‘*systemctl*’	**COMP3500**	**12/03/2014**
Jack Rutherford	**u5179458**	**REDHg1**

1. Summary

The title for this project is ‘Optimizing and maintaining management rules for Linux-based server-side firewalls’ and is being completed for an external client, Red Hat. The mini-project which I have been assigned is to research ‘*systemctl*’, an interface used to control ‘*systemd*’, one of the more recent system and service managers for Linux. For the first period of the project, I have spent my time familiarising myself with the project itself, the requirements of the project, the language and environment in which the project will be conducted in, and the background information required to begin the project.

Short Description	**Important / Contribution**	**Difficulty**	**Hours**	**Links**
Started learning Python	High - Only doing low difficulty tasks to learn the language, in order to get some background for the project. Learning the language is vital for the project however.	Moderate	5	Python
Research	Moderate - hard to find information on systemctl. Much more information on systemd, which is related but not the topic. Involved watching multiple videos on systemd.	High	20	(Find at reference [20])
Set up Fedora Environment	Moderate - Had to clean up a lot of space on pc to make room enough to shrink disk to partition drive and install Fedora.	Moderate	7	Proof of installation at references [10] and [21] from screenshots of user terminal (note username)
Downloaded and studied source code for systemd (includes systemctl as part of repository).	Very High - At first I was under the impression systemctl was part of another package, and was unable to find the source code. The code is vitally necessary to the projects completion.	Moderate	2	https://github.com/systemd/systemd

In order to properly start the project, I have been researching using resources such as the internet and various colleagues, both professional and academic, and beginning to involve myself in the more practical, coding side of the project. The coding is just tutorial based at a beginner level, as I am not familiar with Python, the language in which the project is to be written in. By learning through online tutorials I hope to achieve a level of proficiency in Python so that I can contribute adequately to the project. I am also beginning on the more difficult background of the project code, by researching areas such as my mini-project - systemctl. My task at the completion of this mini-project is to write a Python daemon that uses systemd to track when a process goes up or down. It will then provide information to another program, “serviceAPI”.

systemctl

1. Introduction
'*systemctl*' is a system call in Linux, used to inspect and manipulate the state of '*systemd*'. *systemd* is a Linux-based system management daemon (as well as being the parent daemon of all processes) and is enabled by default on the following Linux distributions:[1]

- Ångström
- Arch Linux[2]
- CoreOS
- Fedora
- Frugalware Linux[3]
- Mageia[4]
- openSUSE[5]
- Sabayon Linux[6]

The Debian GNU/Linux distribution plan to use *systemd* in their next release, the Red Hat Enterprise Linux distribution is waiting for RHEL 7 to use *systemd*, and there is a possibility that Ubuntu may even incorporate it at some stage. Ubuntu is not confirmed however, they have just provided instructions on how to use *systemd* as an experimental option in their development documentation. [7][8]

systemctl provides the user with an interface to this daemon management system, and is a combination of both the 'service' and 'chkconfig' utility from previous Unix systems. The service aspect, as with the old service system call, implies you can only control the services behaviour for the current session only, whereas the chkconfig aspect, again similarly to the old chkconfig command, implies you can control it permanently. Both the service and chkconfig utilities still function as they did before, but now automatically call the *systemctl* utility to complete their operations. [8]

The project is initially going to be written for *systemd,* but if there is ample time this will be extended to also include functionality for SysVinit, the older linux daemon management system.

2. *systemd background*
systemd was designed to replace the old and well known System V init daemon - the daemon management system in most current Unix operating systems, and as such is compatible with SysV and LSB init scripts. It has been designed with the intent to improve on init, and does so in the following ways: [9]

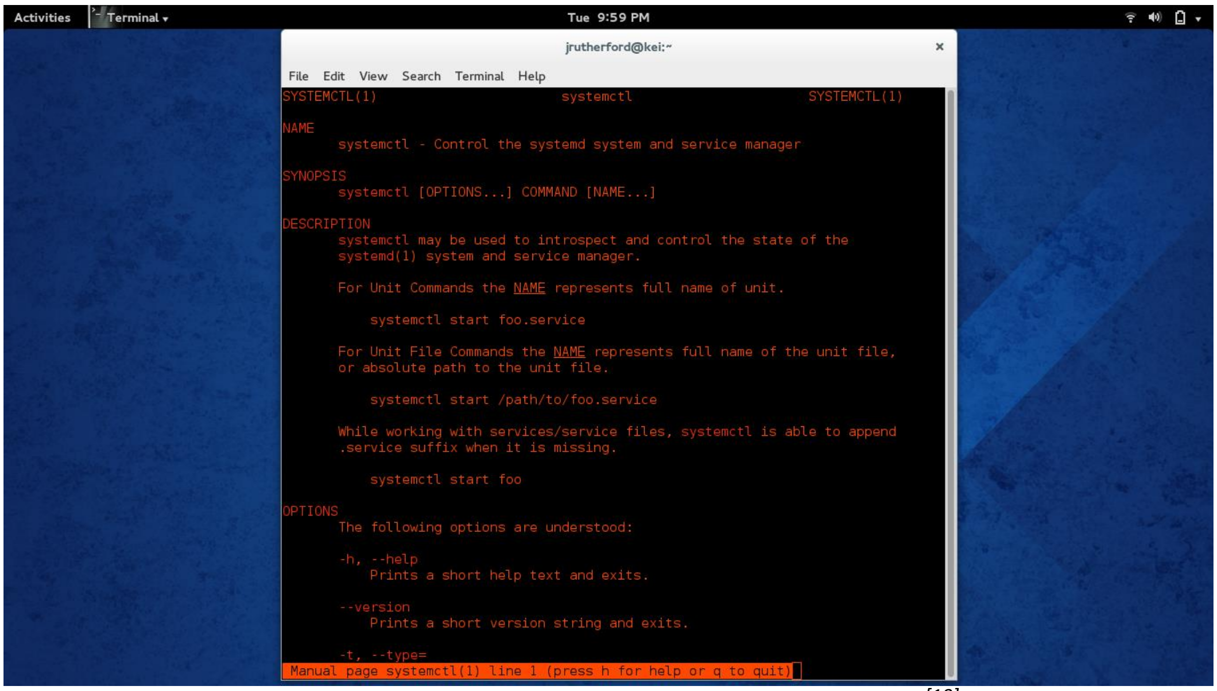
- provides aggressive parallelization capabilities (resulting in faster boot time)
- uses socket and D-Bus activation for starting services
- offers on-demand starting of daemons
- keeps track of processes using Linux cgroups
- supports snapshotting and restoring of system state
- maintains mount and automount points
- implements an elaborate transactional dependency-based service control logic

In a lecture given by Lennart Poettering, one of the founders and main contributors to *systemd*, he outlined the following points and advantages about *systemd,* when he described it in comparison to SysVinit: [15]

- systemd is a system and session manager for Linux. Controls what processes are being run.
- Compatible with SysV and LSB init scripts.
- Provides aggressive parallelization capabilities.
- Uses socket and D-Bus activation for stating services.
- Means you can start services if something happens on a network socket, or if someone requires D-Bus activation.
- Offers on-demand starting of daemons.
- Keeps track of processes using Linux cgroups - cgroups short for control groups - useful to not only start and stop services but also keep precise tracking about everything a service does, what it responds to.
- Supports snapshotting and restoring of system state
- Maintains mount and automount points - used to parallelize bootup.
- Points and implements an elaborate transactional dependency-based service control logic.
- It can work as a drop-in replacement for sysvinit.
- On boot, all sockets are created at the same time and socket listening done by init system. This means Syslog for example can crash and not miss any packets
- Socket has a quite large buffer, rarely runs over size.
- Doesn’t need to be a schedule in init system anymore because everything starts at same time
- Socket based activation:
    - Kernel orders and buffers requests
    - Implicit dependencies
    - Patching daemons

3. *systemctl* Use

systemctl is simple to use for elementary system calls. The format, as you can see in the synopsis section of the terminal man page in figure 1, is ‘systemctl [OPTIONS…] COMMAND [NAME…]. There are a variety of options available, all of which can be found at the man page. The options are not necessary for most of the simple systemctl system calls, but can provide useful information and further manipulation of the command output. [10]



```
SYSTEMCTL(1)                    systemctl                    SYSTEMCTL(1)

NAME
       systemctl - Control the systemd system and service manager

SYNOPSIS
       systemctl [OPTIONS...] COMMAND [NAME...]

DESCRIPTION
       systemctl may be used to introspect and control the state of the
       systemd(1) system and service manager.

       For Unit Commands the NAME represents full name of unit.

           systemctl start foo.service

       For Unit File Commands the NAME represents full name of the unit file,
       or absolute path to the unit file.

           systemctl start /path/to/foo.service

       While working with services/service files, systemctl is able to append
       .service suffix when it is missing.

           systemctl start foo

OPTIONS
       The following options are understood:

       -h, --help
           Prints a short help text and exits.

       --version
           Prints a short version string and exits.

       -t, --type=
 Manual page systemctl(1) line 1 (press h for help or q to quit)
```

Figure 1: Screenshot of the 'systemctl(1)' man page.[10]

Table 1 contains a list of some of the more basic *systemctl* commands which could prove useful during the course of the project. Some of these commands provide the user with information about services, which are available to any user, however it must be noted that any manipulation of state using a *systemctl* system call will require sudo user access to execute, such as starting and stopping services. On successful execution of a command, the exit status will always be 0, otherwise a non-zero exit status will indicate a failure. A full list of commands is available on the *systemctl* man page. [11]

Description	**Example Command(s)**
List all running services	systemctl
Pipe output of *systemctl* into grep. Example 1 will list all service files and their status. Example 2 will find the ntp process.	systemctl list-units | grep .service systemctl | grep -i ntp
Start a service	systemctl start foo.service
Start a service and check return code (to ensure it worked)	systemctl start bluetooth.service && echo $?
Stop a service	systemctl stop foo.service
Immediately stop a process (kill)	systemctl kill foo.service
Restart a service	systemctl restart foo.service
Get status of a service	systemctl status foo.service

Enable a service for boot	systemctl enable foo.service
Disable a service for boot	systemctl disable foo.serivce
Find all active services	systemctl --all | grep active
Find all inactive services	systemctl --all | grep inactive

Table 1: A list of simple systemctl uses and examples[11]

There may be some need to convert commands from *SysVinit*, the old daemon management system, to *systemd* format. For that reason, following in table 2 is a list of some simpler *SysVinit* commands (which use *chkconfig* and *service* commands) as they translate to *systemctl* (using *frobozz* as an example service).

SysVinit	**Systemd**	**Notes**
service frobozz start	systemctl start frobozz.service	Used to start a service (not reboot persistent)
service frobozz stop	systemctl stop frobozz.service	Used to stop a service (not reboot persistent)
service frobozz restart	systemctl restart frobozz.service	Used to stop and then start a service
service frobozz reload	systemctl reload frobozz.service	When supported, reloads the config file without interrupting pending operations.
service frobozz condrestart	systemctl condrestart frobozz.service	Restarts if the service is already running.
service frobozz status	systemctl status frobozz.service	Tells whether a service is currently running.
ls /etc/rc.d/init.d/	systemctl list-unit-files --type=service (preferred) ls /lib/systemd/system/*.service /etc/systemd/system/*.service	Used to list the services that can be started or stopped Used to list all the services and other units
chkconfig frobozz on	systemctl enable frobozz.service	Turn the service on, for start at next boot, or other trigger.
chkconfig frobozz off	systemctl disable frobozz.service	Turn the service off for the next reboot, or any other trigger.
chkconfig frobozz	systemctl is-enabled frobozz.service	Used to check whether a service is configured to start or not in the current environment.
chkconfig --list	systemctl list-unit-files --type=service(preferred) ls /etc/systemd/system/*.wants/	Print a table of services that lists which runlevels each is configured on or off

chkconfig frobozz --list	ls /etc/systemd/system/*.wants/frobozz.service	Used to list what levels this service is configured on or off
chkconfig frobozz --add	systemctl daemon-reload	Used when you create a new service file or modify any configuration

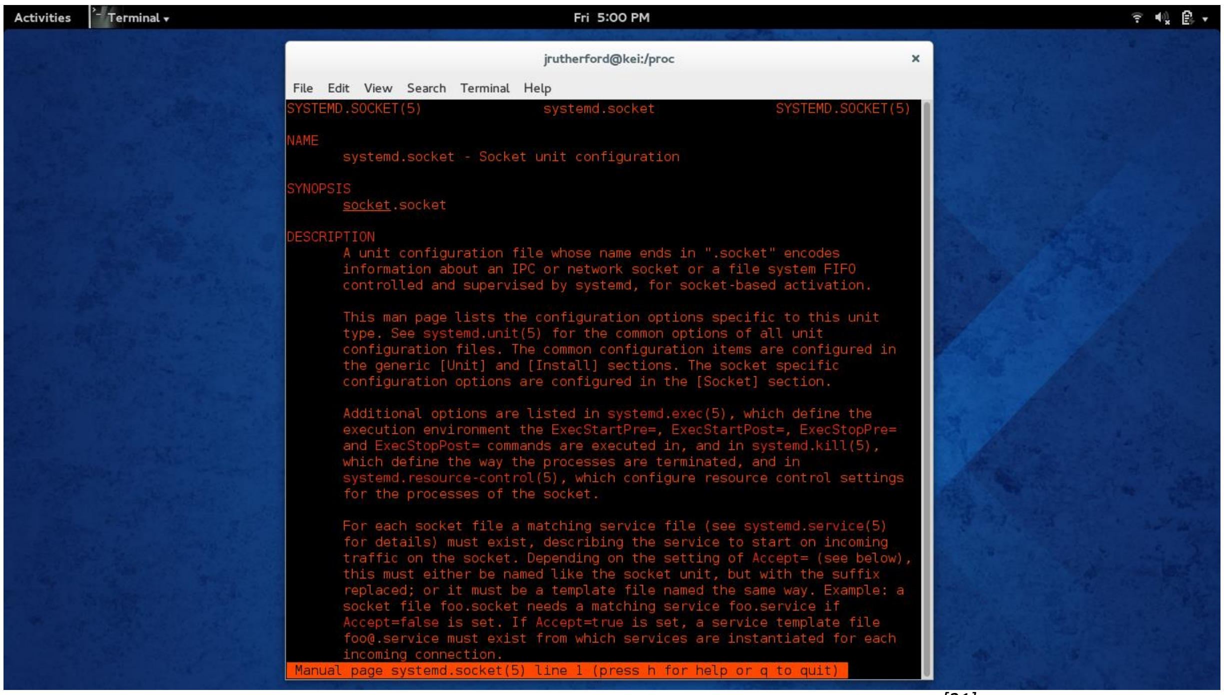
Table 2: A list of SysVinit commands and how they translate to systemd[12]

4. Developing a patch to *systemctl*

To complete the project, it was outlined in the project overview that it would be necessary to "develop a patch to *systemctl* that will use this project to activate, deactivate and show the status of firewall rules for individual network services." In order to research what this would involve, the source code was obtained for *systemctl.* It was contained within the *systemd* open-source repository, which will also be used and useful for the project. Developing a patch to *systemctl* will involve modifying the code within this repository to communicate with the code written by the project. *systemctl* is written in C, which most members of the project are already proficient in, so this will be a task that won't involve python code, but will have to communicate with python code. At reference 14, the reader can find code for a patch to systemctl, initiated earlier this year.[13][14]

5. Project inputs

As part of the project, the project team will be required to write various inputs to the projects that the program will react to. Some of these include *systemd* calls, and from user based function calls. These will both involve listening to find when changes occur. One option of approaching this is to use *systemd,* as it employs socket-based activation. This means that it listens on various ports, and when a connection is requested, a socket is created and control of the socket is handed over to the application. See figure 2 for an insert of the *systemd.socket(5)* man page.



```
SYSTEMD.SOCKET(5)             systemd.socket             SYSTEMD.SOCKET(5)

NAME
       systemd.socket - Socket unit configuration

SYNOPSIS
       socket.socket

DESCRIPTION
       A unit configuration file whose name ends in ".socket" encodes
       information about an IPC or network socket or a file system FIFO
       controlled and supervised by systemd, for socket-based activation.

       This man page lists the configuration options specific to this unit
       type. See systemd.unit(5) for the common options of all unit
       configuration files. The common configuration items are configured in
       the generic [Unit] and [Install] sections. The socket specific
       configuration options are configured in the [Socket] section.

       Additional options are listed in systemd.exec(5), which define the
       execution environment the ExecStartPre=, ExecStartPost=, ExecStopPre=
       and ExecStopPost= commands are executed in, and in systemd.kill(5),
       which define the way the processes are terminated, and in
       systemd.resource-control(5), which configure resource control settings
       for the processes of the socket.

       For each socket file a matching service file (see systemd.service(5)
       for details) must exist, describing the service to start on incoming
       traffic on the socket. Depending on the setting of Accept= (see below),
       this must either be named like the socket unit, but with the suffix
       replaced; or it must be a template file named the same way. Example: a
       socket file foo.socket needs a matching service foo.service if
       Accept=false is set. If Accept=true is set, a service template file
       foo@.service must exist from which services are instantiated for each
       incoming connection.
 Manual page systemd.socket(5) line 1 (press h for help or q to quit)
```

Figure 2: An insert of the systemd.socket(5) man page.[21]

It is possible to view which sockets *systemd* is currently listening on by using the command *'systemctl --all list-sockets'*. This will more than likely be necessary for listening to system calls, however function calls made by the user may be done differently - accepting a call passed by the user rather than listening and reacting to the call.

6. Build tool chain for *systemctl* (and *systemd*) package
The build tool chain will involve a few tools, but not a large number. The first tool required to build the project will be, most obviously, the Fedora distribution of the Linux operating system.[17] This will be equipped with Yellowdog Updater, Modified (YUM)[18], which will be sufficient for more most file installation for the smaller dependencies, such as PEP8. PEP8 is a program formatting standard for python, which the team has been instructed to use during the project. PEP8 also provides an application which the user can input their code to, run, and be notified of any breaches of the standards by the output.[19] As such, PEP8 will be part of the build tool chain, indirectly. Python code does not need to be compiled, just run, and this can be done by the linux terminal, which is obviously included as part of Fedora 20. This means there is also no need for an IDE, as the user can just utilise Vim, an inbuilt text editor for the terminal. This has been suggested as a possibility, however it has also been suggested that the team members may also consider using an IDE to enhance efficiency for things like dynamic error checking. For this reason, Eclipse was suggested to be used, combined with the "integrated python" plugin in order to support the language.

7. Alternatives to using systemctl
For *systemd*, the only systemctl alternatives possible to use are native calls, that were originally built for SysVinit. Although distributions such as Fedora have adopted *systemd* as their default daemon management system, the old SysVinit calls are still implemented; that is, *chkconfig* and *service*. In some distributions, it is possible to use these to initiate SysVinit, however in Fedora, these calls literally just point to the *systemctl* equivalent command when used, in order to make them operate with *systemd,* and to provide some functionality to users who are more familiar with SysVinit. [1][16]

The alternative approach for the project as a whole is to implement it using SysVinit. As previously stated, this will be done if time allows when it is correctly implemented for *systemd. systemd* is the team's priority, as the latest version of Fedora (version 20 - heisenbug) uses *systemd,* and Fedora is RedHat, the client's, preferred operating system. This is still advantageous, because as previous information indicates, more and more distributions of Linux are adopting *systemd,* and it looks to be the future of daemon management in Linux, one it becomes more stable and widespread.

References:

[1] *Systemd.* 2014. The Fedora Project. [ONLINE] Available at: https://fedoraproject.org/wiki/Systemd.
[2] *Systemd is now the default on new installations.* 2012. Arch Linux. [ONLINE] Available at: https://www.archlinux.org/news/systemd-is-now-the-default-on-new-installations/.
[3] *Frugalware*. 2014. Frugalware. [ONLINE] Available at: http://frugalware.org/news/223.
[4] *Mageia 2 arrives with GNOME and systemd.* 2012. [ONLINE] Available at: http://web.archive.org/web/20131208100442/http://www.h-online.com/open/news/item/Mageia-2-arrives-with-GNOME-3-and-systemd-1582479.html.
[5] *Index of /pub/opensuse/discontinued/distribution/11.4/repo/oss/suse/i586.* 2011. OpenSUSE. [ONLINE] Available at: http://ftp5.gwdg.de/pub/opensuse/discontinued/distribution/11.4/repo/oss/suse/i586/.
[6] *systemd as default init system.* 2013. Sabayon. [ONLINE] Available at: http://www.sabayon.org/release/press-release-sabayon-1308
[7] *Ubuntu Package Search*. 2014. Ubuntu. [ONLINE] Available at: http://packages.ubuntu.com/search?suite=raring&arch=amd64&keywords=systemd.
[8] *systemd.* 2014. Wikipedia. [ONLINE] Available at: http://en.wikipedia.org/wiki/Systemd#cite_note-37.
[9] *systemd*. 2014. freedesktop. [ONLINE] Available at: http://www.freedesktop.org/wiki/Software/systemd/.
[10] *systemctl man page.* 2014. freedesktop. [ONLINE and USER ACCESSED] Available at: http://www.freedesktop.org/software/systemd/man/systemctl.html. Man page accessed on Fedora 20 terminal.
[11]*Useful Systemd Commands.* 2014. Dynacont. [ONLINE] Available at: http://dynacont.net/documentation/linux/Useful_SystemD_commands/.
[12] *SysVinit to Systemd cheatsheet.* 2014. fedoraproject. [ONLINE] Available at: http://fedoraproject.org/wiki/SysVinit_to_Systemd_Cheatsheet.
[13] *Github - Systemd.* 2014. Github. [ONLINE] Available at: https://github.com/systemd/systemd
[14] *[PATCH] systemctl: improve readability on failed commands.* 2014. gmane. [ONLINE] Available at: http://comments.gmane.org/gmane.comp.sysutils.systemd.devel/15813.
[15] *systemd, beyond init.* 2011. Lennart Poettering. [ONLINE VIDEO LECTURE] Available at: http://www.youtube.com/watch?v=TyMLi8QF6sw.
[16]*SysVinit.* 2014. Archlinux. [ONLINE] Available at: https://wiki.archlinux.org/index.php/SysVinit.
[17] *Get Fedora.* 2014. Fedoraproject. [ONLINE] Available at: https://fedoraproject.org/en_GB/get-fedora.
[18] *Yum.* 2012. Fedoraproject. [ONLINE] Available at: https://fedoraproject.org/wiki/Yum.
[19] *Style Guide for Python Code.* 2013. PEP8. [ONLINE] Available at: http://legacy.python.org/dev/peps/pep-0008/.
[20] Find report as pdf file named “Jack – systemctl.pdf” in mercurial repository - REDHg1
[21] *system.socket(5) man page.* 2014. [USER ACCESSED] Man page accessed on Fedora 20 terminal.