The Plan 9 Front Concurrent C Extensions

Benjamin Purcell (spew)
benjapurcell@gmail.com

ABSTRACT

The Plan 9 Front compilers extend the C programming language to allow for built-in CSP style concurrency operations. This paper describes the usage and implementation of the extension.

1. Introduction and Motivation

CSP-style concurrency operations are an essential part of many programs in the Plan 9 operating system. Concurrent programs were originally written in Alef which had built-in concurrency operations. When Alef was retired, a library was written to allow access to CSP operations from programs written in C (see *libthread*(2)). However, there are a number of deficiencies with *libthread*(2): thread creation is awkward, receiving or sending of multiple channels requires a large temporary structure definition, and the send/receive operations are not type-safe. The extension aims to address those concerns to make threaded programs easier and safer to write without the need to maintain a separate compiler infrastructure such as Alef. This document assumes familiarity with *libthread*(2).

2. The Extensions

The compiler extension provides capabilities for launching new threads and processes, declaring and allocating storage for typed channels, and type-safe sending and receiving from channels. It also provides a new control structure that allows for type-safe sending or receiving of multiple channels.

2.1. Thread and Process Creation

Threads and processes are created using the keywords `coproc` and `cothread` which have a syntax similar to a function that takes two arguments. The first argument must be a function application, and the second is an int that specifies stack size to use to create the process or thread. The calls `coproc` and `cothread` return the resultant thread id.

```
int tid, pid
void fn(int arg1, double arg2, char *arg3);
...
tid = cothread(fn(a, b, c), 8192);
pid = coproc(fn(a, b, c), 8192);
```

The function passed to `coproc` and `cothread` can have any signature, though its return value will not be used. Instead of applying the function to its arguments, the calls to `cothread` and `coproc` tell the compiler to check the arguments to the function and then compile a call to libthread to start the function in a new thread or process with a memory allocated stack (see *malloc*(2)). Thus, if `a`, `b`, and `c`, are of an incompatible type to `int`, `double`, and `char*` respectively, then the above will not compile.

2.2. Channel Declarations

The extension reserves the character @ for declarations of Channels.