Administração de Sistema: Apontamentos

Eduardo Morgado

janeiro, 2020

Conteúdo:

Figuras:

Tabelas:

Lista de Code Snipets

1 Processos

1.1 Componentes de um processo

Estruturas de dados no Kernel armazenam vários tipos de informação sobre cada processo, algumas dessas informações são [1]:

- O **estado de endereçamento** do processo
- O **estado actual** do processo (*sleeping, stopped, runnable, etc*)
- A **prioridade de execução** do processo
- Informação sobre
  - Os **recursos utilizados**
  - Os **ficheiros e portas de rede** abertos
- A **máscara de sinal**[1] do processo
- O dono do processo

Os processos possuem outros atributos tais como o PID, PPID, etc.

1.1.1 PID: Process ID

O PID é um **número único** atribuído pelo Kernel a cada processo à medida que novos processos são gerados, sendo este necessário para vários comandos. **Nota:** Podem existir *namespaces* que permitem o mesmo PID para processos diferentes.

1.1.2 PPID: Parent PID

No UNIX ou no LINUX não existe nenhuma *syscall* que gere um novo processo, dessa forma um processo deve-se **clonar** para gerar um novo processo (através da *syscall* **fork**), quando um processo é clonado o processo gerado apresenta um PID diferente do pai, o processo original é referido ao novo como sendo o seu pai, dessa forma o novo processo guarda uma referência ao seu pai, neste caso, armazena do PID do progenitor, ou seja, guarda o **PPID**.

1.1.3 UID e EUID: User ID real e efectivo

O UID de um processo é no **número de identificação** (uma cópia do UID do processo pai) da pessoa que o gerou.

O EUID é um UID extra, utilizado para determinar quais os ficheiros e recursos é que um processo tem permissão para aceder, geralmente este valor é igual ao UID, excepto para processos ***setuid***.

A utilização destes dois valores permite manter uma distinção entre **identidade e permissão**.

[1] Um registo/histórico de quais sinais estão bloqueados

1.1.4 GID e EGID: Group ID e efectivo

O GID é um atributo vestigial, utilizado para determinação de acesso, um processo pode ter vários grupos, a lista de todos os grupos de um processo é guardada em separado do processo.

O GID é mais utilizado quando um processo cria novos ficheiros, para atribuir o seu grupo aos ficheiros gerados. Para questões de permissões de acesso é mais utilizado o EGID e a lista de grupos do processo.
Valores de GID e EGID são iguais em geral, excepto para processos ***setgid***.

1.1.5 Niceness

Os processos possuem uma componente de prioridade. A prioridade de um processo determina o **tempo de CPU** que será alocado ao processo, a computação de prioridades é feita de forma dinâmica pelo Kernel, no entanto, o Kernel também tem em consideração a *niceness* do processo.

A *niceness* de um processo é um **valor definido adiministrativamente**, fornecendo informação do quão "simpático" o processo será para os utilizadores do sistema (esse valor pode ir do 19, o mais simpático, até ao -20, o menos simpático).

1.1.6 Terminal de controlo

Associado a processos *nondaemon*, determina as ligações *default* para o *stdin, stdout, stderr*.

1.2 Sinais

Os sinais são pedidos de interrupção ao nível de processos e podem ser utilizados de várias formas [1]:

- Meio de comunicação entre processos
- Envidados pelo terminal para **terminar**,**interromper** e **suspender** processos
- Podem ser envidados pelo administrados
- Podem servir como avisos de interesse enviados pelo kernel aos processos (morte de filhos, disponibilidade de informação em canal I/O,etc)

Quando um sinal é recebido ele deve ser **tratado**, se o processo **possui um *handler*** designado para o sinal em particular, esse *handler* é chamado caso contrário o **kernel toma uma ação *default*** em lugar do processo, essa acção varia, consoante o tipo de sinal [1]. A especificação de um *handler* num programa para um sinal é denominada como **catching** do sinal [1].
Um processo pode impedir a chegada de sinais, **ignorando** ou **bloqueando** sinais [1]. Um sinal ignorado é descartado pelo processo, no entanto, um sinal bloqueado é **colocado em fila** e permanecerá nela até o processo desbloquear esse sinal [1]. A figura 1 apresenta alguns desses sinais.

#	Nome	Descrição	Por omissão	Pode ser apanhado?	Pode ser bloqueado?	Coredump?
1	HUP	Hangup	Terminate	Yes	Yes	No
2	INT	Interrupt	Terminate	Yes	Yes	No
3	QUIT	Quit	Terminate	Yes	Yes	Yes
9	KILL	Kill	Terminate	No	No	No
*	BUS	Bus error	Terminate	Yes	Yes	Yes
11	SEGV	Segmentation fault	Terminate	Yes	Yes	Yes
15	TERM	Software termination	Terminate	Yes	Yes	No
*	STOP	Stop	Stop	No	No	No
*	TSTP	Keyboard stop	Stop	Yes	Yes	No
*	CONT	Continue after stop	Ignore	Yes	No	No
*	WINCH	Window changed	Ignore	Yes	Yes	No
*	USR1	User-defined #1	Terminate	Yes	Yes	No
*	USR2	User-defined #2	Terminate	Yes	Yes	No

- Lista disponível com

  `kill -l`

* Varia de acordo com o sistema

Figure 1: Alguns sinais

1.3 Estados de processos

Apesar de um processo existir em memória, este pode ser **ilegível** para receber tempo de CPU, dessa forma, é necessário considerar quatro estados de processo representados na figura 2 [1].

Um processo no estado **runnable** pode ser executado assim que a CPU esteja disponível uma vez que, **apresenta todos os recursos necessários** à execução, e o processo realizar uma *syscall* que não pode ser solucionada imediatamente o Kernel muda o seu estado para *sleeping* [1]. Estes processos têm valor R no comando ps.

Um processo no estado **sleeping** espera algum evento, um programa interactivo é um exemplo de uma programa no estado *sleeping*, ou seja, é um programa que está bloqueado até a sua necessidade ser satisfeita (podendo ser um evento I/O ou de sistema), dessa forma, estando nesse estado, se estiver a executar na CPU irá disponibilizar o seu tempo na CPU até receber o evento, estes processos têm um valor S no comando ps. Certas operações e ações podem colocar um processo num estado **uninterruplitble sleep state**, este estado não é visível através do comando **ps** (estado com valor D) [1]. Uma vez que estes processos não podem ser interrompidos, nem quando recebem um sinal, **não podem ser terminados**, deve ser resolvido o problema ou reiniciar a máquina.

Um processo no estado **zombie** é um processo que, apesar de ter terminado a sua execução ainda não foram repetidos pelo pai, estes processos têm valor Z no comando ps.

Um processo no estado **stopped** são processos que foram **proibidos administrativamente** de serem executados, estes processos têm valor T no comando ps. Estes processos apresentam um comportamento similar ao estado *sleeping*, no entanto, apenas podem sair deste estado se forem terminados (*kill*) ou modificados de estado por outro programa.

State	Meaning
Runnable	The process can be executed.
Sleeping	The process is waiting for some resource.
Zombie	The process is trying to die.
Stopped	The process is suspended (not allowed to execute).

Figure 2: Estados de processos

```
$ nice -n 5 ~/bin/longtask
    // launch process with lower priority (raise nice) by 5
$ sudo renice -5 8829
    // Sets nice value to -5
$ sudo renice 5 -u boggs
    // Sets nice value of boggs's procs to 5
```

Figure 3: Aplicação do comando nice e renice

1.4 Niceness: Influência na prioridade

Tal como referido anteriormente, o valor *nice* fornece uma indicação do quão "simpático" o programa irá ser para outros utilizadores. Um valor alto corresponde a uma **prioridade baixa**, ou seja, o processo irá ser simpático e um valor baixo/negativo corresponde a uma **prioridade alta**. O intervalo de valores depende do sistema, o mais comum e entre 19 e -20.

Caso não seja especificado pelo utilizador quando cria um novo processo, o valor *nice* do novo processo é **herdado** do processo pai [1]. O dono do processo pode *aumentar* o seu valor *nice* mas **não pode diminuir o valor nice**[2] O *superuser* pode definir o valor *nice* arbitrariamente.

O valor *nice* do processo pode ser definido quando o processo é criado com o comando **nice** e pode mais tarde ser reajustado com o comando **renice** (recebe como argumentos o valor e o PID do processo), a figura 3 mostra a utilização destes comandos. A figura 4 apresenta os intervalos de valores *nice* e os primeiros argumentos dos comandos, dependendo do tipo de sistema a que referem.

[2]Dessa forma, processos com baixa prioridade (alto valor *nice*) não podem gerar processos com alta prioridade (baixo valor *nice*) ou menor prioridade que o seu pai [1].

System	Range	OS nice	csh nice	renice
Linux	-20 to 19	*-incr* or **-n** *incr*	*+incr* or *-incr*	*prio*
Solaris	0 to 39	*-incr* or **-n** *incr*	*+incr* or *-incr*	*incr* or **-n** *incr*
HP-UX	0 to 39	*-prio* or **-n** *prio*	*+incr* or *-incr*	**-n** *prio*[a]
AIX	-20 to 19	*-incr* or **-n** *incr*	*+incr* or *-incr*	**-n** *incr*

Figure 4: Intervalos e expressão de prioridade de nice e renice

1.5 PS: Monitor de processos

Existem várias versões deste comando (ps) dependendo do sistema onde está implementado, o comando ps pode mostrar o PID, UID, prioridade e terminal de controlo de processos, fornecendo também informações sobre a **utilização de memória** do processo, o tempo de CPU utilizado e o **estado do processo**

Bibliografia

[1] Trent R. Hein Evi Nemeth Garth Snyder and Ben Whaley. “Unix and Linux System Administration Handbook”. In: fourth edition. Prentice Hall, 2010. Chap. 5 Controlling Processes.