Making make parallel - legacy code nightmare

Marc Espie <espie@openbsd.org>

September 27, 2014
Make is simple... or is it?

```
.SUFFIXES: .c .o
c.o:
    cc -c $*.c
a.o: a.h
.SUFFIXES:
# disable all suffixes
b.o: b.h
.SUFFIXES: .c .o
# later
c.o: c.h

...
The #later line *reactivates* the .c.o rule, it never really went away.
Hindsight makes perfect

This is work I began 10 years ago
In retrospect, some things are obvious
So I would like to share the journey of discovery.
Why did I do that

Not my code
make is not even Unix code
Comes from a distributed OS called "sprite"
Epitome of student project gone wrong
It was not production code.

Impossible to avoid
Legacy makefiles all over the system
Ports system heavily uses peculiarities
Mission critical, as much a part of Unix as bash sh.
Initial goal: make things faster
Why did I do that

Undocumented features

- make was badly specified
- realize that even `.PHONY` is not standard
- parallel completely an extension

We got to have a plan

... actually, I didn’t. I started looking at small changes
In retrospect, my initial goal was to make it faster *without changing anything.*
Long story short

Success

Amiga port build took *seven seconds* to start. It went down to 1 second.

...

Marc Espie <espie@openbsd.org>  Making make parallel - legacy code nightmare
Long story short

Success
Amiga port build took *seven seconds* to start. It went down to 1 second.

Cheating
Admittedly, half of it was due to work on the ports tree!
make handles a lot of strings, but actually it doesn’t.

```bash
cd ${WRKSRC} && \
    ${ALL_FAKE_FLAGS} ${RUBY} install.rb --destdir=${WRKINST}
```

- handle string intervals
- hashing tables
- memory buffers
How bad was the code

- Make features "character buffer handling" functions: growable string buffers that you can add to.
- Those buffers were doubly terminated.
- I killed the second zero.
- Code crashed.
- Fixed the bug.
- Code crashes again.
Designed to be fast and "all purposes". There are seven distinct hashing tables in make:

- Variable names
- Target names
- Known directories
- Transformation suffixes
- Timestamps per-directory
- Archive names
- Archive members per-archive ...
open hashing

Designed to be fast and "all purposes". There are seven distinct hashing tables in make:

- Variable names
- Target names
- Known directories
- Transformation suffixes
- Timestamps per-directory
- Archive names
- Archive members per-archive ...

Make that 8, target equivalence
Went on to be used

- in m4
- in tsort
- in mandoc
- in signify
**RTFM**

```c
ohash_init(&t, sz, info);

hv = ohash_interval(start, &end);
slot = ohash_lookup_interval(&t, start, &end, hv);

ohash_find(&t, slot);
ohash_remove(&t, slot);
ohash_insert(&t, slot);
```

**Advantages**

direct access to the hv hash value means we can do switches on constants used for magic variables, for instance
Ouch

GNode *
Targ_NewGN(char *name)
{
    gn = emalloc(sizeof GNode);
    gn->name = strdup(name);
    ...
    gn->children = Lst_Init(FALSE);
    ...
    Lst_AtEnd(allGNs, (ClientData)gn);
}
Notion of Ctor distinct from memory allocation

Better

GNode *Targ_NewGNi(const char *name, const char *ename)
{
    gn = ohash_create_entry(&gnode_info, name, &ename);
    ...
    Lst_Init(&gn->children);
    ...
}

(later)

ohash_insert(&targets, slot, gn);
Divided number of memory allocations by

> 10
Make is full of small modules that call each other
None of them sane
The only way to make progress is through small changes
Until you understand one module
And can go on the rest
A simple list

- Vars - handles vars and substitution
- Targets - handles targets and the file system
- Parser - builds structure from makefile
- Cond - every dot bsd command
- Suff - suffixes handling
- Compat - old sequential engine
- Job - funky parallel engine
- Dir - directory caching
- Buf - string construction
- Arch - ar(1) handling
Apparently, each module is "seperated". But there are interactions built over the years.

**Baby dragons**

- Buffer handling
- Variable contents

But there might be (smallish) dragons...
variables in the Makefile
variables on the command line
environment
dynamic variables

Initially, four lists.
Now, just one global list, and one per-node
Plus lazyness, expanded as late as possible.
Tricky ? you bet

Still found a bug this year, related to variable expansion
Nice benefits, such as "recursive variables". (we could already do that through loops)
Also, pass command line recursively through .MAKEFLAGS.
Borrow netbsd extended .for loops (awesome idea)

```
.   for lnk file in ${MLINKS}
    @l=${DESTDIR}${MANDIR}${lnk:E}${sub}${lnk}; \
    t=${DESTDIR}${MANDIR}${file:E}${sub}${file}; \
    echo $$t -\> $$l; \n    rm -f $$t; ln $$l $$t;
.   endfor
```
BSDmake has two basic kind of useful extensions

- variable modifiers, e.g., ${VAR:L}
- dot keywords, .for, .if ...

Tricky part is evaluation of variables in dot stuff! Diverged from other BSDs in variable modifiers.
Parallel make wasn’t working.
Specifically, one shell to run all commands (experimental)
More output
#define REMOTE execution from sprite
By that point I knew enough about the basic structure

- shell execution: why depart from everybody else
- extra display breaks things too
- REMOTE is unlikely to come back

So I killed REMOTE entirely, made the extra display debug-only, and removed the possibility to use other shells.
Model: one target forks a job, job is responsible for spawning its commands. Output comes out garbled. Setup a pipe to catch output. At that point, things good enough for kernel build through make -j. Needed to add lots of dependencies ...
Model: one target forks a job, job is responsible for spawning its commands. Output comes out garbled. Setup a pipe to catch output. At that point, things good enough for kernel build through make -j. Needed to add lots of dependencies ...and boom, make build works too.
Issues

- pipe means no stdin
- jobs that create several files race, e.g. yacc production

```
 a.c a.h: a.y
   yacc a.y
```

Hack manual synchronization through a timestamp

```
 a.c a.h: stamp

 stamp: a.y
   yacc a.y
   touch $@
```

- make doesn’t understand the file system: ./a and a are different things.
Writing documentation.
Reading again POSIX specifications.
Changing the manpage to conform.
Foregoing the quaint little things.
What’s POSIX and what’s not.
We don’t have a POSIX mode and warn.
People write non-portble makefile
make sh... Sounds familiar?
make -j4 on recursive makefile:

\[ 4\text{jobs} + 16\text{jobs} + 64\text{jobs} + \ldots \ = \text{lots} \]

Other systems use a kind of "token" system, but
- finding a socket name can be difficult. Find a file system you can write to.
- fd passing is a hack. There’s no guarantee the shell will let you.

So let’s recognize recursive rules in makefiles.
make -j4

rulea:
    normal

ruleb:
    normal

rulec:
    cd dir && make # <- hey I’m recursive

when we meet rulec, notice that’s recursive.
Don’t allow any other jobs to start while that one is running.
Replace exponentiation with sum \textit{in the worst case}.
Because you can expect the cheap jobs to terminate early.
Yeah, it’s the same as DPB_PARALLEL in dpb land.
Works very well in practice, just needs some kind of heuristic to say "this is a kind of make".
... Because you can expect the cheap jobs to terminate early.
Solution came from totally something else  
Better location of error messages  
Convergence with dpb  
Replace the "job control handler" with a job automaton: One single job-handling loop  
Unintended benefit: no need for pipe, as most printing comes from make itself
Wait can’t be interrupted by signal.
Naive approach doesn’t work:

1. fork jobs
2. wait for any to finish
3. check for signals
4. go back to 1.
Recent signal handling

1. setup empty handler for SIGCHLD and handlers for the rest that just say "got that signal"
2. fork jobs
3. block all signals
4. check for signals that happened before 3, including SIGCHLD (that's just wait3(... WNOHANG))
5. pause suspend until something happens

Marc Espie <espie@openbsd.org>
Making make parallel - legacy code nightmare
Side-notes

**sudo**
If you test this with `sudo` it won’t work, because it can’t pass signals through (at least in the OpenBSD version).
One more reason to not be root while building ports...

**expensive**
Works with the recursive make optimization… because just one command will need to be tagged expensive.

**Single shell ?**
Heuristics to NOT fork a shell for simple commands Could be expanded to also do

```
cd somedir && run_some_cmd
```
Multiple targets semantics changed.
Comment from netbsd (possibly David Holland ?)

```
  a b: deps
      somecommands
```

is no longer a shorthand for

```
  a: deps
      somecommands
  b: deps
      somecommands
```

but it ties a and b together, so lock one target while building the other
Almost

```plaintext
a b: deps
   somecommands

if somecommands refer to $@, then it's actually old-style stuff. Otherwise, we assume it
really builds a and b together.
(Note that we already scan command lines before execution)
```
This is the worst bug we still have.

make doesn’t know that a and ./a are the same file

It’s worse with VPATH constructs.
It only really matters for parallel make.
Sequential make is myopic: it relies on the file system each step of the way.
This breaks autoconf builds, for instance
Partial solution

Equivalence
Build a whole structure of 'equivtargets": hash filenames without the directories, and link all those targets together (as potential siblings), then check through filesystem semantics and VPATH handling for actual equivalence.

Pitfall
Can’t actually use this all the time, make loops.

Sadly
This doesn’t really work yet and is very nasty.
Work in progress.
Remove the difference between the parallel builder and the compat builder by using the compact builder with the new job engine...
... By filling a queue instead of building stuff right away.
Counter-intuitive, but the parallel engine is still partly broken (not lazy enough) and incompatible with the sequential builder...
Beyond skynet

Recursive make is bad
Because each make rescans part of the file system
Because dependencies are not handled
Ways to do that?

Shhhh! Ninja
Redesign that fixes most of make issues
But 0% compatible.
Not wide adoption yet.
Questions !!!