Automatic Deployment Set Generation

The Problem

- Developer creates a software service on a developer box
- Must now deploy the service to a server in a data center
- Must calculate a **deployment set**--the set of files that need to be transferred to the server

The Problem--Difficulties

- Copying all files has security and licensing ramifications
- Developer may have forgotten which middleware they are using and how middleware is configured
- Can a tool automatically determine the deployment set?

Inspiration

- Look for cross-over ideas between source code management and configuration management
- In particular, look at the Vesta system
- Vesta has a mechanism for automatically tracking build dependencies via NFS

Possible Approaches

- Runtime
- Binary
- Source Code
- Metadata

Gathering Runtime Data

- I-Node
- System Call
- Application Specific
- Disk Blocks?

System Overview

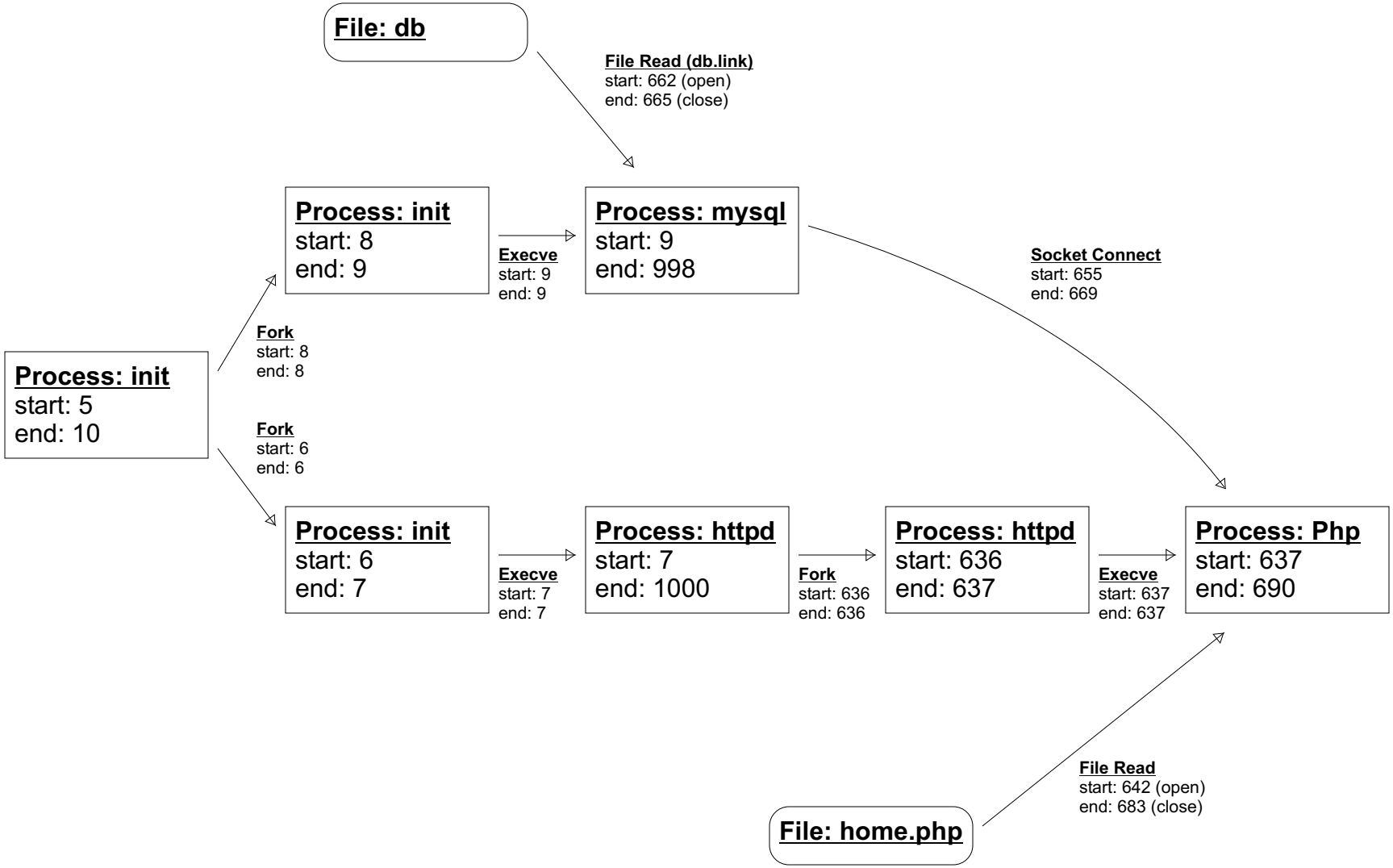
- Run the application
- System calls will be logged
- Process and file dependencies will be uncovered
- User specifies which processes are important
- Tool will use dependency information to generate as small a deployment set as possible

Let's Get Started

Prototype Tool Overview

- Fedora Core 3
- Configured as a web development system (3.96 GB)
- Log system calls using User Mode Linux
- Monitor process creation, exit, execve, file open, file close, socket stuff

Dependency Framework



Framework Operations

- Who are the descendants of a set of processes?
- Who are the ancestors of a set of processes?
- What is the path between two nodes?
- Show me a graph of ...
- Show me the ancestors/descendants of a set of processes up n levels deep

But Wait! What about Eval?

The usual approach:

> Here's my amazing algorithm. Let's run it against some tests. Look at how well it performs!

But Wait! What about Eval?

It is always possible to create a test case that stymies the algorithm

Therefore, bad results are meaningless

It is always possible to tweak the algorithm to pass a test case

Therefore, good results are meaningless

But Wait! What about Eval?

Machine learning solution is impractical here.

Instead

- Code the algorithms beforehand
- Do not allow further code changes unless they are bugs or procedure is documented as being a typical issue that a user might have to deal with

Evaluation? The Reality

Actually, I was just lazy, but that previous stuff sounds better

TEST SEQUENCES

ls

login> root
> ls
> halt

LAMP

login> root
> ifconfig eth0 192.168.1.3

Someone visits a web page that uses PHP to read a MySQL DB

> halt

Sanity Test

Take all accessed files as the deployment set

Problem: Certain system calls such as chdir, readlink, and stat64 are not monitored

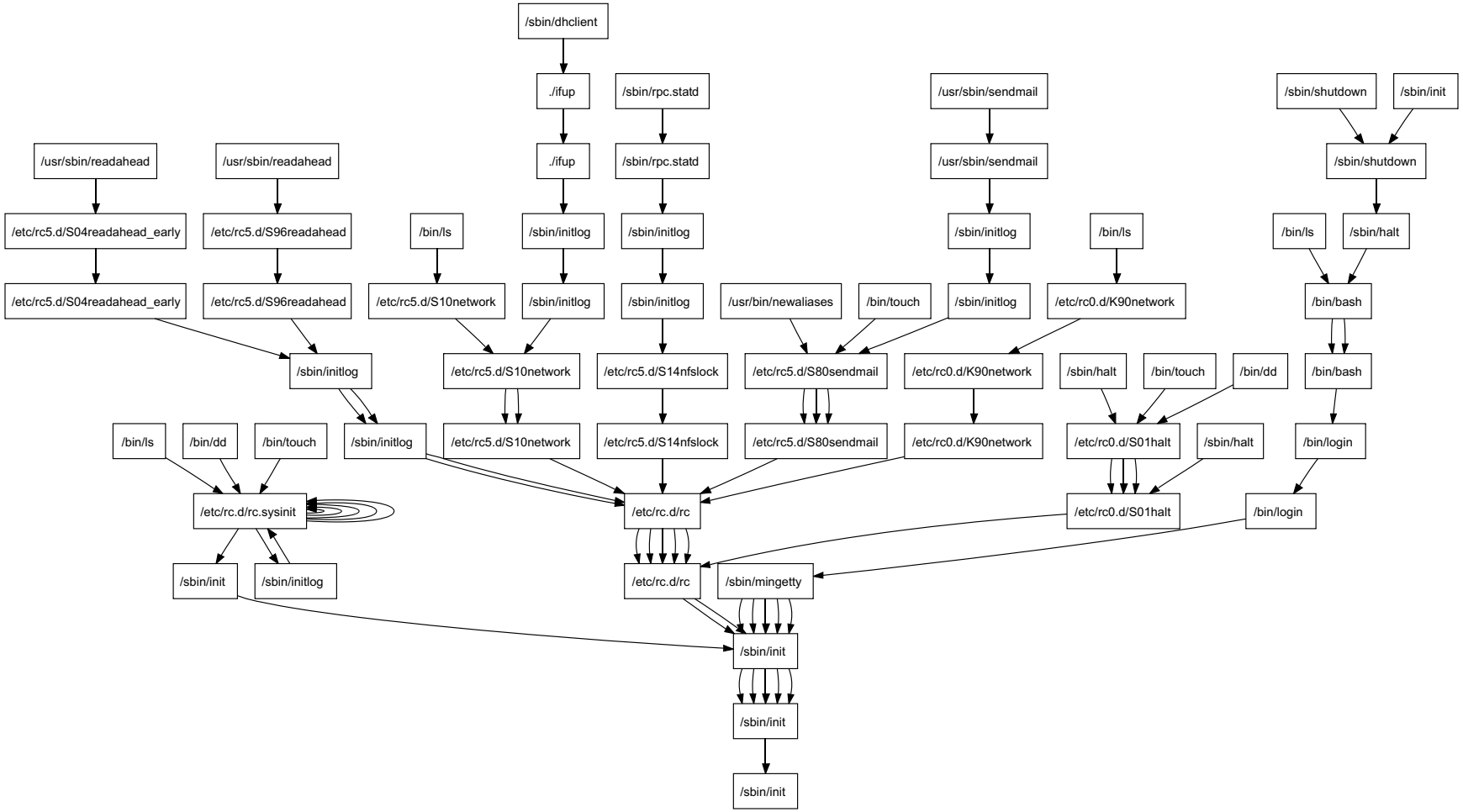
Strict Dependencies

Mark certain processes as important, and gather all dependencies. Use these files as the deployment set.

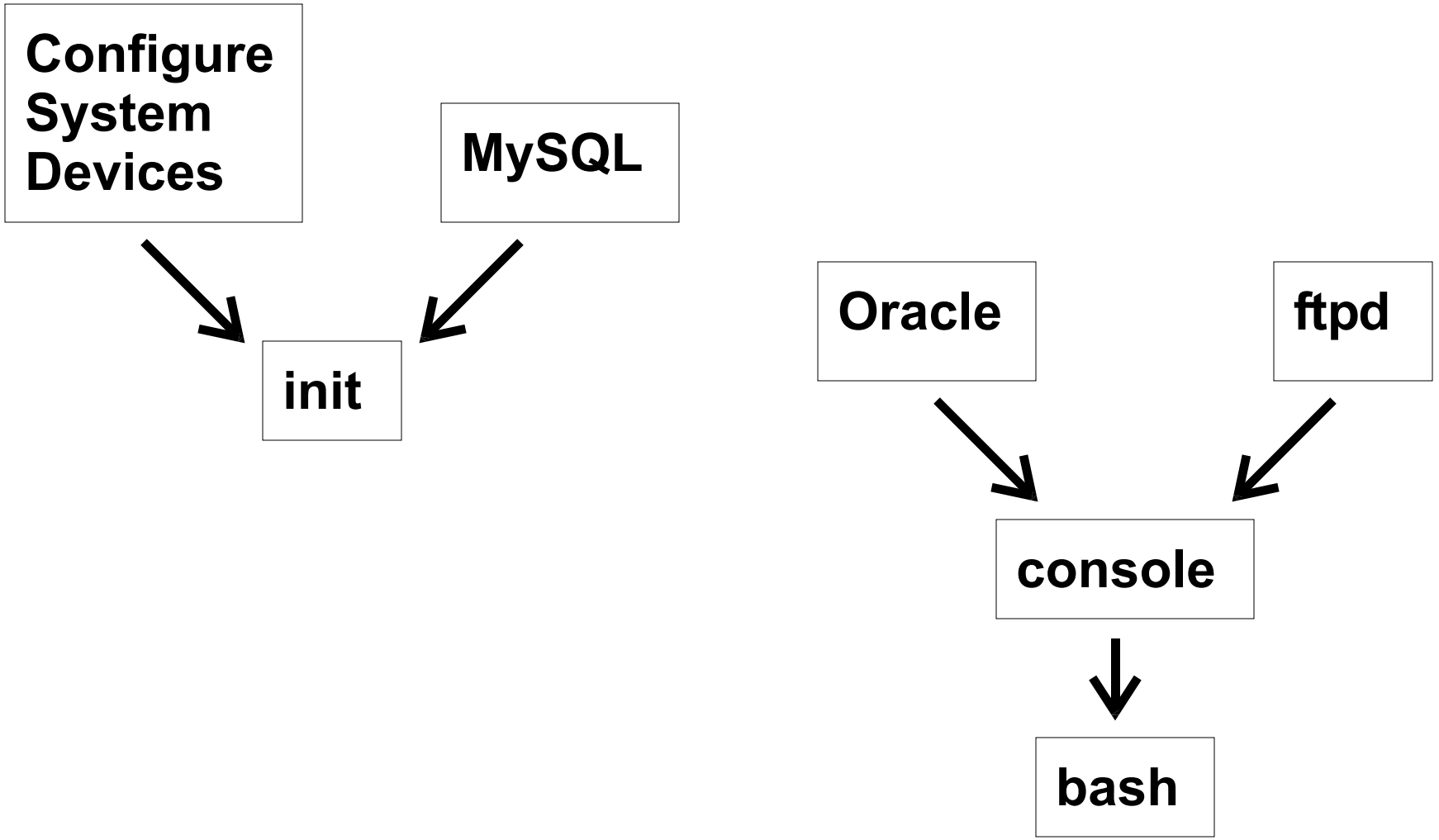
Problem: Some files (e.g. /dev/console) are read from and written to by many processes. These had to be manually tweaked.

Problem: No system startup processes

Dependencies of LS



Strict Dependencies

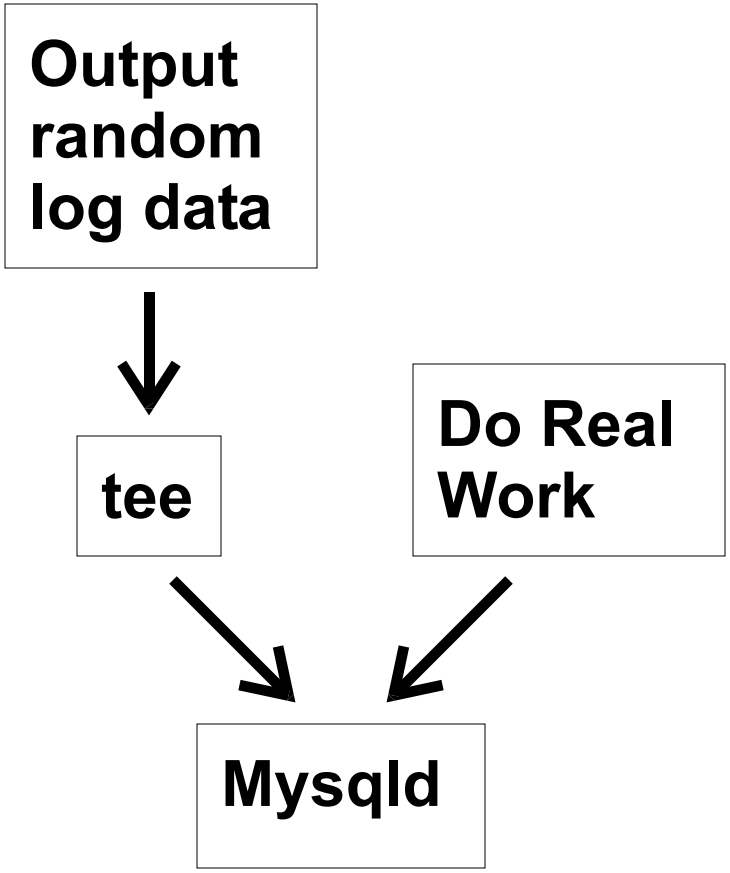


Dependencies & System Files

Add descendants of etc/rc.d/rc.sysinit, /sbin/runlevel, processes with no binary.

Problem: Sub-processes that serve no useful purpose are pruned, but parent processes still check whether these children were successfully exec'd or not.

Dependencies & System Files



Dependencies & Exec'd Files

Any process that is exec'd is included in deployment set (except those of Linux service initialization process /etc/rc.d/rc)

Problem: Files that are written to but never read (i.e. log files) may cause failures in key processes because they cannot be created if parent directory does not exist.

Summary of Results

	Test	Size	Functional
Strict Dependencies	ls	86 MB	No
	LAMP	99 MB	No
With System Files	ls	94 MB	Yes
	LAMP	107 MB	No
With Exec'd Files	ls	97 MB	Yes
	LAMP	110 MB	Yes *
All Accessed Files	ls	138 MB	Yes
	LAMP	138 MB	Yes *
Complete Installation	ls	3.96 GB	Yes
	LAMP	3.96 GB	Yes

Conclusions

- Take a Niche Problem
- Try the Obvious Solution
- Arrive at Expected Results

Conclusions

- Tool is not fully automatic
- User must intervene for tool to work well
- As such, the visualization capabilities of the tool are most important because they allow the user to understand problems and apply corrections

Future Work

- Handle multiple machines
- Support more platforms
- Deployment set diffing and merging
- Improved visualisation
- Meta-data