandos memorizados pelo c-shell, e prompt que especifica a prompt primária do c-shell. As últimas linhas contêm vários exemplos da utilização do comando alias do c-shell.

Na figura seguinte pode ver-se um exemplo de um ficheiro .logout:

```
clear
echo " Removendo temporários"
rm -r  ~/tmp
mkdir  ~/tmp
chmod 777  ~/tmp
date
echo "ADEUS"
```

Figura 7. Exemplo de um ficheiro .logout

Este ficheiro apaga o directório tmp abaixo do directório de login, volta a cria-lo, muda-lhe a permissão e escreve na saída de dados standard a data e hora e a palavra ADEUS. Os administradores de sistema devem sempre prestar a maior atenção ao conteúdo dos ficheiros de execução automática pois alguns dos problemas mais vulgares em sistema Unix relaciona-se com estes ficheiros; por exemplo, o comando exit num ficheiro .profile ou .login impede um utilizador de aceder ao sistema.

2.5. PERMISSÕES

As permissões são um dos mais importantes atributos dos ficheiros de UNIX que controlam as possibilidades de acesso á informação e, no caso dos ficheiros que contêm programas, as possibilidades e modos de execução dos programas. As permissões dividem-se em três grupos: para o dono do ficheiro, para os utilizadores do mesmo grupo que o ficheiro e para outros utilizadores. Neste texto vamos tratar as permissões nos directórios, e as permissões especiais associadas a ficheiros executáveis (suid, sgid, stiky-bit).

2.5.1. CASO ESPECIAL DOS DIRECTÓRIOS

Como é sabido os directórios de UNIX são ficheiros que identificam outros ficheiros (ou outros directórios) atribuindo-lhes nomes, dando ao utilizador a ideia de uma organização em árvore. Sendo tratados pelo sistema operativo como ficheiros têm também associadas permissões que merecem uma análise cuidada á luz da sua função.

Leitura esta permissão indica a possibilidade de lêr o directório, ou seja, a possibilidade de saber quais os ficheiros contidos no directório. Por exemplo o comando ls dá uma mensagem de erro quando se tenta listar um directório que não tem permissão de leitura. A inibição desta permissão provoca a total impossibilidade de acesso aos ficheiros de um directório e mesmo de saber quais os ficheiros nele contidos.

Escrita esta permissão controla a possibilidade de alterar o conteúdo do directório. Com esta permissão controlam-se as possibilidades de criar, mover ou apagar os ficheiros contidos no directório, pois qualquer dessas operações implica a escrita no directório. Por exemplo, o comando

```
ln f1 d1/f2
```

quando o directório d1 não tem permissão de escrita, dá uma mensagem de erro "permission denied".

Execução esta permissão controla nos ficheiros regulares a possibilidade de executar um programa. Um directório nunca pode conter um programa, logo o sistema operativo trata a permissão de execução de um directório de uma forma especial: associa-lhe a possibilidade de um processo tornar um directório no seu directório corrente através da instrução ***cd*** ou da chamada ao sistema **chdir**, consoante a uid efectiva do processo.

2.5.2. PERMISSÕES ESPECIAIS PARA FICHEIROS EXECUTÁVEIS

Para além das 9 permissões habitualmente descritas ao nivel introdutório do UNIX existem 3 outras permissões que podem ser utilizadas apenas para ficheiros que contenham programas. Essas permissões controlam o modo de execução dos programas e são conhecidas pelos seguintes nomes: substitute user-id (***suid***), substitute group-id (***sgid***), e ***sticky-bit***.

suid esta permissão controla o uid efectivo do processo criado ao executar o programa, se estiver ligada o programa assume a identificação do dono do ficheiro onde está contido em vez da identificação do utilizador que lançou o programa. Pode-se assim criar programas que assumem poderes independentemente do utilizador que os executa. Um exemplo tipico desta utilização é o programa passwd acima referido: só um super-utilizador tem permissão para escrever no ficheiro /etc/passwd, no entanto, através do programa passwd qualquer utilizador pode alterar o campo da sua palavra chave, pois o ficheiro /bin/passwd tem a permissão suid ligada. Esta permissão é representada pelo código octal 4000 e só pode ser mudada usando a notação octal do programa chmod. Quando se usa o comando ls -l para visualizar as permissões de um ficheiro a permissão suid aparece representada por um caracter s em vez do caracter x da permissão de execução para o dono do ficheiro.

sgid esta permissão controla o gid do processo gerado ao executar o programa, se estiver ligada o processo

assume o gid do dono do ficheiro onde está contido. Tal como a permissão suid permite a qualquer utilizador executar um programa que realiza acções para as quais esse utilizador não tem permissão. A representação octal da sgid é 2000 e tal como a suid só pode ser mudada com o comando chmod na notação octal. No output do comando "ls -l" a permissão sgid aparece representada com um caracter s substituindo o x da permissão de execução de grupo.

stiky-bit a permissão que controla o que acontece á imagem de um processo após o fim da execução. Habitualmente após um processo terminar a sua imagem é removida da memória, no entanto, se o ficheiro onde o programa está contido tiver a permissão sticky-bit ligada a imagem do processo permanecerá em memória após o fim da execução do processo, tornando assim mais rápidas as execuções subsequentes do mesmo programa. Esta permissão pode ser alterada usando a notação octal do comando chmod e é representada pelo número 1000 ou usando a letra t na notação simbólica do comando chmod. No output de um comando "ls -l" a permissão sticky-bit aparece representada pelo caracter t substituindo o x da permissão de execução para outros. Só um super-utilizador pode alterar a permissão sticky-bit.

Estas três permissões especiais devem ser utilizadas com o máximo cuidado pois podem criar problemas de segurança ou performance da máquina. As permissões suid ou sgid em ficheiros possuidos pelo super-utilizador podem dar a qualquer utilizador a possibilidade de contornar a segurança do sistema. As permissões suid e sgid devem apenas ser associadas a ficheiros que realizam acções perfeitamente controladas.

Um exemplo do output do comando "ls -l" para um ficheiro com as permissões suid e sgid ligadas é o seguinte:

```
-rwsr-sr-x  1 Pedro   usr                    10 Oct  6 11:12 tt
```

O uso indiscriminado da permissão sticky-bit pode provocar a permanência desnecessária em memória de um grande número de processos levando o sistema operativo a usar uma área de disco como extensão da memória (memória virtual ou swaping), o que pode baixar drásticamente a

performance. Os administradores de sistema devem usar do máximo cuidado na utilização desta permissão ligando-a apenas num pequeno número de programas que sejam executados com muita frequência (ex: em máquinas com aplicações de RM-COBOL o intrepretador rmcobol).

Um exemplo de output do comando "ls -l" de um ficheiro com a permissão sticky-bit ligada é o seguinte:

```
-rwxr-xr-t  1 Pedro    usr                    10 Oct  6 11:12 tt
```

Para finalizar dão-se alguns exemplos do uso do comando chmod para ligar as permissões especiais:

- Ligar as permissões suid e sgid para um ficheiro com permissões 755:

```
chmod 6755 f1
```

- Ligar a permissão sticky-bit num ficheiro com permissões 750:

```
chmod 1750 f2
```

- Ligar a permissão sticky-bit num ficheiro, usando a notação simbólica:

```
chmod + t f3
```

As permissões são representadas internamente em dois bytes. Os quatro primeiros bits classificam o tipo do ficheiro (directório, regular, especial, etc.), os doze bits seguintes codificam as permissões da seguinte forma:

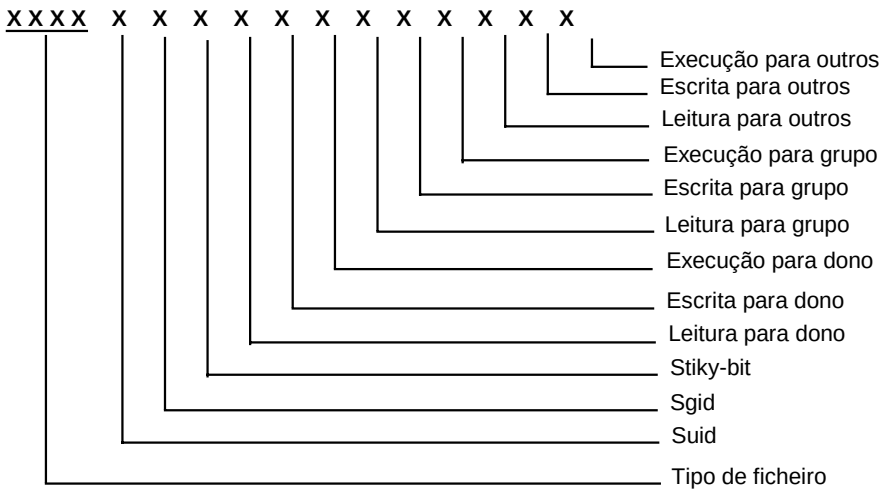


Figura 8. estrutura dos bytes de permissão e tipo

Um ficheiro regular cujo output do comando "ls -l" seja:

```
-r-sr-sr-x   1 lp                    15696 Sep  2  1988
                                /usr/bin/lp
```

Terá o seguinte valor nos bytes de permissões/tipo:

```
0001110101101101
```

3. SISTEMAS DE FICHEIROS

Este capitulo trata da noção de sistema de ficheiros e da descrição do método usado no UNIX para armazenar os ficheiros.

Os sistemas de ficheiros são associados a dispositivos que tenham a possibilidade de acesso aleatório (ex: discos, disquetes, RAM, etc.) e contêm uma estrutura na qual armazenam uma árvore de directórios e respectivos ficheiros.

Os dispositivos de acesso aleatório são também conhecidos por block-devices ou dispositivos de blocos. Tal nome deve-se ao facto de a informação neles contida ser dividida em unidades chamados blocos com um número fixo de caracteres cada. Num dispositivo de acesso aleatório cada bloco pode ser acedido directamente mas cada caracter num bloco só pode ser acedido sequencialmente. A cada dispositivo de bloco corresponde um ficheiro do sistema UNIX que usualmente se encontra abaixo do directório /dev.

No capítulo seguinte discute-se a manipulação de sistemas de ficheiros num sistema UNIX.

Um sistema UNIX pode ter vários sistemas de ficheiros, no entanto, um deles (usualmente o que se encontra na primeira partição do primeiro disco) é usado para carregar o sistema operativo, a esse sistema de ficheiros chama-se ***root file system***. O directório / do sistema é sempre identificado com o directório inicial do root file system, os directórios iniciais dos outros sistemas de ficheiros são identificados com outros directórios através do uso do comando mount.

3.1. ESTRUTURAS DOS SISTEMAS DE FICHEIROS

A estrutura de um sistema de ficheiro é a seguinte:

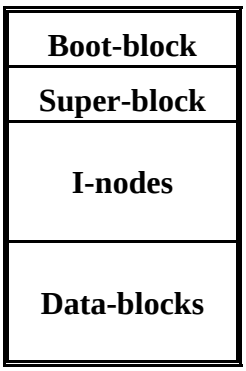


Figura 9. Estrutura de um sistema de ficheiros

onde:

Boot-block	contém um pequeno programa que inicializa o sistema carregando o sistema operativo em memória. Apesar de todos os sistemas de ficheiros terem boot-block o sistema operativo usa sempre o programa boot do sistema de ficheiros principal (root file system).
Super-block	contém os dados globais do sistema de ficheiros. Por exemplo: o número de blocos, o número de I-nodes, a lista dos primeiros blocos de dados livres, a lista dos primeiros I-nodes livres, os instantes de actualização do sistema de ficheiros, etc.
I-nodes	contém os dados referentes aos atributos e á localização dos ficheiros.
Data blocks	contém os dados dos ficheiros, directórios e continuação de dados do super-bloco.

3.1.1. A ESTRUTURA DO SUPER BLOCO

Em cada sistema de ficheiros o super bloco contém a informação de carácter global necessária á gestão dos dados nele contidos. Essa informação é a seguinte:

Número de blocos da I-list
Número de blocos de dados
Número de blocos de dados na lista
Lista dos primeiros blocos livres
.
.
.
Endereço da continuação da lista de blocos livres
Número de I-nodes na lista seguinte
Lista dos primeiros I-nodes livres
.
.
.
Endereço da continuação da lista de I-nodes livres
Lock por manipulação da lista de blocos livres
Lock por manipulação da lista de I-nodes livres
Alterações ao super bloco
Indicações de read-only
Instante da última alteração
Número de blocos livres
Número de I-nodes livres
Nome do sistema de ficheiros
Nome do volume
Estado do sistema de ficheiros
Indicação novo sistema de ficheiros
Tipo do novo sistema de ficheiros

Figura 10. Estrutura do super bloco

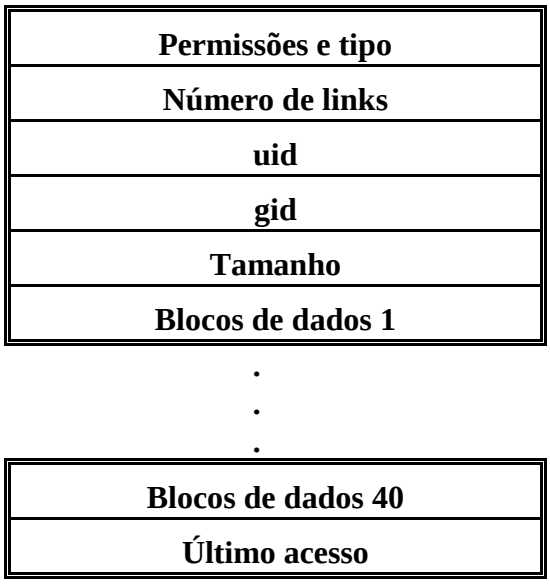
Como se pode ver na figura anterior o super bloco contém um conjunto de informação vital para o sistema de ficheiros, desta destacam-se as listas de blocos de dados e I-nodes disponiveis. O super bloco apenas memoriza um número reduzido de blocos ou I-nodes livres devido ao seu tamanho, as listas são continuadas na área de blocos de dados.

Para garantir uma maior rapidez na manipulação de dados o sistema UNIX mantem em memória uma cópia de cada super-bloco dos sistemas de ficheiros com que está a trabalhar. Qualquer acção que implique a alteração do sistema de ficheiros (ex: criação de um ficheiro) fará uma alteração da cópia em memória do super-bloco e não directamente do super bloco. O super bloco em disco é apenas actualizado periodicamente pelo sistema operativo (tipicamente todos os 30 segundos) ou pelos utilizadores atavés do comando sync.

3.1.2. A ESTRUTURA DOS I-NODES

Cada ficheiro de UNIX é associado a uma entrada numa área do sistema de ficheiros onde está contido chamada I-node. O I-node de um ficheiro contém os dados dos atributos do ficheiro e a informação necessária para localizar os blocos de dados do ficheiro no disco.

A estrutura de um I-node é a seguinte:



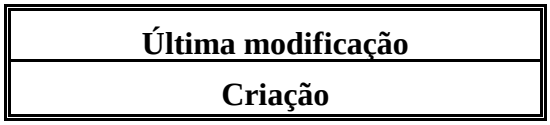


Figura 11. Estrutura do i-node

onde:

Permissões e tipo	dois bytes que indicam o tipo de ficheiro e as permissões.
Número de links	um contador do número de links do ficheiro, ou seja, o número de entradas em directórios que referenciam o ficheiro.
uid	o uid do dono do ficheiro.
gid	o gid associado ao ficheiro.
Tamanho	o número de bytes do ficheiro.
Blocos de dados	os endereços da zona de dados do sistema de ficheiros onde estão contidos os dados do ficheiro.
Último acesso	o instante do último acesso ao ficheiro.
Última modificação	o instante em que foi realizada a ultima modificação do ficheiro.
Criação	o instante em que foi criado o ficheiro.

O UNIX gere automaticamente os blocos de dados ocupados por um ficheiro, ocupando novos blocos quando um ficheiro cresce, e libertando blocos quando um ficheiro diminui. Toda esta gestão é feita usando os campos dos blocos de dados do I-node, no entanto, o I-node é uma estrutura de tamanho fixo com a possibilidade de armazenar apenas 40 endereços de blocos. Se esses endereços de blocos contivessem apenas as áreas do disco onde se encontram os dados de um ficheiro isso provocaria que o tamanho máximo possivel de um ficheiro fosse de 40 blocos (em sistema com blocos de 1 Kbyte 40 Kbytes).

Para tornear esta limitação o UNIX criou um sistema em que os últimos blocos da lista apontam para um bloco que contém endereços dos blocos de dados e ainda para blocos que contém endereços de blocos que contém

endereços de blocos de dados. Para explicar esta metedologia nada melhor que um bom desenho:

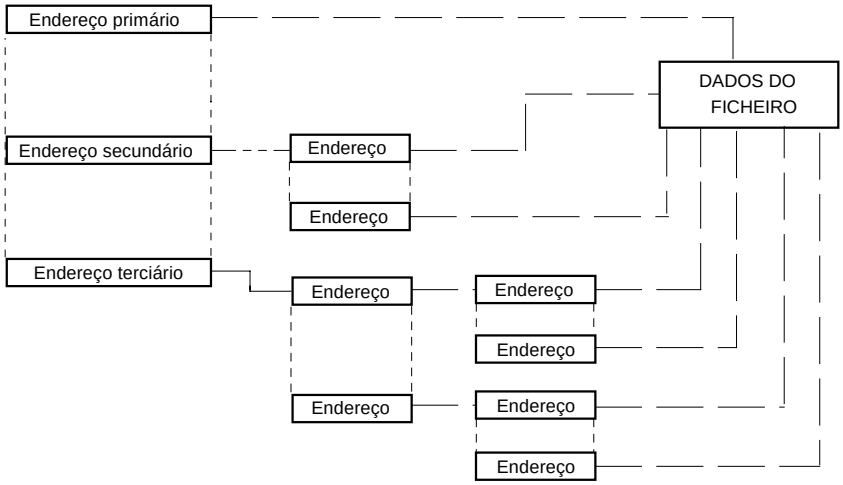


Figura 12. Endereçamento do espaço em disco

3.1.3. A ESTRUTURA DOS DIRECTÓRIOS

A estrutura de I-node não contém nenhuma informação acerca do nome do ficheiro ou da sua posição na árvore de directórios. Tal estrutura é gerida através dos directórios que são tratados como os outros ficheiros tendo apenas um tipo diferente dos ficheiros regulares. Os directórios têm uma estrutura muito simples que contém apenas o número sequencial do I-node na lista de I-nodes (I-number) e o nome atribuído ao ficheiro. Por exemplo o directório que dá o seguinte output para o comando "ls -ia":

```
6956
3012
6962 UNIX_ADM
6495 bib_unix
7015 sccs.man
6616 t
5959 t.c
5918 tt
```

Figura 13. Output de ls -ia

terá a seguinte estrutura:

I-NODE	NOME
6956	**.**
3012	**. .**
6962	**UNIX_ADM**
6495	**bib_unix**
7015	**sccs.man**
5918	**tt**
5959	**t.c**
0	**t.o**
5918	**t**

Figura 14. Conteúdo de um directório

note-se que o penúltimo registo corresponde a um ficheiro que foi apagado, de facto, o sistema quando apaga os ficheiros coloca um zero no número do I-node não removendo o registo do directório. Se for necessário criar um novo ficheiro o sistema ocupa os registos correspondentes a ficheiros apagados. Esta organização tem como consequência que os directórios nunca diminuem de tamanho.

No exemplo acima pode ver-se também que os ficheiros tt e t correspondem ao mesmo I-node sendo de facto dois nomes (links) para os mesmos dados. Se o ficheiro tt já existisse tal resultado poderia ter sido conseguido com o comando:

```
ln tt t
```

Quando os directórios se tornam muito grandes por conterem muitas entradas correspondentes a ficheiros apagados a busca de ficheiros neles contidos torna-se muito lenta. Por isso torna-se necessário às vezes comprimir um directório que por qulquer motivo cresceu demasiado. A única forma de atingir tal fim é copiar os ficheiros e subdirectórios para um directório criado na altura, remover o directório inicial e criá-lo de novo.

Por exemplo para comprimir o directório d1 usar-se-iam as seguintes sequênciasde comandos:

```
$ mkdir d2
$ cd d1
$ find . -print |cpio -pvdulm ../d2
$ cd ..
$ rm -r d2
$ mv d2 d1
```

Figura 15. Compressão de um directório

Os directórios demasiado grandes, seja por conterem muitos ficheiros, seja por conterem muitos registos desocupados, podem fazer baixar muito a performance do sistema. Por exemplo: se incluirmos um directório demasiado grande na variável PATH o interpretador de shell levará muito tempo a iniciar a execução de um programa que se encontre num directório ou no seguinte. Por estas razões os administradores de sistema devem detectar eventuais directórios com tamanho anormal. A melhor forma de conseguir esse fim é a utilização do comando find. Por exemplo: o seguinte comando procura em toda a árvore directórios com mais de 5 blocos de tamanho:

```
find / -type d -size +5 -print
```

A opção "-type d" do comando find indica que se procura ficheiros do tipo directório (outras opções possíveis f-ficheiros regulares, c-ficheiros especiais de caracteres e b-ficheiros especiais de bloco (ver cap. 4)). A opção "-size +5" indica que se procuram ficheiros com tamanho superior a 5 blocos.

Como foi visto um sistema UNIX pode conter vários sistemas de ficheiros sendo necessário identificar os directórios principais dos sistemas de ficheiros com um directório de forma a ser possivel aceder á informação contida no sistema de ficheiros. O comando que faz essa identificação chama-se ***mount*** e tem o seguinte formato:

```
mount [ ficheiro-especial
directório [ -r ] ]
```

os argumentos são o ficheiro-especial que indica o dispositivo onde se enconta o sistema de ficheiros (ex: /dev/dsk/0s1), o directório que fica associado ao directório principal do sistema de ficheiros e a opção -r indica que o sistema de ficheiros será montado sem possibilidade de escrita tornando assim impossivel escrever em qualquer ficheiro que se encontre abaixo do directório especificado. Se se omitirem os argumentos e as opções o comando mount lista todos os sistemas de ficheiros que se encontram montados.

Exemplos:

Montar um sistema de ficheiros:

```
$ mount /dev/dsk/0s2
/usr/acct
```

faz com que os ficheiros da 2ª partição do 1º disco fiquem acessíveis a partir do directório /usr/acct.

Listar os sistemas de ficheiros montados:

```
$ mount
/ on /dev/dsk/60s1 read/write on Fri Oct  6 05:34:26 1989
```

O UNIX esconde do utilizador a existência de vários sistemas de ficheiros através da noção de mount, depois de montado um sistema de ficheiros não é necessário identificar o volume onde se encontra um determinado ficheiro quando se lhe faz referência. Esta organização torna completamente transparentes as cópias e transferências de dados entre sistemas de ficheiros sendo apenas necessário usar os comandos habituais cp ou mv.

Uma limitação importante provocada pela existência de vários sistemas de ficheiros montados é o uso do comando ln pois é proibido establecer links entre ficheiros de sistema diferentes já que o número do I-node só é um identificador único dentro de um sistema de ficheiros. Existe no entanto, um comando chamado rln que tem a mesma sintaxe do comando ln e que serve para estabelecer links entre ficheiros de diferentes sistemas de ficheiros.

Quando se deseja desmontar um sistema de ficheiros da árvore de directórios, tornando assim inacessíveis os seus ficheiros usa-se o comando **umount** que tem o seguinte formato:

```
umount ficheiro-especial
```

onde ficheiro-especial é o dispositivo de blocos onde se encontra o sistema de ficheiros.

3.2. MONITORIZAÇÃO DO ESPAÇO LIVRE EM SISTEMAS DE FICHEIROS

Uma boa administração de sistema requer o acompanhamento do espaço disponível em cada sistema de ficheros. A observação periódica de tal espaço permite a detecção de situações anómalas (por exemplo ficheiros que crescem desmesuradamente), e a realização de previsões e estimativas quanto a necessidades de evolução da configuração do sistema em termos de hardware.

Existe no UNIX um comando que permite saber o número de blocos disponiveis na área de blocos de dados, e o número de I-nodes desocupados de um sistema de ficheiros, tal comando chama-se ***df (disk-free)*** e tem o seguinte formato:

df [opções]
[sistema_de_ficheiros...]

As opções possiveis são:

-t mostra o número de blocos e I-nodes livres e o número total de blocos e I-nodes no sistema de ficheiros.

-f mostra apenas o número de blocos livres (não mostra o número de I-nodes) calculado a partir da lista de blocos livres (ao contrário do habitual que mostra o valor do campo de número de blocos livres do super-bloco).

Se não se indicarem os sistemas de ficheiros o comando df dá a informação relativa a todos os sistemas de ficheiros montados. Os argumentos podem ser indicados através do nome de um ficheiro especial ou através do nome do directório no qual se encontra montado.

Este comando poderá ser melhor entendido através dos seguintes exemplos:

```
$ df -t /
/          (/dev/dsk/60s1   ):   25552 blocks    4108 i-nodes
  total:    183480 physical (22935 4096-byte logical) blocks  11456
i-nodes
$
```

Figura 16. Exemplo do comando df -t

mostra o número de blocos livres e o número total de blocos no sistema de ficheiros principal.

```
$ df
/            (/dev/dsk/60s1    ):     25552 blocks
4108 i-nodes
/usr         (/dev/dsk/60s2    ):      7981 blocks
2876 i-nodes
$
```

Figura 17. Exemplo do comando df

mostra o número de blocos livres em todos os sistemas de ficheiros montados na altura da sua execução.

```
$ df -f /dev/dsk/60s1
/          (/dev/dsk/60s1):    25560 blocks
```

Figura 18. Exemplo do comando df -f

calcula e mostra o número de blocos livres no sistema de ficheiros contido no disco associado ao ficheiro /dev/dsk/60s1 por contagem do número de elementos da lista de blocos livres respectiva, independentemente do sistema de ficheiros se encontrar ou não montado.

Note-se que a opção -f faz com que o comando df seja muito mais lento e dá o mesmo resultado se não se usar nenhuma opção excepto se o sistema de ficheiros se encontrar corrompido.

3.3. VERIFICACÇÃO E CORRECÇÃO DA CONSISTÊNCIA DOS SISTEMAS DE FICHEIROS

Os sistemas de ficheiros sendo o suporte de toda a informação guardada num sistema UNIX são uma peça muito sensível. Como foi visto anteriormente a estrutura dos sistemas de ficheiros é bastante complexa, e como tal sujeita a

eventuais incoerências. Essas incoerências podem provocar perdas de informação, ou mesmo a inoperacionalidade total do sistema.

Entre outros problemas é possivel um ficheiro ficar sem nome em nenhum directório, o número de blocos ocupados por um ficheiro não corresponder ao tamanho deste, ou mesmo corronperem-se os dados do super-bloco tornando um sistema de ficheiros totalmente inutilizável.

Para aumentar a rapidez o UNIX mantém em memória uma cópia do super bloco de cada sistema de ficheiros montado na qual faz as actualizações sempre que necessário, tais cópias são escritas periódicamente em disco com vista a manter os super blocos dos discos actualizados. Um utilizador pode em qualquer momento actualizar os super blocos através do comando ***sync***.

Uma falha de corrente, ou qualquer outro imponderável pode deixar um sistema de ficheiros com o respectivo super-block desactualizado perdendo-se a coerência dos dados. Para verificar a coerência da informação de um sistema de ficheiros existe o comando ***fsck*** que detecta e repara enventuais incoerências.

O comando fsck tem o seguinte formato:

fsck [-y | -n] [-s] [ficheiro-especial]

a opção -y responde automáticamente que sim a todas as perguntas, a opção -n responde automáticamente que não a todas as perguntas, a opção -s força a reconstrução da lista de blocos livres, ficando a nova lista segundo uma ordem óptima o que permite melhorar alguns tempos de acesso a dados.
Se for indicado o nome de um ficheiro especial será verificada a coerência do sistema de ficheiros nele contido. Caso não se indique nenhum ficheiro especial será verificado o sistema de ficheiros principal.

O programa fsck funciona em 6 fases:

Fase 1	***Verificação de blocos e tamanhos***: verifica a coerência da informação contida na lista de I-nodes; procura blocos duplicados (usados por mais de um ficheiro), verifica os tipos dos ficheiros e o formato dos I-nodes.
Fase 2	***Verificação de nomes:***.serve para limpar as referência a I-nodes que tenham sido apagados na primeira fase.

Fase 3	***Verificação de conectividade:*** trata-se de verificar se para cada I-node existe pelo menos uma entrada num directório do sistema de ficheiros. Caso se encontrem ficheiros sem nome em nenhum directório o sistema cria um nome para o ficheiro no directório lost+found abaixo do directório principal do sistema de ficheiros que se está a verificar.
Fase 4	***Verificação do contador de referência:*** Conecção dos erros gerados pela fase anterior (por exemplo o directório lost+found cheio) e verifica-se a consistência do contador de links em cada I-node. Verifica-se também a coerência da lista e do contador de I-nodes livres mantidos no super-bloco.
Fase 5	***Verificação da lista de blocos livres:*** trata-se de verificar se a lista de blocos de dados livres mantida no super-bloco corresponde de facto aos blocos não ocupados por ficheiros.
Fase 6	***Reconstrução da lista de blocos livres:*** só se realiza se a fase anterior detectou algum erro na lista de blocos livres, ou se o utilizador usou a opção -s ao executar o programa fsck.

O comando fsck só deve ser executado sobre sistemas de ficheiros que não se encontrem montados para evitar alterações durante a verificação. Caso haja alterações ao sistema de ficheiros durante a verificação da coerência as acções de reparação podem destruir dados e deixar um sistema originalmente coerente totalmente descoordenado. O sistema de ficheiros principal por se encontrar sempre montado é a única excepção a esta regra, no entanto, deve apenas ser verificado quando a máquina se encontra na situação de single-user (mono-posto) para garantir que nenhum processo está a alterá-lo.

Os sistemas de ficheiros devido á sua importância são uma das mais sensiveis partes de um sistema UNIX. As incoerências dos sistemas de ficheiros podem provocar grandes desastres com enormes perdas de informação. Qualquer pequeno erro pode com o passar do tempo agravar-se até tornar o sistema de ficheiros completamente inutilizável. Face a tudo isto é aconselhável correr o programa fsck frequentemente sobre todos os sistemas de ficheiros. Em geral os sistemas UNIX vêm configurados de forma a que o programa fsck é executado sobre todos os sistemas de ficheiros na altura em que a máquina é ligada. Nos sistemas que se mantêm permanentemente ligados é conveniente correr o programa fsck manualmente com frequência.

Segue-se um exemplo de uma execução do programa fsck:

```
$fsck /dev/dsk/60s1
   /dev/dsk/60s1
   File System:  Volume:

   ** Phase 1 - Check Blocks and Sizes
   POSSIBLE FILE SIZE ERROR I=3181
Number of Blocks = 0, File Size = 1

   POSSIBLE FILE SIZE ERROR I=6560
Number of Blocks = 0, File Size = 1

   POSSIBLE FILE SIZE ERROR I=7216
Number of Blocks = 0, File Size = 1

   POSSIBLE FILE SIZE ERROR I=7227
Number of Blocks = 0, File Size = 1

   ** Phase 2 - Check Pathnames
   ** Phase 3 - Check Connectivity
   ** Phase 4 - Check Reference Counts
   LINK COUNT FILE I=111  OWNER=root MODE=20622
   SIZE=0 MTIME=Oct  6 17:20 1989
   COUNT 5 SHOULD BE 2
   ADJUST? Y
   UNREF FILE I=7216  OWNER=dge MODE=10000
   SIZE=0 MTIME=Oct  6 17:17 1989  -- CLEARED
   UNREF FILE I=7227  OWNER=dge MODE=10000
   SIZE=0 MTIME=Oct  6 17:17 1989  -- CLEARED
   FREE INODE COUNT WRONG IN SUPERBLK
   FIX? Y
   ** Phase 5 - Check Free List
   2 BLK(S) MISSING
   BAD FREE LIST
   SALVAGE? Y
   ** Phase 6 - Salvage Free List
   7224 files 147856 blocks 34176 free
   *** ROOT FILE SYSTEM WAS MODIFIED ***
   *****  BOOT  THE  OS  (DO  NOT  PERFORM  A  SYNC
OPERATION) *****
```

Figura 19. Exemplo de um programa fsck

A primeira mensagem de erro indica que o ficheiro cujo I-number é 3181 tem um tamanho incorrecto (deveria ocupar 1 bloco e ocupa 0 blocos), esta

mensagem não representa um erro grave e pode ser resolvida copiando o ficheiro e movendo o novo ficheiro para o nome original.

A mensagem "LINK COUNT ..." indica que o ficheiro cujo I-number é 111 tem o contador de links a 5 e só existem duas referências ao ficheiro em directórios. Se se responder Y á pergunta ADJUST? o sistema altera o contador para 2.

As mensagens "UNREF FILE ..." indicam ficheiros que não são referênciados em nenhum directório, como os ficheiros estavam vazios (SIZE=0) o I-node foi apagado, caso se encontrasse um ficheiro não vazio nestas condições o programa perguntaria RECONECT? e caso a resposta fosse Y seria criado um novo ficheiro no directório lost+found cujo nome seria o I-number.

A mensagem "FREE INODE COUNT WRONG IN SUPERBLK" indica que o contador de I-nodes livres está errado. A resposta Y á pergunta FIX? provoca a reactualização do contador com um valor correcto.

Finalmente a mensagem "2 BLK(S) MISSING ..." significa que a lista de blocos livres está incorrecta. A resposta Y á pergunta SALVAGE? provoca a reactualização da lista de blocos livres com valores correctos.

Note-se que as respostas positivas a algumas destas perguntas podem implicar a perda de dados, enquanto a resposta negativa deixa o sistema de ficheiros incongruente.

Para obter alguns dados importantes sobre sistemas de ficheiros existe um utilitário bastante útil: o programa ***ncheck***. Este programa tem o seguinte formato:

ncheck [-i num] [-s] [-a] [sistema-de-ficheiros]

Sem nenhuma opção é listado na saída de dados standard uma lista de todos os ficheiros do sistema ou sistemas de ficheiros em consideração precedidos do número do respectivo I-node. Se não se indicar nenhum sistema de ficheiros (através do respectivo ficheiro especial) o comando trata todos os sistemas de ficheiros listados em ***/etc/checklist***. A opção -i não lista apenas os nomes correspondentes ao I-node cujo número é pedido. A opção -a lista também os nomes . e .. que não são listados normalmente. A opção -s lista apenas os ficheiros com a permissão suid ou sgid ligadas.

Uma das utilidades mais imediatas deste programa é descobrir o nome de um ficheiro através do seu I-number, por exemplo para descobrir um ficheiro

listado com tamanho incorrecto no output de um comando fsck. Suponhamos que o comando fsck tinha enviado uma mensagem de erro a propósito do ficheiro com I-number 1351, então para descobrir o nome desse ficheiro poderia fazer-se:

```
ncheck -i 1351
/dev/dsk/60s1
1351  /usr/acct/pedro/tx/abc
```

Figura 20. Exemplo do comando ncheck

3.4. CÓPIAS DE SEGURANÇA DE SISTEMAS DE FICHEIROS

Uma das formas de garantir a segurança dos sistemas de ficheiros é a cópia destes para meios removíveis (por exemplo: disquetes ou bandas magnéticas). O sistema unix fornece um programa chamado ***volcopy*** cuja função é copiar todo um sistema de ficheiros para um meio removível.

Os dispositivos que se usam nas cópias com o comando volcopy têm que ter uma marca reconhecível por software a que se chama "label". O programa para verificar e actualizar as marcas chama-se ***labelit***.

O comando labelit tem o seguinte formato:

/etc/labelit fich-especial [sf volume [-n]]

onde:

fich-especial representa o ficheiro especial associado ao dispositivo que se quer tratar.

sf representa o nome do sistema de ficheiros que irá ficar contido no volume.

volume representa o nome do volume associado ao dispositivo (o programa volcopy faz cópias multi-volume).

-n é uma opção que obriga o programa labelit a escrever uma nova marca.

Se se usar o programa labelit só com o primeiro argumento obtém-se uma lista da marca do dispositivo.

Um exemplo do uso do programa labelit é o seguinte:

```
# labelit /dev/rtp
Current fsname: root, Current volname: rtp,
Blocks: 1989282899, Inodes: 24222
```

```
FS Units: 512b, Date last mounted: Thu Sep 24
12:08:05 1989
# labelit /dev/rtp root tape01 -n
Skipping label check!
NEW fsname=root, NEW volname=tape01 -- DEL if
wrong!!
```

Figura 21. Exemplos do comando labelit

O programa volcopy tem o seguinte formato:

volcopy fs dev-from vol-from dev-to vol-to

onde:

fs o nome do sistema de ficheiros a copiar (ex: /, /usr).

dev-from o nome do ficheiro especial associado ao dispositivo que contém o sistema de ficheiros a copiar.

vol-from a marca do volume a copiar.

dev-to o nome do ficheiro especial associado ao dispositivo de destino da cópia.

vol-to a marca do volume de destino.

Segue-se um exemplo de um comando volcopy:

```
# volcopy root /dev/rdsk/0s1 ds1 /dev/rtp tape01
You will need 1 reels.
(The same size and density is expected for all
reels)
From: /dev/rdsk/0s1, to: /dev/rtp? (DEL if wrong)
```

Figura 22. Exemplo de um comando volcopy

As cópias de segurança criadas com o comando volcopy só podem ser usadas para repôr todo o sistema de ficheiros. As cópias de segurança que possibilitam a reposição de ficheiros individuais devem ser feitas com os programas de arquivo de ficheiros tar ou cpio.

3.5. REORGANIZAÇÃO DE SISTEMAS DE FICHEIROS

Os sistemas de ficheiros vão perdendo performance com o passar do tempo devido ao crescimento dos directórios e ao facto não existirem blocos disponiveis para fazer crescer os ficheiros nas posições óptimas para a velocidade de acesso aos dados.

O programa ***dcopy*** permite copiar um sistema de ficheiros de um dispositivo para outro do mesmo tipo e tamanho optimizando o tempo necessário para aceder aos ficheiros.

O formato básico do comando dcopy é o seguinte:

```
dcopy fs-ori fs-dest
```

onde fs-ori é o ficheiro especial que contém o sistema de ficheiros que se quer optimizar e fs-dest é o o nome do ficheiro especial associado ao dispositivo onde irá ficar a cópia optimizada do sistema de ficheiros. Por exemplo o comando:

```
# dcopy /dev/dsk/0s1
/dev/dsk/0s2
```

copia o sistema de ficheiros contido no disco /dev/dsk/0s1 para o disco /dev/dsk/0s2 que fica com uma versão optimizada.

O comando dcopy é de uso muito restrito pois implica a existência de dois dispositivos de acesso aleatório com o mesmo tamanho estando um deles vazio, o que só muito raramente é possivel.

3.6. CRIAÇÃO DE SISTEMAS DE FICHEIROS

A introdução de novos discos, a utilização de disquetes ou a necessidade de racionalização da utilização dos discos existentes tornam por vezes necessário criar novos sistemas de ficheiros. Em alguns computadores usando o UNIX Sistema V Release 3 ou posterior é tambem possivel utilizar parte da memória central (RAM) para criar sistemas de ficheiros voláteis de acesso ultra-rápido para instalação de aplicações de uso muito frequente.

As necessidades acima descritas tornam necessário o conhecimento por parte dos administradores de sistema dos procedimentos necessários á criação de um novo sistema de ficheiros.

A criação de um sistema de ficheiros deve ser precedida, no caso dos discos e das disquetes, de uma operação de formatação. Esta operação é bastante dependente do sistema e é geralmente realizada por um comando de nome ***format***. As opções e argumentos do comando format variam bastante com o computador e os administradores de sistema devem consultar os manuais da sua máquina para conhecerem os detalhes.
Posteriormente à formatação os discos podem também ter necessidade de uma operação de distribuição do espaço por várias unidades chamadas partições. As partições irão corresponder a diferentes ficheiros especiais de acesso aleatório nos quais se pode criar sistemas de ficheiros. A operação de particionamento dos discos é feita usando um comando dependente do sistema e com um nome variável (por ex: em sistemas XENIX ***Fdisk***, em sistemas TOWER-NCR dkpart).

O comando usado para criar um novo sistema de ficheiros chama-se ***mkfs*** (***make file-sistem***) .R e tem o seguinte formato:

mkfs fich_especial proto

ou

mkfs fich_especial blocos i-nodes

Como se pode ver existem duas versões deste comando: A primeira permite especificar os pârametros necessários num ficheiro de texto (proto), e simultâneamente especificar directórios e ficheiros a carregar na altura. Na segunda versão são especificados na linha de comando os dados minimos necessários à criação de um sistema de ficheiros totalmente vazio (só com o directório / vazio).

Os pârametros da primeira forma do comando mkfs são o número de blocos disponíveis para criar o sistema de ficheiros (ou seja a dimensão fisica do dispositivo), o número de I-nodes do novo sistema de ficheiros (ou seja a quantidade de ficheiros que será possível criar), e o nome do ficheiro especial no qual o sistema de ficheiros irá ficar contido.

É necessário ter a maior atenção com o uso deste comando pois ele destrói toda a informação contida no dispositivo.

A segunda versão do comando mkfs cria o sistema de ficheiros com algum conteúdo inicial utilizando os pârametros descritos num ficheiro de texto designado por proto. Um ficheiro proto contém instruções separadas por espaços ou caracteres de mudança de linha. A primeira instrução é o nome do ficheiro donde será copiado o bloco de boot. A segunda instrução é o número de blocos ***físicos*** disponíveis para o novo sistema de ficheiros. A instrução seguinte é o número de I-nodes. As instruções seguintes são a descrição do ficheiro / do novo sistema de ficheiros: o tipo, as permissões, o uid, o gid e o conteúdo inicial que depende do tipo. O tipo e permissões do ficheiro são indicados usando 6 caracteres:

- o primeiro pode ser **-, b, c, d** e especifica o tipo (respectivamente: regular, especial de bloco, especial de caracter e directório).

- o segundo especifica a permissão ***suid*** e pode ser **-** ou **u** consoante essa permissão está desactivada ou activada.

- o terceiro especifica a permissão ***sgid*** e pode ser **-** ou **g** consoante essa permissão está desactivada ou activada.

- os três últimos caracteres especificam as restantes permissões em notação octal.

Seguidamente ao tipo e permissões indicam-se o uid e gid do ficheiro. Se o ficheiro é regular indica-se no fim o nome do ficheiro do sistema do qual será copiado o conteúdo a colocar no novo ficheiro, se o ficheiro é um ficheiro especial então os argumentos a seguir ao gid são os major e minor-number (ver adiante) do ficheiro. Se o ficheiro é um directório o mkfs cria automáticamente os directórios . e .. e lê recursivamente uma lista de especificação de ficheiros (com o formato acima descrito) para inicializar o directório. A leitura do conteúdo de um directório termina com o caracter $.

Pode ver-se de seguida um exemplo de um ficheiro proto:

```
/dev/null
```

```
4872 110
d--777 3 1
usr     d--777 3 1
        sh      ---755 3 1 /bin/sh
        ken     d--755 6 1
                $
        b0      b--644 3 1 0 0
        c0      c--644 3 1 0 0
        $
$
```

Figura 23. Exemplo de um ficheiro proto

Na primeira linha indica-se o ficheiro do qual se vai copiar o bloco de boot, neste caso o ficheiro /dev/null o que significa que o bloco de boot ficará vazio.

Na segunda linha especifica-se que o sistema de ficheiros será um total de 4872 blocos fisicos e 110 i-nodes.

As linhas três a nove definem os ficheiros e directórios que ficaram no novo sistema de ficheiros.

4. PROCESSOS

4.1. CONTROLO DE PROCESSOS

4.1.1. INTRODUÇÃO

"Processo" é a designação num ambiente UNIX de uma tarefa que o sistema está encarregue de cumprir. ***É fundamental distinguir processo de programa***. Programa é um conjunto de instrucções que descrevem como realizar determinada tarefa e que tipos de dados vão ser tratados. Processo é uma cópia de um programa, instalada na memória do computador, a correr ou em estado de espera, com uma área de memória reservada para dados que estão a ser tratados por esse programa. Pode-se portanto dizer que um processo é uma imagem ***viva*** de um programa, quando este foi chamado a tratar dados.

O sistema operativo encarrega-se de distribuir a atenção da CPU pelos diversos processos, pondo o processador ao seu dispôr sequencialmente. Os programas contidos em cada um dos processos tratam os dados da sua área de memória ou requesitam serviços ao sistema operativo. Estes serviços podem ser diversos: enviar uma mensagem para um terminal, escrever num ficheiro em disco, criar um novo processo, etc. .P Os processos são numerados ao longo de uma sessão e os números não se repetem. Se determinado utilizador chamar o comando "ls" duas vezes seguidas isso dará origem a dois processos diferentes, com diferentes números.

4.1.2. GESTÃO INTERNA

A gestão de processos apoia-se numa tabela interna ao Kernel - a tabela de processos, que conté informação diversa sobre estes. O leitor deve já ter travado conhecimento com o comando ***ps***, que permite visualizar a tabela de processos do sistema. Sem argumentos, o comando apresenta uma lista de processos associados ao terminal que emite o comando. O comando ***ps*** tem algumas opções que a seguir se explicam.

É possivel consultar a tabela de processos do computador através do comando "ps -ef", que dá um output parecido com:

```
$ ps -ef
     UID   PID  PPID  C    STIME   TTY      TIME COMMAND
    root     0     0  0  05:25:38    ?      0:01 sched
    root     1     0  0  05:25:38    ?      0:10 /etc/init
    root     2     0  0  05:25:38    ?      0:01 vhand
    root     3     0  0  05:25:38    ?      0:00 bdflush
    root   109     1  0  05:26:12    tty00  0:00 /etc/getty tty00 d
vt100
  Manel   623     1  0  07:16:18    tty02  0:02 -csh
      lp   90     1  0  05:26:03    ?      0:00 /usr/lib/lpsched
    root   99     1  0  05:26:05    ?      0:01 /etc/cron
    ncrm  105     1  0  05:26:06    ?      0:00 /usr/lib/errdemon
  Manel   630   623  0  07:16:53    tty02  0:02 vi UNIX-ADM.II
      lp  603    90  0  07:10:54    ?      0:00 /usr/lib/lpsched
  Manel   634   632  6  07:20:18    tty02  0:00 ps -f -ef
$
```

Figura 24. Comando ps -ef

A lista completa de informações sobre processos pode ser obtida com o comando:

ps -eafl

e contém os seguintes campos:

NOME	OPÇÕES	DESCRIÇÃO
F	1	Flags do processo (octal e aditivas) *
S	1	Estado do processo *
UID	f, 1	Identidade do utiliador dono do processo
PID	todas	Identidade do processo
PPID	f, 1	Identidade do processo pai
C	f, 1	Utiliação do processador
PRI	1	Prioridade do processo (invertida) **
NI	1	Valor "Nice", p/ o calculo da prioridade
ADDR	1	Endereço em memória ou disco (swap)
SZ	1	Tamanho em blocos do processo
WCHAN	1	Evento pelo qual está à espera o processo
STIME	f	Hora de arranque do processo
TTY	todas	Terminal que controla o processo
TIME	todas	Tempo de execução acumulado
CMD	todas	Nome do comando (completo com a opção -f)

* ver manual PS(1) para mais detalhes
** quanto maior o valor menor a prioridade

Figura 25. Opções do comando ps

4.1.3. SINAIS

Quando um processo está a correr é possivel interrompê-lo através do envio de ***sinais***. Os sinais são mensagens simples que o Kernel passa ao processo. Certos sinais podem provocar a morte do processo, outros são ignorados. No entanto, estas reacções são na sua maior parte alteráveis pelo próprio processo. Se este assim decidir, pode não morrer ou ignorar quando recebe um determinado sinal, mas reagir de uma outra forma. Os sinais podem ser enviados com o comando kill (a partir de outro terminal) ou com certas teclas (a partir do terminal que deu origem ao processo). Só o dono do processo lhe pode enviar sinais ou o SuperUser.

Os sinais que podem ser enviados através do terminal são:

INTERRUPT (2) "Premindo a tecla <Delete> ou <Rubout> o processo recebe este sinal, e se não está protegido (comando trap, se fôr um shell script) morre; esta tecla pode ser escolhida com o comando "stty intr <TECLA>".

QUIT (3)" Premindo as teclas <CTRL>+<\\> o processo recebe este sinal, que também pode ser ignorado pelo processo, se este assim decidir; se não estiver a ignorá-lo, o processo morre, deixando uma imagem sua em disco com o nome "core".

HANGUP (1)" Ao desligar o terminal a meio de um processo, o sistema encarrega-se de enviar este sinal ao processo, que morre se não tiver optado por ignorá-lo.

Todos os sinais sem execpção podem ser enviados através do comando ***kill -X PID*** em que X é o código numérico do sinal e PID é a identidade do processo.

De entre todos os sinais, o único que o processo não pode ignorar é o ***SIGKILL (9)***" que faz com que qualquer processo morra de certeza.

4.1.4. EXEMPLO DE GESTÃO DE SINAIS

Do ponto de vista do processo o sinal pode aparecer em qualquer altura, pelo que o sistema operativo e a linguagem de programação têm que conter um mecanismo de gestão de sinais. Vamos avaliar este mecanismo para a linguagem de programação ***shell***.

O programa que seguidamente se mostra serve para preencher uma tabela, mas devido á sua importância é fundamental que esta seja bem preenchida, pelo que só se coloca a tabela no ficheiro quando os valores estão todos num ficheiro temporário.

```
$ cat prog1

TMP=/tmp/temp$$
TAB=/usr/acct/maria/tab.values

rm $TMP

for val in alpha beta gamma zeta
do
     echo "$val :\c"    read value
     echo $val $value >> $TMP
done

# tabela completa, substituir velha tabela
cp $TMP $TAB
rm $TMP

$
```

Figura 26. Sinais - programa sem cuidados

Se por acaso o utilizador deste programa desligasse o terminal na altura, o Kernel enviaria um sinal HANGUP ao processo, e este morreria, deixando um ficheiro no directório /tmp. Mas o problema podia ser ainda maior: o HANGUP podia atingir o processo a meio do comando cp, e isto faria com que nem a tabela velha escapasse intacta.

O melhor seria inserir no programa instrucções de gestão de sinais, que "apanhassem" o HANGUP quando este chegasse.

```
$ cat prog1

TMP=/tmp/temp$$
TAB=/usr/acct/maria/tab.values

# preparar o programa para nao deixar o temporario
se receber um sinal trap "rm $TMP; echo PROGRAMA
ABORTADO; exit 1" 1 2 3 15

rm $TMP

for val in alpha beta gamma zeta
do
     echo "$val :\c"    read value
     echo $val $value >> $TMP
done

# ignorar os sinais perigosos nesta fase mais
critica
trap "echo SINAL IGNORADO" 1 2 3 15
# tabela completa, substituir velha tabela
cp $TMP $TAB
rm $TMP

$
```

Figura 27. Sinais - programa cauteloso

O programa fica assim com duas atitudes diferentes em relação aos sinais: na primeira fase apanha-os e morre de uma forma mais limpa (removendo o temporário), e na segunda fase, em que é fundamental que não seja interrompido ignora simplesmente os sinais que chegarem.

4.2. SEQUÊNCIA DE INICIALIZAÇÃO (BOOT) DE UM SISTEMA UNIX

4.2.1. PRIMEIRA FASE - ROM/BOOT/UNIX/INIT

Na fase de Boot (arranque, inicialização) de qualquer máquina, o processador vai ler as suas primeiras instrucções à ROM (FirmWare), que contém um pequeno programa que lhe diz o que deve fazer seguidamente. O objectivo deste programa é encontrar e carregar para a memória um sistema operativo que se encarregue de gerir o hardware daí em diante.

Muitos destes programas armazenados em ROM perguntam ao operador da máquina onde devem ir buscar o S.O., mas outros tomam a iniciativa de o ir buscar a um disco. A descrição seguinte pode não ser exacta para a máquina com que o leitor trabalha mas dá uma ideia de como o boot é efectuado.

O firmware da ROM começa por ir carregar um pequeno programa que está no disco principal do sistema e passa a execução para este. Este programa, no caso das máquinas UNIX, já conhece a organização dos ficheiros e procura na árvore do file-system o ficheiro "/unix", que contém o núcleo do S.O. Depois de o encontrar carrega-o para memória e passa-lhe o controlo das operações. O ***Kernel*** (assim se chama o núcleo do UNIX) começa por organizar-se internamente, avaliando a memória, preparando os acessos aos ficheiros, a gestão de processos (tarefas), etc.

Nesta altura lança o primeiro processo da sessão, cujo programa é carregado do ficheiro "/etc/init". Este processo fica com o número 1 e o Kernel continua em memória como processo número 0. Consoante as versões ou configurações do UNIX em jogo, o processo 0 pode dar inicio a outros processos (de gestão de memória virtual, por exemplo), mas isso só acontece com casos especiais, sendo todos os outros processos, directa ou indirectamente gerados pelo processo init.

4.2.2. O PROGRAMA/PROCESSO INIT

O init é responsável directa ou indirectamente por todos os outros processos existentes numa máquina UNIX. Se um terminal pede ou não login, a responsabilidade da decisão é do init. Se o sistema passa ou não a "multi-user mode", quem decide é o init.

Para compreender isto é necessário conhecer o conceito de ***run-level*** (nivel de execução). O run-level é o estado da máquina. Normalmente os S.O.'s UNIX vêm configurados de modo a que no run-level 0 só funcione a consola, e no run-level 1 funcionem todos os terminais. A isto costuma chamar-se de ***single-user mode*** e ***multi-user mode***. Mas esta configuração pode alterar-se fácilmente, de modo, por exemplo, a que funcionem apenas metade dos terminais em run-level 1, a outra metade em run-level 2 e todos os terminais em run-level 3.

Pelo facto de em run-level 0 só funcionar a consola (na configuração mais habitual) isto não significa que não possa haver multi-programação (existência de mais do que um processo em simultâneo). Pode sempre existir. Repare-se que é apenas uma mudança interna ao init. O Kernel do UNIX mantém-se a funcionar, e como é ele que trata a gestão dos processos a multi-programação continua possivel.

4.2.3. A TABELA DE CONTROLO DO INIT: /ETC/INITTAB

Depois de ser chamado pelo Kernel, o init está em run-level 0. Com base neste parâmetro vai ler a tabela /etc/inittab que tem o seguinte aspecto:

```
$  cat  /etc/inittab
su:0:respawn:/etc/getty syscon C vt100 # Single-
user console
rc:1:wait:/etc/rc < /dev/syscon > /dev/syscon 2>&1
#multiuser start t00:1:respawn:/etc/getty tty00 d
4970
t01:1:respawn:/etc/getty tty01 d tvi925
t02:1:respawn:/etc/getty tty02 d 4970
t03:1:respawn:/etc/getty tty03 y vt100
$
```

Figura 28. Ficheiro /etc/inittab

Esta tabela é composta por quatro campos, com o seguinte significado:

ID um a quatro caracteres que identificam a linha do ficheiro.

RSTATE o run-level a que é relativa a linha, se o sistema está em run-level 1, só as linhas que contêm 1 neste campo são processadas.

ACTION uma palavra chave que diz ao init como tratar o processo, algumas destas são:

- ***respawn*** o processo deve ser criado sempre que não exista.
- ***bootwait*** o processo deve ser criado quando a máquina estiver a arrancar e o init deve esperar que o processo acabe sem fazer mais nada;
- ***wait*** quando o init atinge o run-level definido pelo .I rstate .R da linha, dá inicio ao processo e espera pelo fim deste; o processo nunca é reiniciado

once quando o init atinge o run-level definido pelo rstate da linha, dá inicio ao processo, não espera pelo fim deste, e quanto este termina não o relança

PROCESS um comando de shell (sh) que vai ser executado; qualquer sintaxe de shell pode ser usada no resto da linha, inclusivé comentários começados por #

Veja-se o exemplo de ficheiro e repare-se que a configuração tem dois run-level's; um em que a máquina fica em 'single-user' e só funciona com a consola (run-level 0), e outro que permite trabalhar com mais do que um terminal (run-level 1).

Um administrador pode dar ordem ao init para mudar de run-level desde que seja SuperUser, utilizando o comando ***telinit X*** em que X é o novo run-level. A partir desse momento o init passa a trabalhar no novo run-level, lendo e executando as acções correspondentes.

4.2.4. O SCRIPT DE PASSAGEM A MULTI-USER: /ETC/RC

No exemplo de /etc/inittab dado acima existe uma linha (LABEL=rc) a ser executada quando o init passa para run-level 1. Esta linha diz que deve ser chamado o programa /etc/rc, e que o init não deve fazer mais nada enquanto este não terminar. Neste programa, que é normalmente um script de shell, é habitual existirem instrucções de inicialização de impressoras, de sistemas de ficheiros (através do comando mount), etc.

Muitas das configurações habituais não são tão simples quanto a deste exemplo. Pode acontecer que em determinadas máquinas as funções necessárias à passagem a multi-user se distribuam por vários scripts. Um caso habitual é a verificação da coerência dos sistemas de ficheiros (utilizando o comando fsck) feita por um script chamado /etc/bcheckrc, que corre antes do /etc/rc, através da adição de uma linha ao inittab, colocada antes da linha correspondente à do /etc/rc, ou da utilização da palavra-chave ***bootwait*** que faz com que o programa seja executado na altura do boot.

Noutras versões ou configurações não existe apenas um único /etc/rc mas vários ficheiros equivalentes, um para executar à entrada do run-level 0 (/etc/rc0), outro para a entrada no run-level 1 (/etc/rc1), etc.

4.3. SHUTDOWN - COMO DESLIGAR O SISTEMA

Quando se quer desligar um computador que trabalhe com sistema UNIX não se pode simplesmente comutar o interruptor, porque o sistema utiliza um método de acesso aos ficheiros em disco que acelera a sua utilização usando a memória do computador. O sistema mantém partes dos ficheiros na memória e actualiza os verdadeiros ficheiros, que estão colocados em disco, a intervalos de tempo pré-estabelecidos.

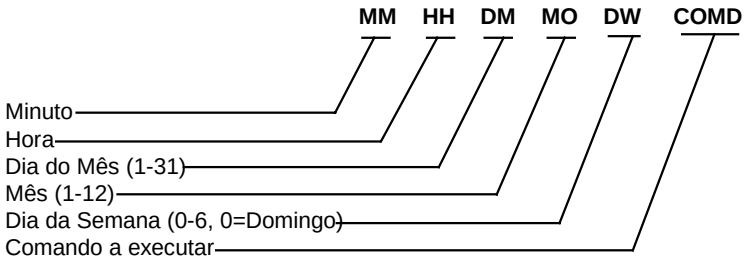
Todos os ficheiros são actualizados através da operação ***sync*** que é automática mas que pode ser invocada pelo operador usando o comando com o mesmo nome. Esta operação é fundamental antes de cada shutdown, já que ao desligar o computador se perde o conteúdo da memória central.

Esta e outras operações necessárias antes de desligar o computador são relizadas por um script que é habitual existir em todos os sistemas UNIX, que se chama ***/etc/shutdown***. O operador chama o programa digitando "/etc/shutdown". O script emite algumas mensagens, avisa normalmente todos os utilizadores que a máquina vai ser desligada, faz sync, desmonta os sistemas de ficheiros, etc. No fim, o programa emite uma mensagem a informar o operador que pode finalmente desligar o computador.

4.4. O UTILITÁRIO CRON

Os comandos at e batch (referidos anteriormente) utilizam um utilitário designado ***cron*** que está sempre a correr (e que pode ser visto com o comando "ps -e"), e que foi lançado (provavelmente) pelo /etc/rc. Este utilitário (contido no ficheiro /etc/cron) acorda uma vez em cada minuto, consulta um ficheiro de controlo para saber se há tarefas que devam ser executadas nessa altura, executa-as se as houver e adormece de novo.

Qualquer utilizador pode criar um ficheiro de controlo do cron através da utilização do comando ***crontab***. A primeira fase é a criação de um ficheiro de texto através do editor. Este ficheiro deve conter uma linha por cada tarefa que se quer efectuada com a indicação das horas e do comando. O ficheiro deve ter o seguinte formato:



Exemplo

```
$ cat cron.jobs

# Crontab file for user: Mary
# Created:                Fri Oct  6 05:42:39 EST
1989

30 17  *   *  1,2,3,4,5 /usr/acct/mary/bin/end-of-
day
  0          0     27          *                 *
/usr/acct/mary/bin/pay-check
  0         1             *         *             0
/usr/acct/mary/bin/check-all

$
```

Figura 29. Exemplo crontab

A análise do exemplo permite-nos observar algumas das possibilidades da utilização do cron pelos utilizadores. As linhas começadas por # são ignoradas e podem ser usadas como comentários. Nas outras linhas encontra-se o formato especificado para os ficheiros de controlo do cron, em que o sinal * significa "todos". Assim, dizer

0 0 27 * * /usr/acct/mary/bin/pay-check

quer dizer "executar /usr/acct/mary/bin/pay-check uma vez todos os dias 27 do mês às zero horas", dia em que a Mary recebe o ordenado mensal. Identicamente, dizer

30 17 * * 1,2,3,4,5 /usr/acct/mary/bin/end-of-day

significa "executar /usr/acct/mary/bin/end-of-day às segundas, terças, quartas, quintas e sextas, às 17H 30M. E dizer

0 1 * * 0 /usr/acct/mary/bin/check-all

significa "executar "/usr/acct/mary/bin/check-all à uma da manhã de todos os domingos.

Depois da criação do ficheiro de texto, o utilizador (neste caso Mary) deverá executar o comando "crontab filename" (no exemplo seria "crontab cron.jobs"). O comando crontab copiará o ficheiro indicado para o directório /usr/spool/cron/crontabs, onde ficará com o nome do utilizador.

Exemplo:

```
$ ls /usr/spool/cron/crontabs
root uucp
$ contab cron.jobs
$ ls /usr/spool/cron/crontabs
mary root uucp
$ crontab -l

# Crontab file for user: Mary
# Created:                Fri Oct  6 05:42:39 EST
1989
30 17   *   *   1,2,3,4,5 /usr/acct/mary/bin/end-of-
day
 0   0 27   *   *           /usr/acct/mary/bin/pay-
check
 0   1   *   *   0          /usr/acct/mary/bin/check-
all

$ crontab -r
$ ls /usr/spool/cron/crontabs
root uucp
$
```

Figura 30. Exemplo da utilização do cron

O utilizador pode rever o conteúdo do ficheiro de controlo do cron que criou através do comando "crontab -l", e pode removê-lo usando o comando "crontab -r", como se pode ver pelo exemplo.

O cron inclui um mecanismo de ***permissões*** e ***segurança*** apoiado em dois ficheiros, que só devem ser acessiveis ao administrador de sistema. O primeiro destes ficheiros chama-se /usr/lib/cron/cron.allow e o segundo chama-se /usr/lib/cron/cron.deny. Ambos contêm nomes de utilizadores, um por linha, que podem ou não podem (respectivamente) utilizar o cron. Se o cron.allow existir, só os utilizadores com o nome nele incluido podem usar o cron. Se não existir e existir o cron.deny, é negada a utilização aos utilizadores neste

incluidos. Não existindo nenhum dos ficheiros de controlo só ao super-user é permitida a utilização.

Exemplo:

```
$ cat /usr/lib/cron/cron.allow
root
sys
adm
uucp
Manel
$
```

Figura 31. Ficheiros de controlo do cron

5. PERIFÉRICOS

Vamos agora analizar a forma de trabalhar com periféricos em sistemas UNIX, nomeadamente terminais e impressoras.

5.1. FICHEIROS ESPECIAIS

No sistema operativo UNIX, existem, não só ficheiros que contêm dados, e que estão normalmente colocados em disco ou disquete, mas também ficheiros de outros tipos que representam terminais, impressoras, discos, streaming tapes, drives de disquetes, etc. Estes ficheiros são chamados ***ficheiros especiais*** e podem ser distinguidos através do comando "ls -l".

```
$ ls -l `cat /tmp/jk `
drwxrwxr-x  2 bin                1616 May  2 07:19
/bin
cr--r-----   1 sys           1,    0 Dec   1   1986
/dev/mem
crw-rw-rw-   1 root          1,    2 Oct   6  06:34
/dev/null
crw-rw----   1 lp            3,    0 Oct   3  13:11
/dev/pp00
cr--r-----   1 root         30,    0 Feb  20   1989
/dev/rdsk/60s0
crw-rw-rw-   2 root         33,    3 Oct   5  15:05
/dev/rtp
br--r-----   2 root         30,   15 Feb  20   1989
/dev/swap
crw--w--w-   5 root          2,    0 Oct   5  14:43
/dev/tty00
crw--w--w-   1 dge           2,    1 Oct   6  08:05
/dev/tty01
crw--w--w-   3 Manel         2,    2 Oct   6  08:05
/dev/tty02
crw--w--w-   1 Fernando      2,    3 Oct   6  08:05
/dev/tty03
```

```
prw-------   1 root                0 Oct   6 07:16
/usr/lib/cron/FIFO
-rw-rw-rw-  1 root          865 Feb  1  1989 /xsum
$
```

Figura 32. Alguns ficheiros especiais

5.1.1. TIPOS DE FICHEIROS ESPECIAIS

No campo das permissões, o primeiro caracter indica o tipo de ficheiro:

'-' ficheiro regular, contendo dados diversos, que vão desde texto em ascii a dados de aplicações, ou programas.

'd' directório - um ficheiro quase normal, com representação em disco, que contém informações sobre outros ficheiros.

'c' ficheiro especial de caracteres - um "nó" de ligação entre um periférico de leitura sequencial (um terminal, por exemplo) e o sistema de ficheiros; ao escrever neste ficheiro o Kernel reconhece que não tem de escrever no disco mas enviar a mensagem a um dispositivo (mensagem esta que pode ser enviada, por exemplo, através de uma linha RS-232).

'b' ficheiro especial de blocos - um nó de ligação a dispositivos tais como discos, disquetes, etc, com os quais se comunica por blocos; no caso de um destes ficheiros representar um disco o super-user pode facilmente destruir um sistema de ficheiros escrevendo directamente sobre o disco (que faria perder a informação da localização dos dados, do espaço livre, dos tamanhos dos ficheiros, etc).

'p' ficheiro especial tipo FIFO (no exemplo dado acima "FIFO" também é o nome do ficheiro), First In First Out, um dispositivo de comunicação inter-processos.

5.1.2. MAJOR E MINOR NUMBER E O COMANDO MKNOD

Nos ficheiros tipo "b" ou tipo "c", o comando ls substitui o campo do tamanho do ficheiro (que não teria significado) por dois campos designados ***major number*** e ***minor number*** que servem de referência ao Kernel para indicar qual o dispositivo físico a aceder. Vejamos: o ficheiro /dev/tty01 do exemplo acima tem como major number o numero 2 e como minor number o número 1, e isto tem muita informação para o Kernel - indica que o ficheiro corresponde à porta 1 do controlador de portas RS-232. Claro que isto não é assim tão directo. O que acontece é que ao reconhecer o código 2 no major number o Kernel entra nas rotinas de tratamento de comunicação série e ao reconhecer o código 1 no minor number essas rotinas ficam a saber que é a porta 1.

Um super-user pode criar um ficheiro especial através da utilização do comando ***mknod*** e indicando o tipo e nome do ficheiro a criar:

```
mknod name b/c major minor - especiais de bloco ou
caracteres
mknod name p               - especiais FIFO
```

Figura 33. Utilização do comando mknod

O utilizador tem que ter cuidado já que o major e minor numbers têm que fazer parte dos aceites pelo Kernel. Se não fizerem, qualquer tentativa de acesso a esse ficheiro darão origem a um erro que será qualquer coisa como "no such device or adress". Novos periféricos só podem ser incorporados numa máquina UNIX se o Kernel estiver preparado para eles. Caso contrário será preciso alterar o Kernel e recompilá-lo com as alterações.

5.2. A CONFIGURAÇÃO DE TERMINAIS

5.2.1. O PROGRAMA GETTY

Já vimos que os terminais estão representados na estructura de ficheiros do UNIX como ficheiros (embora especiais). Assim sendo, a interacção do sistema com os terminais pode ser feita como com qualquer ficheiro. Como funciona um terminal? Os caracteres digitados no teclado são recebidos por quem estiver a ler o ficheiro associado, e os que são escritos no ficheiro são apresentados no ecrãn do terminal.

O init, referido anteriormente lê o ficheiro inittab e encontra muitas vezes linhas como esta:

t02:1:respawn:/etc/getty tty02 d vt100

Isto significa que deve lançar um processo (sempre que esteja em run-level 1) e relançá-lo se este morrer. Este processo deve incluir o programa getty, que receberá como argumentos "tty02 d vt100".

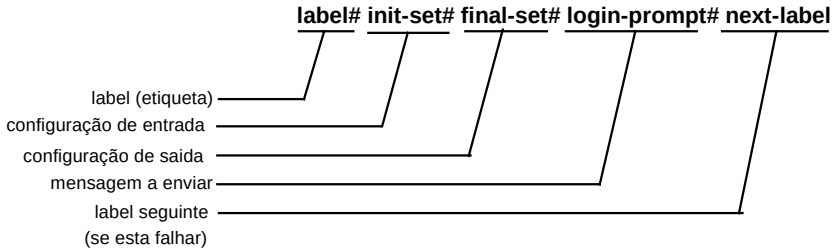
tty02 é o nome do ficheiro associado ao terminal, e está colocado no directório /dev. O processo do getty vai abrir este ficheiro e vai escrever e ler mensagens dele.

5.2.2. O FICHEIRO GETTYDEFS E O COMANDO STTY

O segundo argumento serve ao getty para descobrir como configurar a linha de comunicação do terminal. Como muitos dos leitores devem saber, as linhas de comunicação série têm vários parâmetros que podem ser alterados, como a velocidade de transmissão de dados (baud rate), a paridade, o número de bits de uma palavra, e o protocolo. O computador tem geralmente portas configuráveis por software, e quem se encarrega de as configurar é o getty.

Como é que o getty sabe os parâmetros da comunicação? Quem indica isto é o seu segundo argumento, neste caso a letra 'd', que é uma label referente a um ficheiro - o /etc/gettydefs, onde estão descritas várias combinações possiveis dos parâmetros. Cada uma destas combinações está etiquetada, de forma a que quando se pretende ligar outro terminal não é preciso indicar todos os parâmetros, bastando escolher uma combinação existente.

A forma das combinações no ficheiro gettydefs é a seguinte:



Exemplo:

```
d# B9600 ECHO PARENB # B9600 PARENB TAB3 ECHO
#login: #f
```

Figura 34. Formato do /etc/gettydefs

A seguir descreve-se sumariamente o significado dos campos:

Label	a string com a qual o getty vai comparar o seu segundo argumento, não precisa ter nenhum significado especial mas pode tê-lo.
Init-set	a configuração da linha que o getty deve ligar quando arranca (ver TERMIO(7)).
Final-set	a configuração da linha que o getty deve ligar imediatamente antes de chamar o programa 'login'.
Login-prompt	a mensagem que o getty envia para o terminal.
Next-label	a label que deve ser usada se a comunicação com o terminal falhar, para nova tentativa.

O getty começa por configurar a linha de comunicações usando a informação que encontra no /etc/gettydefs, relativa à label que vem no seu segundo argumento. Seguidamente apaga o ecrãn (com base na informação do tipo de terminal), envia uma mensagem com o conteúdo do ficheiro /etc/issue e com a login-prompt, e aguarda resposta do utilizador. Este pode indicar ao getty que a linha está mal configurada através do caracter 'break' (que na realidade não é

um caracter), e o getty nesse caso utilizaria o valor next-label para tentar outra configuração, e começaria de novo.

No caso de a configuração estar correcta, o utilizador deverá indicar um código de entrada (normalmente o seu nome), e o getty, após configurar a linha segundo a indicação final-set, chamará o programa ***login*** que dará inicio à sequência de entrada do utilizador.

Existe ainda outra forma de configurar os parâmetros de uma linha de comunicações série, que é a utilização do comando ***stty***.

```
stty < /dev/tty03
```

permite saber como está configurada a linha correspondente ao ficheiro /dev/tty03. A opção -a do stty faz com que este exponha todos os parâmetros da linha:

EXEMPLO: Utilização do stty para consultar a configuração de uma linha

```
$ stty -a < /dev/tty03
speed 9600 baud; line = 0; intr = DEL; quit = ^|;
erase = ^h; kill = @; eof = ^d; eol = ^`; swtch = ^`
parenb -parodd cs7 -cstopb -hupcl cread clocal -hdx -loblk
-ignbrk brkint ignpar -parmrk -inpck istrip -inlcr -igncr
icrnl -iuclc ixon -ixany -ixoff -irts -err_bel -extend
isig icanon -xcase echo echoe echok -echonl -noflsh
opost -olcuc onlcr -ocrnl -onocr -onlret -ofill -ofdel tab3
$
```

Figura 35. Comando stty -a

O comando stty "trabalha" o ficheiro associado ao seu standard input. Se não se redireccionar o standard input pode consultar-se ou alterar-se o estado da linha a que o nosso terminal está ligado. A forma de alterar a configuração é dar ao stty argumentos:

```
stty erase ^L
```

é uma maneira de indicar que em vez do ^H a tecla de correcção passa a ser o ^L (que em muitos terminais é o código da seta para a esquerda);

```
stty 19200
```

é a forma de indicar que a nossa linha deve passar a trabalhar a um baud rate de 19200 BPS, em vez dos habituais 9600 BPS. Claro que nesse momento a configuração própria do terminal teria que ser mudada, já que para que haja comunicação ambas as máquinas (computador e terminal) têm que falar a mesma língua. Ao terminal também seria necessário dizer para começar a falar a 19200 BPS, o que seria feito alterando os dip-switches ou o setup-menu.

No fundo, quando o getty configura a linha, está de facto a fazer o mesmo que o stty. Por dentro, os mecanismos são os mesmos.

5.2.3. AS BASES DE DADOS TERMCAP E TERMINFO

Para apagar o ecrãn, o getty baseia-se na informação contida numa das bases de dados sobre terminais existentes no UNIX. O terceiro argumento que o getty recebe do init é o tipo de terminal. No exemplo dado, "vt100" corresponde ao famoso DEC VT100. Uma das coisas que o getty faz antes de chamar o login é afectar a variável de ambiente TERM com o valor do seu terceiro parâmetro. Se este não existir TERM será (provávelmente) afectada com o valor "unknown".

A existência destas bases de dados permite que os programas saibam como mover o cursor, apagar o ecrãn, reconhecer as teclas de função, as setas, etc, sem que os programadores tenham que se preocupar em saber com que terminais os programas vão trabalhar. Permitem até que os programas funcionem com terminais desconhecidos na altura, desde que o administrador crie uma entrada para o terminal nas bases de dados, com a descrição do funcionamento destes.

A base de dados termcap reside num único ficheiro de texto - /etc/termcap - e consiste numa sequência de registos referentes a terminais.

ENTRADA DO TERMCAP RELATIVA AO DEC VT100

```
d1|vt100|DEC    vt100:\
co#80:li#24:cl=70\E[;H\\E[2J:bs:cm=\E[%i%2;%2H:\
ti=\E[0;0H:\
kh=Â:kr=\E[C:nd=\E[C:kl=\E[D:bc=\E[D:ku=\E[A:\
up=\E[A:kd=\E[B:do=\E[B:\
```

```
ce=\E[0K:cd=\E[0J:dl=\E2K:\
so=\E[7m:se=\E[0m:sg#0:us=\E[4m:ue=\E[0m:ul:ug#0:\
MP=:MR=\E[0;0H:HM=\E\\+O\\+P:\
PU=\E\\+1:PD=\E\\+2:PL=\E\\+3:PR=\E\\+4:NU=\E\\+5:\
CW=\E\\+6:\
EN=\E\\+7:WL=\E\\+8:WR=\E\\+9:CL=\E\\+0:\
CR=\E\\+-:DL=\E\\+=:\
CN=:CF=:RS=\E[0m:NM=\E[0m:NR=\E[7m:NB=\E[5m:\
NS=:AB=:AR=:AS=:OV#0:\
k0=:k1=:k2=:k3=:k4=:k5=:k6=:k7=:k8=:k9=:\
kA=\Eq:kB=\Ew:kC=\Ee:kD=\Er:kE=\Et:kF=\Ey:kG=\Eu:\
kH=\Ei:kI=\Eo:kJ=\Ep:\
kK=\Ea:kL=\Es:kM=\Ed:kN=\Ef:kO=^x:kP=:kQ=:kR=:\
kS=:kT=:\
kU=\E1:kV=\E2:kW=\E3:kX=\E4:kY=\E5:kZ=\E6:lA=\E7:\
lB=\E8:lC=\E9:lD=\E0:\
lE=\EOS:lF=\EOP:lG=\EOQ:lH=\EOR:lI=^L:lJ=\E[A:\
lK=\010:lL=^M:lM=^I:lN=^B:lO=:\
lP=^V:lQ=^K:lR=^N:lS=^S:lT=^W:lU=Ê:lV=^R:lW=^D:\
lX=^Q:lY=^C:\
lZ=\177:za=\E[m:
```

Figura 36. Descrição de um terminal no /etc/termcap

Para acelerar a leitura da informação referente é hábito não ter a descrição de todos os terminais da base de dados no ficheiro /etc/termcap, mas manter os não usados num outro ficheiro chamado /etc/btermcap.

A base de dados ***terminfo*** foi criada mais recentemente e está aos poucos a substituir o termcap, já que se provou ser mais potente e de mais rápido acesso. Está localizada no directório /usr/lib/terminfo, e as informações estão dispersas em sub-directórios e ficheiros mantidos por um utilitário de nome ***tic***. Este programa lê um ficheiro ascii com a descrição das capacidades do terminal, compila-o e organiza a informação na estructura da base de dados. Eis um exemplo deste tipo de ficheiro, que pode ser tratado pelo tic, mais uma vez descrevendo o terminal VT 100 da DEC:

```
$ cat /usr/lib/terminfo/vt100.ti

vt100|vt100-am|dec vt100,
   cr=^M, cud1=^J, ind=^J, bel=^G, cols#80, lines#24, it#8,
   clear=\E[H\E[2J$<50>, cub1=^H, am, cup=\E[%i%p1%d;%p2%dH$<5>,
   cuf1=\E[C$<2>, cuu1=\E[A$<2>, el=\E[K$<3>, ed=\E[J$<50>,
   cud=\E[%p1%dB, cuu=\E[%p1%dA, cub=\E[%p1%dD, cuf=\E[%p1%dC,
   smso=\E[7m$<2>, rmso=\E[m$<2>, smul=\E[4m$<2>, rmul=\E[m$<2>,
   bold=\E[1m$<2>, rev=\E[7m$<2>, blink=\E[5m$<2>, sgr0=\E[m$<2>,
   sgr=\E[%?%p1%t;7%;%?%p2%t;4%;%?%p3%t;7%;%?%p4%t;5%;%?%p6%t;1%;m,
   rs2=\E>\E[?3l\E[?4l\E[?5l\E[?7h\E[?8h,
smkx=\E[?1h\E=,rmkx=\E[?1l\E>,
   tbc=\E[3g, hts=\EH, home=\E[H,
   kcuu1=\EOA, kcud1=\EOB, kcuf1=\EOC, kcub1=\EOD, kbs=^H,
```

```
    kf1=\EOP, kf2=\EOQ, kf3=\EOR, kf4=\EOS, ht=^I, ri=\EM$<5>,
    vt#3, xenl, xon, sc=\E7, rc=\E8, csr=\E[%i%p1%d;%p2%dr,
$
```

Figura 37. Descrição de terminal no terminfo

Entre os programas standard do UNIX que usam as bases de dados encontram-se o vi, o more, o getty e o tput. Estes programas utilizam a variável de ambiente TERM para procurar informação de como tratar os terminais numa das bases de dados.

A base de dados termcap tem tendência a desaparecer mas ainda existem alguns programas que a utilizam, nomeadamente aplicações comerciais já com alguma idade.

Resta ainda abordar a existência de duas variáveis de ambiente TERMCAP e TERMINFO, que permitem usar versões destas bases de dados colocadas noutros locais. Se um utilizador assim quizer, pode criar o seu próprio termcap e executar o comando:

```
$ setenv TERMCAP /usr/acct/mary/mytermcap
```

Figura 38. Utilização da variável TERMCAP

em 'csh' ou:

```
$ TERMCAP=/usr/acct/mary/mytermcap
$ export TERMCAP
```

Figura 39. Utilização da variável TERMCAP

se estiver a trabalhar em 'sh' (shell). Do mesmo modo, pode criar outra base de dados terminfo, afectando a variável TERMINFO com o directório onde esta fica colocada.

5.3. O SPOOLER LP E A CONFIGURAÇÃO DE IMPRESSORAS

O leitor que chega a esta fase já conhece a forma de pedir a impressão de um determinado ficheiro de texto através do comando ***lp*** e como cancelar as impressões através do comando ***cancel*** e ainda como verificar o estado da sua lista de espera com o comando ***lpstat***. Vamos agora analizar mais profundamente o funcionamento do spooler LP, um conjunto de programas e ficheiros que permitem utilizar mais racionalmente os recursos de impressão de determinada instalação.

O Spooler Lp é tão genérico e poderoso que não é costume encontrar mais nenhuma ferramenta de impressão nos sistemas UNIX. Devido à filosofia de contruir ferramentas que fazem trabalhos especifícos, a formatação, paginação, etc, são levadas a cabo por aplicações especiais e filtros em vez de o ser pelo spooler. Isto permite ao Spooler Lp especializar-se em gestão de listas de espera e do hardware.

5.3.1. FUNCIONAMENTO

Um utilizador envia um texto para uma impressora com o seguinte comando:

```
lp filename
```

Figura 40. Imprimir um ficheiro

O que acontece a seguir deixa de ser da responsabilidade dele e passa para as "mãos" de um processo conhecido como ***spooler***. Nenhuma impressora é tão rápida como nós gostariamos que fosse (e tão silenciosa, já agora), e estar à espera que a impressão acabe sem poder continuar a trabalhar é um inconveniente grande. Aqui entra o spooler. Cada pedido de impressão entra e é guardado numa lista de espera e o texto respectivo é guardado também. Isto é responsabilidade do comando 'lp'.

Deve já ser sabido que a consulta à lista de espera pode ser feita com o comando 'lpstat' e a remoção de um pedido da lista de espera (quer esteja ou

não a ser já impresso) pode ser feita com o comando 'cancel'. Este comando pode também ser utilizado pelo Super-User para cancelar impressões de outros utilizadores, sendo o utilizador avisado que a sua impressão foi cancelada via 'mail'.

O utilizador pode requerer que o pedido seja impresso numa determinada impressora, em vez do destino principal (default destination), através da adição da flag -d (destino):

```
lp -d printername filename
```

Figura 41. Imprimir um ficheiro indicando a impressora

No LP Spooler existe ainda o conceito de ***classe de impressoras***, que permite ao utilizador usar a impressora que na altura esteja com menos pedidos, de entre um conjunto (classe) de impressoras designado pelo administrador do sistema. Isto é feito usando também a opção '-d' do comando lp, em que o nome da impressora é substituido pelo nome da classe.

A aceitação ou não do pedido por parte do 'lp' pode ser controlada com dois comandos: ***accept/reject*** que são usados do seguinte modo:

/usr/lib/accept destinos*
/usr/lib/reject [-r razão] destinos*

* por destinos entendem-se impressoras ou classes.

Exemplos:

```
/usr/lib/reject -r"disco cheio" ibm_prop laser
/usr/lib/accept ibm_prop laser
```

Figura 42. Accept e reject

No exemplo dado para a utilização do reject imagina-se que há um conjunto de utilizadores a fazerem pedidos de impressão que vão aos poucos enchendo o disco. O administrador decide então fazer com que os pedidos não sejam aceites durante um certo período de tempo, enquanto as impressões se vão fazendo, e as cópias temporárias que o spooler mantém vão sendo apagadas do

disco. Nesta situação, quando um utilizador pede para imprimir determinado ficheiro depara com a seguinte situação:

```
$ lp -dlaser myfile
lp: can't accept requests for destination "laser"
- disco cheio
$
```

Figura 43. Resultado da utilização do reject

Quem se encarrega de despejar a lista de espera e de imprimir os ficheiros é um processo que está constantemente a correr (que é normalmente lançado pelo /etc/rc) chamado ***lpsched*** (nome completo: /usr/lib/lpsched). A existência do ficheiro /usr/spool/lp/SCHEDLOCK indica que o lpsched está (ou deveria estar) a correr. Este ficheiro serve de indicador ao spooler do facto de estar a correr um lpsched, evitando a existência de dois processos destes na máquina, o que seria bastante confuso e perigoso. É possivel parar o processo lpsched através da utilização do comando ***lpshut*** (comando completo: /usr/lib/lpshut) que envia um sinal ao processo lpsched e remove o SCHEDLOCK. O spooler pode ser reactivado com a invocação do comando '/usr/lib/lpsched'.

Imagine-se agora que o papel de uma impressora encravava. Seria de toda a conveniência parar as impressões nessa impressora ***no momento*** para seguidamente as recomeçar. Mais ainda: não existiria razão para que se deixasse de aceitar pedidos para aquela impressora. Para isto, o mecanismo reject/accept não é o mais conveniente. O que haveria a fazer seria utilizar o comando ***disable*** que faz com que o spooler (na "pessoa" do lpsched) deixe de enviar impressões para aquela impressora.

Quando a impressora estivesse pronta a recomeçar as impressões bastaria fazer ***enable*** `a impressora e o lpsched recomeçaria a enviar os textos, começando por aquele que tinha sido interrompido.

disable [-rrazão] printer
enable printer

Exemplos:

```
$ disable -r"papel outra vez encravado" ibm_prop
$ enable ibm_prop
```

Figura 44. Utilização do disable e do enable

5.3.2. CONCEITOS FUNDAMENTAIS NO LP SPOOLER

Dispositivo:	(Device) Aparelho físico, associado a um ficheiro no directório /dev.
Impressora:	(Printer) Aparelho virtual, associado a um dispositivo e a um programa (interface), e que pode eventualmente pertencer a uma classe.
Classe:	(Class) Um conjunto de impressoras do mesmo tipo, que compartilham a mesma fila de espera.
Destino:	(Destination) Referência a uma classe ou a uma impressora.
Interface:	Programa que trata o texto que chega uma impressora antes de o enviar para o dispositivo, e faz a gestão de algumas opções especiais do comando lp (ver adiante).
Modelo:	(Model) Programa de interface inactivo que pode ser copiado para servir de interface a uma impressora residente em /usr/spool/lp/model.
Destino Principal:	(Default Destination) Impressora ou classe para a qual são enviados os pedidos se não fôr usada a opção '-d' do comando 'lp'.

5.3.3. GESTÃO DE IMPRESSORAS

Uma impressora é ligada ao computador tal qual um terminal, e se fôr de comunicação série a linha também pode ser configurada através do getty ou do stty. Se fôr uma impressora de comunicação paralela, não é geralmente

necessário configurar a porta (nota: muitas vezes "porta" e "linha" podem ser usadas neste texto como sinónimos).

Imediatamente após a ligação fisica da impressora ao computador, é aconcelhável testar o funcionamento do hardware, enviando texto directamente para a porta (imaginemos que a impressora estava ligada à porta 02):

```
$ ls -l > /dev/tty02                    (a impressora escrevia caracteres
esquisitos)
$ stty 9600 cs7 parenb < /dev/tty02   (alterar a configuração da porta)
$ ls -l > /dev/tty02              (a impressão realizava-se correctamente)
```

Figura 45. Configuração manual de uma porta série

Nesta altura, o hardware funciona, pelo que a fase seguinte é adicionar a impressora sistema de spooling. As tarefas de administração do LP Spooler são realizadas através do ***lpadmin*** (comando completo: /usr/lib/lpadmin). Este comando é usado no LP Spooler para descrever impressoras, classes e dispositivos; adiciona e remove destinos, altera membros de classes, muda os dispositivos associados às impressoras, muda os programas de interface das impressoras e pode designar como principal um qualquer destino do sistema.

/usr/lib/lpadmin -pprinter [options]	**- comandos relativos a impressoras (descritos abaixo)**
/usr/lib/lpadmin -xdest	**- remover um destino (seja classe ou impressora)**
/usr/lib/lpadmin -ddest	**- designar 'dest' como destino principal do LP Spooler**

Figura 46. Utilização do comando lpadmin

O lpadmin tem obrigatóriamente que ser usado numa das três formas acima descritas. Uma das três opções ('-p', '-x' ou '-d') tem que ser usada.

```
UTILIZAÇÃO DO LPADMIN REFERENTE ÀS IMPRESSORAS
==============================================
/usr/lib/lpadmin -pXX -cCC                          - adicionar a
impressora 'XX à classe 'CC'
/usr/lib/lpadmin -pXX -rCC                           - remover a
impressora 'XX' da classe 'CC'
/usr/lib/lpadmin -pXX -vDD                          - associar a
impressora 'XX' ao dispositivo 'DD'
/usr/lib/lpadmin -pXX -eYY              - copiar o programa
interface da impressora existente 'YY' para servir de interface à
impressora 'XX'
/usr/lib/lpadmin -pXX -ifile               - copiar o programa
contido no ficheiro 'file' para servir de interface à impressora
'XX' ('file' é começado por '/')
/usr/lib/lpadmin -pXX -mmodel     - copiar o programa contido
noficheiro 'file' do directório '/usr/spool/lp/model' para servir
de interface à impressora 'XX'
```

Figura 47. Opções do lpadmin

O estado do LP Spooler pode ser consultado em qualquer altura através do comando ***lpstat*** e da utilização da flag '-t'. Esta opção dá um output parecido com o seguinte:

```
$ lpstat -t
scheduler is not running
system default destination: lp0
members of class IBM:
        lp0
        ibm_prop
device for lp0: /dev/centronics
device for ibm_prop: /dev/tty02
device for laser: /dev/tty06
lp0 accepting requests since Oct  7 12:49
ibm_prop accepting requests since Oct  7 15:26
laser accepting requests since Oct  7 15:26
IBM accepting requests since Oct  7 15:31
printer lp0 is idle.  enabled since Oct  7 15:26
printer ibm_prop disabled since Oct  7 15:26 -
      papel encravado
printer laser is idle.  enabled since Oct  7 15:25
$
```

Figura 48. Comando lpstat

6. BIBLIOGRAFIA

Este manual não chega. O curso correspondente foi intensivo e de curta duração. Se o leitor quer aprofundar os conhecimento que adquiriu até agora sobre o UNIX e a sua administração propomos-lhe que consulte os seguintes livros:

"UNIX FOR SUPER-USERS"
Eric Foxley

Addison-Wesley Publishing Company
(International Computer Science Series)

"UNIX SYSTEM SECURITY"
P. H. Wood & Stephen G. Kochan

Hayden Books
(UNIX System Library)

"THE DESIGN OF THE UNIX OPERATING SYSTEM"
Maurice J. Bach

Prentice-Hall

"THE UNIX SYSTEM GUIDEBOOK (2ND EDITION)"
Peter P. Silvester

Springer-Verlag .DE

"UNIX SYSTEM ADMINISTRATION"
David Fiedler & Bruce H. Hunter

Hayden Books
(UNIX System Library)