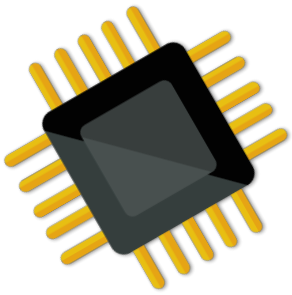
Laboratório 1.5: Processos, Daemons, Serviços, Inicialização do Sistema

Introdução aos Processos



Um ***processo*** nada mais é do que um programa em execução. Em um sistema multitarefa como o Linux, centenas de processos estão rodando ao mesmo tempo e usando os recursos do processador. O tempo de CPU é dividido entre os processos pelo sistema operacional através do seu escalonador de processos.

Todo processo possui um número único no sistema, conhecido como **PID** (*Process Identifier*). O PID, como o nome indica, é usado para identificar, unicamente, um processo. Este número é parâmetro para vários comandos relacionados a processos que veremos neste laboratório.

Além disso, todo processo possui também um pai, mais conhecido como **PPID** (*Parent Process Identifier*). Por exemplo, quando você está no terminal usando o interpretador de comandos `bash` e pede para executar um comando/processo, o pai desse novo processo será o próprio `bash`.

Um processo especial, chamado **`init`**, possui PID 1 e ele é o pai, direto ou indireto, de todos os outros processos. Ele é o único processo que não possui pai (`PPID`). O motivo disso é que o `init` é o primeiro processo executado na inicialização do Linux e ele é chamado/iniciado diretamente pelo Kernel. O `init` é responsável por iniciar os outros processos do sistema.

Alguns processos, como o próprio `init`, são utilitários e proveem um determinado serviço ou funcionalidade para outros processos e, devido a isso, estão sempre em execução. Estes processos são conhecidos como ***daemons***. Os *daemons* são geralmente iniciados no boot do sistema e só terminam quando o sistema é desligado.

Nesta parte do laboratório, exploraremos os comandos relacionados à visualização e gerenciamento de processos em um sistema Linux.

1. Listando os Processos em Execução

No Linux, o comando para listar processos em execução é o `ps`:

```
$ ps
```

```
  PID TTY          TIME CMD
    9 pts/0    00:00:00 bash
   71 pts/0    00:00:00 ps
```

Entretanto, como você deve ter visto no comando anterior, se você executar o `ps` sem opções, ele só mostra os processos relacionados ao seu usuário e ao seu terminal atual. Por exemplo, execute o comando `cat` e deixe-o rodando em *background* conforme passado no primeiro comando abaixo e, em seguida, execute o `ps` novamente:

```
$ cat &                          # Executa o cat sem parâmetros e em background (não
termina)
$ ps                             # Lista novamente os processos
```

```
  PID TTY          TIME CMD
    9 pts/0    00:00:00 bash
   72 pts/0    00:00:00 cat
   73 pts/0    00:00:00 ps
```

O comando `cat` executado sem parâmetros basicamente repete (na saída padrão) tudo que é digitado no teclado (entrada padrão). Mas como no comando acima nós executamos ele sem ficar preso ao terminal (em *background*, usando o caractere `&`), ele ficará para sempre esperando uma entrada que não virá. Fique tranquilo que mataremos esse processo mais adiante. Mas, seguindo para o segundo comando (o `ps`), note como o processo `cat` aparece, agora, na lista de processos em execução.

Ainda na saída do comando anterior, note a coluna `PID` que mostra os *ids* dos processos. A coluna `TTY` (*TeleTYpewriter*) indica o número do terminal atual. Se você abrir um novo terminal e executar o comando, verá que os valores de coluna mudam. Já a coluna `TIME` mostra o tempo de uso da CPU. Note que o tempo efetivo de uso da CPU por um processo é muito menor do que o tempo que ele está executando.

Observando os processos listados, note que o `bash`, processo que você está usando neste momento como interpretador dos seus comandos, aparece. Por fim, note como o próprio `ps` aparece na lista de processos pois, enquanto ele está acessando os dados do kernel, ele também é um dos processos em execução.

Para ver mais alguns detalhes dos processos, use a opção "`-f`":

```
$ ps -f                          # Lista mais detalhes (full format)
```

```
UID        PID  PPID  C STIME TTY          TIME CMD
marcos       9     8  0 18:53 pts/0    00:00:00 -bash
marcos      72     9  0 19:10 pts/0    00:00:00 cat
marcos      74     9  0 19:10 pts/0    00:00:00 ps -f
```

Note que agora a coluna `UID` (*user id*), que mostra o login do usuário que iniciou os processos. Tem-se também a coluna `PPID` que mostra o pai dos processos. Note como o `PPID` dos processos `cat` e `ps` é o `PID` do processo `bash`, pois foi o `bash` que interpretou seu comando e efetivamente pediu para iniciar os programas.

Listando Todos os Processos

Vamos agora pedir para mostrar todos os processos:

```
$ ps -ef                         # Lista todos os processos com detalhes
```

```
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 18:53 ?        00:00:00 /init
root         7     1  0 18:53 ?        00:00:00 /init
root         8     7  0 18:53 ?        00:00:00 /init
marcos       9     8  0 18:53 pts/0    00:00:00 -bash
--- Continua com vários outros processos ---
```

Tire alguns minutos para ver com cuidados as centenas de processos rodando. Note como é uma quantidade grande de processos, executados por usuários diferentes. Olhando para as primeiras linhas, note como o `init` é o primeiro processo. Ele possui `PID` 1 e não possui PPID (indicado por zero). Você pode notar também alguns processos especiais conhecidos como *kernel threads*. Eles são identificados pelos colchetes, por exemplo, `[kthreadd]`. Estes processos são iniciados e gerenciados diretamente pelo kernel, e não pelo `init`. Fora eles, todos os outros processos são inicializados diretamente ou indiretamente a partir do `init`.

Para ver a quantidade de processos rodando, execute o comando abaixo:

```
$ ps -ef | wc -l                 # Executa o ps, conta as linhas
```

```
200
```

Top: Processos Ordenados

Outro comando muito útil para ver o que está acontecendo no seu sistema é o `top`:

```
$ top                            # Pressione Q para sair
```

O `top` mostra os processos que estão consumindo mais recursos. Por padrão, os processos são ordenados por consumo atual de CPU. Note que ele atualiza a cada 3 segundos. Além dos processos, na parte de cima, é possível ter um resumo dos recursos do sistema. Por exemplo, na primeira linha é possível ver o *uptime*, a quantidade de usuários logados no sistema, e o *loadavg*. Pressione `Q` para sair.

Para que o `top` ordene por consumo de memória, por exemplo, use a opção `-o` identificando a coluna "%MEM":

```
$ top -o %MEM                          # Pressione Q para sair
```

Você pode pedir para ordenar por qualquer uma das colunas (`PID`, `USER`, etc).

2. Matando um Processo

No Linux, "matar" um processo significa terminá-lo forçadamente. Apenas o usuário *root* pode matar qualquer processo. Os usuários normais, só podem matar os processos que eles iniciaram. Para matar um processo, usamos o comando `kill`. Para isso, precisamos primeiro descobrir o seu PID. Vamos identificar o PID do processo `cat`, que iniciamos anteriormente:

```
$ ps | grep cat | xargs | cut -d" " -f1
```

```
72
```

Se o comando `cat` não estiver rodando, inicie-o novamente executando `cat &`. O comando `xargs` é usado apenas para remover espaços extras entre as palavras.

Comando `kill`

Uma vez identificado o PID, usamos o `kill` para matá-lo:

```
$ kill 72
$ ps | grep cat                        # Verifica se o processo realmente foi terminado
```

```
72 pts/0    00:00:00 cat
```

Como você pode ter visto, muito provavelmente o processo não foi realmente terminado! O motivo disso é que o comando `kill`, por padrão, é educado. Ao invés de realmente matar o processo, ele envia um sinal ao mesmo solicitando que ele termine. Muitos programas ouvem esse sinal e realmente terminam, aproveitando para "limpar a casa" antes, fechando arquivos abertos e etc. Entretanto, outros programas ignoram o pedido, como foi o caso do `cat`.

Para forçar o término do programa e realmente matá-lo, usamos a opção `-9` do `kill`:

```
$ kill -9 72
$ ps | grep cat                        # Verifica se o processo realmente foi terminado
```

```
[1]+  Killed                  cat
```

Note como, agora, o processo `cat` foi realmente finalizado.

Comando `killall`

Outra forma de matar programas é usando o comando `killall`. Vamos, primeiro, iniciar um processo novo do `cat` para o usarmos como sacrifício novamente. Mas, melhor do que um, vamos iniciar três processos!

```
$ cat &
$ cat &
```

```
$ cat &
$ ps | grep cat
```

```
89 pts/0    00:00:00 cat
   90 pts/0    00:00:00 cat
   91 pts/0    00:00:00 cat
```

Usaremos o comando `killall` para matá-lo. Este comando, diferentemente do `kill`, não precisa do PID. Ele mata os processos pelo nome mesmo. Como, diferentemente do PID, o nome do processo não é único, o `killall` irá matar todos os processos que possuem o nome especificado:

```
$ killall -9 cat
```

```
[1]   Killed                  cat
[2]-  Killed                  cat
[3]+  Killed                  cat
```

Por termos menos controle dos processos que são terminados, o uso do comando `kill` é mais recomendado, principalmente se você estiver usando o usuário *root*.

3. Verificando a Memória Livre no Sistema

Enquanto arquivos consomem espaço em disco, processos consomem espaço em memória. Vimos que o comando `top` é capaz de ordenar os processos por consumo de memória e que esse comando também mostra o total de memória livre no sistema.

Outro comando mais prático para vermos rapidamente o consumo de memória do sistema é o `free`:

```
$ free
```

```
               total        used        free      shared  buff/cache   available
Mem:       12179960      251408    11796248         280      132304    11714064
Swap:       3145728           0     3145728
```

A linha `Mem` mostra os dados da memória principal, enquanto a linha `Swap` mostra os dados da memória virtual. A memória virtual usa o disco como memória, quando a memória principal acaba, o que deixa o sistema mais lento.

Novamente, o comando anterior mostra os dados em *bytes*. Para ter uma saída mais amigável, use a opção "`-h`":

```
$ free -h
```

```
               total        used        free      shared  buff/cache   available
Mem:            11Gi       243Mi        11Gi       0.0Ki       129Mi        11Gi
Swap:          3.0Gi          0B       3.0Gi
```

4. Daemons do Linux

Conforme mencionado, *daemons* são processos utilitários iniciados no *boot* do sistema e terminados apenas quando o computador é desligado. Tais processos proveem ao sistema e aos processos uma série de serviços. O Linux possui vários processos deste tipo. A seguir, será listado apenas alguns deles:

```
$  ps -ef | grep -v grep | grep "systemd-udevd\|systemd-resolved\|systemd-
timesyncd\|systemd-logind\|cron\|rsyslogd\|atd\|thermald\|acpid\|cupsd\|Xorg"
```

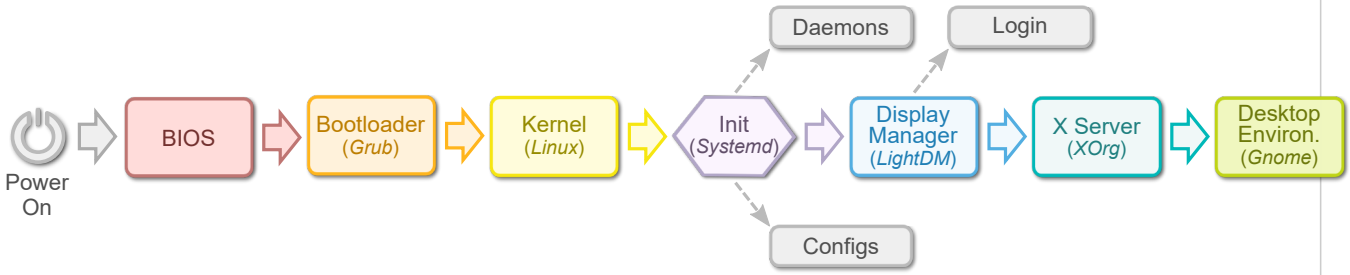
```
root         374       1  0 Apr20 ?        00:00:08 /lib/systemd/systemd-udevd
systemd+     541       1  0 Apr20 ?        00:00:02 /lib/systemd/systemd-timesyncd
systemd+     584       1  0 Apr20 ?        00:00:04 /lib/systemd/systemd-resolved
root         601       1  0 Apr20 ?        00:00:02 /usr/sbin/cron -f
syslog       611       1  0 Apr20 ?        00:00:01 /usr/sbin/rsyslogd -n -iNONE
root         615       1  0 Apr20 ?        00:00:03 /lib/systemd/systemd-logind
daemon       618       1  0 Apr20 ?        00:00:00 /usr/sbin/atd -f
```

Talvez você não tenha alguns desses *daemons*, dependendo da sua instalação. A tabela a seguir descreve rapidamente cada um deles:

Daemon	**Descrição**
`systemd-udevd`	Como veremos futuramente, o `systemd` é a implementação mais usada do `init` hoje. Na verdade, o `init` é uma parte do `systemd`. Outra parte dele é o *daemon* `systemd-udevd`, que monitora e recebe mensagens de eventos de hardware do kernel e permite configurar os arquivos do `/dev`, `/proc` e `/sys`.
`systemd-resolved`	Mais uma parte do `systemd`, responsável por "resolver" (descobrir o IP) de um nome DNS (*Domain Name System*).
`systemd-timesyncd`	Mais uma parte do `systemd`, responsável por manter o relógio do computador sincronizado com um servidor da Internet usando o protocolo NTP (*Network Time Protocol*).
`systemd-logind`	Gerencia o login e as sessões de usuários no sistema.
`cron`	É um serviço que permite o agendamento de comandos a serem executados rotineiramente (diariamente, a cada hora, etc).
`atd`	Outro serviço que permite o agendamento de comandos, mas estes são executados apenas uma vez em um determinado dia/horário.
`rsyslogd`	É um utilitário do sistema que provê suporte às mensagens de log (do diretório `/var/log`).
`thermald`	Usado para prevenir o aquecimento descontrolado do sistema. Este *daemon* monitora a temperatura da CPU e é capaz de tomar medidas para reduzir essa temperatura, caso necessário, antes que o próprio hardware tome medidas mais drásticas.
`acpid`	*Advanced Configuration and Power Interface*, monitora eventos de hardware e notifica programas de usuários. Exemplos de eventos: tela do laptop abaixada, bateria desconectada, volume do som, eventos wifi, etc.
`cupsd`	*Common UNIX Printing System*, é o servidor de impressão do Linux. Controla o acesso à impressora e permite o seu compartilhamento.
`Xorg`	Servidor X. Oferece os serviços básicos de interface gráfica, incluindo a configuração e controle da tela, do mouse e do teclado.

Inicialização do Linux

Vários passos compõem a inicialização de um computador desde o botão *power* até a inicialização do ambiente gráfico. Nesta parte do laboratório, iremos mostrar, detalhar e tentar explorar um pouco cada um desses passos. A figura a seguir mostra uma visão geral de todo o processo de inicialização de um sistema Linux:



Segue uma descrição rápida de cada uma das partes mencionadas na figura:

Etapa	Descrição
BIOS	A BIOS (*Basic Input/Output System*) é um *firmware* responsável por inicializar o hardware, incluindo a memória principal. Ele provê também diversos serviços ao sistema operacional. Após a sua inicialização, a BIOS executa o *bootloader* chamando um pequeno programa na parte inicial do disco, conhecida como MBR (*Master Boot Record*).
Bootloader	O *bootloader* do Linux é o GRUB. Ele é dividido em dois estágios. No primeiro estágio, um pequeno programa localizado no início do disco (MBR) é chamado. Como seu tamanho é limitado (menos de 500 bytes), ele basicamente chama outro estágio, localizado dentro de alguma partição. Ao ser iniciado, um menu pode ser mostrado ao usuário para permitir escolher qual sistema operacional será iniciado. Para iniciar o Linux, o *bootloader* carrega o kernel para a memória e o executa.
Kernel	O Kernel inicializa e carrega os drivers do hardware. Ele provê também uma série de serviços para os programas de usuários tais como gerenciamento de memória, escalonamento do processador e facilita o acesso aos dispositivos de hardware. Após inicializado, o kernel chama o primeiro processo do sistema: o `init`.
Init/Systemd	O `init` é responsável por configurar o sistema e iniciar os outros programas/daemons necessários. Atualmente, no Linux, o `init` é implementado pelo `systemd`, que fornece também uma série de outras funcionalidades, como o gerenciamento dos serviços iniciados. Uma das principais características do Systemd é permitir a inicialização de diversos serviços em paralelo, tirando proveito máximo do processador. Um desses serviços é o *Display Manager*.
Display Manager	O *display manager* é responsável por gerenciar as sessões dos usuários. Tais sessões são iniciadas, normalmente, via login gráfico. Portanto, uma das primeiras coisas que o *display manager faz* é iniciar o *X Server*. Após feito o login, ele inicia o *desktop environment* padrão ou o selecionado pelo usuário (é possível ter vários no mesmo sistema).
X Server	O *X Server*, mais conhecido como *X Window System display server*, é responsável por gerenciar o ambiente gráfico e os dispositivos de entrada e saída como mouse, teclado, touchscreen, etc. O mais usado atualmente é o XOrg.
Desktop Environment	O *desktop environment* é a interface gráfica que o usuário normal usa. Ele pinta as janelas, possui uma série de utilitários, configurações, etc. Conforme mencionado, existem vários *desktop environments* para Linux, sendo os mais conhecidos: GNOME, KDE, Cinnamon, Xfce, Mate, LXQt, Deepin, dentre vários outros.

Um sistema Linux pode ser configurado para não usar interface gráfica, como acontece com muitos servidores. Neste caso, o `init` não inicia o *X Server* e os passos seguintes são ignorados. Ao invés disso, apenas um sistema de login em modo texto estará disponível.

Na seção anterior, dissemos que um dos componentes do `systemd` era o `init`, um dos principais componentes da arquitetura de inicialização de um sistema Linux. Podemos provar isso, olhando mais cuidadosamente para o

5. Inicialização do Init pelo Kernel

O Linux Kernel inicia o `init` procurando por seu executável em vários lugares, como pode ser visto aqui. Inicialmente ele verifica se a opção "`init=`" foi passada quando o kernel foi carregado (variável `execute_command`). Caso a opção não tenha sido passada, o código verifica a macro `CONFIG_DEFAULT_INIT`, que não é usada pois seu valor é uma string vazia, como pode ser visto aqui.

No código mencionado, que define o valor (vazio) da variável DEFAULT_INIT, existe uma ajuda descrevendo a variável. Cole abaixo o texto da ajuda.

```
This option determines the default init for the system if no init= option is passed on the
kernel command line. If the requested path is not present, we will still then move on to
attempting further locations (e.g. /sbin/init, etc). If this is empty, we will just use
the fallback list when init= is not passed.
```

Voltando para o código-fonte do kernel, em seguida ele procura pelo `init` em três lugares:

Analise o código-fonte do Kernel Linux e diga abaixo os três arquivos que o kernel verifica procurando pelo binário do `init`.

```
"/sbin/init" "/etc/init" "/bin/init"
```

Por fim, caso o `init` não tenha sido encontrado, o kernel tenta iniciar um shell (`/bin/sh`) para permitir ao administrador resolver o problema.

Caso nem o shell tenha sido encontrado, qual a mensagem impressa pelo kernel?

```
No working init found. Try passing init= option to kernel.
See Linux Documentation/admin-guide/init.rst for guidance.
```

Em qual das formas acima o Kernel do Linux encontra o `init` no seu sistema? Vamos investigar! O primeiro passo é verificar as opções passadas para o kernel:

```
$ cat /proc/cmdline
```

```
initrd=\initrd.img panic=-1 nr_cpus=8 swiotlb=force pty.legacy_count=0
```

Como você pode ver (provavelmente), a opção `init=` não foi passada. Vamos agora verificar qual dos três arquivos procurados pelo kernel existem.

```
$ ls -fd /sbin/init /etc/init /bin/init      # f: não ordena, d: mostra nome de diretório
(não o seu conteúdo)
```

```
ls: cannot access '/etc/init': No such file or directory
ls: cannot access '/bin/init': No such file or directory
/sbin/init
```

Muito provavelmente, o `/bin/init` não existe no seu sistema. O `/etc/init` é um diretório, e não um executável. Portanto, o `init` realmente chamado pelo kernel é o `/sbin/init`.

6. O Init e o Systemd

Anteriormente, mencionamos que o `init` é uma de várias partes do `systemd`. Podemos confirmar isso olhando mais detalhadamente para o `/sbin/init`:

```
$ ls -lh /sbin/init
```

```
lrwxrwxrwx 1 root root 20 Jul 21  2021 /sbin/init -> /lib/systemd/systemd
```

Note como o arquivo `/sbin/init` é, na verdade, um link simbólico para o `/lib/systemd/systemd`. Como o comando `systemd` é usado para outras coisas além ser o `init`, no código-fonte dele é verificado o PID do processo atual para

saber se o que está sendo executado é o `init` ou algum outro serviço.

As configurações dos serviços gerenciados pelo `init` ficam, principalmente, no diretório `/lib/systemd/system/`:

```
$ ls /lib/systemd/system/*.service
```

Note como é uma quantidade grande de serviços. Nem todos resultam na execução de um *daemon*. O comando abaixo mostra um exemplo de configuração de serviço:

```
$ cat /lib/systemd/system/cron.service
```

```
[Unit]
Description=Regular background program processing daemon
Documentation=man:cron(8)
After=remote-fs.target nss-user-lookup.target

[Service]
EnvironmentFile=-/etc/default/cron
ExecStart=/usr/sbin/cron -f $EXTRA_OPTS
IgnoreSIGPIPE=false
KillMode=process
Restart=on-failure

[Install]
```

Conhecer muito bem o `systemd` é um dos principais requisitos de um administrador de sistemas Linux. Entretanto, para um usuário normal e também para o entendimento do AOSP, tais conhecimentos não são muito importantes. No caso dos usuários normais, em geral o `systemd` simplesmente funciona, sem necessidade de se lidar com ele. Já no caso do AOSP, como veremos futuramente, o Android usa outro `init` que não é o `systemd`. Portanto, neste curso, não entraremos nos detalhes do funcionamento, configuração e gerenciamento do `systemd`.

The more I learn, the more I realize how much I don't know. — Albert Einstein

π