tvix: waiting for the store and trampolines

Problem

Sometimes, we're partway through evaluating an expression, and we reach a point where we need to wait for some long-running operation to continue. **Usually** this is something involving the store - a fetcher, an addPath, building a derivation to IFD from it, etc. We'd like to be able to continue making progress with evaluation as much as reasonably possible if this happens - and potentially kick off **other** long running operations that aren't directly dependent on ours, and run them in parallel - then resume the evaluation once the long-running op is finished.

Background: Evaluation order

There seems to be some consensus, both inside and outside TVL, that nix evaluation order basically doesn't matter - and in all the places where it **can** currently be observed (eg nix#3249) that should be considered either unspecified behavior or a full-blown bug. If we trust this assumption, then we should consider tvix allowed to evaluate expressions out-of-order, as long as the same eventual result is returned.

Suspended evaluation

The general idea here is to define a special kind of error return, which a thunk can use to indicate to the VM that it must be "suspended" until an indicated background process has completed. By default, this error can bubble up all the way into the toplevel `VM::run()` loop which forced the thunk, and the VM can then block evaluation until the background process is completed. However, certain points of evaluation can be considered "fork points" for thunk suspension (initially this will likely be forcing the elements of a list or values of an attrset, but in the future we may be able to determine other fork points). In these cases, we can place the blocking thunk in a suspension queue, but continue to force the other siblings within the fork point (potentially suspending them as well if necessary).

I see two options for how to actually implement suspension:

1. Just save the state of the VM (the stack and instruction pointer) to the suspended thunk queue, then pick back up where we left off once we resume. This initially seems quite simple to implement, and likely to be correct, but might present disappointing tradeoffs in terms of memory locality and copying. Regardless, I wouldn't do anything else without benchmarking
2. Start the thunk from the beginning, but remember the result of the operation that we were waiting on. Thunks tend to be quite small (on the order of 4 or 5 instructions, from what I've seen) and my instinct is that they're all idempotent, so it's possible this will work fine (though it's also possible there's something major I'm missing here). We end up re-doing work, but the tradeoff might make sense in terms of keeping things cache-local. Alternatively, we could implement both approaches and decide which to do based on the size of the thunk or something like that

Trampolines

The notion of suspending a thunk's evaluation while waiting on something else to happen is actually quite similar to the notion of a stack trampoline (as implemented for tail-recursive lisps, not

the tens of concepts with the same name). Essentially, to avoid being bound on the OS stack, we can allow a thunk to return an error (again, bubbling all the way up to the VM) that it would like to be suspended pending *the evaluation of another thunk*. At this point, the suspension mechanism looks very similar: we place the suspended thunk on a stack, then force the thunk it would like to wait for, then pop off the stack and resume execution of the original thunk.