The Remodel Build System

The Remodel Build System, hereinafter referred to as remodel, is a replacement for make, which ships with most Linux distributions.

Remodel – The Good

Happens-before relationship

Make uses timestamps on files to calculate which parts of the compilation have to be redone. This is a very ineffective model. For one, source files copied in from another machine may have older timestamps due to clock skew between the machines. When make is used to rebuild, the files which were modified do not get recompiled and we are left with a stale binary.

Remodel, on the other hand, uses MD5 hashes to determine happens-before relationships. It stores the MD5 hashes of the files that it deals with in a special .remodel directory. When another compilation is done, it recalculates the MD5 hashes of the relevant files, compares them with that in the stored files, and uses this comparison to decide whether or not to recompile the file.

One non-obvious benefit is that files which are just saved again without any modifications, or identical files that are dumped in from a machine whose clock leads, will not lead to recompilation. This improves compilation time for large systems.

Parallelism

The other big advantage of remodel is that it is able to execute all the non-conflicting commands in parallel. It is astonishing that even after so many years of multi-core computing, an essential utility like make does all its work linearly. It is not quite surprising when we consider that timestamp based computation of happens-before does not allow for easy parallel execution.

Make does have a parallel execution mode, but it is broken and even simple programs do not build properly with the parallel mode on.

Remodel computes all the dependency relations as soon as it is initiated, and then creates a dependency tree out of the information. With the dependency tree, it is able to know exactly which productions are dependent on a production. This enables it to dispatch jobs to different processes.

Remodel has a central dispatcher that spawns processes as and when a production can be executed without any dependency issues. The central dispatcher does not have a lot of overhead, and thus, a lot of parallelism can be realized.

Remodel – The Bad

Lacking in feature-set

As remodel is still in its adolescence, there are many features demonstrated by make that are still missing here.

One prominent missing feature is macro support. Make allows for elegant representations of the build instructions using macros that can be reused in multiple places. Remodel’s architecture does not preclude the possibility of including this feature in the future. It requires an additional parse step while building the production list.

Currently, there is no support for comments, and blank lines in the file will lead to errors. There is no reason why these cannot be implemented trivially in future versions.

Command line options are missing beyond the ability to specify the root of the tree. This means that remodel can be executed only if remodfile is present in the present working directory.

Not a Drop-In Replacement

An existing makefile cannot be used by remodel. It requires a specially designed remodfile. This may slow adaptation.

Architectural Details

The program can be divided broadly into 3 parts.

The first part parses the remodfile and stores the production information into a list of productions. Each production has a list of targets, a list of dependencies and a command that builds the targets from the dependencies.

The second part constructs the dependency tree. This uses the optional parameter that the user may have supplied to determine the root of the tree. Each node of the tree can have multiple parents and multiple children – essentially a directed acyclic graph with a single root. This was essential to represent the myriad ways in which compilations have dependencies.

The final part obtains the leaves of the tree and spawns children processes to deal with each leaf. Even if a single child has returned, the program reprocesses the tree and determines whether any new leaves have evolved due to the completion of execution of the previous production. This ensures that if a command can be executed in parallel, it is.

The choice of a central dispatcher is potentially contentious, but it has been adopted due to a significant reduction in complexity and because it does not affect parallelism too much. The dispatcher is simple and scales reasonably well because the tree structure that it utilizes does not have a lot of dead-weight. The assumption is that the compiler will consume a lot more time than the dispatcher. This is a reasonable assumption to make.

Child processes have a simple job. They determine if the dependency has changed. If it has, they re-execute the command to build the new target.

The module used to do MD5 hash calculations is the work of Aladdin Enterprises.

Challenges

The main challenge was the construction of the tree. A lot of container nodes had to be created to store nodes in different contexts.

Deletion after a production had finished executing was also tricky because a lot of references had to be removed carefully without affecting the integrity of the tree structure.

Outcome

The program has been tested with 10 test cases which exercise different capabilities.

While timing results are unavailable, remodel builds significantly faster than the native make utility in multiprocessor environments, especially for tree structures with large branching factors. No perceivable difference was found on single processor systems, but some slowdown is expected.