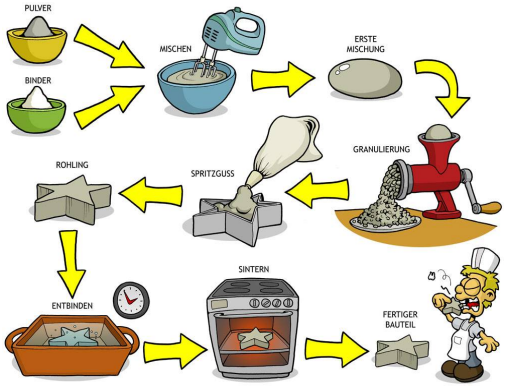
Befehle zur Dienst- u. Systemverwaltung (Services, Prozesse u. Tasks)

Quellen:



http://www.nt.fh-koeln.de/fachgebiete/inf/diplom/proc_sync/index2.html Task – Prozesse – Threads mit Java erklärt

http://msdn.microsoft.com/en-us/library/ms684841(VS.85).aspx Quellcode in C++ zur Prozessverwaltung

http://technet.microsoft.com/de-de/library/aa998749(EXCHG.65).aspx SC-Systemverwaltung und Registry-Einträge

http://www.serverhowto.de/Applikationen-als-Dienste-einrichten.228.0.html Installation und registrieren von Systemdiensten http://www.pcwelt.de/suche?searchStr=Windows+Dienste&category=Alle+Rubriken&period=Jederzeit Dienste und Infos bei PC-

http://www.pcwelt.de/ratgeber/Windows-7-Dienste-Windows-7-Dienste-verstehen-aufraeumen-120117.html Dienste Aufräumen

https://en.wikipedia.org/wiki/Service_Control_Manager
https://en.wikipedia.org/wiki/List_of_Microsoft_Windows_components#Services
} Windows-Service-Dienste-

http://www.pcwelt.de/ratgeber/Best-of-Praxis-100-Prozent-Windows-293554.html?redirect=1 Windows 2000 / XP -Dienste

http://www.pcwelt.de/ratgeber/Vista-Dienste-aufraeumen-492154.html?redirect=1 Windows-Vista und Folgende – Dienste aufräumen

http://www.pcwelt.de/ratgeber/Dienste-bearbeiten-Windows-Control-msconfig-sc-Programm-oder-Registry-Windows-Dienste-richtig-aufraeumen-so-geht-s-5769946.html Windows-Dienste aufräumen

http://openwbem.sourceforge.net/ OPENWBEM Open-Source-Dienste- und Management-Framework

http://www.pcwelt.de/ratgeber/Windows-Tuning-Vista-Dienste-aufraeumen-221441.html http://de.wikipedia.org/wiki/Desktop_Management_Interface

http://www.linux-mag.com/id/7768/ BIOS

http://linux.dell.com/libsmbios/main/cmdlinetools.html Bios-Befehle

http://www.linux-mag.com/id/8794/ Dateisystem-Dienste-Prüfungen

http://channel9.msdn.com/Events/TechEd/NorthAmerica/2014/WIN-B354#fbid= Windows-Probleme lösen mit den sysinternal-tools

https://technet.microsoft.com/de-de/magazine/2008.03.kernel.aspx Window-Kernel 2008 Neuerungen

http://en.wikipedia.org/wiki/Service_Control_Manager#Delayed_auto-start_services Delayed auto-start-Service

https://www.youtube.com/watch?v=XoHz9Umb Infos die Google nicht zeigen darf

Aufgaben:

1. Speichern Sie diese Datei lokal auf Ihrem PC/USB-Stick!
2. Versehen Sie alle Befehle mit einer Erläuterung oder einen Hyperlink in Spalte zwei bzw. Erklären Sie den Skript-Befehl bzw. das Skript!
3. Wenn möglich finden Sie zusätzlich ein aussagefähiges und nützliches (batch-)Skript oder sonstige Infos (z. B. als Web-Links) zum jeweiligen Windows- und Linux-Befehl! Tragen Sie die Links auf der rechten Seite in die Datei-Vorlage ein!

Windows-Befehl mit Übergabeparameter:	*Kurze Erläuterung bzw. Bedeutung des Befehls:*
procexp.exe	**Sehr ausführlicher Windows Prozess- und Taskmonitor!**
instsrv.exe [Dienstname] [Pfad zur Anwendung] \srvany.exe	**Installiert eine Anwendung als Dienst (Service < Win2008)!**
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\[Dienstname]	**Registry-Eintrag für einen angelegten (System)dienst**
instsrv.exe [Dienstname] remove	**Deinstallation eines (System)dienstes**
sc create „Testdienst" start= auto binpath= „C:\Programme\ Beispieldatei.exe"	**Einrichten einer Anwendung als Dienst (>win2000)**
sc config „Testdienst" start= disabled	**Nachkonfiguration der Startoption eines Systemdienstes hier speziell auf „disabled" stellen**
sc description „Testdienst" „Dieser Dienst ist nur ein Test-Dienst"	**Dies ist nur eine Änderung der Dienstbeschreibung**
PS C:\>Get-Help <Cmdlet>	**Hilfe zu den Powershell-Befehlen**
PsExec	**Startet eine Befehlskonsole remote und fürht dort einen Konsolenbefehl aus.**
PsFile	**Zeigt die auf dem remote PC gegenwärtig geöffneten/verwendeten Dateien an.**
PsInfo	**Remote Prozessinformationen z.B. Prozess-Ids abrufen**
PsKill	**Remote Prozesse beenden (kill the prozess)**
PsList	**Remote Infos über die laufenden Prozesse**
PsLoggedOn	**Wer ist gegenwärtig über welche Resource angemeldet.**
PsLogList	**Zeigt die Logdateien (Protokoll) des Systems auch remote an**
PsService	**Anzeigen und einstellen (start/stopp) der Dienste**
PsShutdown	**Remote Herunterfahren und Neustart des PC's.**

Windows-Befehl mit Übergabeparameter:	*Kurze Erläuterung bzw. Bedeutung des Befehls:*
PsSuspend	**Unterbrechen und reinitialisieren/wieder starten eines (remote) Prozesses**
sc query state= all	**Er zeigt den Status aller Dienste an.**
sc sdset	**Setzen einer Sicherheitsbeschreibung in der Form des „sddl"**
msconfig	**Microsoft-Startup-Konfigurator**
sc query state= all type= service	**Siehe oben (Mischform)**

Linux-Befehl mit Übergabeparameter:	***Kurze Erläuterung bzw. Bedeutung des Befehls:***
tail -f /var/log/messages	**Zeige das Ende der Datei: .../messages fortlaufend an = Monitor von .../messages**
lsb_release -a	**Zeigt die Betriebssystemversion an.**
strace -c ls >/dev/nullSummarise/profile	**Anzeigen von Systemaufrufen und Systemrückmeldungen**
strace -f -e open ls >/dev/nullList	**-„-**
ltrace -f -e getenv ls >/dev/null	**Anzeige von Aufrufen, die aus einer Programmbibliothek herrühren**
lsof -p $$	**Liste der offenen (gegenwärtig laufenden) Prozesse.**
lsof ~	**Liste ....**
tcpdump not port 22	**Zeigt den Netzwerkverkehr an ohne Port 22an.**
ps -e -o pid,args –forest	**Hierarchisches auflisten der Prozesse**
ps -e -o pcpu,cpu,nice,state,cputime,args --sort pcpu | sed '/^ 0.0 /d'	**Listet die prozentuale CPU-Prozess-Auslastung auf.**
ps -e -orss=,args= | sort -b -k1,1n | pr -TW$COLUMNS	**Listet die Speicher-Prozessbelegung sortiert auf. (oder: ps_mem.py)**
ps -C firefox-bin -L -o pid,tid,pcpu,state	**Listet die laufenden Threads eines einzelnen Prozesses auf.**
ps -p 1,2	**Prozessinfos zu den gelisteten Prozess-IDs**
last reboot	**Zeigt eine „history" der letzten Rebootvorganänge!**
free -m	**Zeigt den vorhandenen und freien RAM-Speicher in MByte.**
watch -n.1 'cat /proc/interrupts'	**Alle Sekunden wird die Ausgabe der Interrupts erneuert.**
udevadm monitor	**Kontrolle/Anzeige von Geräte- und Systemzugriffen (oder: sysinfo)**
uname -a	**Kernel-Architektur und Version**
head -n1 /etc/issue	
cat /proc/partitions	**Zeige den Inhalt der Datei ... Diese Datei enthält die registrierten Prozesse, die die Partitionen angeben – also im System registrierte Partitionen**
grep MemTotal /proc/meminfo	
grep "model name" /proc/cpuinfo	**Zeige Infos über den Prozessor bzw. die Prozessoren.**

Linux-Befehl mit Übergabeparameter:	*Kurze Erläuterung bzw. Bedeutung des Befehls:*
lspci -tv	Zeige Infos über die PCI-Komponenten (Steckkarten, Grafikkarte usw.)
lsusb -tv	Zeige Infos über die USB-Busgeräte und Bus-Hubs.
mount | column -t	Zeige die eingebundenen Partitionen tabellarisch angeordnet.
grep -F capacity: /proc/acpi/battery/BAT0/info	Ladezustand der Akkus anzeigen
dmidecode -q | less	Zeige SMBIOS/DMI- Informationen
smartctl -A /dev/sda | grep Power_On_Hours	Zeige die S.M.A.R.T. - Werte, greife die Lauf-Stunden heraus.
hdparm -i /dev/sda hdparm -tT /dev/sda	Zeige die Eigenschaften der Festplatte sda Zeige die Werte für die Lesegeschwindigkeit der Festplatte
badblocks -s /dev/sda	Suche nach defekten Blöcken (Lesbarkeit) auf der Festplatte
dmidecode -t system	Die Originaldaten der Hersteller aus den Systemkomponenten auslesen.
biosdecode	Ausgabe der Biosdaten
Vmstat -m oder -a	Anzeige der virtuellen Speicherstatistik (VRAM-Statistik)
w oder who	Wer ist gerade auf der Maschine? Who is here?
fuser und alternativ finger fuser -k -i filename.txt fuser -n tcp 21	Welcher Prozess (welche ID) verwendet die Datei *.txt? -k bedeutet kill. Welcher Prozess verwendet den tcp-Port 21?
uptime	Seit wann läuft der PC?
ps -aAlLux oder ps -eo euser,ruser,suser,fuser,f,comm,label	Prozessinformationen
ps -auxf | sort -nr -k 4 | head -10	-"-
iostat	Infos zur CPU und zur Input- Output-Vorgängen
sar -n DEV -f /var/log/sa/sa24 | more	System-activity-request Sammeln von logging-infos
mpstat -P ALL	Multiprozessorstatistik
pmap -d 47394	Protokoll Mamory-MAP
ss	Socket-Statistik
netstat -nat | grep {IP-address} | awk '{print $6}' | sort | uniq -c | sort -n	Netzwerkstatistik
iptraf	ip-traffic
tcpdump -ni eth0 'dst 192.168.1.5 and tcp and port http'	Tcp-dump: Aufzeichnen des tcp-traffic
strace	Dient dem Debugging von Systemaufrufen
cat /proc/zoneinfo	Diese „Datei" stellt Prozessinformationen zu den RAM-Speicher-Zonen dar. Dies wird benötigt um das Verhalten der virtuellen Speicherzuordnung verstehen zu können.

Linux-Befehl mit Übergabeparameter:	*Kurze Erläuterung bzw. Bedeutung des Befehls:*
nagios	SNMP-Tool Überwachungssystem für PCs und Switche etc.
cacti	SNMP-Tools Überwachungssystem für PCs und Switche etc.
KSysguard	KDE-Systemmonitor
Gnome System Monitor	Gnome-Systemmonitor
nmap	Scannen der offenen Ports eines PCs.
lsof	Zeigt offene Dateien, Netzwerksockets, und andere Verbindungen
ntop oder htop	web based tool - ntop is the best tool to see network usage in a way similar to what top command does for processes i.e. it is network traffic monitoring software. You can see network status, protocol wise distribution of traffic for UDP, TCP, DNS, HTTP and other protocols. htop is an enhanced version of top, the interactive process viewer, which can display the list of processes in a tree form.
Conky	Another good monitoring tool for the X Window System. It is highly configurable and is able to monitor many system variables including the status of the CPU, memory, swap space, disk storage, temperatures, processes, network interfaces, battery power, system messages, e-mail inboxes etc.
GKrellM	It can be used to monitor the status of CPUs, main memory, hard disks, network interfaces, local and remote mailboxes, and many other things.
mtr	mtr combines the functionality of the traceroute and ping programs in a single network diagnostic tool.
vnstat	vnStat is a console-based network traffic monitor. It keeps a log of hourly, daily and monthly network traffic for the selected interface(s).
`iperf -s -B 202.54.1.1`	Ip-Performance, -s Server, -B Broudcast an ...
smartctl -t long /dev/sda oder `smartd`	Kontrolle/Ausgabe der S.M.A.R.T. - Werte
Smart-notifier smartmontools	Kontrolle/Ausgabe der S.M.A.R.T. - Werte
http://openwbem.sourceforge.net/	OpenSource-Tools zur PC- und Netzwerkanalyse und -Monitoring

Beschriebung einiger wichtiger Befehle

1 Befehle zum Auflisten der laufenden Prozesse

1.1 ps

Um alle Prozesse aufzulisten, die gerade laufen, gibt es den Befehl ps (Process-Status), der zunächst mal nur die eigenen Prozesse auflistet. ps stellt aber immer nur eine Momentaufnahme dar, d.h., es werden genau die Prozesse aufgelistet, die im Augenblick laufen. Dynamische Veränderungen sind nicht darstellbar.

ps kennt eine ganze Menge verschiedener Kommandozeilenparameter (Schalter) die sein Verhalten entsprechend ändern können. Die wichtigsten sind:

l	Langes Format
u	User Format (Mit User und Startzeit)
a	Alle Prozesse, auch die anderer User
x	Auch Daemon-Prozesse (Ohne eigene TTY-Leitung)
f	Forest (Wald) Format - Der Prozessbaum wird dargestellt
w	Wide (breite) Ausgabe - Zeilen werden nicht abgeschnitten

So kann also mit dem Befehl

ps uax

eine Liste aller Prozesse ausgegeben werden, die insgesamt auf dem System laufen, auch die anderer User (a) und Daemonprozesse (x). Sie werden zusammen mit dem Namen ihres Eigentümers (u) dargestellt.

Um das gleiche Ergebnis als Baumstruktur zu bekommen kann

ps fax

benutzt werden. Die einzelnen Felder sind identisch mit den weiter unten dargestellten Feldern des top-Befehls.

1.2 top

Wenn statt der Auflistung der Momentaufnahme eine ständig aktualisierte Liste gewünscht wird, so ist das Programm top das benötigte Werkzeug. Dieses Programm gibt eine Liste der Prozesse aus und aktualisiert diese nach einer bestimmten Wartezeit (voreingestellt sind 5 Sek). Der Nachteil ist, dass nur soviele Prozesse aufgelistet werden, wie auf den entsprechenden Bildschirm passen. Eine vernünftige Anwendung ist somit nur in einem entsprechend großem xterm-Fenster möglich.

Als Programm, das interaktiv benutzbar ist, hat top natürlich auch Befehlstasten, die den Ablauf verändern. Die folgenden Tasten sind die wichtigsten Befehle:

Leertaste	Sofortiges Update der Prozesse
Strg-L	Bildschirm neu aufbauen
h oder ?	Darstellung einer Hilfeseite
i	Ignoriere schlafende und Zombie-Prozesse (i ist ein Wechselschalter, erneutes Drücken bewirkt, dass diese Prozesse wieder angezeigt werden.
r	Renice - Damit kann einem Prozeß ein neuer Nice-Wert gegeben werden (sofern der User das Recht dazu hat). Das Programm fragt nach PID und Nice-Wert.

k	Kill - Entspricht dem Programm kill - siehe weiter unten. Damit können Signale an Prozesse geschickt werden.
s	Damit kann die Zeit verändert werden, die zwischen dem Auffrischen des Bildschirms gewartet wird. Eingabe in Sekunden. Vorsicht, eine zu kurze Zeit bringt den Rechner schnell in die Knie. Eine 0 steht für dauernde Neudarstellung ohne Wartezeit, Voreingestellt sin meist 5 Sekunden.
f	Fügt Felder in die Ausgabe ein oder entfernt sie. Die einzelnen Felder werden weiter unten erklärt.
W	Schreibt die aktuelle Konfiguration in die Datei ~/.toprc - Damit wird nach dem Neustart von top diese Konfiguration wieder geladen.
q	Quit - Beendet das Programm

top kann verschiedene Felder darstellen, die mit dem f-Befehl ausgewählt werden können. Die einzelnen Felder haben folgende Bedeutung:

PID	Die Process-ID des Prozesses
PPID	Die Parent Process ID des Prozesses
UID	Die User ID des Users, dem der Prozeß gehört
USER	Der Username des Users, dem der Prozeß gehört
PRI	Die Priorität des Prozesses. Höhere Werte bedeuten höhere Priorität.
NI	Der Nice-Wert des Prozesses. Höhere Werte bedeuten geringere Priorität.
SIZE	Die Größe des Codes plus Daten plus Stack in KiloByte
TSIZE	Die Größe des Codes in KiloByte. ELF Prozesse werden nicht korrekt dargestellt
DSIZE	Die Größe der Daten und Stack in Kilobyte. ELF Prozesse werden nicht korrekt dargestellt
TRS	Text Resident Size - Die größe des residenten Code-Blocks in KiloByte
SWAP	Größe des ausgelagerten Bereichs des Tasks
D	Größe der als Dirty markierten Speicherseiten
LIB	Größe der Library-Speicherseiten - Funktioniert nicht bei ELF-Prozessen.
RSS	Die Größe des physikalische Speichers, den das Programm benutzt. Für ELF-Format werden hier auch die Libraries mitgezählt, bei a.out Format nicht.
SHARE	Die Größe der benutzten Shared-Libraries des Prozesses.
STAT	Der Status des Prozesses. Das kann entweder ein **S** für schlafend, **D** für ununterbrechbar schlafend (dead), **R** für laufend (running) oder **T** für angehalten (traced). Dieser Angabe kann noch ein **<** für einen negativen Nice-Wert, ein **N** für einen positiven Nice-Wert oder ein **W** für einen ausgelagerten Prozeß folgen. (Das W funktioniert nicht richtig für Kernel-Prozesse)
WCHAN	Die Kernelfunktion, die der Task gerade nutzt.
TIME	Die gesamte CPU-Zeit, die der Prozeß verbraucht hat, seit er gestartet wurde.
%CPU	Die CPU-Zeit, die der Prozeß seit dem letzten Bildschirm-Update verbraucht hat, dargestellt als Prozentsatz der gesamten CPU-

	Zeit.
%MEM	Der Anteil des Speichers, den der Task nutzt.
COMMAND	Das Kommando, mit dem der Prozeß gestartet wurde.
TTY	Die Terminalleitung des Prozesses.

2 Befehle für die Inter-Process-Communikation

Die folgenden Befehle, die alle Signale an Prozesse schicken sind in ihrer Handhabung für den Normaluser eingeschränkt. Der Normaluser darf selbstverständlich nur die Prozesse ansprechen, die ihm gehören, die er also selbst aufgerufen hat. Nur der Systemverwalter (root) darf auch Prozessen anderer User solche Signale schicken.

2.1 kill

Mit dem kill-Befehl kan man einem oder mehreren Prozessen Signale schicken. Diese Signale können den Prozeß dann dazu bewegen, bestimmte Aktionen vorzunehmen. Der Befehl kill erwartet als Parameter zuerst das zu sendende Signal (mit vorgestelltem Bindestrich), entweder als Zahl oder als symbolischer Name, dann die PID des Prozesses, dem es geschickt werden soll. Es können auch mehrere ProzeßIDs angegeben werden.

Mit kill -l erhält man eine Liste der gültigen Signale, bei Linux sind das in der Regel die Folgenden:

1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL
5) SIGTRAP	6) SIGABRT	7) SIGBUS	8) SIGFPE
9) SIGKILL	10) SIGUSR1	11) SIGSEGV	12) SIGUSR2
13) SIGPIPE	14) SIGALRM	15) SIGTERM	17) SIGCHLD
18) SIGCONT	19) SIGSTOP	20) SIGTSTP	21) SIGTTIN
22) SIGTTOU	23) SIGURG	24) SIGXCPU	25) SIGXFSZ
26) SIGVTALRM	27) SIGPROF	28) SIGWINCH	29) SIGIO
30) SIGPWR			

Der Programmierer eines Programms kann festlegen, wie das Programm auf ein bestimmtes Signal reagieren soll. Das Signal 15 ist das voreingestellte Signal, das geschickt wird, wenn dem kill-Befehl kein Signal angegeben wird und fordert ein Programm auf, sich zu beenden. Ein Prozeß muß aber nicht zwangsläufig abbrechen, wenn es dieses Signal empfängt. Das Signal 9 hingegen ist der "Todesschuß", diesem Signal kann sich kein Prozeß entziehen.

2.2 skill

Der Befehl skill arbeitet im wesentlichen wie kill, nur dass er sich nicht auf Prozeß-IDs bezieht, sondern auf Namen von Programmen, Usern oder TTY-Leitungen. Das heißt, skill schickt ein bestimmtes Signal an alle Programme eines Namens, alle Prozesse eines Users oder alle Prozesse, die ein bestimmtes Terminal benutzen. Die Aufrufform ist wie bei kill, zunächst das Signal, wie bei kill mit vorgestelltem Bindestrich, entweder als symbolischer Name (z.B. -HUP oder -SIGHUP) oder als Nummer (-1). Dann folgen aber nicht PIDs sondern die Namen der Programme, User oder TTYs. Der Befehl

skill -9 hans

schickt also allen Prozessen, die dem User hans gehören das tödliche Signal Nummer 9. Oder um alle Prozesse zu beenden, die auf der Terminalleitung TTY4 laufen, reicht es zu schreiben:

skill -9 TTY4

Das hat natürlich den Haken, dass es z.B. Programme geben kann, die den gleichen Namen wie User tragen. Um also Verwechslungen auszuschließen, kann dem Usernamen ein -u vorgestellt werden, dem Programmnamen ein -c (command), einer PID ein -p und dem Namen der Terminalleitung ein -t. Diese Angaben sind kombinierbar.

Um sicherzustellen, dass es in den beiden obigen Beispielen nicht zu Verwechslungen kommen kann wäre es also besser zu schreiben:

skill -9 -u hans
skill -9 -t TTY4

2.3 snice

Der Befehl snice ist nah verwandt mit skill, nur dass er statt ein Signal an Prozesse eines Users, eines Namens oder einer TTY-Leitung eben deren Nice-Wert ändert. Wie früher schon erklärt ist der Nice-Wert eines Prozesses der Wert, wie nett ein Prozeß zu anderen Prozessen ist, wie viel Rechenzeit er also benutzt. Je höher der Wert, umso niedriger die Priorität. Nur der Systemverwalter darf einem Prozeß einen negativen Nice-Wert geben.

Der Befehl

snice +10 -p 1034

stellt den Nice-Wert des Prozesses mit der PID 1034 auf +10 ein.

2.4 killproc

Der Befehl killproc schickt, ähnlich wie skill, Signale an Prozesse, aber er schickt sie an alle Prozesse, die von genau einem Programm abstammen. Als zweiten Parameter erwartet er also den vollen Pfad zu einem Programm, und schickt dann das angegebene Signal an alle Prozesse, die dieses Programm laufen lassen. Er entspricht damit fast dem Befehl

skill -SIGNAL -c Programmname

mit dem Unterschied, dass skill sich hier nur um den Namen kümmert, nicht um den Pfad. Wären also im System zwei Programme gleichen Namens, eines heißt /home/hans/prog1, das andere /home/otto/prog1, beide Programme sind gerade geladen und laufen, so würde ein

skill -9 -c prog1

beide Prozesse killen. Der Befehl

killproc -9 /home/hans/prog1

schickt aber das Signal 9 nur an das eine von beiden.

Was ist ein Dämon?

Unter "daemon" versteht man in der LINUIX-Welt, im Gegensatz zur realen Welt, einen hilfreichen Geist, der für den Benutzer quasi unsichtbar nützliche Dienste leistet.

"Unsichtbar" heißt in diesem Zusammenhang, daß es sich um Prozesse handelt, die keine Interaktion des Benutzers erfordern und somit auch keine graphische Benutzeroberfläche benötigen. Einmal gestartet wartet er auf bestimmte Ereignisse und tritt dann in Aktion. Dies kann beispielsweise ein Datenbankserver sein, der darauf wartet, daß jemand eine Verbindung zu ihm aufbaut und Daten aus der Datenbank z.B. mittels SQL abfragt.

In einem UNIX-System gibt es zahlreiche daemons, die allerhand nützlicher Dienste leisten, ohne daß der Benutzer es merkt. Dies sind nicht selten wichtige Systemfunktionen, die mit root-Rechten ausgestattet sein müssen. Beispiele hierfür sind der secure shell daemon (sshd), der ssh Sitzungen abwickelt, der at-daemon (atd), der zu bestimmten Zeiten jobs im Hintergrund startet, u.v.a. mehr. Diese "traditionellen" UNIX-Dienste werden üblicherweise in den Init-Skripten beim wechseln des runlevels (z.B. beim booten) automatisch gestartet und auch gestoppt. Es gibt aber durchaus Fälle, in denen man dieses Verhalten nicht wünscht. Zum einen hat nicht jeder Benutzer root-Privilegien oder auch die Erfahrung, die man braucht, um die Init-Skripte zu modifizieren. Zum anderen gibt es Dienste, die nicht in den üblichen Linux-Distributionen vorhanden sind (zu Recht). Zwei praktische Beispiele, die keine root-Rechte benötigen, seien hier erläutern:

- File-sharing Systeme
- Dedicated game server

Zunächst jedoch, soll das "Skelett" eines Dämonen (eigentlich ein "Wrapper-Skript", das den eigentlichen Dämon einhüllt) vorgestellt werden, das durch einfachste Konfiguration an die eigenen Bedürfnisse angepasst werden kann.

Das Skelett

In der folgenden Tabelle ist das Skript dargestellt.

skeleton.sh

```
 1  #!/bin/sh
 2  #------------------------------------------------------
 3  #
 4  # generic daemon wrapper script
 5  #
 6  # OPTIONS: (start|stop|restart|status)
 7  #
 8  #------------------------------------------------------
 9
10  # --- edit here ---
11
12  DAEMON_DESC="my fancy daemon"
13  DAEMON_BIN="/path/to/my_daemon"
14  DAEMON_PROC="daemon_ps_name"
15  DAEMON_LOGFILE="/path/to/my/logfile/${DAEMON_PROC}.log"
16  DAEMON_OPTIONS=" -foo -bar "
17
18  # --- don't edit below ---
19
20  #------------------------------------------------------
21  #
22  # get_pid()
23  #
24  #------------------------------------------------------
25
26  get_pid()
27  {
28          PID=`ps aux | grep -v grep | grep ${DAEMON_PROC} | awk '{print $2}'`
29  }
30
31  #------------------------------------------------------
32  #
33  # start_daemon()
34  #
35  #------------------------------------------------------
```

```
 36
 37 start_daemon()
 38 {
 39         echo -n "Starting ${DAEMON_DESC}"
 40         ${DAEMON_BIN} ${DAEMON_OPTIONS} 2> ${DAEMON_LOGFILE}&
 41         echo " ... done."
 42 }
 43
 44 #-----------------------------------------------------
 45 #
 46 # stop_daemon()
 47 #
 48 #-----------------------------------------------------
 49
 50 stop_daemon()
 51 {
 52         echo -n "Stopping ${DAEMON_DESC}"
 53         kill -s SIGKILL $PID
 54         echo " ... done."
 55 }
 56
 57 #-----------------------------------------------------
 58 #
 59 # main
 60 #
 61 #-----------------------------------------------------
 62
 63 case "$1" in
 64         start)
 65                 get_pid
 66                 if [ -z "$PID" ] ; then
 67                         start_daemon
 68                 else
 69                         echo "${DAEMON_DESC} is running."
 70                         exit 1
 71                 fi
 72                 ;;
 73
 74         stop)
 75                 get_pid
 76                 if [ -z "$PID" ] ; then
 77                         echo "${DAEMON_DESC} is not running."
 78                         exit 1
 79                 else
 80                         stop_daemon
 81                 fi
 82                 ;;
 83
 84         restart)
 85                 get_pid
 86                 if [ ! -z "$PID" ] ; then
 87                         stop_daemon
 88                 fi
 89                 start_daemon
 90                 ;;
 91
 92         status)
 93                 get_pid
 94                 if [ -z "$PID" ] ; then
 95                         echo "${DAEMON_DESC} is not running."
 96                 else
 97                         echo "${DAEMON_DESC} is running. PID is $PID"
 98                 fi
 99                 ;;
100
101         *)
102                 echo "Usage: $0 {start|stop|status|restart}"
103                 exit 1
104                 ;;
```

```
105  esac
106
107  exit 0
108
109  # --- EOF ---
```

Erklärung:

Zeile 10-18: Hier wird der daemon durch eigene Modifikationen instanziiert.

Zeile 26-29: Funktion findet die process ID des daemons.

Zeile 37-42: Funktion startet den daemon.

Zeile 50-55: Funktion stoppt den daemon.

Zeile 63-107: Body des Skripts. Hier werden die Parameter ausgewertet.

Das Skript akzeptiert jeweils einen von vier Parameter:

start: Startet den daemon.

stop: Stoppt den daemon, wenn er läuft.

status: Informiert darüber, ob der daemon läuft oder nicht.

restart: Stoppt den daemon (falls er läuft) und startet ihn danach wieder neu.

Vorteil eines solchen generisches Ansatzes ist, dass die Steuerung der *daemons* einheitlich ist und man sich nicht für jeden diverse Besonderheiten merken muss.

Beispiele

Indem man zwischen Zeile 10 und Zeile 18 des Skripts Änderungen macht, kann man sich leicht eigene Dämonen bauen.

File-sharing (overnet)

Erstes Fallbeispiel: Der user "horst-kevin" möchte am file sharing teilnehmen und zwar mit einem overnet core (man kann natürlich auch den edonkey core benutzen). Zu diesem Zweck hat er sich in seinem Home-Verzeichnis ein Verzeichnis ed2k angelegt, in dem das Executable liegt und später auch das Logfile. Die Änderungen am Skelett sind einfach folgende:

Änderungen für overnet core
DAEMON_DESC="overnet core 0.51.2"
DAEMON_BIN="/home/horst-kevin/ed2k/overnet0.51.2"
DAEMON_PROC="overnet0.51.2"
DAEMON_LOGFILE="/home/horst-kevin/ed2k/${DAEMON_PROC}.log"
DAEMON_OPTIONS=" - ! -g -l "

Wenn man kein logfile erzeugen möchte nimmt man einfach /dev/null. Man benötigt hier keine root-Rechte. Zum Starten:

>> ./overnetd.sh start

eingeben. Der overnet core hat die (nicht ungewöhnliche) Eigenschaft mehrere Threads zu erzeugen, mit der unangenehmen Nebenwirkung, daß diese alle in der Prozessliste mit dem selben Namen auftauchen. Dies ist aber kein Problem. Mit

>> ./overnetd.sh stop

werden alle wieder gekillt (und der Spuk ist vorbei).

Dedicated game server (Unreal Tournament)

Zweites Fallbeispiel: Der user "horst-kevin" spielt gern UT mit anderen Spielern. Nun möchte er auf seinem älteren Zweitrechner, dessen Resourcen für heutige Verhältnisse recht bescheiden sind, für einen UT dedicated server aber völlig ausreichend sind, einen eben solchen aufbauen. Er macht folgende Änderungen:

```
DAEMON_DESC="UT dedicated server version 436"DAEMON_BIN="/usr/local/games/ut/ucc"
DAEMON_PROC="ucc-bin"
DAEMON_LOGFILE="/home/horst-kevin/${DAEMON_PROC}.log"
DAEMON_OPTIONS1=" server \"DM-Turbine?game=Botpack.DeathMatchPlus\" " DAEMON_OPTIONS2=" -
port=7777 ini=ucc.ini log=ucc.log"
DAEMON_OPTIONS="${DAEMON_OPTIONS1}${DAEMON_OPTIONS2}"
```

Der Server läßt sich problemlos in der Konsole oder remote über ein Netzwerk in einer telnet oder ssh Sitzung Starten und Stoppen wie oben beschrieben.

Fazit

Die Dämonen in einem UNIX-System sollten nun ihren Schrecken verloren haben. Der geneigte Leser sollte darüber hinaus nun in der Lage sein, auch eigene Dämonen zu erschaffen und zu steuern. Das Erstellen beispielsweise eines Quake 3 dedicated servers ist analog zu obigem UT-Beispiel problemlos möglich.

Weitere Quellen:

http://www.netzmafia.de/skripten/unix/linux-daemon-howto.html

https://sites.google.com/site/wzhang85/creatingadaemonindebianlinux

http://esa-matti.suuronen.org/blog/2012/07/22/creating-kiss-daemons-in-linux/

http://www.principlesofprogram.com/concepts/Linux-Daemon

Linux System Information Decoded

Quelle: http://www.linux-mag.com/cache/7768/1.html

System Administrator Dilemma #942: Send a Data Center Service Tech out to pop open the case to tell you what's in your system or use two simple commands.

Ken Hess Friday, April 23rd, 2010

Do you rely on *proc* files or *dmesg* to tell you everything you need to know about a system? If you do, you're only seeing part of the picture. But what about when you want more detailed information about the system on which you're working? Do you have to have the manufacturer's spec sheet handy to know which components your system contains? It's inconvenient to have someone open a system case in a remote data center to tell you how many memory DIMMs are in, or are not in, your target system. Linux systems include two native commands that tell you almost everything you need to know: *dmidecode* and *biosdecode*.

These two utilities live in the */usr/sbin* directory and are standard Linux executables (not shell scripts). You must have root priveleges, via *sudo* or *su* -, to run either of these commands.

bios, dmi, linux, smbios system information

BIOSDECODE

The *biosdecode* command prints, to screen, information from BIOS memory about all of its known entry points. Entry point types are:

BIOS Information

- SMBIOS - System Management BIOS
- DMI - Desktop Management Interface
- SYSID
- PNP - Plug and Play
- ACPI - Advanced Configuration and Power Interface
- BIOS32 - Compaq-specific
- PIR - PCI IRQ Routing
- SNY - Sony-specific
- VPD - IBM-specific

To show information about your computer, enter the *biosdecode* command at a prompt. Remember to use *sudo* or *su* -. The output shown is for my Linux system.

```
$ sudo biosdecode

# biosdecode 2.7
SMBIOS 2.3 present.
        Structure Table Length: 2008 bytes
        Structure Table Address: 0x000F0000
        Number Of Structures: 46
        Maximum Structure Size: 258 bytes
ACPI 1.0 present.
        OEM Identifier: ACRSYS
        RSD Table 32-bit Address: 0xAFFF3040
BIOS32 Service Directory present.
        Revision: 0
        Calling Interface Address: 0x000FADB0
PNP BIOS 1.0 present.
        Event Notification: Not Supported
        Real Mode 16-bit Code Address: F000:BA38
        Real Mode 16-bit Data Address: F000:0000
        16-bit Protected Mode Code Address: 0x000FBA10
        16-bit Protected Mode Data Address: 0x000F0000
```

As you can see, my system doesn't offer a huge amount of information or detail. Some systems show much more useful information. For example, if you have a Compaq, Sun, IBM or Sony system, your *biosdecode* output will display specific information about your system that generic ones (Acer), like mine, do not.

DMIDECODE

The more interesting and more verbose of these two commands is *dmidecode*. This command dumps the DMI or SMBIOS information, to screen, in a human-readable format. While this tool is respectable in terms of its speed and volume of information, the information could prove unreliable. As stated by *dmidecode*'s authors (Alan Cox and Jean Delvare): "More often than not, information

contained in the DMI tables is inaccurate, incomplete or simply wrong."

```
$ sudo dmidecode

# dmidecode 2.7
SMBIOS 2.3 present.
46 structures occupying 2008 bytes.
Table at 0x000F0000.

Handle 0x0000, DMI type 0, 20 bytes.
BIOS Information
        Vendor: Phoenix Technologies,
LTD
        Version: R01-B4
        Release Date: 04/27/2007
        Address: 0xE0000
        Runtime Size: 128 kB
        ROM Size: 512 kB
        Characteristics:
                ISA is supported
                PCI is supported
                PNP is supported
                APM is supported
                BIOS is upgradeable
                ...
```

As discomforting as the above statement seems, *dmidecode* can give you a "quick and dirty" peek into your hardware configuration. Matching its output against your *proc* files (cpuinfo, meminfo, pci) and *dmesg* information should provide you with enough verification to either accept its output or to send someone to the data center for a physical inspection.

For the moment, we're going to consider the information displayed by *dmidecode* accurate. By default, *dmidecode* assumes that you want all of the information it has about your system. If you enter the command with no options at the command line, you'll see that it directs pages of output to your screen.

Since this amount of information is a bit much to digest all at once, you can request specific information from *dmidecode* by using options. To have a look at all possible options, enter the following command.

Now you have the ability to query specific DMI *types* from the DMI table. A type is a number (0 through 39) or one of the keywords shown above.

The DMI type numbers, shown below, used individually or in a comma separated list provide you with targeted information.

```
$ sudo dmidecode -t system

# dmidecode 2.7
SMBIOS 2.3 present.

Handle 0x0001, DMI type 1, 25 bytes.
System Information
        Manufacturer: Acer
        Product Name: Aspire E380
        Version: R01-B4
        Serial Number: PTS550X050723069912704
        UUID: 001921EA-AD8E-2007-0606-082404000000
        Wake-up Type: Power Switch

Handle 0x002C, DMI type 32, 11 bytes.
System Boot Information
        Status: No errors detected
```

```
$ sudo dmidecode -t

Type number or keyword expected
Valid type keywords are:
  bios
  system
  baseboard
  chassis
  processor
  memory
  cache
  connector
  slot
```

0 BIOS 1 System 2 Base Board 3 Chassis 4 Processor 5 Memory Controller 6 Memory Module 7 Cache 8 Port Connector 9 System Slots 10 On Board Devices 11 OEM Strings 12 System Configuration Options 13 BIOS Language 14 Group Associations 15 System Event Log	16 Physical Memory Array 17 Memory Device 18 32-bit Memory Error 19 Memory Array Mapped Address 20 Memory Device Mapped Address 21 Built-in Pointing Device 22 Portable Battery 23 System Reset 24 Hardware Security 25 System Power Controls 26 Voltage Probe 27 Cooling Device 28 Temperature Probe 29 Electrical Current Probe 30 Out-of-band Remote Access 31 Boot Integrity Services	32 System Boot 33 64-bit Memory Error 34 Management Device 35 Management Device Component 36 Management Device Threshold Data 37 Memory Channel 38 IPMI Device 39 Power Supply

Type Information

I know my system very well and the information reported from *dmidecode* is accurate but your mileage may vary. You can see the value and the amount of information extracted from *biosdecode* and *dmidecode* and why you need to know about these valueable tools.

If your system, gives you inaccurate information, please let us know and you should also report it to the developers. Send them as much info as you can, including the output and point out the inaccuracies to them.

```
$ sudo dmidecode -t 1
# dmidecode 2.7
SMBIOS 2.3 present.

Handle 0x0001, DMI type 1, 25 bytes.
System Information
        Manufacturer: Acer
        Product Name: Aspire E380
        Version: R01-B4
        Serial Number: PTS550X050723069912704
        UUID: 001921EA-AD8E-2007-0606-082404000000
        Wake-up Type: Power Switch

$ sudo dmidecode -t 1,19
# dmidecode 2.7
SMBIOS 2.3 present.

Handle 0x0001, DMI type 1, 25 bytes.
System Information
        Manufacturer: Acer
        Product Name: Aspire E380
        Version: R01-B4
        Serial Number: PTS550X050723069912704
        UUID: 001921EA-AD8E-2007-0606-082404000000
        Wake-up Type: Power Switch

Handle 0x0027, DMI type 19, 15 bytes.
Memory Array Mapped Address

        Starting Address: 0x00000000000
        Ending Address: 0x000BFFFFFFF
        Range Size: 3 GB
        Physical Array Handle: 0x0022
        Partition Width: 0
```

Erweiterte Systemüberwachung mit rsyslog

Andrea Müller

rsyslog überwacht unter Linux Anwendungen und Betriebssystem

Linux ist die reinste Plaudertasche – was die Systemdienste und der Kernel so melden, landet, oft recht unübersichtlich, in Log-Dateien. Der Dienst rsyslog erleichtert es dem Admin, relevante Informationen in den Systemprotokollen zu finden.

Auf einem Linux-System geschieht mehr, als der Nutzer auf dem Desktop sieht: Dienste starten, der Kernel erkennt neue Hardware, andere Nutzer melden sich per **SSH[1]** an und Cron-Jobs erledigen wichtige Aufgaben wie Backups oder das Aktualisieren der updatedb. Solange das System rund läuft, ist man zufrieden, davon nichts mitzubekommen, aber spätestens wenn es einmal hakt, freut man sich über die Linux-typische Geschwätzigkeit. Kernel und Dienste führen penibel Buch über jedes Ereignis und vermerken es in den Log-Dateien des Systems, die im Verzeichnis /var/log und seinen Unterverzeichnissen liegen. Viele Linux-Distributionen sind von Haus aus so eingerichtet, dass sie den Dienst syslogd zum Erfassen der Protokollinformationen installieren. Diese Implementierung des **Syslog-Protokolls[2]** ist sehr verbreitet, wird aber spätestens dann unübersichtlich, wenn man auf einem Server Dienste anbietet und gigantische Mengen an Protokollinformationen anfallen.

Sich per grep & Co. die relevanten Einträge herauszufischen, kostet nicht nur Zeit, sondern auch Nerven. Ein Ausweg aus dem Dilemma ist der Log-Daemon **rsyslog[3]**, der 2004 unter der Federführung von Rainer Gerhards entstand und in Konkurrenz mit **syslog-ng[4]** treten sollte, einem weiteren Log-Dienst, der mehr Funktionen als der klassische syslogd bietet. rsyslog hat seinem älteren Verwandten einiges voraus: Er lässt sich flexibler konfigurieren, beherrscht diverse Filterfunktionen, kann die Systemprotokolle per TCP an andere Rechner übermitteln und lässt sich leicht mit den beiden Datenbanken MySQL und PostgreSQL verbinden. Filterfunktionen und Datenbankanbindung helfen dabei, wichtige Einträge schneller zu finden. Und wer sich nicht mit SQL-Syntax anfreunden mag, greift stattdessen zum Helferlein PHPLogCon, mit dem man die in eine Datenbank eingespeisten Log-Dateien bequem via Web-Browser sichtet. Komplett umstellen müssen sich Admins beim Umstieg nicht, denn rsyslog kennt in seiner Konfigurationsdatei dieselben Optionen wie syslog und packt nur noch einige neue dazu.

Die meisten Distributionen, wie hier OpenSuse, teilen rsyslog in mehrere Pakete auf. Bei den meisten Distributionen besteht die erste Aufgabe darin, den standardmäßig installierten syslogd durch rsyslog zu ersetzen – Fedora-Nutzern bleibt dieser Arbeitsschritt erspart, da die Distribution rsyslog als Default-Log-Dienst verwendet. Alle anderen verbreiteten Distributionen bringen zwar rsyslog-Pakete mit, aber die muss der Anwender über die Software-Verwaltung des Systems selbst einspielen.

Einige Distributionen wie OpenSuse und Mandriva teilen rsyslog in mehrere Pakete auf, sodass Sie beispielsweise neben dem Log-Dienst selbst auch das Paket *rsyslog-mysql* einspielen müssen, wenn Sie das Einspeisen der Systemprotokolle in eine Datenbank nutzen möchten. Wer PostgreSQL bevorzugt, wählt stattdessen das Paket *rsyslog-postgresql* aus. Je nach Distribution ist im rsyslog-Paket hinterlegt, dass es mit dem klassischen syslogd einen Konflikt gibt und das System wird Ihnen anbieten, den syslogd zu entfernen. Ist das nicht der Fall, wie beispielsweise bei Mandriva 2009, müssen Sie selbst Hand anlegen. Beenden Sie zunächst den syslogd mit dem Befehl

```
/etc/init.d/syslogd stop
```

und entfernen Sie ihn dann mit

```
chkconfig -del syslogd
```

aus der Liste der automatisch startenden Dienste. Autostart und sofortiges Hochfahren des rsyslog-Diensts erreichen Sie mit den beiden Befehlen

```
chkconfig -add rsyslogd
```

```
/etc/init.d/rsyslogd start
```

Das syslog-Paket können Sie nun entfernen oder Sie lassen es installiert, falls Sie später doch wieder zum klassischen Log-Dienst zurückkehren möchten.

Als Distributionspaket installiert, ändert sich erst einmal nichts, da den Paketen bereits fertige Konfigurationsdateien für rsyslog beiliegen, und auch die Log-Dateien im Verzeichnis /var/log unterscheiden sich nicht von denen des Vorgängers. Wer in der Konfiguration stöbern und sie auch anpassen möchte, findet zum einen die zentrale Einrichtungsdatei /etc/rsyslog.conf, die bei vielen Distributionen jedoch nur die Include-Anweisungen

```
$IncludeConfig /etc/syslog.d/*.conf

$IncludeConfig /etc/syslog.conf
```

enthält. Sie weisen rsyslog an, alle Optionen in der Datei syslog.conf sowie die in den auf „.conf“ endenden Dateien im Verzeichnis /etc/syslog.d zu beachten. In diesem Ordner liegt nach der Installation des Daemons zumeist nur die Datei 00_common.conf und eventuell noch 01_mysql.conf oder 01_postgresql.conf, sofern Sie die Module für die Datenbankanbindung installiert haben.

Eine Spezialität von rsyslog sind Templates, mit denen Sie das Ausgabeformat der Meldungen anpassen. Einige dieser Vorlagen sind fest in rsyslog verdrahtet und per Default benutzt er das traditionelle Ausgabeformat, das auch der syslogd schreibt. Eine praktische Alternative ist das mitgelieferte Template, das vor jeder Meldung die sogenannte Facility und die Priorität der Meldung vermerkt. Insgesamt kennt das Syslog-Protokoll laut RFC sechzehn vordefinierte Facilities mit den Nummern 0–15, unter anderem Kernel-Messages, solche des Mail-Subsystems sowie Netzwerk- und Authentifizierungsmeldungen. Weitere acht Facilities (local0 bis local7) sind für allgemeine Meldungen vorgesehen. Die Priorität der Meldungen reicht von Debug (Priority 7) über Warnungen (Priority 4) bis zu Emergency-Meldungen (Priority 0).

Das Template, das die Facility und Priority in die Log-Datei schreibt, heißt TraditionalFormatWithPRI und Sie binden es über die folgende Zeile in die rsyslog.conf ein:

```
$template TraditionalFormatWithPRI,"%PRI-text%:%timegenerated% %HOSTNAME% -
%syslogtag%msg:::drop-last-lf%\n"
```

Das Schlüsselwort *$template* sagt rsyslog, dass diese Zeile eine Vorlage enthält, danach folgt der Name des Templates, über das man die Vorlage später einbindet. Der Text in Anführungszeichen ist die eigentliche Vorlage, die das Format festlegt, mit dem rsyslog die Meldungen ins Protokoll schreibt. Bei allen in Prozentzeichen eingeschlossenen Anweisungen handelt es sich um Variablen, im Beispiel oben Facility und Priorität, Zeitstempel, Rechnername, Tag (der Name des Programms, das die Meldung erzeugt) und die Meldung. Zeichen, die nicht von Prozentzeichen umschlossen sind, wie der Doppelpunkt vor dem Zeitstempel, landen 1:1 in der Logdatei.

Damit rsyslog diese Vorlage benutzt, muss man sie mit einer Ausgaberegel verknüpfen. So sorgt beispielsweise die Zeile

```
*.* -/var/log/syslog
```

in der 00_common.conf dafür, dass alle Meldungen in der Datei /var/log/syslog landen. Mit einem Semikolon gefolgt vom Template-Namen, verbinden Sie die Vorlage mit der Ausgaberegel:

```
*.* -/var/log/syslog;TraditionalFormatWithPRI
```

Nach einem Neueinlesen der Konfiguration mit SIGHUP schreibt rsyslog Meldungen nach dieser Vorlage, hier einen gescheiterten Anmeldeversuch via SSH:

```
auth.info<38>: Sep 14 11:15:39 doomtrain sshd[4247]:- Failed password for andi
from 192.168.0.66 port 43229 - ssh2
```

Ein eigenes Template zu erstellen, funktioniert nach demselben Muster. So befüllt

```
$template Myown,"%programname% meldet am - %timegenerated%: %msg:::drop-last-lf%
 mit Priorität- %%syslogseverity%\n" *.* -/var/log/blubber;Myown
```

die Datei /var/log/blubber mit Meldungen der Form:

```
CROND meldet am Sep 14 12:01:01: (root) CMD (nice -n - 19 run-parts –report
/etc/cron.hourly) - mit Priorität 6
```

Eine vollständige Liste der Variablen, die rsyslog kennt, finden Sie in der Manpage von rsyslog.conf.

---

rsyslog kennt eine ganze Reihe Filteroptionen, mit denen Sie Meldungen häppchenweise auf mehrere Dateien verteilen. Die klassische Form der Filterung haben Sie schon im Beispiel oben mit der Zeile `*.* -/var/ log/syslog` kennengelernt – sie leitet alle Meldungen jeder Priorität in die Datei /var/log/syslog. Möchte man die Meldungen des Mail-Subsystems lieber gesondert in der Datei /var/log/mail speichern, erreicht man das mit der Regel

```
mail.* -/var/log/mail
```

Sollen Meldungen der Prioritätsstufe Critical in einer eigenen Datei landen, tragen Sie eine Zeile wie die folgende in die rsyslog-Konfiguration ein:

```
*.=crit -/var/log/critical
```

Fehlermeldungen des Mail-Systems würden Sie mit dem Filter mail.=err herausfischen. Haben Sie weitergehende Filterbedürfnisse, können Sie auch mit sogenannten „Expression Based Filters“ arbeiten. Dabei können Sie unter anderem if-Abfragen mit mehreren Bedingungen nutzen:

```
if $syslogfacility-text == 'CROND' and ($msg contains – 'backup' and $msg
 contains 'error') – then /var/log/backup-problem
```

Bei diesem Beispiel schreibt rsyslog alle Meldungen der Cron-Facility, die die Zeichenketten „backup“ und „error“ enthalten, in die Datei /var/log/backup-problem. Bei diesem Beispiel unterscheidet rsyslog zwischen Groß- und Kleinschreibung. Soll der Dienst das nicht tun, verwenden Sie contains_i anstelle von contains. Beim Speichern der Meldungen sind Sie nicht auf lokale Dateien beschränkt. rsyslog kann die Systemprotokolle via UDP oder TCP an einen anderen Rechner mit rsyslog-Daemon übertragen. Das Remote-Logging ist schnell eingerichtet: Auf dem sendenden Rechner ergänzen Sie die Konfigurationsdatei um die Zeile der Art

```
 *.* @@IP/hostname
```

beim empfangenden Rechner tragen Sie Folgendes in die rsyslog-Konfiguration ein:

```
$ModLoad imtcp.so
$InputTCPServerRun 514
```

Soll die Remote-Übertragung via UDP laufen, entfernen Sie auf dem sendenden Rechner einfach eines der @-Zeichen und tragen in die Empfängerkonfiguration diese beiden Zeilen ein:

```
$ModLoad imudp.so
$UDPServerRun 514
```

Noch komfortabler wird das Auswerten der Systemprotokolle, wenn rsyslog die Einträge nicht nur in Textdateien im Verzeichnis /var/log schreibt, sondern die Einträge in der Tabelle einer Datenbank hinterlegt. Besonders fix richten Sie das mit einer MySQL-Datenbank ein. Installieren Sie dazu zunächst die Pakete mysql und mysql-client Ihrer Distribution, sofern die Datenbank noch nicht bei Ihnen läuft. Als ersten Schritt nach der Installation sollten Sie ein Passwort für den Datenbankadministrator root setzen, der nichts mit dem Root-Account des Systems zu tun hat. Das erledigen Sie mit dem Kommando

```
mysqladmin -u root password geheimes_passwort
```

Das Anlegen der Datenbank für rsyslog müssen Sie nicht zu Fuß erledigen, denn dafür bringt der Dienst das Skript createDB.sql mit, das Sie im Ordner /usr/share/doc/rsyslogmysql finden. Mit dem Befehl

```
mysql -u root -p < /usr/share/doc/rsyslog-– mysql/createDB.sql
```

gefolgt vom zuvor festgelegten Root-Passwort für den Datenbankserver, erzeugen Sie die Datenbank Syslog mit den beiden Tabellen SystemEvents und SystemEventsProperties. Jetzt fehlt nur noch der Nutzer, der auf die Datenbank zugreifen darf. Aus Sicherheitsgründen sollte das nicht der Datenbankadministrator root sein.

Process Monitor v3.5 (Windows)

By Mark Russinovich
Published: February 13, 2018

 **Download Process Monitor (981 KB)**
Run now from Sysinternals Live.

Einführung

Process Monitor ist ein erweitertes Überwachungstool für Windows, mit dem die Dateisystem-, Registrierungs- und Prozess-/Threadaktivitäten in Echtzeit angezeigt werden. Dieses Tool vereint die Features der beiden älteren Dienstprogramme *Filemon* und *Regmon* von Sysinternals und ergänzt diesen Funktionsumfang mit einer Fülle von Verbesserungen, beispielsweise der ausführlichen und nichtdestruktiven Filterung, umfassenden Ereigniseigenschaften (z. B. Sitzungs-IDs und Benutzernamen), zuverlässigen Prozessinformationen, vollständigen Threadstapeln mit integrierter Symbolunterstützung für jeden Vorgang, simultaner Protokollierung in eine Datei und vielem mehr. Die äußerst leistungsfähigen Features machen Process Monitor zu einem wichtigen Bestandteil im Repertoire für die Fehlerbehebung und die Suche nach schädlichen Programmen.

Process Monitor ist unter Windows 2000 SP4 mit Updaterollup 1, Windows XP SP2, Windows Server 2003 SP1 und Windows Vista funktionsfähig, außerdem unter den x64-Versionen von Windows XP, Windows Server 2003 SP1 und Windows Vista.

Zum Seitenanfang

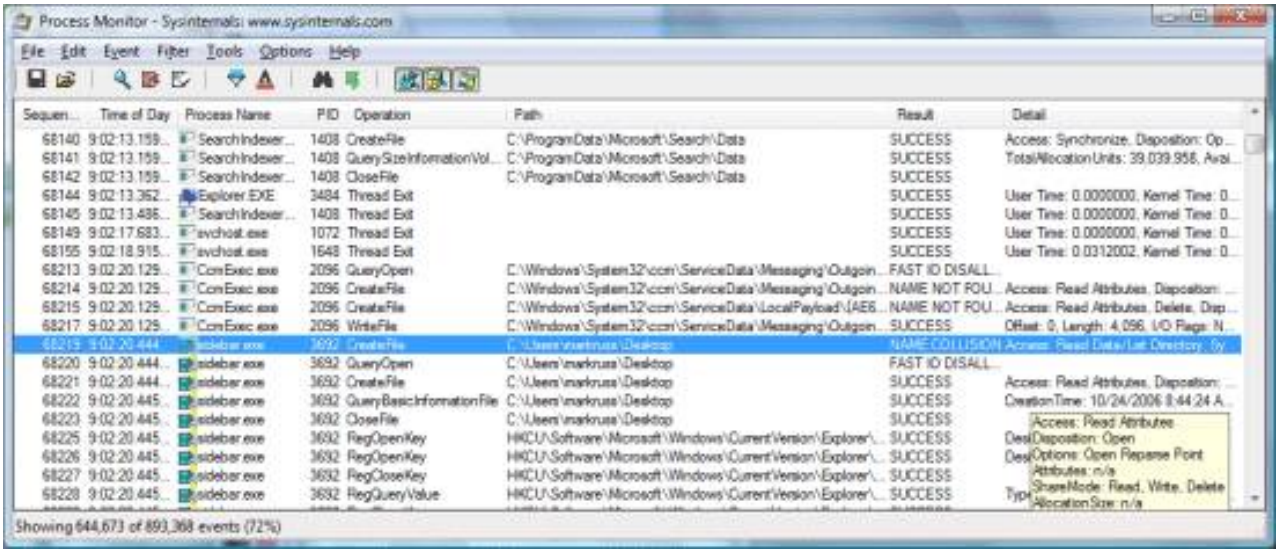
Verbesserungen bei Process Monitor im Vergleich zu Filemon und Regmon

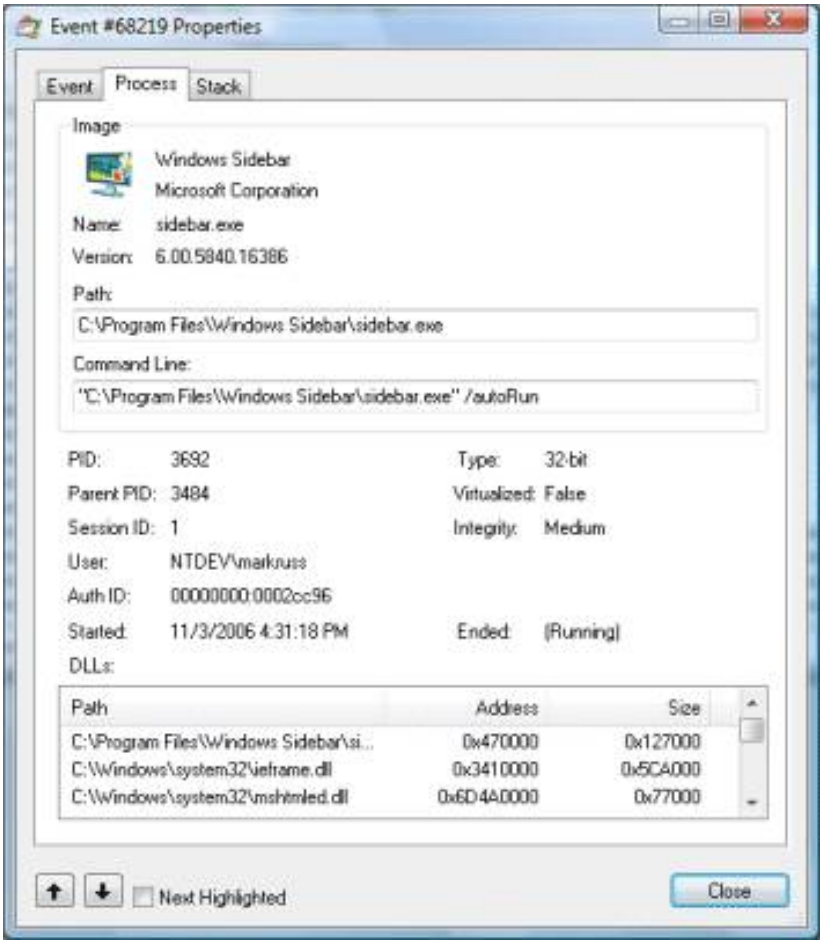
Die Benutzeroberfläche und die Optionen von Process Monitor sind ähnlich aufgebaut wie bei Filemon und Regmon, wurden allerdings von Grund auf neu entwickelt und dabei mit zahlreichen bedeutenden Verbesserungen ausgestattet, beispielsweise:

- Überwachung des Startens und Beendens von Prozessen und Threads (mit Beendungsstatuscodes)
- Überwachung von Abbildladevorgängen (DLL- und Kernelmodus-Gerätetreiber)
- Erfassung umfangreicherer Daten für die Ein- und Ausgabeparameter von Vorgängen
- Nichtdestruktive Filter zum Festlegen von Filtern ohne Datenverlust
- Erfassung von Threadstapeln für jeden Vorgang, sodass die Hauptursache für einen Vorgang in vielen Fällen ersichtlich wird
- Zuverlässige Erfassung von Prozessdetails, z. B. Abbildpfad, Befehlszeile, Benutzer, Sitzungs-ID
- Konfigurierbare und verschiebbare Spalten für sämtliche Ereigniseigenschaften
- Filter für alle Datenfelder festlegbar, auch für Felder, die nicht als Spalten konfiguriert sind
- Erweiterte Protokollierungsarchitektur, auf viele Millionen von erfassten Ereignissen und mehrere Gigabyte an Protokolldaten skalierbar
- Prozessstrukturtool zur Anzeige der Beziehungen aller Prozesse, die in einer Spur genannt werden

- Systemeigenes Protokollformat und damit Beibehaltung aller Daten zum Laden in eine andere Process Monitor-Instanz
- Process-QuickInfo zur unkomplizierten Anzeige von Informationen zu Prozessabbildern
- Detail-QuickInfo für den komfortablen Zugriff auf formatierte Daten, die nicht in die Spalten passen
- Abbrechbare Suche
- Protokollierung aller Vorgänge zum Startzeitpunkt
  Damit Sie rasch mit allen Features von Process Monitor vertraut werden, sollten Sie die Hilfedatei lesen und anschließend die einzelnen Menübefehle und Optionen auf einem Livesystem durchgehen und aufrufen.

Screenshots



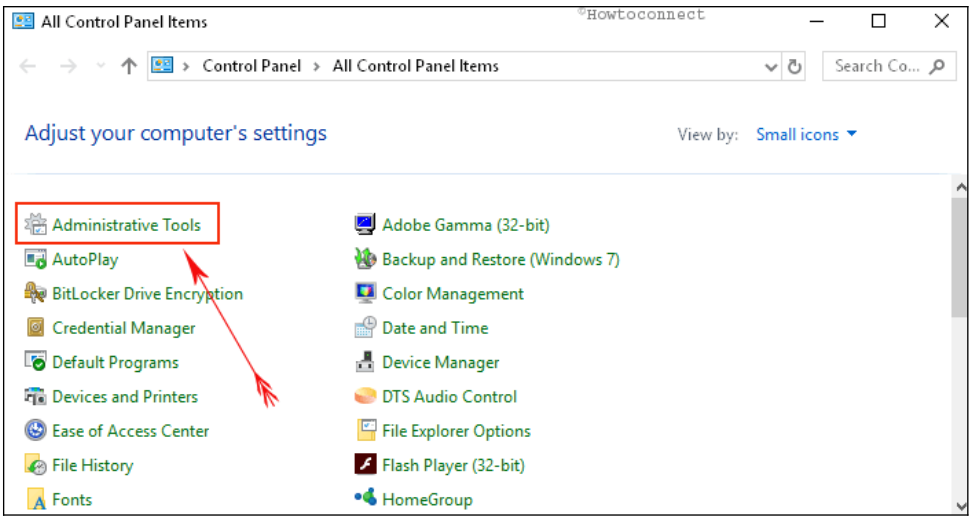


[nixcraft - insight into linux admin work]

Related Links

- **Windows Internals Book**
  The official updates and errata page for the definitive book on Windows internals, by Mark Russinovich and David Solomon.
- **Windows Sysinternals Administrator's Reference**
  The official guide to the Sysinternals utilities by Mark Russinovich and Aaron Margosis, including descriptions of all the tools, their features, how to use them for troubleshooting, and example real-world cases of their use.

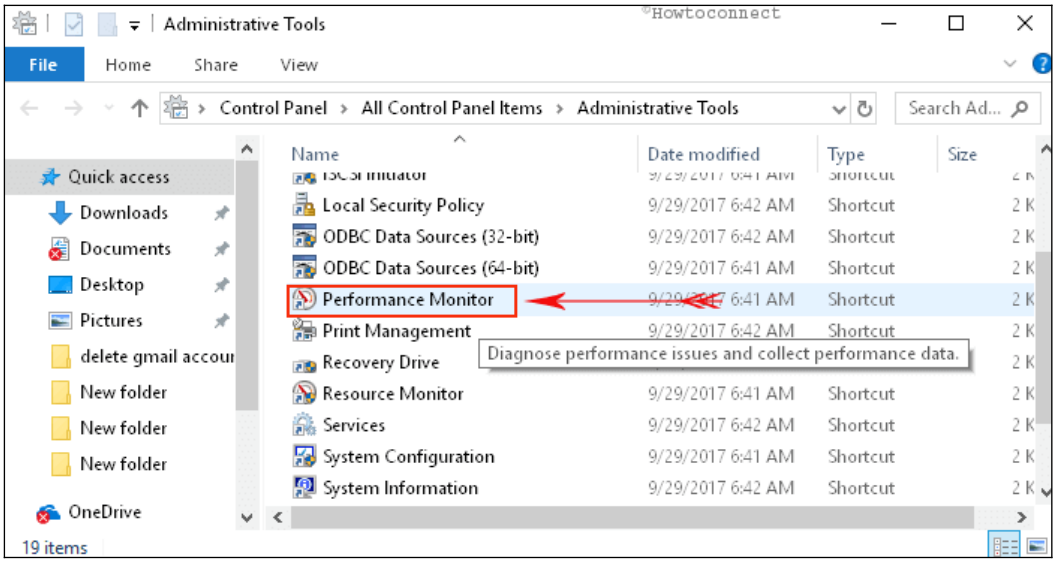
Der Windows10 Performance-Monitor



Get Into Performance Monitor on Windows 10

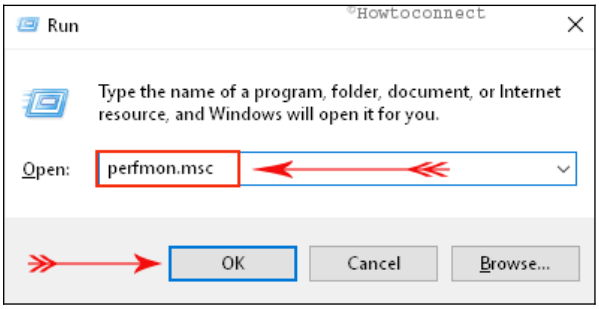
To begin the process, you will have to know the way through which you can get the entry to the **Performance Monitor** application. There are two ways of exploring **Performance Monitor** application.

The first method that I will show you is by getting into the **Control Panel**. In the **Category** view of **Control Panel** select **Administrative Tools.**



A new window of **Administrative Tools** will arrive on the screen. Among the tools listed there look for **Performance Monitor** tool and double-click on it to open.

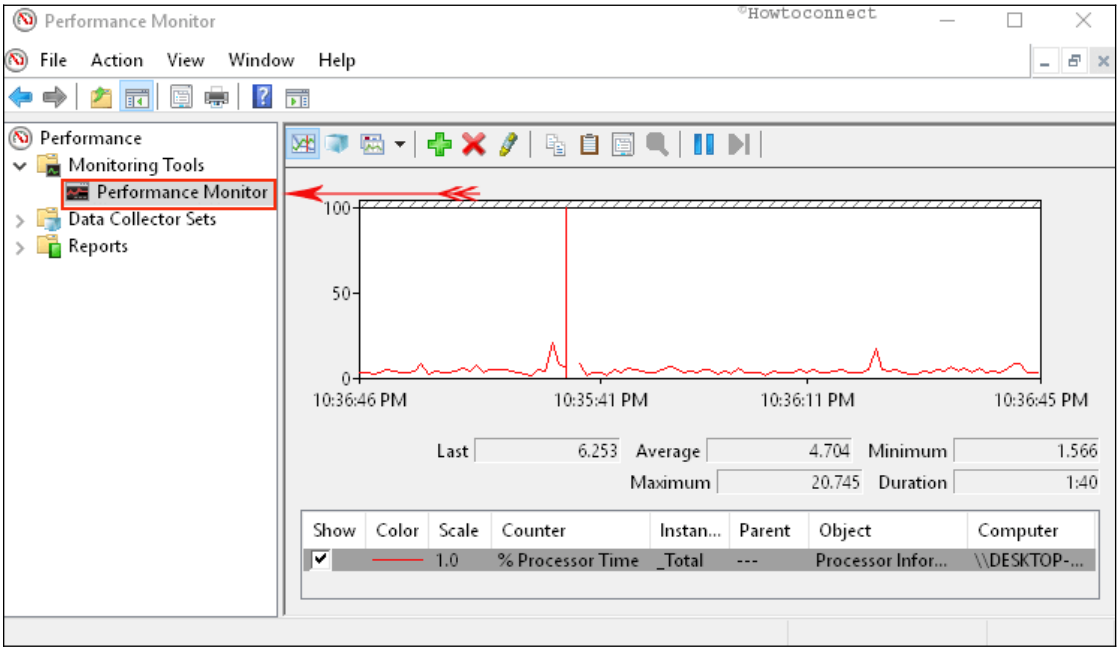
Another way is by launching **Run** command window. When you hit **Windows** and **R** keys **Run** command box will appear on the screen. Type ***perfmon.msc*** in the provided command field and click **Ok**.



Use any of the ways you are into the **Performance Monitor**. Take a look at the below screenshot for its first view.

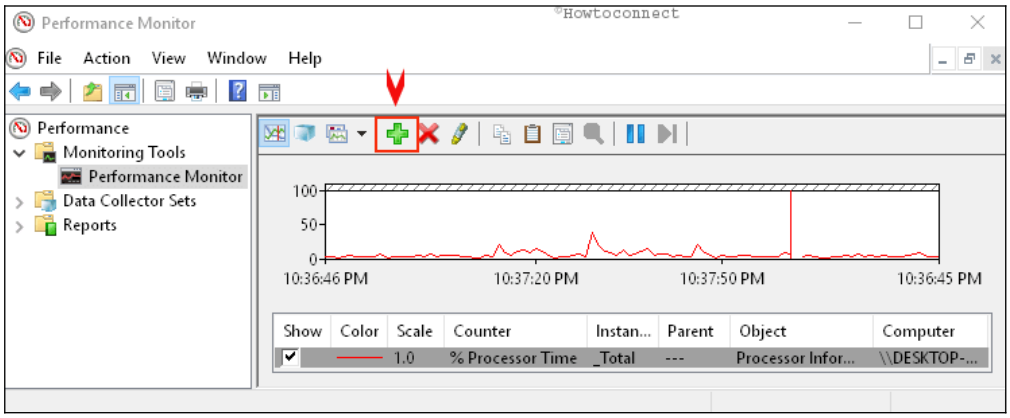
Analyze Your System's Performance on Windows 10

Navigate the following path on the left pane of the **Performance Monitor** window – **Monitoring Tools -> Performance Monitor**.

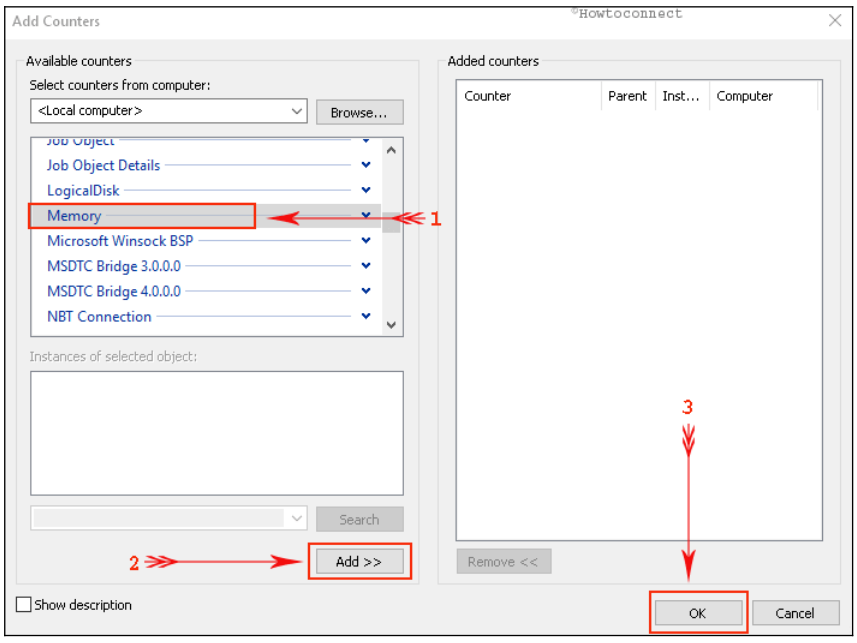


Open some programs on your system so that you will be able to see your system's performance when the programs are being active. The graph will represent you the effects of the programs on your system's mechanism. As you have opened the **Performance Monitor** on Windows 10 for the first time so you will notice a graph. This graph represents the **Processor Time**. The amount of time being displayed in the graph is in terms of percentage. It shows the amount of time being consumed by the Processor when the programs are active on your system. The horizontal line depicts the time and the vertical line is the percentage used.

Performance Monitor allows you to add your required counters so that you can analyze its performance on your system. So click on the **green colored plus sign** on the top of the window.



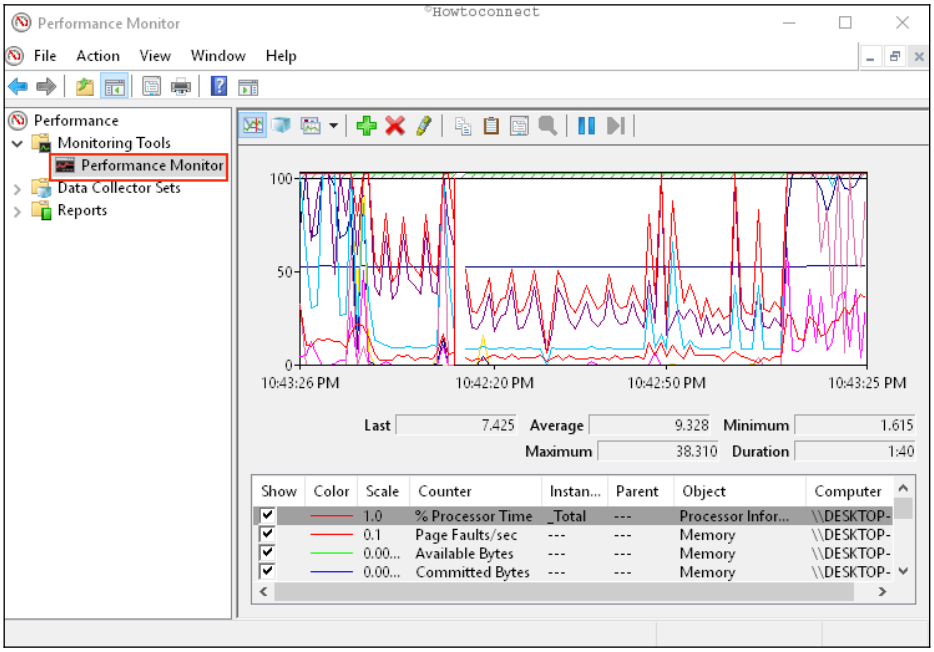
Add counters window will appear now. On the left pane of the window, you will a list of counters available. Scroll up and down to find your required counter (For instance, I have selected the **Memory** counter). Then click on **Add** button followed by **Ok** option.



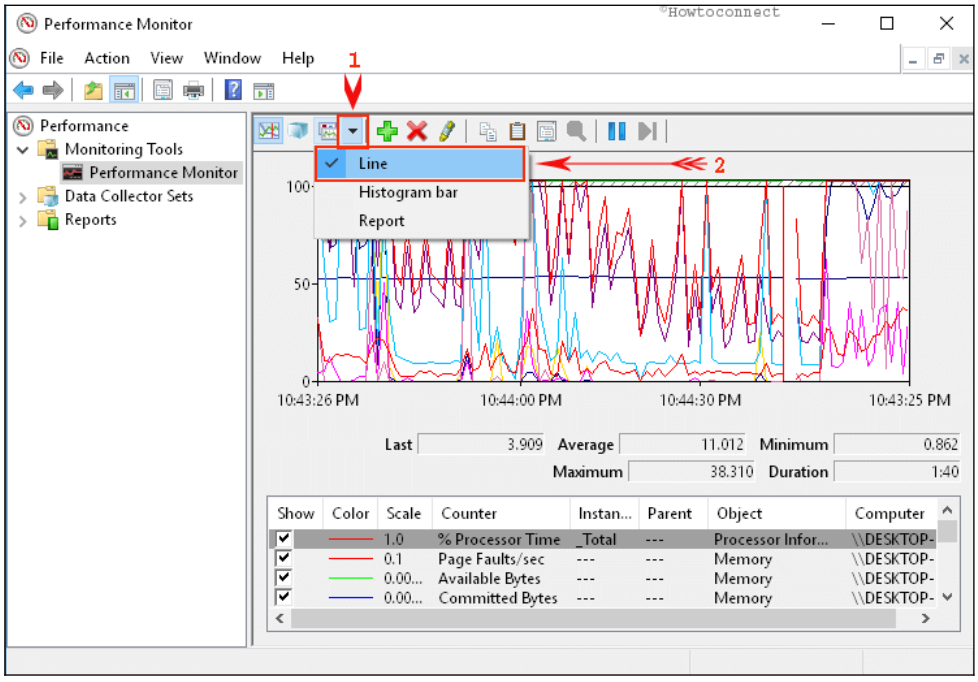
You will view a complicated graph on your **Performance Monitor** on windows 10 Don't get scared with the graph. If looked carefully you will be able to understand its depth. In the lower half of the graph, you will notice many parameters whose performance is shown the above graph. The colors corresponding to

each parameter is the shown in the above graph with the same colors.

If you deselect any parameter then its performance will not be shown in the above graph. Many important parameters like Available Bytes or Committed Bytes are displayed.



For the users who are getting confused with so many colored lines, (I suppose we all are getting confused ;-)) there is a way to make things bit clear. You can view the details in Histogram bar or in Report. When you give a click on the **downward arrow** located beside the green colored plus sign you will notice three options – **Line, Histogram bar and Report**.



To view the details in the **Histogram Bar** select **Histogram bar** option.

To get the **Report** click on **Report** option.

CONCLUSION

I know understanding it is not an easy task and so I request you all to know the casual reports only. You will at least be able to keep a track on your system's performance. Though a tough job but if you try then it will be helpful.

Tagged with

Administrative Tool Guides How-to

Weitere Übungen mit Herrn Scharrer folgen!

Wichtige Linux System Monitoring Tools Every SysAdmin Should Know

by Vivek Gite

Need to monitor Linux server performance? Try these built-in command and a few add-on tools. Most Linux distributions are equipped with tons of monitoring. These tools provide metrics which can be used to get information about system activities. You can use these tools to find the possible causes of a performance problem. The commands discussed below are some of the most basic commands when it comes to system analysis and debugging server issues such as:

1. Finding out bottlenecks.
2. Disk (storage) bottlenecks.
3. CPU and memory bottlenecks.
4. Network bottlenecks.

#1: top - Process Activity Command

The top program provides a dynamic real-time view of a running system i.e. actual process activity. By default, it displays the most CPU-intensive tasks running on the server and updates the list every five seconds.

```
top - 04:14:53 up 1 day, 20:07,  5 users,  load average: 0.53, 0.69, 0.55
Tasks: 187 total,   2 running, 184 sleeping,   0 stopped,   1 zombie
Cpu(s):  4.6%us,  0.5%sy,  0.0%ni, 94.8%id,  0.1%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   8299944k total,  8020784k used,   279160k free,   221276k buffers
Swap:  1951888k total,     2976k used,  1948912k free,  6454792k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 8352 vivek     20   0  248m  75m  25m S   10  0.9   6:21.26 liferea-bin
 4575 vivek     20   0  227m 123m  38m S    5  1.5  86:26.28 delage
25031 vivek     20   0  798m 472m  34m S    3  5.8  43:02.18 firefox-bin
 4073 root      20   0  388m  85m  17m S    2  1.2  88:44.31 Xorg
 7268 vivek     20   0 31412 5240 3820 S    1  0.1  10:26.03 pulseaudio
12582 root      15  -5     0    0    0 S    1  0.0  10:54.52 ntos_wq
32484 vivek     20   0 84512  26m  11m S    0  0.3   0:07.67 gnome-terminal
    1 root      20   0  1908  884  652 S    0  0.0   0:01.69 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.02 migration/0
    4 root      15  -5     0    0    0 R    0  0.0   0:27.71 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/0
    6 root      RT  -5     0    0    0 S    0  0.0   0:00.01 migration/1
    7 root      15  -5     0    0    0 S    0  0.0   0:06.38 ksoftirqd/1
    8 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/1
    9 root      RT  -5     0    0    0 S    0  0.0   0:00.02 migration/2
   10 root      15  -5     0    0    0 S    0  0.0   0:05.59 ksoftirqd/2
   11 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/2
   12 root      RT  -5     0    0    0 S    0  0.0   0:00.03 migration/3
   13 root      15  -5     0    0    0 S    0  0.0   0:05.85 ksoftirqd/3
   14 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/3
   15 root      15  -5     0    0    0 S    0  0.0   0:00.27 events/0
   16 root      15  -5     0    0    0 S    0  0.0   0:00.52 events/1
   17 root      15  -5     0    0    0 S    0  0.0   0:00.44 events/2
   18 root      15  -5     0    0    0 S    0  0.0   0:00.50 events/3
   19 root      15  -5     0    0    0 S    0  0.0   0:00.01 khelper
   61 root      15  -5     0    0    0 S    0  0.0   0:00.00 kintegrityd/0
   62 root      15  -5     0    0    0 S    0  0.0   0:00.00 kintegrityd/1
   63 root      15  -5     0    0    0 S    0  0.0   0:00.00 kintegrityd/2
   64 root      15  -5     0    0    0 S    0  0.0   0:00.00 kintegrityd/3
   66 root      15  -5     0    0    0 S    0  0.0   0:00.05 kblockd/0
   67 root      15  -5     0    0    0 S    0  0.0   0:00.16 kblockd/1
   68 root      15  -5     0    0    0 S    0  0.0   0:00.23 kblockd/2
   69 root      15  -5     0    0    0 S    0  0.0   0:00.11 kblockd/3
   71 root      15  -5     0    0    0 S    0  0.0   0:00.00 kacpid
   72 root      15  -5     0    0    0 S    0  0.0   0:00.00 kacpi_notify
  153 root      15  -5     0    0    0 S    0  0.0   0:00.00 cqueue
  157 root      15  -5     0    0    0 S    0  0.0   0:00.01 kseriod
  211 root      15  -5     0    0    0 S    0  0.0   0:02.25 kswapd0
```

Fig.01: Linux top command

Commonly Used Hot Keys

The top command provides several useful hot keys:

Hot Key	Usage
t	Displays summary information off and on.
m	Displays memory information off and on.
A	Sorts the display by top consumers of various system resources. Useful for quick identification of performance-hungry tasks on a system.
f	Enters an interactive configuration screen for top. Helpful for setting up top for a specific task.
o	Enables you to interactively select the ordering within top.
r	Issues renice command.
k	Issues kill command.
z	Turn on or off color/mono

=> **Related:** How do I Find Out Linux CPU Utilization?

#2: vmstat - System Activity, Hardware and System Information

The command vmstat reports information about processes, memory, paging, block IO, traps, and cpu activity.

```
# vmstat 3
```

Sample Outputs:

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0 2540988 522188 5130400    0    0     2    32    4    2  4  1 96  0  0
 1  0      0 2540988 522188 5130400    0    0     0   720 1199  665  1  0 99  0  0
 0  0      0 2540956 522188 5130400    0    0     0     0 1151 1569  4  1 95  0  0
 0  0      0 2540956 522188 5130500    0    0     0     6 1117  439  1  0 99  0  0
 0  0      0 2540940 522188 5130512    0    0     0   536 1189  932  1  0 98  0  0
 0  0      0 2538444 522188 5130588    0    0     0     0 1187 1417  4  1 96  0  0
 0  0      0 2490060 522188 5130640    0    0     0    18 1253 1123  5  1 94  0  0
```

Display Memory Utilization Slabinfo

```
# vmstat -m
```

Get Information About Active / Inactive Memory Pages

```
# vmstat -a
```

=> **Related:** How do I find out Linux Resource utilization to detect system bottlenecks?

#3: w - Find Out Who Is Logged on And What They Are Doing

w command displays information about the users currently on the machine, and their processes.

```
# w username
# w vivek
```

Sample Outputs:

```
 17:58:47 up 5 days, 20:28,  2 users,  load average: 0.36, 0.26, 0.24
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0    10.1.3.145       14:55    5.00s  0.04s  0.02s vim /etc/resolv.conf
root     pts/1    10.1.3.145       17:43    0.00s  0.03s  0.00s w
```

#4: uptime - Tell How Long The System Has Been Running

The uptime command can be used to see how long the server has been running. The current time, how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15 minutes.

```
# uptime
```

Output:

```
 18:02:41 up 41 days, 23:42,  1 user,  load average: 0.00, 0.00, 0.00
```

1 can be considered as optimal load value. The load can change from system to system. For a single CPU system 1 - 3 and SMP systems 6-10 load value might be acceptable.

#5: ps - Displays The Processes

ps command will report a snapshot of the current processes. To select all processes use the -A or -e option:

```
# ps -A
```

Sample Outputs:

```
  PID TTY          TIME CMD
    1 ?        00:00:02 init
    2 ?        00:00:02 migration/0
    3 ?        00:00:01 ksoftirqd/0
    4 ?        00:00:00 watchdog/0
    5 ?        00:00:00 migration/1
    6 ?        00:00:15 ksoftirqd/1
.....
 4881 ?        00:53:28 java
 4885 tty1     00:00:00 mingetty
 4886 tty2     00:00:00 mingetty
 4887 tty3     00:00:00 mingetty
 4888 tty4     00:00:00 mingetty
 4891 tty5     00:00:00 mingetty
 4892 tty6     00:00:00 mingetty
 4893 ttyS1    00:00:00 agetty
12853 ?        00:00:00 cifsoplockd
12854 ?        00:00:00 cifsdnotifyd
14231 ?        00:10:34 lighttpd
14232 ?        00:00:00 php-cgi
54981 pts/0    00:00:00 vim
55465 ?        00:00:00 php-cgi
55546 ?        00:00:00 bind9-snmp-stat
55704 pts/1    00:00:00 ps
```

ps is just like top but provides more information.

Show Long Format Output

```
# ps -Al
```

To turn on extra full mode (it will show command line arguments passed to process):

```
# ps -AlF
```

To See Threads (LWP and NLWP)

```
# ps -AlFH
```

To See Threads After Processes

```
# ps -AlLm
```

Print All Process On The Server

```
# ps ax
# ps axu
```

Print A Process Tree

```
# ps -ejH
# ps axjf
# pstree
```

Print Security Information

```
# ps -eo euser,ruser,suser,fuser,f,comm,label
# ps axZ
# ps -eM
```

See Every Process Running As User Vivek

```
# ps -U vivek -u vivek u
```

Set Output In a User-Defined Format

```
# ps -eo pid,tid,class,rtprio,ni,pri,psr,pcpu,stat,wchan:14,comm
# ps axo stat,euid,ruid,tty,tpgid,sess,pgrp,ppid,pid,pcpu,comm
# ps -eopid,tt,user,fname,tmout,f,wchan
```

Display Only The Process IDs of Lighttpd

```
# ps -C lighttpd -o pid=
```
OR
```
# pgrep lighttpd
```
OR
```
# pgrep -u vivek php-cgi
```

Display The Name of PID 55977

```
# ps -p 55977 -o comm=
```

Find Out The Top 10 Memory Consuming Process

```
# ps -auxf | sort -nr -k 4 | head -10
```

Find Out top 10 CPU Consuming Process

```
# ps -auxf | sort -nr -k 3 | head -10
```

#6: free - Memory Usage

The command free displays the total amount of free and used physical and swap memory in the system, as well as the buffers used by the kernel.

```
# free
```

Sample Output:

```
             total       used       free     shared    buffers     cached
Mem:      12302896    9739664    2563232          0     523124    5154740
-/+ buffers/cache:    4061800    8241096
Swap:      1052248          0    1052248
```

=> **Related:** :

1. Linux Find Out Virtual Memory PAGESIZE
2. Linux Limit CPU Usage Per Process
3. How much RAM does my Ubuntu / Fedora Linux desktop PC have?

#7: iostat - Average CPU Load, Disk Activity

The command iostat report Central Processing Unit (CPU) statistics and input/output statistics for devices, partitions and network filesystems (NFS).

```
# iostat
```

Sample Outputs:

```
Linux 2.6.18-128.1.14.el5 (www03.nixcraft.in)   06/26/2009

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           3.50    0.09    0.51    0.03    0.00   95.86

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda              22.04        31.88       512.03   16193351  260102868
sda1              0.00         0.00         0.00       2166        180
sda2             22.04        31.87       512.03   16189010  260102688
sda3              0.00         0.00         0.00       1615          0
```

=> **Related:** : Linux Track NFS Directory / Disk I/O Stats

#8: sar - Collect and Report System Activity

The sar command is used to collect, report, and save system activity information. To see network counter, enter:

```
# sar -n DEV | more
```

To display the network counters from the 24th:

```
# sar -n DEV -f /var/log/sa/sa24 | more
```

You can also display real time usage using sar:

```
# sar 4 5
```

Sample Outputs:

```
Linux 2.6.18-128.1.14.el5 (www03.nixcraft.in)          06/26/2009

06:45:12 PM       CPU     %user     %nice   %system   %iowait    %steal     %idle
06:45:16 PM       all      2.00      0.00      0.22      0.00      0.00     97.78
06:45:20 PM       all      2.07      0.00      0.38      0.03      0.00     97.52
06:45:24 PM       all      0.94      0.00      0.28      0.00      0.00     98.78
06:45:28 PM       all      1.56      0.00      0.22      0.00      0.00     98.22
06:45:32 PM       all      3.53      0.00      0.25      0.03      0.00     96.19
Average:          all      2.02      0.00      0.27      0.01      0.00     97.70
```

=> **Related:** : How to collect Linux system utilization data into a file

#9: mpstat - Multiprocessor Usage

The mpstat command displays activities for each available processor, processor 0 being the first one. mpstat -P ALL to display average CPU utilization per processor:

```
# mpstat -P ALL
```

Sample Output:

```
Linux 2.6.18-128.1.14.el5 (www03.nixcraft.in)          06/26/2009

06:48:11 PM  CPU   %user   %nice    %sys %iowait    %irq   %soft  %steal   %idle
intr/s
06:48:11 PM  all    3.50    0.09    0.34    0.03    0.01    0.17    0.00   95.86
```

```
1218.04
06:48:11 PM    0    3.44    0.08    0.31    0.02    0.00    0.12    0.00   96.04
1000.31
06:48:11 PM    1    3.10    0.08    0.32    0.09    0.02    0.11    0.00   96.28
34.93
06:48:11 PM    2    4.16    0.11    0.36    0.02    0.00    0.11    0.00   95.25
0.00
06:48:11 PM    3    3.77    0.11    0.38    0.03    0.01    0.24    0.00   95.46
44.80
06:48:11 PM    4    2.96    0.07    0.29    0.04    0.02    0.10    0.00   96.52
25.91
06:48:11 PM    5    3.26    0.08    0.28    0.03    0.01    0.10    0.00   96.23
14.98
06:48:11 PM    6    4.00    0.10    0.34    0.01    0.00    0.13    0.00   95.42
3.75
06:48:11 PM    7    3.30    0.11    0.39    0.03    0.01    0.46    0.00   95.69
76.89
```

=> **Related:** : Linux display each multiple SMP CPU processors utilization individually.

#10: pmap - Process Memory Usage

The command pmap report memory map of a process. Use this command to find out causes of memory bottlenecks.

```
# pmap -d PID
```

To display process memory information for pid # 47394, enter:

```
# pmap -d 47394
```

Sample Outputs:

```
47394:   /usr/bin/php-cgi
Address           Kbytes Mode  Offset           Device    Mapping
0000000000400000    2584 r-x-- 0000000000000000 008:00002 php-cgi
0000000000886000     140 rw--- 0000000000286000 008:00002 php-cgi
00000000008a9000      52 rw--- 00000000008a9000 000:00000   [ anon ]
0000000000aa8000      76 rw--- 00000000002a8000 008:00002 php-cgi
000000000f678000    1980 rw--- 000000000f678000 000:00000   [ anon ]
000000314a600000     112 r-x-- 0000000000000000 008:00002 ld-2.5.so
000000314a81b000       4 r---- 000000000001b000 008:00002 ld-2.5.so
000000314a81c000       4 rw--- 000000000001c000 008:00002 ld-2.5.so
000000314aa00000    1328 r-x-- 0000000000000000 008:00002 libc-2.5.so
000000314ab4c000    2048 ----- 000000000014c000 008:00002 libc-2.5.so
......
..
00002af8d48fd000       4 rw--- 0000000000006000 008:00002 xsl.so
00002af8d490c000      40 r-x-- 0000000000000000 008:00002 libnss_files-2.5.so
00002af8d4916000    2044 ----- 000000000000a000 008:00002 libnss_files-2.5.so
00002af8d4b15000       4 r---- 0000000000009000 008:00002 libnss_files-2.5.so
00002af8d4b16000       4 rw--- 000000000000a000 008:00002 libnss_files-2.5.so
00002af8d4b17000  768000 rw-s- 0000000000000000 000:00009 zero (deleted)
00007fffc95fe000      84 rw--- 00007ffffffea000 000:00000   [ stack ]
ffffffffff600000    8192 ----- 0000000000000000 000:00000   [ anon ]
mapped: 933712K    writeable/private: 4304K    shared: 768000K
```

The last line is very important:

- **mapped: 933712K** total amount of memory mapped to files
- **writeable/private: 4304K** the amount of private address space
- **shared: 768000K** the amount of address space this process is sharing with others

=> **Related:** : Linux find the memory used by a program / process using pmap command

#11 and #12: netstat and ss - Network Statistics

The command netstat displays network connections, routing tables, interface statistics, masquerade connections, and multicast memberships. ss command is used to dump socket statistics. It allows showing information similar to netstat. See the following resources about ss and netstat commands:

- ss: Display Linux TCP / UDP Network and Socket Information
- Get Detailed Information About Particular IP address Connections Using netstat Command

#13: iptraf - Real-time Network Statistics

The iptraf command is interactive colorful IP LAN monitor. It is an ncurses-based IP LAN monitor that generates various network statistics including TCP info, UDP counts, ICMP and OSPF information, Ethernet load info, node stats, IP checksum errors, and others. It can provide the following info in easy to read format:

- Network traffic statistics by TCP connection
- IP traffic statistics by network interface
- Network traffic statistics by protocol
- Network traffic statistics by TCP/UDP port and by packet size
- Network traffic statistics by Layer2 address

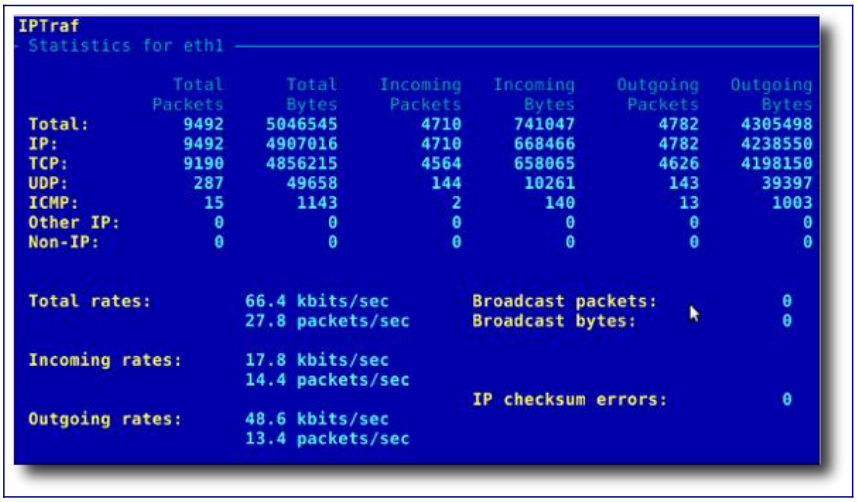


Fig.02: General interface statistics: IP traffic statistics by network interface

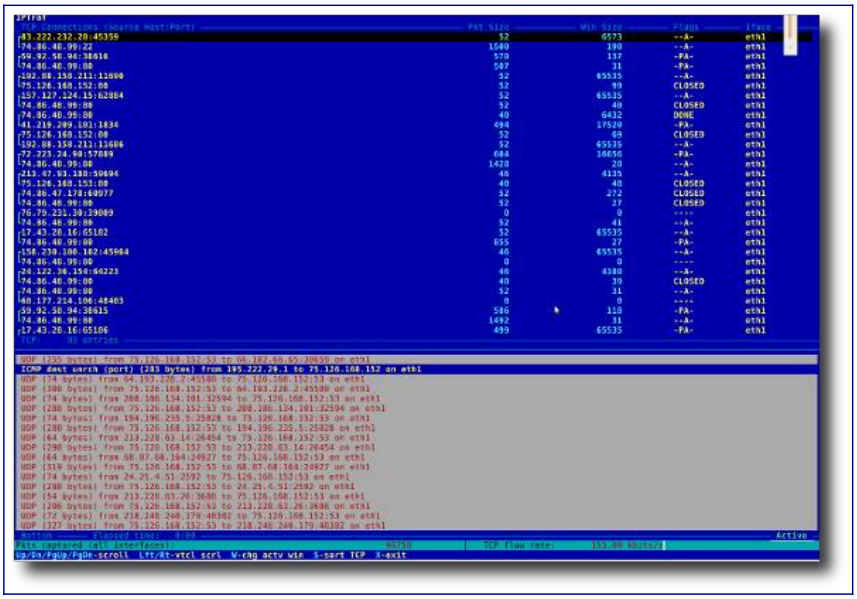


Fig.03 Network traffic statistics by TCP connection

#14: tcpdump - Detailed Network Traffic Analysis

The tcpdump is simple command that dump traffic on a network. However, you need good understanding of TCP/IP protocol to utilize this tool. For.e.g to display traffic info about DNS, enter:

```
# tcpdump -i eth1 'udp port 53'
```

To display all IPv4 HTTP packets to and from port 80, i.e. print only packets that contain data, not, for example, SYN and FIN packets and ACK-only packets, enter:

```
# tcpdump 'tcp port 80 and (((ip[2:2] - ((ip[0]&0xf)<<2)) -
((tcp[12]&0xf0)>>2)) != 0)'
```

To display all FTP session to 202.54.1.5, enter:

```
# tcpdump -i eth1 'dst 202.54.1.5 and (port 21 or 20'
```

To display all HTTP session to 192.168.1.5:

```
# tcpdump -ni eth0 'dst 192.168.1.5 and tcp and port http'
```

Use wireshark to view detailed information about files, enter:

```
# tcpdump -n -i eth1 -s 0 -w output.txt src or dst port 80
```

#15: strace - System Calls

Trace system calls and signals. This is useful for debugging webserver and other server problems. See how to use to trace the process and see What it is doing.

#16: /Proc file system - Various Kernel Statistics

/proc file system provides detailed information about various hardware devices and other Linux kernel information. See Linux kernel /proc documentations for further details. Common /proc examples:

```
# cat /proc/cpuinfo
# cat /proc/meminfo
# cat /proc/zoneinfo
# cat /proc/mounts
```

17#: Nagios - Server And Network Monitoring
(neuere Nachfolger sind: Icinga und Check_MK)

Nagios is a popular open source computer system and network monitoring application software. You can easily monitor all your hosts, network equipment and services. It can send alert when things go wrong and again when they get better. FAN is "Fully Automated Nagios". FAN goals are to provide a Nagios installation including most tools provided by the Nagios Community. FAN provides a CDRom image in the standard ISO format, making it easy to easilly install a Nagios server. Added to this, a wide bunch of tools are including to the distribution, in order to improve the user experience around Nagios.

18#: Cacti - Web-based Monitoring Tool

Cacti is a complete network graphing solution designed to harness the power of RRDTool's data storage and graphing functionality. Cacti provides a fast poller, advanced graph templating, multiple data acquisition methods, and user management features out of the box. All of this is wrapped in an intuitive, easy to use interface that makes sense for LAN-sized installations up to complex networks with hundreds of devices. It can provide data about network, CPU, memory, logged in users, Apache, DNS servers and much more. See how to install and configure Cacti network graphing tool under CentOS / RHEL.

#19: KDE System Guard - Real-time Systems Reporting

KSysguard is a network enabled task and system monitor application for KDE desktop. This tool can be run over ssh session. It provides lots of features such as a client/server architecture that enables monitoring of local and remote hosts. The graphical front end uses so-called sensors to retrieve the information it

displays. A sensor can return simple values or more complex information like tables. For each type of information, one or more displays are provided. Displays are organized in worksheets that can be saved and loaded independently from each other. So, KSysguard is not only a simple task manager but also a very powerful tool to control large server farms.

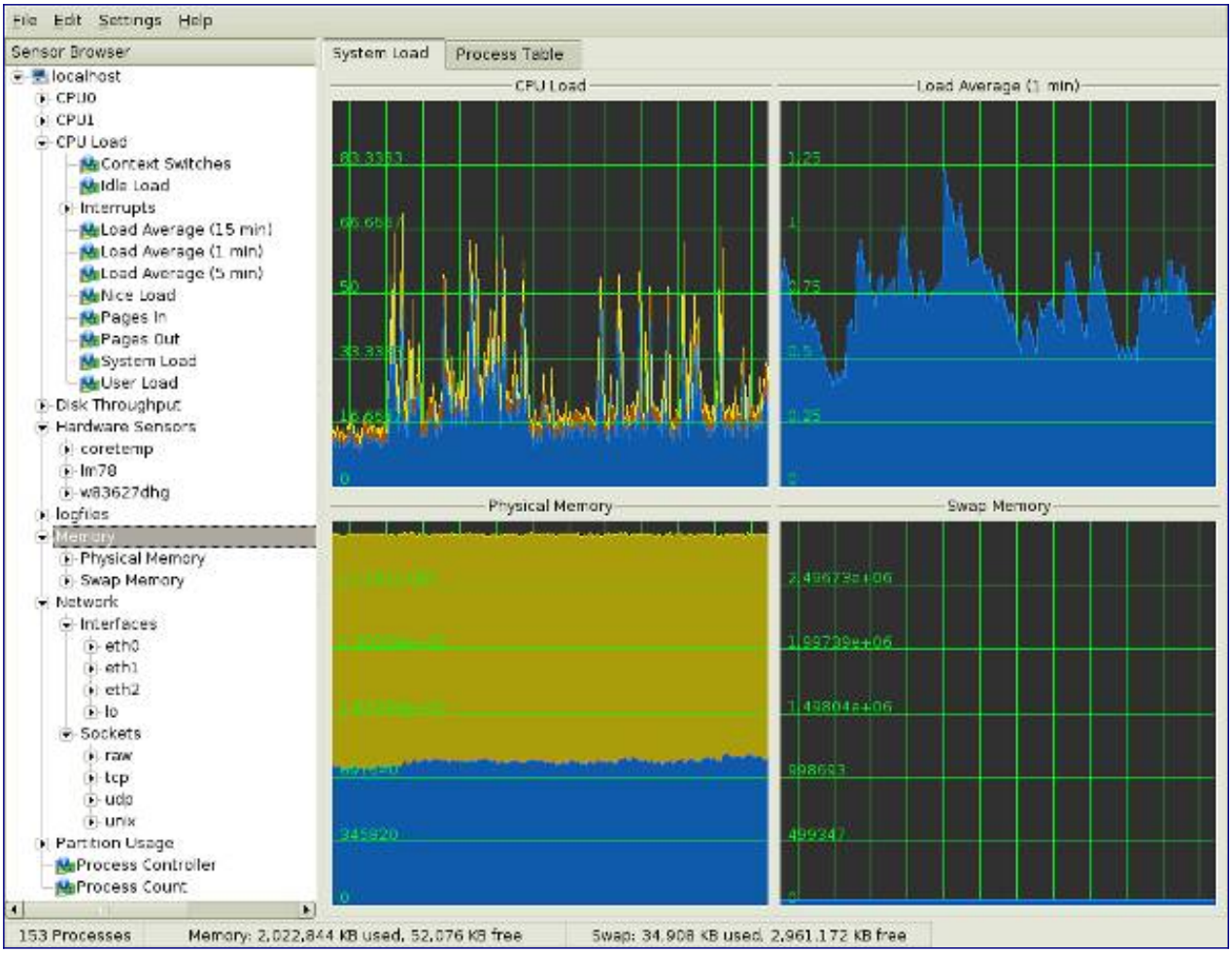


Fig.05 KDE System Guard {Image credit: Wikipedia}

See the KSysguard handbook for detailed usage.

#20: Gnome System Monitor - Real-time Systems Reporting and Graphing

The System Monitor application enables you to display basic system information and monitor system processes, usage of system resources, and file systems. You can also use System Monitor to modify the behavior of your system. Although not as powerful as the KDE System Guard, it provides the basic information which may be useful for new users:

- Displays various basic information about the computer's hardware and software.
- Linux Kernel version
- GNOME version
- Hardware
- Installed memory
- Processors and speeds
- System Status
- Currently available disk space
- Processes
- Memory and swap space
- Network usage
- File Systems
- Lists all mounted filesystems along with basic information about each.

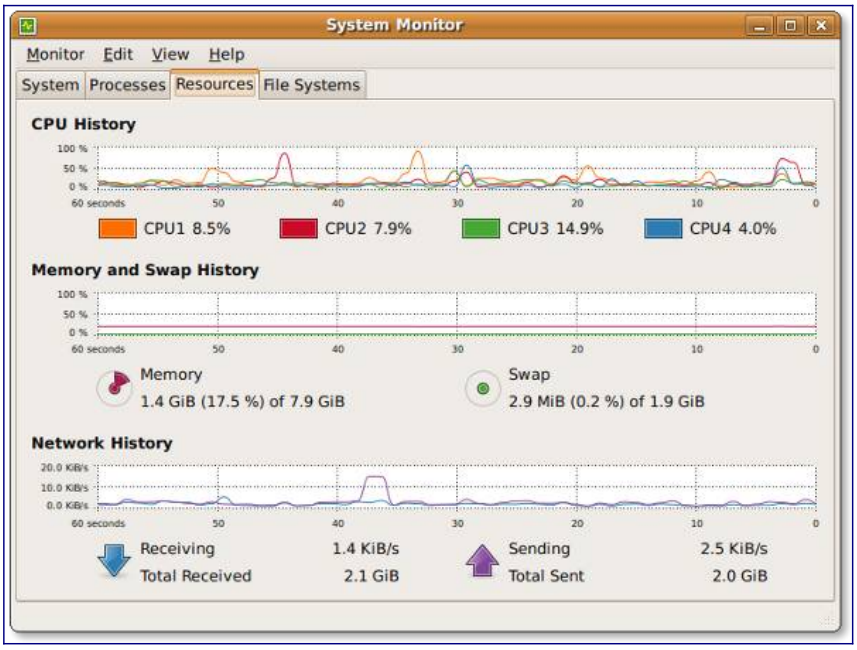


Fig.06 The Gnome System Monitor application

Bounce: Additional Tools

A few more tools:

- nmap - scan your server for open ports.
- lsof - list open files, network connections and much more.
- ntop web based tool - ntop is the best tool to see network usage in a way similar to what top command does for processes i.e. it is network traffic monitoring software. You can see network status, protocol wise distribution of traffic for UDP, TCP, DNS, HTTP and other protocols.
- Conky - Another good monitoring tool for the X Window System. It is highly configurable and is able to monitor many system variables including the status of the CPU, memory, swap space, disk storage, temperatures, processes, network interfaces, battery power, system messages, e-mail inboxes etc.
- GKrellM - It can be used to monitor the status of CPUs, main memory, hard disks, network interfaces, local and remote mailboxes, and many other things.
- vnstat - vnStat is a console-based network traffic monitor. It keeps a log of hourly, daily and monthly network traffic for the selected interface(s).
- htop - htop is an enhanced version of top, the interactive process viewer, which can display the list of processes in a tree form.
- mtr - mtr combines the functionality of the traceroute and ping programs in a single network diagnostic tool.

Did I miss something? Please add your favorite system motoring tool in the comments.

Featured Articles:

- 20 Linux System Monitoring Tools Every SysAdmin Should Know
- 20 Linux Server Hardening Security Tips
- 10 Greatest Open Source Software Of 2009
- My 10 UNIX Command Line Mistakes
- Top 5 Email Client For Linux, Mac OS X, and Windows Users
- Top 20 OpenSSH Server Best Security Practices
- Top 10 Open Source Web-Based Project Management Software
- Top 5 Linux Video Editor Software

Weitere Hinweise:

ulimit

User limits – limit the use of system-wide resources.

Syntax: **ulimit [-acdfHlmnpsStuv] [limit]**

Options:
-S Change and report the soft limit associated with a resource.
-H Change and report the hard limit associated with a resource.

-a All current limits are reported.
-c The maximum size of core files created.
-d The maximum size of a process's data segment.
-f The maximum size of files created by the shell(default option)
-l The maximum size that may be locked into memory.
-m The maximum resident set size.
-n The maximum number of open file descriptors.
-p The pipe buffer size.
-s The maximum stack size.
-t The maximum amount of cpu time in seconds.
-u The maximum number of processes available to a single user.
-v The maximum amount of virtual memory available to the process.

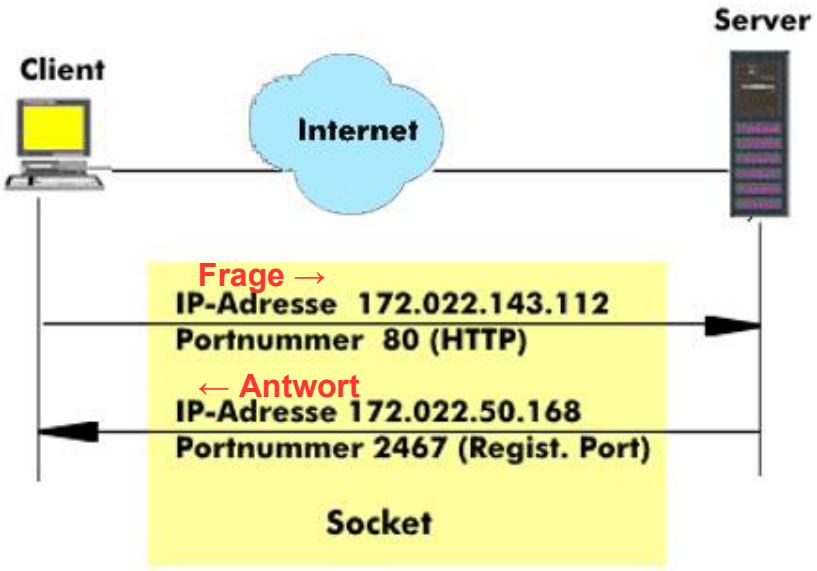
ulimit provides control over the resources available to the shell and to processes started by it, on systems that allow such control.

2 top monitoring tools:

monit: http://mmonit.com/monit/
mrtg : http://oss.oetiker.ch/mrtg/

Der „Socket“

Als **Socket** wird die Adressenkombination aus IP-Adresse und Portnummer bezeichnet mit der eine bestimmte Anwendung auf einem bestimmten Rechner angesprochen werden kann. Mit der IP-Adresse wird das Netzwerk und der Rechner bestimmt und mit der Portnummer die Anwendung ausgewählt.



Merke:

A. Prozesse

1. laufen in einer Ausführungsumgebung (API,Kommandozeile etc.)
2. haben eine Prozess-ID (PID, ganzzahlig positiv)
3. führen ein Programm mit bestimmten Funktionsumfang aus, das mehrere Tasks oder Threads enthalten kann

B. Prozesse greifen auf Ressourcen zu bzw. belegen Ressourcen

1. Speicher
2. Adressraum
3. Netzwerkschnittstellen z.B. TCP-Ports
4. Dateisystem (geöffnete Dateien)

...

C. Prozesse müssen vom Betriebssystem koordiniert und im Ablauf überwacht werden. Mögliche Befehle zur Visualisierung sind z.B.:

1. pstree
2. htop
3. nice
4. wait
5. waitpid
6. kill -9 …

die Signale zur Terminierung von Prozessen:

`SIGALRM` `Term`: Timer abgelaufen (`alarm(2)`,`setitimer(2)`)
`SIGCHLD` (`Ign`): Statusänderung eines Kindprozesses
`SIGINT` (`Term`): Interrupt (Shell: `CTRL-C`)
`SIGQUIT` (`Core`): Quit (Shell: `CTRL-@`)
`SIGKILL` (nicht behandelbar): beendet den Prozess
`SIGTERM` (`Term`): Terminierung; Standardsignal für `kill(1)`
`SIGSEGV` (`Core`): Speicherschutzverletzung
`SIGUSR1, SIGUSR2` (`Term`): Benutzerdefinierte Signale

Unix-Signale

Ein Ereignis wird hier als Signal bezeichnet. Unter Unix gibt es vorgegebene durchnummerierte Signale. Unter Unix kann mit *kill(Prozessnummer, Signalnummer)* einem Prozess ein Ereignis signalisiert werden.

Die Reaktion eines Prozesses auf ein Signal kann verschieden ausfallen:

- Aktivierung einer Prozessfunktion, die der Prozess vorher für dieses Signal mit der Funktion **signal(Signalnummer, Funktionsadresse)** angemeldet hat
- Er kann das Signal mit **signal(Signalnummer, SIG_IGN);** ignorieren
- Der Prozess kann beendet werden.
- Hat der Prozess für eine Signalnummer nichts festgelegt, dann gibt es Voreinstellungen:

	#define SIGFPE	8	/* Floating point trap */
	#define SIGILL	4	/* Illegal instruction */
	#define SIGINT	2	/* voreingestelle Interrupttaste z.B. Ctl -c*/
	#define SIGSEGV	11	/* Memory access violation */
	#define SIGTERM	15	/* Kill -Kommando ohne Angabe*/
	#define SIGKILL	9	/* unbedingter Prozessabbruch*/

7. ps
8. fork
9. exec
10. exit
11. trap sammelt events und reagiert darauf.

Wenn der Prozess in der Shell bereits gestartet ist:

- mit der Eingabe: <Ctrl> + <z> unterbricht man den laufenden Prozess (suspend)

- mit dem Befehl: bg versetzt man den vorher angehaltenen Prozess in den Hintergr. Dieser wird dann im Hintergrund fortgesetzt.

- mit dem Befehl: jobs [<optionen>] kann man sich die Liste der aktuellen Background-Jobs anzeigen lassen.
  Beispiel: jobs -l
  [1]+ 45817 Running xemacs &
- mit dem Befehl: *kill %1 kann man den Backgroundjob durch den id-Bezug (hier 1) löschen.*
- mit dem Befehl: *fg %1* kann man den Hintergrundprozess wieder in den Fordergrund holen.
- mit nohup <Skriptname/Befehl> & wird ein Skript vom Vaterprozess entkoppelt und im Hntergund ausgeführt!

Wenn man Prozesse sucht, die von einem bestimmten Benutzer gestartet wurden oder die eine bestimmte Datei verwenden:

- mit fuser [-fMuvw] [-a|-s] [-4|-6] [-c|-m|-n RAUM] [-k [-i] [-SIGNAL]] <Name der Datei oder Sockets etc.> kann man die Prozessnummer herausfinden.
- mit finger [<Optionen>] wird der Benutzername, der auf die Datei bzw. den Socket etc. zugreift.

Weitere Übungsaufgaben:

4. Erläutern sie wie sie den nice-Wert eines laufenden Prozesses ermitteln und welche Bedeutung dieser Wert hat!
5. Verfassen sie ein bash-Skript mit einer Endlosschleife und das den eigenen Speicherbedarf und die prozentuale CPU-Belastung ausgibt.
6. Starten sie den ssh-Server-Dienst. Fordern sie einen Kollegen auf, dass er per ssh auf ihren PC per ssh-Protokoll zugreift! Versuchen Sie nun mittels „finger“- bzw. „fuser“-Befehl den Benutzer und desse Prozessnummer herauszufinden.
7. Verfassen Sie ein bash-Skript, das fortlaufend die Eingaben eines Buchstaben oder einer Ziffer vom Bediener fordert. Gibt man einen Buchstaben ein, gibt das Programm „OK“ zurück. Gibt man eine Ziffer ein, so soll eine Errormeldung „*Skriptpfad und -Name* meldet: kein Buchstabe“ auf der Konsole. Dieser Fehler wird auch unter „/var/log/messages“ in die errror-Datei eingetragen.
8. Wie schaltet man einen Daemon ein, aus und wie bestimmt man dessen Status und dass er bei jedem PC-Neustart automatisch startet?

https://superuser.com/questions/127863/manually-closing-a-port-from-commandline
https://coderwall.com/p/dkuptg/show-what-pid-is-listening-on-a-port-linux

https://shapeshed.com/unix-tee/
http://www.linux-community.de/ausgaben/linuxuser/2004/06/zu-befehl-kanaele-pipes-und-tee/2/

http://www.admin-magazin.de/Das-Heft/2014/10/Systemstart-mit-Systemd-unter-Linux/(offset)/8

Unix / Linux – Signals and Traps

Quelle: https://www.tutorialspoint.com/unix/unix-signals-traps.htm

In this chapter, we will discuss in detail about Signals and Traps in Unix.

Signals are software interrupts sent to a program to indicate that an important event has occurred. The events can vary from user requests to illegal memory access errors. Some signals, such as the interrupt signal, indicate that a user has asked the program to do something that is not in the usual flow of control.

The following table lists out common signals you might encounter and want to use in your programs −

Signal Name	Signal Number	Description
SIGHUP	1	Hang up detected on controlling terminal or death of controlling process
SIGINT	2	Issued if the user sends an interrupt signal (Ctrl + C)
SIGQUIT	3	Issued if the user sends a quit signal (Ctrl + D)
SIGFPE	8	Issued if an illegal mathematical operation is attempted
SIGKILL	9	If a process gets this signal it must quit immediately and will not perform any clean-up operations
SIGALRM	14	Alarm clock signal (used for timers)
SIGTERM	15	Software termination signal (sent by kill by default)

List of Signals

There is an easy way to list down all the signals supported by your system. Just issue the **kill -l** command and it would display all the supported signals −

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      34) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

The actual list of signals varies between Solaris, HP-UX, and Linux.

Default Actions

Every signal has a default action associated with it. The default action for a signal is the action that a script or program performs when it receives a signal.

Some of the possible default actions are −

- Terminate the process.
- Ignore the signal.

- Dump core. This creates a file called **core** containing the memory image of the process when it received the signal.
- Stop the process.
- Continue a stopped process.

Sending Signals

There are several methods of delivering signals to a program or script. One of the most common is for a user to type **CONTROL-C** or the **INTERRUPT key** while a script is executing.

When you press the ***Ctrl+C*** key, a **SIGINT** is sent to the script and as per defined default action script terminates.

The other common method for delivering signals is to use the **kill command**, the syntax of which is as follows −

```
$ kill -signal pid
```

Here **signal** is either the number or name of the signal to deliver and **pid** is the process ID that the signal should be sent to. For Example −

```
$ kill -1 1001
```

The above command sends the HUP or hang-up signal to the program that is running with **process ID 1001**. To send a kill signal to the same process, use the following command −

```
$ kill -9 1001
```

This kills the process running with **process ID 1001**.

Trapping Signals

When you press the *Ctrl+C* or Break key at your terminal during execution of a shell program, normally that program is immediately terminated, and your command prompt returns. This may not always be desirable. For instance, you may end up leaving a bunch of temporary files that won't get cleaned up.

Trapping these signals is quite easy, and the trap command has the following syntax −

```
$ trap commands signals
```

Here *command* can be any valid Unix command, or even a user-defined function, and signal can be a list of any number of signals you want to trap.

There are two common uses for trap in shell scripts −

- Clean up temporary files
- Ignore signals

Cleaning Up Temporary Files

As an example of the trap command, the following shows how you can remove some files and then exit if someone tries to abort the program from the terminal −

```
$ trap "rm -f $WORKDIR/work1$$ $WORKDIR/dataout$$; exit" 2
```

From the point in the shell program that this trap is executed, the two files ***work1$$*** and ***dataout$$*** will be automatically removed if signal number 2 is received by the program.

Hence, if the user interrupts the execution of the program after this trap is executed, you can be assured that these two files will be cleaned up. The **exit** command that follows the **rm** is necessary because without it, the execution would continue in the program at the point that it left off when the signal was received.

Signal number 1 is generated for **hangup**. Either someone intentionally hangs up the line or the line gets accidentally disconnected.

You can modify the preceding trap to also remove the two specified files in this case by adding signal number 1 to the list of signals −

```
$ trap "rm $WORKDIR/work1$$ $WORKDIR/dataout$$; exit" 1 2
```

Now these files will be removed if the line gets hung up or if the *Ctrl+C* key gets pressed.

The commands specified to trap must be enclosed in quotes, if they contain more than one command. Also note that the shell scans the command line at the time that the trap command gets executed and also when one of the listed signals is received.

Thus, in the preceding example, the value of **WORKDIR** and **$$** will be substituted at the time that the

trap command is executed. If you wanted this substitution to occur at the time that either signal 1 or 2 was received, you can put the commands inside single quotes −

```
$ trap 'rm $WORKDIR/work1$$ $WORKDIR/dataout$$; exit' 1 2
```

Ignoring Signals

If the command listed for trap is null, the specified signal will be ignored when received. For example, the command −

```
$ trap '' 2
```

This specifies that the interrupt signal is to be ignored. You might want to ignore certain signals when performing an operation that you don't want to be interrupted. You can specify multiple signals to be ignored as follows −

```
$ trap '' 1 2 3 15
```

Note that the first argument must be specified for a signal to be ignored and is not equivalent to writing the following, which has a separate meaning of its own −

```
$ trap  2
```

If you ignore a signal, all subshells also ignore that signal. However, if you specify an action to be taken on the receipt of a signal, all subshells will still take the default action on receipt of that signal.

Resetting Traps

After you've changed the default action to be taken on receipt of a signal, you can change it back again with the trap if you simply omit the first argument; so −

```
$ trap 1 2
```

This resets the action to be taken on the receipt of signals 1 or 2 back to the default.

Beim `Präfix` SIG handelt es sich jeweils um die `Kurzform` von Signal.

> Tasks und Prozesse können aus unterschiedlichen Quellen unterschiedliche Signale erhalten. Die Signalverarbeitung kann je Betriebssystem, Architektur und Prozessverwaltung sehr verschieden verarbeitet werden! Viele Grundfunktionen werden jedoch übergreifend über alle Architekturen hinweg einheitlich behandelt.

Der kill-Befehl:

Mit dem Befehl `kill -l` werden gewöhnlich alle unterstützten Signalnummern mit den zugehörigen Namen ausgegeben. Die Tabelle unten enthält beispielhaft die folgenden Werte:

- Spalte A: GNU-C-Bibliothek
- Spalte B: Linux, Architekturen Alpha und SPARC
- Spalte C: Linux, x86, AMD64, ARM und die meisten anderen Prozessorarchitekturen
- Spalte D: Linux, MIPS

Signal	Werte, Synonym				Bedeutung (englisch)	Bedeutung (übersetzt)	Ursprüngliche Verwendung, Standards
	A	B	C	D			
SIGHUP	1				Hangup detected on controlling terminal or death of controlling process	Blockierung des Kontrollterminals oder Ende des Kontrollprozess.	POSIX (1990)
SIGINT	2				Interrupt from keyboard; interactive attention signal.	Interrupt durch die Tastatur; interaktives Warnsignal.	C89; POSIX (1990)
SIGQUIT	3				Quit from keyboard.	Beenden durch die Tastatur.	
SIGILL	4				Illegal instruction.	Ungültige Anweisung.	C89; POSIX (1990)
SIGTRAP	5				Trace/breakpoint trap.	Haltemarke erreicht.	SUSv2; POSIX (2001)

Signal	Werte, Synonym				Bedeutung (englisch)	Bedeutung (übersetzt)	Ursprüngliche Verwendung, Standards
	A	B	C	D			
SIGABRT	6 SIGIOT				Abnormal termination; abort signal from abort(3).	abnormale Beendigung.	C89; POSIX (1990)
SIGIOT	6 SIGABRT				IOT trap; abort() on a PDP11.		4.2BSD
SIGEMT	–	7	–	7			
SIGBUS	10	10	7	10	BUS error (bad memory access).	BUS Fehler (Speicherzugriffsfehler).	4.2BSD; SUSv2; POSIX (2001)
SIGFPE	8				„Floating-point exception“: erroneous arithmetic operation.	„Gleitkommaoperation Ausnahmefehler“: fehlerhafte arithmetische Operation.	C89; POSIX (1990)
SIGKILL	9				Kill, unblockable.	Unblockbares Beenden.	POSIX (1990)
SIGUSR1	30	30	10	16	User-defined signal 1.	Benutzerdefiniertes Signal 1.	POSIX
SIGSEGV	11				„Segmentation violation“: invalid memory reference.	„Schutzverletzung“: ungültige Speicherreferenz.	C89; POSIX (1990)
SIGUSR2	31	31	12	17	User-defined signal 2.	Benutzerdefiniertes Signal 2.	POSIX
SIGPIPE	13				„Broken pipe“: write to pipe with no readers.	„Broken pipe“: Schreiben auf eine Pipe ohne Empfänger.	POSIX (1990)
SIGALRM	14				Alarm clock timer signal: alarm(2).	Wecker Signal: Alarm(2).	POSIX (1990)
SIGTERM	15				Termination request.	Beendigungsanfrage.	C89; POSIX (1990)
SIGSTKFLT	–	–	16	–	Stack fault on coprocessor (unused).	Stapelfehler auf Coprozessor (unbenutzt).	
SIGCHLD	20 SIGCLD	20	17	18	Child status has changed (stopped or terminated).	Kindstatus wurde geändert (angehalten oder beendet).	POSIX (1990)
SIGCLD	20 SIGCHLD	–	–	–	Old System V name; child status has changed.	Alte System V Bezeichnung; Kindstatus wurde geändert.	System V
SIGCONT	19	19	18	25	Continue stopped process.	Fahre angehaltenen Prozess fort.	POSIX
SIGSTOP	17	17	19	23	Stop process, unblockable.	Halte Prozess an, unblockierbar.	POSIX
SIGTSTP	18	18	20	24	Stop typed at keyboard.	Eingabe von Stop durch die Tastatur.	POSIX
SIGTTIN	21	21	21	26	Background read from tty.	Lesen vom Terminal im Hintergrund	POSIX
SIGTTOU	22	22	22	27	Background write to tty	Schreiben auf ein Terminal im Hintergrund.	POSIX
SIGURG	16	16	23	21	Urgent condition on socket: high bandwidth data is available.	Wichtiger Zustand auf Socket: Daten mit hoher Bandbreite sind verfügbar.	4.2BSD; SUSv2; POSIX (2001)
SIGXCPU	24	24	24	30	CPU time limit exceeded.	Prozessorzeitbegrenzung überschritten.	4.2BSD; SUSv2; POSIX (2001)
SIGXFSZ	25	25	25	31	File size limit exceeded.	Dateigrößenbeschränkung überschritten.	4.2BSD; SUSv2; POSIX (2001)
SIGVTALR	26	26	26	28	Virtual alarm clock.	Virtueller Wecker.	4.2BSD; SUSv2;

Signal	Werte, Synonym				Bedeutung (englisch)	Bedeutung (übersetzt)	Ursprüngliche Verwendung, Standards
	A	B	C	D			
M							POSIX (2001)
SIGPROF	27	27	27	29	Profiling alarm clock timer expired.		4.2BSD; SUSv2; POSIX (2001)
SIGWINCH	–	28	28	20	Window size change.	Fenstergröße änderte sich.	4.3BSD; Sun
SIGPOLL	23	23/–	29/–	23/–	Pollable event occurred.	Abfragbares Ereignis aufgetreten.	System V; SUSv2; POSIX (2001)
	SIGIO						
SIGIO	23	23	29	23	I/O now possible.	I/O jetzt möglich.	4.2BSD
	SIGPOLL						
SIGINFO	–	(29)	–	–	Status request from keyboard.	Statusanfrage durch die Tastatur.	Mac OS X
	SIGPWR						
SIGLOST	–	(29)/–	–	–	Unused; only on Sparc: file lock lost.	Unbenutzt; nur unter Sparc: Dateisperre verloren.	
SIGPWR	–	(29)	30	19	Power failure restart.	Stromausfall bedingter Neustart.	System V
	SIGINFO						
SIGSYS	12	12	31	12	Bad system call.	ungültiger Systemaufruf.	System Vr4; SUSv2; POSIX (2001)
	SIGUNUSED						
SIGUNUSED	12	–	31/–	–			