LFD107x

# OSS Development: Linux for Developers

Version 2022-01-19

Version 2022-01-19

Published by:

the **Linux Foundation**
https://www.linuxfoundation.org

# Contents

# Chapter 1

# Open Source Software

## Exercise 1.1: Basic Facts about Open Source Software

Which of the following statements are true?

1. Once fees are charged for for the use or distribution of software, it can no longer be termed **open source**.
2. If the control, content and timing of the official software releases is controlled by one entity (such as a company), it can no longer be termed **open source**.
3. The first widely used **OSS** product was the **Linux** kernel, first released in 1991.
4. If a program is written using an open source computer language (such as **Python** or **Perl**) it must be released under an open source license.

## Solution 1.1

1. **False**: When the word **free** is used in conjunction with **OSS**, it is used in the sense of freedom, not no cost. One can charge any sum for the software and still fit in the category of **OSS**.
2. **False**: **Android** is a good example of an important product ultimately controlled by one authority (**GOOGLE**), where all code is openly available, but the company controls the code base and release schedules and in general does not release all information in advance of new versions, although it does give more confidential heads up information to corporate partners using the operating system, such as mobile manufacturers.
3. **False**: Software has been released in source form since the beginnings of the computer age. A big advance as the history progressed was a growing consciousness about the importance of carefully crafted licenses.
4. **False**: The language used does not control the license of an application or other software; it can still be proprietary and closed source.

## Exercise 1.2: Open Source on Critical Software

Consider some of the arguments for whether or not it is a good idea to open the source for some kinds of critical software.

Examples could include:

- Medical devices, such as pacemakers, metering of life signs, controlling drug dosage through intravenous tubes etc.
- Electronic voting machines.

- Automobile software, include that for vehicle performance (emissions, braking, etc.) and newly emergent autonomous vehicles.

Do you consider it less or more safe and/or secure if the source for these kinds of products is freely available?

## ✓ Solution 1.2

Of course the course authors have a certain opinion!

- A hostile actor could grab control of your pacemaker, automobile and hurt or kill you, as has been seen on TV!
- It is a perennial argument, but the OSS community believes the more eyeballs that inspect the code, the more easily flaws and holes can be found and fixed.
- If the source is closed, a bad actor could obtain it one way or another and be able to hack it and use it harmfully, sometimes even quite easily if the code is weak or sloppy.
- There have been many demonstrations, even by high school students, showing how easy it is to hack electronic voting machines with rudimentary tools. See
  https://theguardian.com/technology/2018/aug/22/us-elections-hacking-voting-machines-def-con.

# Chapter 2

# Why Use Open Source Software?

## Exercise 2.1: How OSS Methods Can Produce a Better Product

Enumerate a few reasons how adopting **OSS** methods can improve and accelerate a product's success.

## Solution 2.1

1. Expanding the pool of collaborators brings in more talent, more ideas.
2. Competing groups become collaborators and do not have to keep reinventing the wheel. By standardizing on one implementation and building upon it, progress can be faster.
3. More eyeballs on the code leads to more bug and security hole discovery as well as faster fixes.
4. More eyeballs on the code leads to improved and cleaner style. Furthermore, having style standards throughout a project leads to increased clarity and lessens the path to involvement by new contributors.
5. Getting recognition from a wider audience motivates people.
6. Having a wider variety of use cases induces a project to have more flexibility and also helps unearth problems that might be rarer and harder to identify with a smaller audience.

# Chapter 3

# Examples of Successful OSS Projects

## Exercise 3.1: What OSS Products Do You Use?

List some of the open source projects you probably use day to day.

## Solution 3.1

1. **Android**

   Used in the majority of smart phones and other mobile devices world wide, built on top of the **Linux** kernel

2. **Apache Web Server (httpd)**

   About half of all web servers are running **Apache** and its related products.

3. **Social Media**

   Virtually all social media platforms (**Facebook**, **Twitter**, **Instagram** etc.) are using **OSS** components throughout.

4. **Search Engines**

   **GOOGLE** and other search engines have vast data farms running **Linux**.

5. **Weather Forecasting**

   Essentially 100 percent of the world's supercomputers run **Linux**, and forecasting the weather is one of the most intensive uses.

6. **Personal Fitness Devices**

   Including **FitBit**.

7. **DVRs**

   Almost all set top boxes and video recorders run **Linux**

8. **Medical Devices**

   A large number of medical devices used routinely every day are running **Embedded Linux**.

All these projects make extensive use of **OSS** tooling, such as **gcc**, **make**, **glibc** and languages such as **Python**, **Perl** and **Ruby**.

# Chapter 4

# OSS Licensing and Legal Issues

## Exercise 4.1: Selecting a License

- Go to http://oss-watch.ac.uk/apps/licdiff/ and interactively specify the parameters needed to choose the right type of **OSS** license.
- Try a number of variations.

# Chapter 5

# How to Work in OSS Projects

## Exercise 5.1: Dealing with a Non-Responsive Maintainer

You have submitted a patch to a subproject maintainer and have received no response for a few days. You should:

- Quietly resend the patch every day until there is an acknowledgment of receipt, acceptance, or rejection. You can even use an automated email bot to do this.
- Since the maintainer is ignoring you, you should send the patch set to the entire project, through a mailing list if that is an appropriate channel. Make sure to tell people you are doing this because the maintainer is not doing their job and you are routing around them.
- Send a follow up inquiry referring to the patch to the maintainer, possibly resubmitting. Make sure you know the person's normal workflow and response times to see if you are being singled out. See if you can ascertain if the maintainer seems to be active but ignoring you, or is inactive in general.

## Solution 5.1

Obviously the last answer is most appropriate. You need to be patient and respectful. When you achieve the exalted status of being a maintainer you will learn how stressful, time consuming and often thankless a task it is. Maintainers are squeezed by contributors from below, and by upstream leaders and reviewers from above.

As usual there is not one exact protocol to follow. But be a good citizen and appreciate the work people do and next time you have a change set, your treatment can only be better.

## Exercise 5.2: Is it Better to Submit Small or Large Patches?

Suppose you are making a rather large set of changes to a project. Is it better to:

- Submit a sequential series of bite-size patches that are easier to digest, even if there is not much functionality until all are accepted.
- Wait until the work is essentially complete and submit one or two large change sets.

## Solution 5.2

Usually small and incremental patch sets are preferred. This has the advantage that they are easier to understand, and if something is not quite right, it is preferable to catch it early and nip it in the bud.

Larger patch sets tend to overwhelm reviewers and maintainers. Sometimes they are just refused out of hand for that reason.

There is no hard and fast rule here and some projects might like to see the complete set. Sometimes it is very hard to test anything until the work is complete, as the intermediate evolutionary steps are just that. However, smaller patch sets are more often the preferred method.

# Chapter 6

# Leadership vs Control and Why Projects Fail

## Exercise 6.1: Enabling New Contributors

Enabling new contributors to join a project is essential to both longevity and vitality. It is the normal life cycle that many project members eventually leave a project for many reasons ranging from not enough time or lost interest, to disagreeing with changes in goals or governance issues or personal conflicts. New blood brings in both more labor or fresh ideas and tools.

What are some of the methods that can be used to ease the glide path of new project members and get them to the point where they both feel comfortable with the project and can make useful contributions?

## Solution 6.1

- Have a structure for mentorship, for attaching new members to patient long time contributors. These can acclimate new people to the project ecosystem and teach them proper practices.
- Keep discussions in the open. If decisions are made behind closed doors and it looks like those with the strong connections to project leadership have inordinately favored behaviour, new members will be confused and possibly discouraged.
- Make sure reviews and rejections are based on issues and have clear explanations.
- Avoid abusive, even toxic tones in discussions, even between senior members who know each other well and may not take offense, possibly due to cultural factors. Of course one not should not go crazy and install over-bearing thought police, either internally or externally, but be sensitive. New members may not know that the person you just (humorously) insulted is your oldest friend and they may be intimidated and fear they will be treated with the same irreverence.
- If you are a maintainer, don't just exert control over submissions because you can. Lead by explaining the logic of decisions, by suggesting what would be a more acceptable change set, and encourage others to contribute their views rather than ruling soley by fiat.

# Chapter 7

# Respecting and Encouraging Diversity in OSS

## Exercise 7.1: Offensive but Extraneous Behaviour

A member of an **OSS** project gets into hot water because of behaviour, or views expressed, in a milieu extraneous to the project. Examples might include:

- Expression of strong political views that are sure to offend some project members. This can happen if the views are far right, far left, as well as not easily characterized.
- Support for a particular political leader or candidate, or political system.
- Personal behaviour that never directly impacts the project, but is deeply offensive to members. This might be homophobic, racist, intolerant of a religion etc. Or it might be an arrest or conviction for criminal behaviour etc.

## Solution 7.1

There is no easy answer here and **OSS** projects have had very difficult (and sometimes interminable) debates on these issues. Keep in mind that most contributors have a strong belief in individual freedoms and that includes expression of beliefs and the right to a desired lifestyle.

However, it's not hard to conjure up examples where one simply cannot accept certain extreme individuals into the **OSS** family. About all one can say is that discussion should be respectful, any established codes of conduct should be respected and revised as necessary, and if possible discussion should be placed in channels where ti does not overwhelm technical debate.

## Exercise 7.2: Tone of Debate

You submit a patch to an **OSS** project and it engenders some strong critiques. Some of the technical questions raised deserve a good and productive debate. However, one of the reviewers is rather harsh, and uses rather strong language, some of which could be interpreted as ad hominem personal attacks. How should you respond?

## Solution 7.2

Some of the choices are:

1. Ignore the over the top attacks and just answer the technical points.
2. Directly confront the inappropriate tenor of the remarks and respectfully ask the project member to tone it down. Do this as a preface to answering the technical points.
3. Respond in kind to the bully and slap them down.

4. Rely on senior project members to reach out to the offending commenter, and take action as needed, including expulsion from the project.

There is no easy answer to this question. This kind of situation happens with varying frequency on **OSS** projects. Nobody wants lengthy non-productive digressions and nobody wants to feed **trolls**. The only clear answer is one should not respond in kind and accelerate the conflict. Within a given project there is probably an explicit code of conduct and senior maintainers, and these are what set the framework for dealing with such uncomfortable matters.

# Chapter 9

# GitHub and Other Hosting Providers

## Exercise 9.1: Getting a GitHub Account

Go to https://github.com and open up a free of charge account if you do not already have one. Note that all **GitHub** accounts (including free) can have private repositories. You may use this newly created account later in this course.

## Exercise 9.2: Getting a GitLab Account

Go to https://gitlab.com and open up a free of charge account if you do not already have one. Note that all **GitLab** accounts (including free) can have private repositories. You may use this newly created account later in this course.

# Chapter 10

# Linux and the Operating System

## Exercise 10.1: sudo

- It is very dangerous to run a **root shell** unless absolutely necessary: a single typo or other mistake can cause serious (even fatal) damage. Thus, the sensible procedure is to configure things such that single commands may be run with superuser privilege, by using the **sudo** mechanism. With **sudo** the user only needs to know their own password and never needs to know the root password.
- If you are using a distribution such as **Ubuntu**, you may not need to do this lab to get **sudo** configured properly for the course. However, you should still make sure you understand the procedure.
- To check if your system is already configured to let the user account you are using run **sudo**, just do a simple command like:

  ```
  $ sudo ls
  ```

  You should be prompted for your user password and then the command should execute. If instead, you get an error message you need to execute the following procedure.
- Launch a root shell by typing **su** and then giving the **root** password, not your user password.
- On all recent **Linux** distributions you should navigate to the `/etc/sudoers.d` subdirectory and create a file, usually with the name of the user to whom root wishes to grant **sudo** access. However, this convention is not actually necessary as **sudo** will scan all files in this directory as needed. The file can simply contain:

  /etc/sudoers.d/student

  ```
  student ALL=(ALL)  ALL
  ```

  if the user is `student`.
- An older practice (which certainly still works) is to add such a line at the end of the file `/etc/sudoers`. It is best to do so using the **visudo** program, which is careful about making sure you use the right syntax in your edit.

## Text Editing Basics

- It may not be obvious to you how to create or edit such a file, especially if you are logged in as a normal user and then have used **su** to run as root.
- In particular, using a graphical editor such as **gedit** would require dealing with some complications because root does not own the display screen.
- If you are not yet familiar with using **vi** or **emacs** you can easily use **nano**, which is very simple to use and has information in the bottom panel about all needed commands. So you could do:

```
# log in as root
$ su
....
# nano /etc/sudoers.d/student
```

```
....
Ctr-X (to save file and exit)
```

- Or you can type at the command line:

```
cat << EOF > student
student ALL=(ALL)  ALL
EOF
```

- You probably also need to set proper permissions on the file by typing:

```
$ sudo chmod 440 /etc/sudoers.d/student
```

(Some **Linux** distributions may require `400` instead of `440` for the permissions.)

- After you have done these steps, exit the root shell by typing `exit` and then try to do `sudo ls` again.
- There are many other ways an administrator can configure **sudo**, including specifying only certain permissions for certain users, limiting searched paths etc. The `/etc/sudoers` file is very well self-documented.
- However, there is one more setting we highly recommend you do, even if your system already has **sudo** configured. Most distributions establish a different path for finding executables for normal users as compared to root users. In particular the directories `/sbin` and `/usr/sbin` are not searched, since **sudo** inherits the `PATH` of the user, not the full root user.
- Thus, in this course we would have to be constantly reminding you of the full path to many system administration utilities; any enhancement to security is probably not worth the extra typing and figuring out which directories these programs are in. Consequently, we suggest you add the following line to the `.bashrc` file in your home directory:

```
PATH=$PATH:/usr/sbin:/sbin
```

If you log out and then log in again (you do not have to reboot) this will be fully effective.

# Chapter 11

# Graphical Environments and Interfaces

## Exercise 11.1: Using Multiple Workspaces

- The most common Desktop Managers in use in **Linux** have the capability of using multiple **workspaces** or **desktops** simultaneously.
- This is not the same as using multiple monitors, for which **Linux** has mature functionality.
- On recent **GNOME** desktops you should see the workspace widget in the lower right corner. You can configure properties such as how many alternative desktops are available by running **gnome-tweaks** (or **gnome-tweak-tool on old distributions**) which you can install if it is not already present in your system.
- You can then switch to a different workspace by clicking on the applet in the appropriate space, or hitting `Ctl-Alt-arrow`, where `arrow` can be the left, right, up or down arrow.
- How you can move a running application to a different workspace depends on the **GNOME** version you are running. Right-clicking on the title bar and following suggestions may work. Or you can hit the `Windows` key and drag the window you want to the desktop you want, which will be seen on the right hand side of the screen. make
- You can also choose to have the application show on **all** workspaces.
- If you are running a **KDE** desktop the relevant applet is called the **Pager** and the alternate screens are called **desktops**, not workspaces. It is similarly easy to add the applet and configure it, and to move applications among desktops.

## Exercise 11.2: Using Multiple Tabs in a Terminal Window

- Open up a terminal window: for **GNOME** you can do this by one of these methods:
  - Find the terminal application in the `Application` menu system, either under `Accessories` or `System Tools`.
  - Hit `Alt-F2` to bring up the `Run Command` window and type in `gnome-terminal`.
  - Right-click anywhere on the desktop; this may not work depending on how your system is configured.
- Right-clicking anywhere in the terminal window will offer a choice of opening either a new terminal or a new **tab**. On a few recent **GNOME** versions, you have to select which one you want to be offered by clicking on `Edit->Preferences` and then choose either `Window` or `Tab` under the dialog for `Open new terminals in`. Why this should be considered an improvement is rather obscure

  Or you can hit `Shift->Ctl-T` from within the terminal window.

- If you open multiple tabs, you can switch between them either with the mouse or by hitting `Alt-1`, `Alt-2`, etc.
- While you are at it, it would be a good time to learn how to configure the **profile** or create a new one, changing fonts, colors, etc.
- For **KDE** the **konsole** program can be used in much the same way, with new windows being launched with `Ctl-Shift-M` and new tabs with `Ctl-Shift-N`.

# Chapter 12

# System Administration

## Exercise 12.1: Installing A New Repository

**Please Note**

- This lab can only be done on a **RHEL** or **CentOS** system.
- You may already have the **EPEL** repository installed; for example if you are using a **Linux Foundation CentOS** virtual machine. In such a case, just make sure you know how to obtain the **RPM** file and examine `/etc/yum.repos.d/epel.repo`.

- Enterprise **Linux** distributors often offer only a relatively small subset of the total number of packages available and the versions supported are often not the latest cutting edge ones.
- This is done to permit better control over package interaction as the potential number of problems tends to rise rather dramatically as more software is included.
- However, there are many software packages that are well understood and probably can be added without big problems. While one can always install from source, binary packages are much easier to deal with.
- For **Red Hat Enterprise Linux** installations (including **CentOS**) a handy resource is the **EPEL** repository (**E**xtra **P**ackages for **E**nterprise **Linux**), which can be found at https://fedoraproject.org/wiki/EPEL. For the most part these are packages which have been used on **Fedora** systems, which are very similar to the current **Red Hat** system, and are expected to install cleanly and play well with the rest of the software on the system, although no technical support can be provided.
- To install the **EPEL** repository you need to download the **rpm** file from: https://fedoraproject.org/wiki/EPEL, selecting the version appropriate for your major release version, or you can obtain it from https://training.linuxfoundation.org/cm/LFD107x.
- You can then install as in:

  ```
  $ sudo rpm -Uvh epel-release*.noarch.rpm
  ```

  or to be explicit:

  ```
  $ sudo rpm -Uvh epel-release-8-6.el8.noarch.rpm
  ```

- You will notice this creates a file, `/etc/yum.repos.d/epel.repo` which you should examine as it is a template for how other repositories can be added.

# Chapter 13

# Getting Help

## Exercise 13.1: Multiple man pages

- Often there are multiple **man** pages (in different chapters of the manual) for a given topic. As an example we will give **stat**.
- If you just do:

```
$ man stat
```

```
STAT(1)                          User Commands                        STAT(1)

NAME
       stat - display file or file system status

SYNOPSIS
       stat [OPTION] FILE...

DESCRIPTION
       Display file or file system status.

.....
```

  you will get a description of the command line utility.

- Try doing:

```
$ man -k stat
```

  which is equivalent to doing **apropos stat**. You will get a long list of **man** pages that reference the string **stat**.

- To narrow the list down to pages that are an exact match do:

```
$ man -f stat
```

  and then you can get the page from chapter two by doing:

```
$ man 2 stat
```

- If you want to see all the pages, you can do

```
$ man -a stat
```

and then you can see them all in turn, typing `q` after each one is displayed to see the next one.

## Exercise 13.2: Using info

- Enter the **info** help system for **gcc** by doing:

```
$ info gcc
```

- If you type **?** (or **H** depending on **info** version) a screen will be brought up detailing the various key bindings and commands you can type.
- For example suppose you want to examine optimization levels that can be invoked. You can search forward either by hitting / or **s** and typing in the string `optimization`.
- Navigate around to different levels, search for other items and familiarize yourself with the **info** system.

# Chapter 14

# Text Editors

## Exercise 14.1: vi and emacs Tutorials

- There is no shortage of on-line tutorials on the use of the basic editors available. An excellent interactive **vi** tutorial can be found at: https://openvim.com.
- If you have never used **vi**, or your skills have grown rusty, this would be a good place to start or refresh.
- For **emacs** it is as simple as starting the program (you do not even have to give a file name) and then typing `Ctl-h` and then `t`. This will launch the built in **emacs** tutorial.
- Take a short time familiarizing yourself with one of these tutorials.

# Chapter 15

# Shells, bash, and the Command Line

## Exercise 15.1: Customizing the prompt

- Set the prompt to be:
  - current directory>
    (i.e. `/home/coop>` )
  - machine name:user:current directory>
    (i.e., `quad64:coop:/home/coop>`
  - anything else you would like (play!)

## Solution 15.1

`Please see SOLUTIONS/s_15/lab_prompt.txt`

## Exercise 15.2: Redirection and Pipes

- Try a command such as the following:

  ```
  $ ls /etc/passwd /etc/passwd_not
  ```

  where one file exists and the other does not.
- Get the **stdout** output of the command in one file and the **stderr** output in another.
- Now get them both to go to the same file.
- Now get the **stderr** output to go away to `/dev/null`.
- Now pipe the result of the **ls** command into **sort**, and get the **stderr** output into a separate file.

## Solution 15.2

```
Please see SOLUTIONS/s_15/lab_pipe.sh
Please see SOLUTIONS/s_15/lab_pipe.txt
```

# Chapter 16

# Filesystem Layout, Partitions, Paths and Links

## Exercise 16.1: Adding to the path

- Create a simple executable file with the name **ls** in your current directory, which we will assume to be `/tmp`:

```
$ cd /tmp
$ echo echo Hello, This is MY ls program > ls
$ chmod +x ls
```

- You can run this directly by doing:

```
$ ./ls
```

  but just typing **ls** will bring up the normal `/bin/ls`, which can be verified by typing `which ls`.

- If you do:

```
$ export PATH=/tmp:$PATH
```

  then typing **ls** will bring up your program no matter where you are sitting on the filesystem.

- This is different than doing:

```
$ export PATH=./:$PATH
```

  which puts the current directory first in the path, no matter where you are, or

```
$ export PATH=$PWD:$PATH
```

  which will put the current working directory at this time in your future path.

- Prepending your current directory to the path is generally **a bad idea** as it makes **trojan horses** easy to implement.

# Chapter 17

# System Initialization

## Exercise 17.1: Killing the Graphical User Interface

From within a graphical terminal (**gnome-terminal**, **konsole**, etc), we kill the current graphical desktop.

Your method will depend on your distribution, and your greeter program (**gdm**, **lightdm**, or **kdm**).

First we will bring down the **GUI**, which depending on your system will be done with one of:

```
$ sudo systemctl stop gdm
```

Now we restart the GUI from the text console with one of:

```
$ sudo systemctl start gdm
```

# Chapter 18

# Memory

## Exercise 18.1: Invoking the OOM Killer

- When the **Linux** kernel gets under extreme memory pressure it invokes the dreaded **OOM** (**O**ut **O**f **M**emory) **Killer**. This tries to select the "best" process to kill to help the system recover gracefully.
- We are going to force the system to run short on memory and watch what happens. The first thing to do is to open up a terminal window, and in it type:

```
$ sudo tail -f /var/log/messages
```

in order to watch kernel messages as they appear.

- An even better way to look is furnished by:

```
$ dmesg -w
```

as it does not show non-kernel messages.

- This exercise will be easier to perform if we turn off all swap first with the command:

```
$ sudo /sbin/swapoff -a
```

Make sure you turn it back on later with

```
$ sudo /sbin/swapon -a
```

- Now we are going to put the system under increasing memory pressure. You are welcome to find your own way of doing it but we also supply a program for consuming the memory:

lab_wastemem.c

```
/* simple program to defragment memory, J. Cooperstein 2/04
 */

#include <stdio.h>
#include <stdlib.h>
```

```c
#include <unistd.h>
#include <string.h>
#include <sys/sysinfo.h>
#include <signal.h>

#define MB (1024*1024)
#define BS 16                          /* will allocate BS*MB at each step */
#define CHUNK (MB*BS)
#define QUIT_TIME 20
void quit_on_timeout(int sig)
{
        printf("\n\nTime expired, quitting\n");
        exit(EXIT_SUCCESS);
}

int main(int argc, char **argv)
{
        struct sysinfo si;
        int j, m;
        char *c;

        /* get total memory on the system */
        sysinfo(&si);
        m = si.totalram / MB;
        printf("Total System Memory in MB = %d MB\n", m);
        m = (9 * m) / 10;         /* drop 10 percent */
        printf("Using somewhat less: %d MB\n", m);

        if (argc == 2) {
                m = atoi(argv[1]);
                printf("Choosing instead mem = %d MB\n", m);
        }

        signal(SIGALRM, quit_on_timeout);
        printf("Will quite in QUIT_TIME seconds if no normal termination\n");
        alarm(QUIT_TIME);

        for (j = 0; j <= m; j += BS) {
                /* yes we know this is a memory leak, no free,
                 * that's the idea!
                 */
                c = malloc(CHUNK);
                /* just fill the block with j over and over */
                memset(c, j, CHUNK);
                printf("%8d", j);
                fflush(stdout);
        }
        printf("\n\n    Sleeping for 5 seconds\n");
        sleep(5);
        printf("\n\n    Quitting and releasing memory\n");
        exit(EXIT_SUCCESS);
}
```

It takes as an argument how many MB to consume. Keep running it, gradually increasing the amount of memory requested until your system runs out of memory.

**Please Note**

You should be able to compile the program and run it by just doing:

```
$ gcc -o lab_wastemem lab_wastemem.c
$ ./lab_wastemem 4096
```

which would waste 4 GB. It would be a good idea to run **gnome-system-monitor** or another memory monitoring program while it is running (although the display may freeze for a while!)

- You should see the **OOM** (Out of Memory) killer swoop in and try to kill processes in a struggle to stay alive. Who gets clobbered first?

## Solution 18.1

```
Please see SOLUTIONS/s_18/lab_wastemem.c
Please see SOLUTIONS/s_18/lab_waste.sh
```

# Chapter 19

# Command Details

## Exercise 19.1: Installing and Running ksysguard

- You may already have **ksysguard** installed; check by doing

  ```
  $ which ksysguardd
  ```

  If not you will have to install and it is recommended you use your distributor's packaging system, as installing from source can be a little tricky due to all the development headers and libraries required.

  On **Red Hat**-based systems you need

  ```
  $ sudo [dnf|yum] install ksysguard*
  ```

  On **openSUSE** you will need to do

  ```
  $ sudo zypper install kdebase4-workspace
  ```

  On **Ubuntu**/**Debian** systems the appropriate command is:

  ```
  $ sudo apt-get install ksysguard
  ```

  Of course depending on your exact **Linux** distribution and version there might be some variation of these names. You should be able to find the right package name through the appropriate graphical package management utilities if necessary.

- The following instructions should work with any version of **ksysguard**, but the graphical interface will vary somewhat according to version and **Linux** distribution.
- Launch **ksysguard**. Click on `File->New Worksheet` and then `Edit->Properties` to control the number of rows and columns and the update frequency.
- You can control what **sensors** you want to display by clicking on your machine name in the left pane and then dragging and dropping choices to the worksheet windows.
- You can display multiple sensors of similar type within a given window, including those of different CPU's or cores.
- If you have more than one machine available, you can display at the same time by selecting `File->Connect Host`. However, the other machine must be running a very similar version of **ksysguard** for this to work.

## Exercise 19.2: Using stress

- From time to time it will be useful to us to stress the system by making the cpu labor, wasting memory, or kicking up I/O activity. The appropriately named **stress** utility is a **C** language program written by Amos Waterland at the University of Oklahoma under the **GPL v2**. It is designed to place a configurable amount of stress by generating various kind of workloads on any kind of **POSIX** system.
- A newer enhanced version of **stress** (called **stress-ng**) is also available and has an almost infinite number of options, including over 105 stress tests.
- All major distributions now have both of these programs in their repositories, so you should be able to do one of:

```
$ sudo dnf     install stress stress-ng # RHEL/CentOS/Fedora
$ sudo apt-get install stress stress-ng # Debian/Ubuntu
$ sudo zypper  install stress stress-ng # openSUSE/SLES
```

- **stress-ng** is backwards compatible with **stress**, so wherever we use **stress** for here on, you can substitute **stress-ng**.

```
$ stress-ng --help
```

  for a quick list of options.
- As an example the command:

```
$ stress -c 8 -i 4 -m 6 -t 20s
```

  will:
  - Fork off 8 CPU-intensive processes, each spinning on a `sqrt()` calculation.
  - Fork off 4 I/O-intensive processes, each spinning on `sync()`.
  - Fork off 6 memory-intensive processes, each spinning on `malloc()`, allocating 256 MB by default. The size can be changed as in `--vm-bytes 128M`.
  - Run the stress test for 20 seconds.

## ✓ Solution 19.2

```
Please see SOLUTIONS/s_19/lab_stress.sh
```

# Chapter 20

# Users and Groups

## Exercise 20.1: Working with User Accounts

1. Create a `user1` account using **useradd**.
2. Login as `user1` using **ssh** or **su** or **sudo**. For example, you can just do this with:

   ```
   $ ssh user1@localhost
   ```

   It should fail because you need a password for `user1`; it was never established.
3. Set the password for `user1` to `user1pw` and then try to login again as `user1`.
4. Look at the new records which were created in the `/etc/passwd` and `/etc/group`.
5. Look at the `/etc/default/useradd` file and see what the current defaults are set to. Also look at the `/etc/login.defs` file.
6. Create a user account for `user2` which will use the **Korn** shell (**ksh**) as its default shell. (if you do not have /**bin**/**ksh**, install it or use the **C** shell at /**bin**/**csh**.) Set the password to `user2pw`.