NAME

chanalt chanclose chancreate chanfree channbrecv channbrecvp channbsend channbsendp chanopen chanrecv chanrecvp chansend chansendp proccreate procexit procfree procgetname procinit prockill procrand procsetname procsleep procsrand — parallel programming procedures

SYNOPSIS

```
#include <ccco.h>

typedef struct Chan    Chan;
typedef struct Alt     Alt;
typedef struct Proc    Proc;

Proc*   proccreate(void (*fn)(void*), void* arg, int stksz);
void    procfree(Proc* proc);
int     procinit(Proc* proc, void (*fn)(void*), void* arg, int stksz);
void    prockill(Proc* proc);
void    procsetname(const char*, ...);
char*   procgetname(void);
void    procsleep(long dur);
void    procexit(void);
void    procsrand(long seed);
long    procrand(void);
Chan*   chancreate(long nel, long elsz);
void    chanfree(Chan* chan);
int     chanopen(Chan* chan);
void    chanclose(Chan* chan);
void    chanalt(Alt*);
int     chanrecv(Chan* c, void* p);
void*   chanrecvp(Chan* c);
int     channbrecv(Chan* c, void* p);
void*   channbrecvp(Chan* c);
int     chansend(Chan* c, void* p);
int     chansendp(Chan* c, void* p);
int     channbsend(Chan* c, void* p);
int     channbsendp(Chan* c, void* p);
```

DESCRIPTION

Libccco is a parallel programming library inspired by several programming languages including Limbo, Occam, Alef, and Go, as well as the Plan 9 and Inferno operating systems. Its programming interface similar to that provided by Plan 9's libthread.

Processes

Processes have a 1:1 correspondence with system threads. A *Proc* holds the state of a process. A program's main thread should not call any of the procedures prefixed with *proc;* they only behave correctly if called by a process. Processes may, in turn, create new processes.

Proccreate and *procinit* create a new process that will run *fn(arg)*. The size of the process's stack is specified by *stksz;* if *stksz* is zero, the system's default stack size for threads will be used. *Proccreate* returns a pointer to a newly allocated *Proc,* so the pointer must be freed later by *procfree.* On the other hand, *Proc* structures that were initialized by a prior call to *procinit* must be destroyed by *prockill*. *Proccreate* returns a null pointer if it encounters an error. *Procinit* returns a non-zero value if it encounters an error.

Procexit terminates the calling process; any *Proc* structures associated with it will remain intact.

Procsleep blocks the calling process for at least *dur* seconds.

Procsetname sets the calling process's current name, and *procgetname* returns it. The pointer returned by *procgetname* is valid until the next call to *procsetname.*

Channels

Channels are a means of communication and synchronization between two or more processes (including the program's main thread in this special case). A *Chan* holds the state of a channel.

Chanopen opens a channel of *nel* elements *elsz* bytes in size, and initialises the *Chan* indicated by *chan.* If *nel* is zero, the channel will be unbuffered. *Chanopen* returns a non-zero value if it encounters an error. The channel must later be closed and the *Chan* destroyed by *chanclose.*

Chancreate allocates a new *Chan* and opens a channel of *nel* elements *elsz* bytes in size. If *nel* is zero, the channel will be unbuffered. *Chancreate* returns a null pointer if it encounters an error. The returned pointer must later be freed (and the channel closed) by *chanfree.*

Channel operations

Once a channel is open, it can be used to transfer data. Channel operations follow a naming scheme that indicates the differences in their behaviour. The naming scheme is *chan[nb]send[p]* and *chan[nb]recv[p],* where *nb* indicates that the operation is non-blocking, and *p* indicates that the operation transfers a pointer rather than a value. Channel operations operate on one element in the channel per call.

A *send* operation on a buffered channel proceeds if there is space for an element in the buffer, otherwise it blocks until space becomes available. If the channel is unbuffered, a *send* blocks until a *recv* occurs in another process; likewise, a *recv* blocks until a *send* occurs. A *send* operation returns -1 if it encounters an error, 1 otherwise. A non-blocking *send* returns 0 if it would have blocked.

A *recv* operation on a buffered channel blocks until at least one element becomes available in the buffer. A *recv* operation returns -1 if it encounters an error, 1 otherwise. A non-blocking *recv* returns 0 if it would have blocked. Exceptions to this convention are *chanrecvp* and *channbrecvp,* both of which return a null pointer if they encounter an error, as will *channbrecvp* if it would have blocked.

Random numbers

Because interleaved or simultaneous calls to C's *rand* and *srand* procedures by multiple processes result in undefined behaviour, libccco provides a pseudo-random number generator (henceforward called a RNG).

Procsrand seeds the RNG with *seed. Procrand* returns a random integer between 0 and 2,147,483,647, provided the RNG has been seeded (otherwise the return value is undefined).

Each process has its own RNG, so calls to *procrand* will yield a reproducible period regardless of whether other processes have called *procrand* or *procsrand.* Likewise, calls to *procsrand* will not interfere with other processes.

The RNG's underlying mechanism is a Mersenne twister (MT19937) populated by repeated applications of an equation taken from a Pokémon video game.

BUGS

Alting is not yet implemented.

Unbuffered channels only mimic unbuffered behaviour; they actually use a one-element buffer. This keeps the implementation of unbuffered channel operations as straightforward as possible.