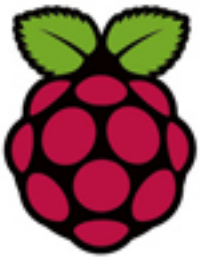


A Punnet of Berries - Documentation

TeamPi: Adrian Zielonka, Alyssa Biasi, Zach Ryan

COSC1114 - Operating Systems Principles: Friday 14:30

CONTENTS

CHAPTER

ONE

INTRODUCTION

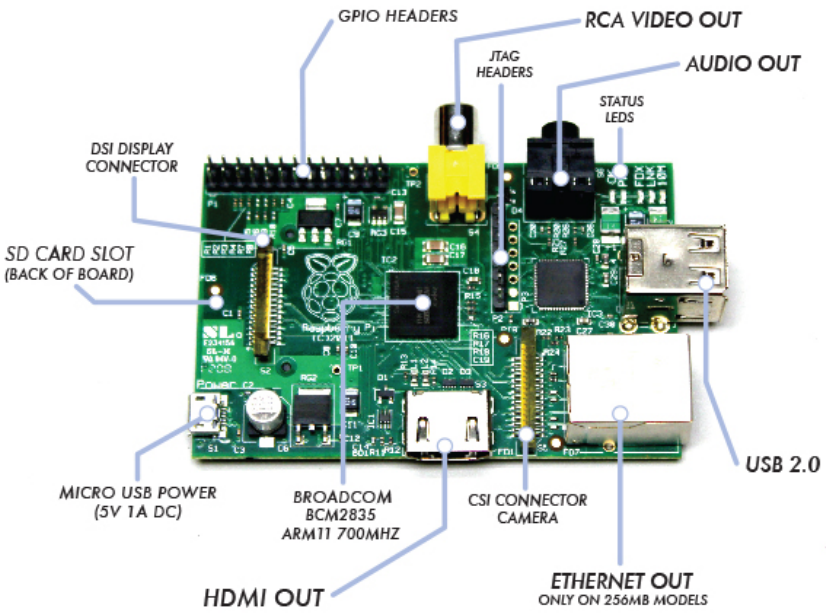
The aim of the **Punnet of Berries** project was to create a financially viable Beowulf cluster, evaluate the power of the Raspberry Pi and demonstrate scheduling concepts.

This document outlines the frameowrk on which the **Punnet of Berries** and its accompanying software were developed. System design, architecture and implementation are all covered in the subsequent sections.

1.1 The Punnet of Berries

A Beowulf cluster is a supercomputer built out of a collection of (typically) inexpensive computers. The computers are networked together by a local area network, usually Ethernet, and run a parallel processing software. The individual computers become the nodes of the supercomputer, which are combined together to form a single system. This concept puts the power of supercomputing into the hands of small research groups and schools. It gives them the ability to access the computational power that normally only large corporations can afford.

The Raspberry Pi is a single-board, credit card sized device. It is capable of running Linux, and other light-weight operating systems, with its ARM processors.



The Raspberry Pi is a powerful, yet, low-cost "mini-computer". It was developed to educate and inspire the next generation of programmers. **TeamPi** believes that this makes the Raspberry Pi an ideal candiate to create a Beowulf cluster.

The **Punnet of Berries** is a *Raspberry Pi Beowulf cluster*.

1.2 The Berry Batch

In order for a compute cluster to function properly and fairly, its resources must be managed and monitored by a batch system scheduler. Batch systems enforce limits on the number of jobs running at one time and the resources available to them. Compute cluster users request resources by submitting jobs to the batch system, which then determines the scheduling that best utilises the resources available.

The **Berry Batch** is a custom batch system, which manages the Punnet of Berrries compute cluster.

As one of the aims of the Punnet of Berries is to demonstrate scheduling concepts, the Berry Batch implements several scheduling algorithms. The user is able to select which algorithm to use when the cluster is initialised.

CHAPTER

TWO

GOALS AND OBJECTIVES

The key goals of the Punnet of Berries project were to:

- Create a Beowulf compute cluster
- Develop a basic batch system scheduler
- Demonstrate scheduling concepts
- Develop descriptive documentation so that others may recreate the Punnet of Berries project

A financially viable Beowulf cluster was created as a part of teh Punnet of Berries project. The cluster is energy efficient and demonstrates the advantages of parallel computing.

CHAPTER

THREE

SELF ASSESSMENT

3.1 Milestone 01

3.1.1 Reflection

Milestone 1 involved following a cross-compiler recipe and becoming familiar working in a command line Unix environment.

The image was build by one TeamPi member, Alyssa Biasi. Alyssa already has considerable experience working on the command line. After spending some time getting to know the recommended tool, `tmux`, she followed the instructions given in the recipe and performed the sanity check. No problems were encountered during the build.

uClibc vs. glibc: Clibc was developed without consideration for other architectures and thus is a C library targeted directly for embedded Linux. glibc was intended for higher output performance and has additional features not required on a Raspberry Pi. The omitted features allow uClibc to function with a smaller amount of memory.

Cross-Compliers: A cross-compiler is critical in the creation of executables that are able to function on architectures that differ from the platform on which the code was developed.

3.1.2 Result

OSP Check sheet 1 – Cross Compiler

Input	Expected result / output	Verified
cd $HOME/cross/cross-tools/bin ldd arm-unknown-linux-uclibcgnueabi-ar | grep `whoami`	If this returns nothing then the student did not build the compiler as this user.	✓
cd $HOME/cross/rootfs	Should return 245	✓
find . -type d | wc -l	Should return 1815 245	✓
cd $HOME/cross/images	boot the image the student created.	✓
/share/tools/bin/qemu-system-arm -M versatilepb -cpu arm1176 -m 256 -kernel kernel-qemu -initrd fs.img -append "root=/dev/ram rdinit=/bin/sh console=ttyAMA0" -nographic	Should see a "#" prompt	✓
which find	Returns /usr/bin/find	✓
find . | wc -l	Returns 2101 2100	✓
vi -v	Should fail. Returns /bin/vi: invalid option -- v BusyBox v1.21.0 (2013-07-27 16:16:25 EST) multi-call binary. Usage: vi [OPTIONS] [FILE]... Edit FILE -c CMD Initial command to run ($EXINIT also available) -R Read-only -H List available features	✓
Ctrl-A x	Exit.	✓

Team name: Team P1 Lab time: Friday 14:30-15:30 Date of demonstration: 16/08/13

Result of demonstration: Requirements fulfilled ☑ Resubmit ☐ Demonstrator name: Peter George Signed:

CHAPTER

FOUR

CONSTRAINTS

4.1 Hardware: Raspberry Pi

The Raspberry Pi is a comparatively low end single-board computer. As such, the design process must accommodate for these hardware specifications:

- SoC (System on a Chip) Broadcom BCM2835
- 700 MHz ARM core
- VideoCore IV GPU
- 512 MB of SDRAM.

Aside from an SD card, the Raspberry Pi does not feature any sort of non-volatile storage. Efficiency is a concern when constrained to a low capacity memory card. There is also an interface bottleneck, as both the USB and Ethernet share the same bandwidth.

An additional concern is of power distribution in regards to scalability; this would likely be resolved with the utilization of custom power supplies, but nonetheless it is something to consider.

4.2 Software: Architectural Considerations and Code Compilation

The Raspberry Pi hosts a single-core ARM processor, operating at 700MHz (with overclocking available). Due to this, any software developed for a Raspberry Pi should be in languages that are high-level, architecturally dependent and need to be recompiled on the Pi itself.

As there are several pre-configured Linux operating system images for the Raspberry Pi, it was decided that it was not necessary to create a custom Linux-From-Scratch image. In order to choose an appropriate operating system, it was necessary to evaluate the compute cluster's basic needs. The key factors in the decision making process were:

- Performance
- Size
- Compatibility

With this in mind, each node in the Punnet of Berries runs the **Arch Linux Arm** operating system. Arch Linux is:

- ARM compatible.
- Light-weight
  - The entire image is ~150 MB.
  - The default installation is bare bones, with nothing extra included.

- Boots in around 10 seconds.
  - Allowing the entire cluster to *boot in under 20 seconds*.

CHAPTER

FIVE

METHODOLOGY

5.1 Collaboration Tools

The collaboration tools utilised for the development of the Punnet of Berries project reflect the Open Source nature of the project.

Version Control: All code and documentation relating to the Punnet of Berries project is hosted in repositories on GitHub under the *rmit-teamPi* organisation. GitHub was chosen as it allows individual team members to use either Git or SVN depending on their own preferences.

Project Management: The project and bug-tracking was managed using Redmine, an open source web-based project management tool. The team's Redmine account was hosted on HostedRedmine, a service that hosts standard Redmine projects for free.

5.2 Development Tools

Given the collaborative effort of the development process, it is prudent to define the best practices and design principles to abide by. All protocols must adhere to a certain standard to promote scalability and ease of use.

The design approach is one that highlights the employment of parallel processing, concurrency, and synchronization. Fundamental operating system concepts will be explored and integrated, exposing the Raspberry Pi as a viable platform for cluster computing as well as comprehension of underlying technologies.

Design Principles:

- **Scalability** A computer cluster is, by nature, designed to be scalable. This inherent design characteristic must be adhered to, avoiding any sort of strict node limitations.
- **Usability** As this project is largely for educational purposes, the cluster should be easily constructed with high cohesion. The final application interface should also be readily accessible.
- **Flexibility** The software should be customizable, allowing users to specialize; secondary choices should be made available to enhance the application.

CHAPTER

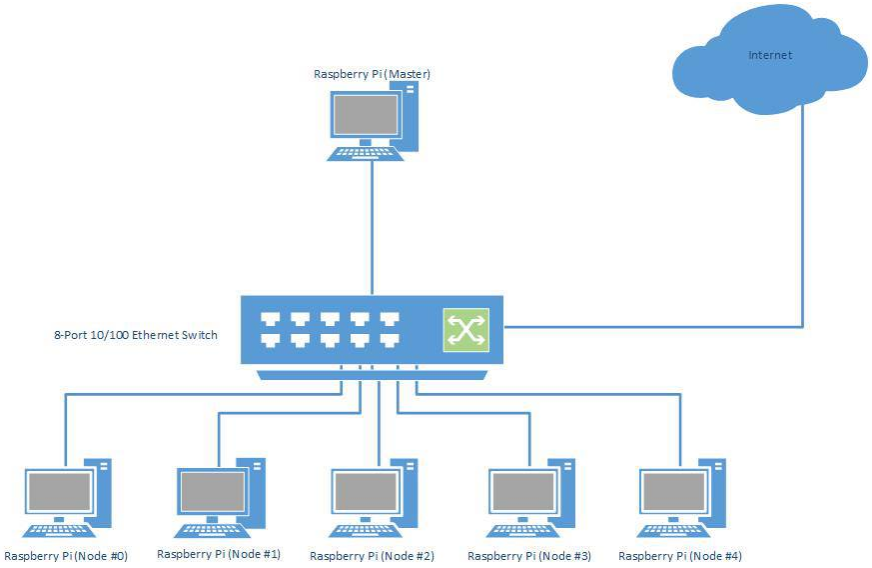
SIX

ARCHITECTURE

6.1 System Design

A chain is only as strong as its weakest link.

In a distributed model super computer, this statement cannot be more true. No matter how quickly each individual node can process its allocated tasks, if the communications link between the master system and each of the slave nodes becomes a bottleneck, then the entire system will suffer in performance.



In order to produce an efficient, flexible and reliable system, the selection of a networking topology which can fulfil those three specific characteristics is crucial. By selecting the star network topology as the basis for the Punnet of Berries super computer, the key requirements of efficiency, flexibility and reliability are met.

6.1.1 Efficiency

By design, star networks utilise a central hub such as a networking switch to handle to flow of packets in a network. This allows data packets to only travel to those nodes which they are intended for, reducing the overall traffic in the network.

6.1.2 Flexibility

By utilising a central hub for communications, a star network can easily be expanded to support additional nodes without affecting the entire network. Furthermore nodes can be removed at any time, if servicing is required.

6.1.3 Redundancy

By utilising a central hub for communications, the likelihood of total system failure is greatly reduced. Whilst failure of the central hub would mean that the entire system is brought down, failure of an individual node wouldn't have any affect on the overall system.

6.2 Program Design

The Berry Batch will consist of a centralised manager daemon and worker daemons.

Each node in the Punnet of Berries cluster will run a Berry Batch worker daemon. These workers exist to execute the jobs submitted to the system. The workers will:

1. Wait to be assigned a job.
2. Execute the job.
3. Returns the job's exit status.
4. Back to 1.

A job finished execution, with one of the following status:

1. **Successful** The job finished execution within the requested time frame and resources and without encountering any errors.
2. **Error** The job encountered an error during execution.
3. **Walltime Exceeded** The job exceeded the requested walltime and was terminated by Berry Batch.
4. **Resources Exceeded** The job required more system resources than requested and was terminated by Berry Batch.

The Berry Batch manager will monitor the system's resources and jobs. Users will interact directly with the Berry Batch manager to manage their job requests. A user is able to:

- Submit jobs.
- View all queued jobs.
- View the status of all of a particular user's jobs.
- View the status of a particular job.
- Cancel their own jobs.

6.2.1 Submitting Jobs

Users will write job scripts, which will be passed to the Berry Batch manager. A job script will consist of header and execution sections. The header section will specify the job details for queueing by Berry Batch. This could include a name for the job, the memory required, an estimated walltime and the number of nodes to run on. The execution section will contain the actual commands to be executed as a part of the job. While the job script's specific format is yet to be decided, an example is:

```
## Header Start ##
Name="JobHello"
Walltime=00:50:00
Nodes=2
Memory=256mb
## Header End ##

mpirun -np 2 helloWorld
```

Once a job has been submitted, the Berry Batch manager will determine which priority queue the job should wait in based on the resources and walltime requested in the header section. The queues could be:

- Short
- Medium
- Long
- Special (requires permission from the Punnet of Berries administration to run)

The *special* queue has first priority, followed by the *long* queue, and so on. If the resources are not available for any job in the *special* queue, the manager will look in the *long* queue for a suitable job, and so on. Each queue will operate in a *First in First Out* fashion.

6.2.2 Job Execution

Once a job has been submitted, it will move through the following states:

1. Queued
2. Running
3. Complete
4. Cancelled

When a job has completed, a Berry Batch job summary file will be written. This summary will contain details of the job's execution, such as the resources and walltime that were requested as well as what was actually used. Any standard output generated during execution will also be included in the summary file.

CHAPTER

SEVEN

TESTING

7.1 Tests Done

7.1.1 Cluster Performance Tests

In order to test the performance and functionality of the 'Punnet of Berries' cluster, the system will be tested using a 'HPL (High-Performance Linpack Benchmark)'. This benchmark provides guesstimates of how many GFLOPS (**G** iga **FL** oating-point **O** perations **P** er **S** econd) of processing performance the Punnet of Berries cluster is able to achieve.

Once the cluster itself has been benchmarked, the 'HPL' will then be run on a single Raspberry Pi as well as seperate system containing a modern x86/x64 based processor. This will provide a meaningful comparison with regards to the performance gain a cluster of six Raspberry Pi's has over various other systems.

7.1.2 Batch System Tests

To prove that the Berry Batch is functioning correctly, it will undergo a range of tests. These include, but are not limited to, the following:

Test Cases	Expected Result
Submit 1 short job.	Confirmation. Job put in short queue.
Submit 1 medium job.	Confirmation. Job put in medium queue.
Submit 1 long job.	Confirmation. Job put in long queue.
Submit 1 special job.	Job set to wait for permission.
One user submits a short job followed by one medium job.	First job put in short queue. Second job put in medium queue.
One user submit a short job. Second user submits a short job.	First user's job put in short queue. Second user's job put in short queue after the first.
One user submit a short job. Second user submits a medium job.	First user's job put in short queue. Second user's job put in medium queue.
One user submit a short job. Second user submits a medium job. Third user submits a medium job.	First user's job put in short queue. Second user's job put in medium queue. Third user's job put in long queue.
User views all current jobs.	List of jobs currently in the system.
User views a particular user's jobs.	List of user's jobs currently in the system.
User views a specific job.	Details of the given job.
First user views all current jobs. Second user views a specific jobs.	First user sees list of jobs currently in the system. Second user sees the details of the given job.
User cancels a queued job.	Confirmation of job cancellation.
User cancels a running job.	Confirmation of job cancellation.
User cancels a completed job.	Warning of invalid state.
User cancels an already cancelled job.	Warning of invalid state.
One user cancels another user's job.	Permission denied.

CHAPTER

EIGHT

ROLES AND RESPONSIBILITIES

The members of TeamPi are:

- **Alyssa Biasi:**

  Primary Role: Documentation, Testing and Management

  Secondary Role(s): Operating System Configuration

- **Adrian Zielonka**

  Primary Role: Hardward and Operating System Configuration

  Secondary Role(s): Application Development, Documentation

- **Zach Ryan**

  Primary Role: Application Development

  Secondary Role(s): Documentation

CHAPTER

NINE

WORK BREAKDOWN BY TEAM MEMBER

9.1 Alyssa Biasi's Log

9.1.1 Week 1

1. Raspberry PI cross complier recipe - Milestone 1.
2. Project research.

9.1.2 Week 3

1. Evaluation of project management tools

   > Trello

   > Gantter

   > Jira

   > Redmine
2. Setting up Redmine

   > Hosted by www.hostedredmine.com
3. Setting up repositories

   > hostedredmine.com only supports SVN

   > Repository hosted on www.github.com/rmit-teamPi

   - Using GitHub's "Organizations" to allow team access
   - GitHub provides support for SVN allowing individual members to pick their preferred method of version control.

9.1.3 Week 4

1. Arch Linux Arm image setup.

   > Partition layout changed in July 2013 (http://davidnelsono.me/?p=218)

9.1.4 Week 5

1. Setting up docutils, rst2pdf and Sphinx on a VM running Ubuntu for the generation of documentation.

    > apt-get install python-docutils

    > apt-get install rst2pdf

    > apt-get install python-sphinx

    > apt-get install texlive-latex-recommended

    > apt-get install texlive-latex-extra

    > apt-get install texlive-fonts-recommended

2. Work on design specification.

9.1.5 Week 6

1. Work on design specification.
2. Completed Milestone 2 - design specification.

9.1.6 Week 7

1. Setting up Arch Linux Arm images on 8GB SD cards.
2. Set up basic documents and sphinx for the final portfolio.

9.1.7 Week 8

1. Fixing SD cards to cluster specifications.

9.1.8 Week 11

1. Portfolio

9.1.9 Week 12

1. Portfolio

9.2 Adrian Zielonka's Log

9.2.1 Week 1

1. Brainstormed and researched viable project ideas.

9.2.2 Week 2

1. Brainstormed and researched viable project ideas.

9.2.3 Week 5

1. Worked on Design Specification.

9.2.4 Week 6

1. Worked on Design Specification.
2. Complete Milestone 2 - design specification.
3. Installed ArchLinux ARM (for Raspberry Pi) image (8GB SD Card).
4. Setup “masterpi” image for MASTER Raspberry Pi.

9.2.5 Week 8

1. Installed ArchLinux ARM (for Raspberry Pi) image (8GB SD Card).
2. Setup “slavepi0” image for SLAVE #0 Raspberry Pi.
3. Created baseline performance benchmark of single Raspberry Pi.

9.3 Zach Ryan’s Log

9.3.1 Week 1

9.3.2 Week 2

9.3.3 Week 3

9.3.4 Week 4

9.3.5 Week 5

9.3.6 Week 6

9.3.7 Week 7

CHAPTER

TEN

SUMMARY

CHAPTER

ELEVEN

REFERENCES

11.1 Raspberry Pi

Murray, M. (2012, July 13). *Raspberry Pi*. Retrieved from http://www.pcmag.com/article2/0,2817,2407058,00.asp

RPi Easy SD Card Setup. Retrieved from http://elinux.org/Rpi_Easy_SD_Card_Setup

Build Guide - Linux From Scratch on the Raspberry Pi. Retrieved from http://www.intestinate.com/pilfs/guide.html

Linux From Scratch. Retrieved from http://www.linuxfromscratch.org/lfs/view/development

11.2 Project Research

About Beowulf. Retrieved from http://yclept.ucdavis.edu/Beowulf/aboutbeowulf.html

Vaughan-Nichols, S. (2013, May 23). *Build your own supercomputer out of Raspberry Pi boards*. Retrieved from http://www.zdnet.com/build-your-own-supercomputer-out-of-raspberry-pi-boards-7000015831

Kiepert, J. (2013, May 22). *RPiCluster*. Retrieved from http://coen.boisestate.edu/ece/files/2013/05/Creating.a.Raspberry.Pi-Based.Beowulf.Cluster_v2.pdf

The Annual Unnamed UD Internet Contest - Problem 6: Supercomputer Job Scheduling. Retrieved from http://www.eecis.udel.edu/~breech/contest.inet.fall.09/problems/sc-sched.html

SImple Linux Utility for Resource Management. Retrieved from https://computing.llnl.gov/linux/slurm/overview.html

Sample PBS Script for Serial Job. Retrieved from http://qcd.phys.cmu.edu/QCDcluster/pbs/run_serial.html

11.3 Arch Linux ARM

Nelson, D. (2013, June 16). *Growing the root filesystem on Arch Linux ARM for the Raspberry Pi*. Retrieved from http://davidnelson.me/?p=218

Norman (2013, June 4). *Beginner's Guide to Arch Linux on the Raspberry Pi*. Retrieved from http://qdosmsq.dunbar-it.co.uk/blog/2013/06/beginners-guide-to-arch-linux-on-the-raspbery-pi/

Norman (2013, June 12). *Beginner's Guide to Arch Linux on the Raspberry Pi - Part 2*. Retrieved from http://qdosmsq.dunbar-it.co.uk/blog/2013/06/beginners-guide-to-arch-linux-on-the-raspbery-pi-part-2/

Beginners' Guide - ArchWiki. Retrieved from https://wiki.archlinux.org/index.php/Beginners'_Guide

Arch Linux Install Guide. Retrieved from http://elinux.org/ArchLinux_Install_Guide