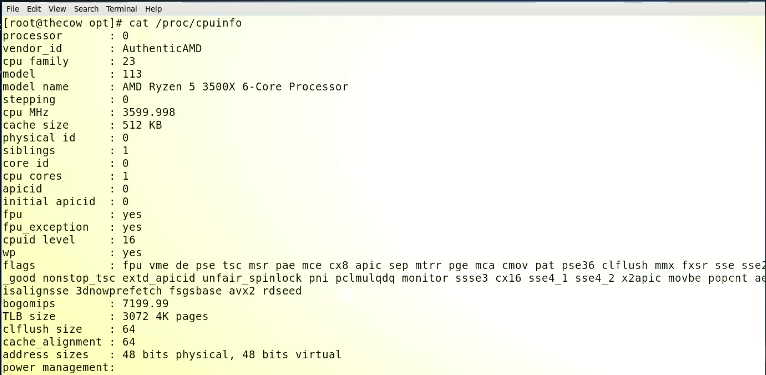
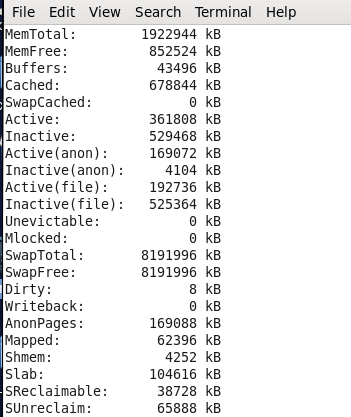
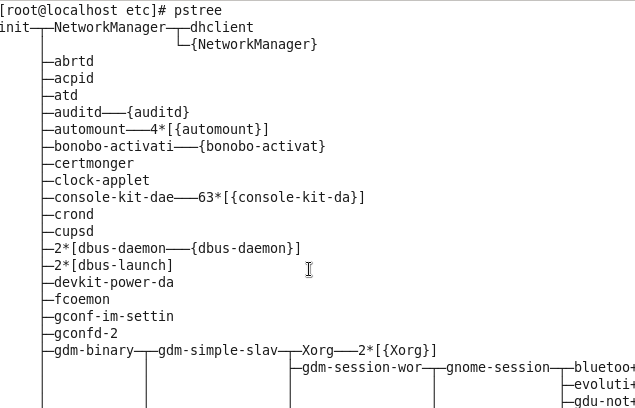
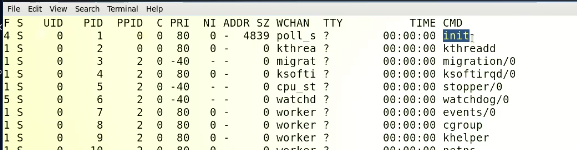
Lecture 20

**Process-daemons-Performance Monitoring-P1**

**Process deamons**

* $ lsof 🡪 all processes in RAM
* Stands for:- **List open Files**
* Each process had PID 🡪 process ID
* “erandom” randomly assign and manages PIDs for each process.
* PID of 20% processes will remain constant
* PID 0f 80% services will change after every reboot or service restart.
* To check no of running processes,
* **$ ps 🡪 very power full command**
* Famous flags are
* $ ps -el 🡪 use “| less” to fit to screen, “| wc -l” to list running processes.
* $ ps -ef 🡪
* $ ps -aux
* **Output form RAM**
* To grep a specific process , e.g firefor
* $ ps -el | grep -i fire
* 
* **To stop or kill a process or throw it out of RAM**
* **$ kill <PID>**
* **To kill a process forcefully**
* **$ kill -9 <PID>**
* **Or**
* **$ pkill <process\_name>**
* $ pkill firefox
* To display directly PID of a specifir process command is,
* **$ pidof <process\_name>**
* $ pidof firefox
* **Or**
* **$ pgrep <process\_name>**
* $ pgrep fire
* What is kill 🡪 for short description
* **$ cat /proc/cpuinfo 🡪 to display “CPU” information**
* **$ lscpu 🡪 CPU information**
* 
* Similarly for “RAM”
* **$ cat /proc/meminfo**
* 
* Another useful command is
* **$ pstree 🡪 to display process tree**
* 
* **$ top 🡪 just like Task Manager in MS Windows**
* **$ :() { :|:&}:: 🡪 it is called “fork bomb” which fill the process, it calls itself and fills the processor with requests and system halts and needs a hard restart**
* **For better understanding it can be ,**
* **$ :() { k|k& };k**
* The fork bomb is a form of denial-of-service (DoS) attack against a Linux or Unix-based system
* This appears to be a command in the Bash programming language. The ":" character is a no-op (no operation) command, which means it does nothing. The "{ }" characters indicate a command block, and the ":" commands within the block also do nothing. The "|" character is a pipe, which takes the output of one command and redirects it as input to another command. The "&" character runs the command in the background, allowing other commands to continue running simultaneously.
* In fact what you posted there is called a fork bomb, because it does exactly that. It is a Bash function which calls itself recursively. You can replace the : with a name if you want so it becomes more obvious:
* **fu {**
* **fu | fu &**
* **}; fu**
* So fu calls itself, piping its output through itself again. In this way it fills up your processor with requests.
* It is not harmful, beside the fact that you may have to reboot your computer the hard way because it becomes unresponsive.
* It is often used by sysadmin to test user process limitations on a server.
* **$ ps -el | less**
* ****
* **“init” at top of the list till RHL 6**
* PID will always remain “1”
* It is called father of all the processes
* **In RHL 7 & 8 🡪 “systemd” is being father of all the processes**
* **Querry what id PID “0”.**
* In a Unix-like operating system, the process identifier (PID) is a number assigned to each running process. The kernel, the core part of the operating system, is also a process and it is assigned a PID of 0. This process, also known as the "idle process," is created by the kernel during the initialization of the operating system, and it continues running until the system is shut down. The idle process does not execute any program code and acts as a placeholder for when no other process is running. It is also known as the swapper or scheduler process as it schedules other processes that are ready to execute.
* **Another fact, PPID 🡪 Parent Process ID**
* In a Unix-like operating system, the parent process identifier (PPID) is a number assigned to the parent process of a running process. Each process in the system has a unique process identifier (PID) and is created by another process, called the parent process. The parent process's PID is the child process's PPID. When a process starts another process, the child process inherits the parent's process ID (PID) as its parent process ID (PPID). When the parent process terminates, the child process becomes an orphan and its PPID is changed to the init process (usually PID 1) which takes the role of the parent process for the orphaned process.

**Further reading below …**

Understanding :(){ :|:& };: fork() bomb code

**WARNING!** These examples may crash your computer if executed.

The :() – Defined the function called :. This function accepts no arguments. The syntax for bash function is as follows:

foo**(){**

arg1=$1

arg2=$2

**echo** 'Bar..'

*#do\_something on $arg argument*

**}**

fork() bomb is defined as follows:

:(){

:|:&

};:

**:|:** – Next it will call itself using programming technique called recursion and pipes the output to another call of the function ‘:’. The worst part is function get called two times to bomb your system.

**&** – Puts the function call in the background so child cannot die at all and start eating system resources.

**;** – Terminate the function definition.

**:** – Call (run) the function aka set the fork() bomb.  
Here is more human readable code:

bomb() {

bomb | bomb &

}; bomb

Properly configured Linux / UNIX box should not go down when fork() bomb sets off. See the [comment # 5 below](https://www.cyberciti.biz/faq/understanding-bash-fork-bomb/#comment-37097) for more fork bomb examples created in Perl, Windows XP (batch) and C.

Preventing fork bomb on Linux

Please note that ulimit is a shell builtin. You can verify this using the [type command](https://bash.cyberciti.biz/guide/Type_command) or [command command](https://bash.cyberciti.biz/guide/Command) as follows:  
$ type -a ulimit  
ulimit is a shell builtin

Type the following ulimit command to find out the current maximum processes you can run on Linux:  
$ ulimit -u  
OR  
$ ulimit -a  
  
The number 128038 indicates that you can run 128038 processes. To protect your Linux system from a fork bomb, you need to lower that number. To limit your session to 5000 processes, use the following command  
$ ulimit -S -u 5000

**WARNING!** Please don’t set ulimit numbers too low. This will prevent you from working on your system.

Now run fork bomb again:  
$ :(){ :|:& };:  
And you will see messages as follows:

bash: fork: Resource temporarily unavailable

bash: fork: Resource temporarily unavailable

bash: fork: Resource temporarily unavailable

bash: fork: Resource temporarily unavailable

bash: fork: Resource temporarily unavailable

bash: fork: Resource temporarily unavailable

bash: fork: Resource temporarily unavailable

You just avoided fork bomb on Linux. Run the following pgrep command to see the current threads limit:  
$ pgrep -wcu $USER  
Sample outputs:

5002

Summing up

You learned about a kind of denial-of-service (DoS) attack upon a Linux or Unix-based machine. Make sure you do read the following man pages using the [man command](https://bash.cyberciti.biz/guide/Man_command) or [help command](https://bash.cyberciti.biz/guide/Help_command):  
$ man bash  
$ ulimit --help  
$ help ulimit  
$ man ulimit

# [**How to: Prevent a fork bomb by limiting user process**](https://www.cyberciti.biz/tips/linux-limiting-user-process.html)

Limiting user processes is important for running a stable system. To limit user process just add user name or group or all users to **/etc/security/limits.conf** file and impose process limitations.

Understanding /etc/security/limits.conf file

Each line describes a limit for a user in the form:  
<domain> <type> <item> <value>  
Where:

* **<domain>** can be:
  + an user name
  + a group name, with @group syntax
  + the wildcard \*, for default entry
  + the wildcard %, can be also used with %group syntax, for maxlogin limit
* **<type>** can have the two values:
  + “soft” for enforcing the soft limits
  + “hard” for enforcing hard limits
* **<item>** can be one of the following:
  + core – limits the core file size (KB)
* **<value>** can be one of the following:
  + core – limits the core file size (KB)
  + data – max data size (KB)
  + fsize – maximum filesize (KB)
  + memlock – max locked-in-memory address space (KB)
  + nofile – max number of open files
  + rss – max resident set size (KB)
  + stack – max stack size (KB)
  + cpu – max CPU time (MIN)
  + ***nproc* – max number of processes**
  + as – address space limit
  + maxlogins – max number of logins for this user
  + maxsyslogins – max number of logins on the system
  + priority – the priority to run user process with
  + locks – max number of file locks the user can hold
  + sigpending – max number of pending signals
  + msgqueue – max memory used by POSIX message queues (bytes)
  + nice – max nice priority allowed to raise to
  + rtprio – max realtime priority
  + chroot – change root to directory (Debian-specific)

**Warning:** This will have no effect on the root user or any process with the CAP\_SYS\_ADMIN or CAP\_SYS\_RESOURCE capabilities are not affected by this kind of limitation on a Linux based system.

Configuration

Login as the root and open configuration file:  
# vi /etc/security/limits.conf  
Following will prevent a “fork bomb”:  
vivek hard nproc 300  
@student hard nproc 50  
@faculty soft nproc 100  
@pusers hard nproc 200

Above will prevent anyone in the student group from having more than 50 processes, faculty and pusers group limit is set to 100 and 200. Vivek can create only 300 process. Please note that KDE and Gnome desktop system can launch many process.

Test it again

Save and close the file. Test your new system by dropping a fork bomb:  
$ :(){ :|:& };:

-