

libcsp

CSP for pure C

libcsp overview

- Libcsp is designed for the purpose of providing the CPS programming model to C
 - Not to match the performance of Occam or C++CSP (libcsp is older though)
- It is a communication library for posix threads

CSP 'feel'

- As with most pure C stuff the API is very detailed and very little estitic
- Some things are cumbersome
- But it is very portable!

Simple test

```
#include <process.h>
#include <channel.h>
#include <stdio.h>
```

```
Channel * chan;
void hello (void) {
    printf ("hello\n");
    ChanOutInt (chan, 2);
}
```

```
void world (void) {
    int i = ChanInInt (chan);
    printf ("world %d\n", i);
}
```

```
int main (int argc, char** argv) {
    Process * a, * b;
    chan = ChanAlloc();
    a = ProcAlloc (hello, 0, 0);
    b = ProcAlloc (world, 0, 0);
    ProcPar (a, b, NULL);
    ProcAllocClean (a);
    ProcAllocClean (b);
    ChanDestroy (chan);
    return 0;
}
```

Shared Channels

```
#include <process.h>
#include <csp.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

const int N = 10;
Channel * chan;

void hello (const char* msg)
{
    int i;
    char text[100];
    for (i = 0; i < N; i++)
    {
        sprintf (text, msg, i);
        CSP_chanOutCopy (chan, \
                        text, strlen(text)+1);
        usleep (drand48() * 100.0);
    }
}
```

```
void hello1 (void){hello ("hello1 %d.\n");}

void hello2 (void){hello ("hello2 %d.\n");}

void world (void)
{
    int i;
    char text[100];
    for (i = 0; i < (2*N); i++)
    {
        CSP_chanInCopy (chan, text, -100);
        printf (text);
    }
}
```

Shared Channels – main

```
int main (int argc, char** argv)
{
    Process * w1, * w2, * b;

    chan = CSP_chanAlloc (CSP_ANY2ONE_CHANNEL, 0);
    w1 = ProcAlloc (hello1, 0, 0);
    w2 = ProcAlloc (hello2, 0, 0);
    b = ProcAlloc (world, 0, 0);
    ProcPar (w1, w2, b, NULL);
    ProcAllocClean (w1);
    ProcAllocClean (w2);
    ProcAllocClean (b);
    CSP_chanDestroy (chan);
    return 0;
}
```

Alternation (1)

```
const int N = 10;
Channel * chan0, * chan1;
int delay = 100;

void hello (const char* msg, Channel * c)
{
    int i;
    char text[100];
    for (i = 0; i < N; i++)
    {
        sprintf (text, msg, i);
        CSP_chanOutCopy (c, text, strlen(text)+1);
        usleep (drand48() * delay);
    }
}

void hello1 (void){hello ("hello1 %d.\n", chan0);}

void hello2 (void){hello ("hello2 %d.\n", chan1);}
```

Alternation (2)

```
void world (void)
{
    int i, selected;
    char text[100];
    CSP_Alt_t alt;
    Channel * clist[2];
    clist[0] = chan0;
    clist[1] = chan1;

    CSP_altInit (&alt);

    for (i = 0; i < (2*N); i++)
    {
        selected = CSP_priAltSelect (&alt, clist, 2);
        if (selected >= 0)
        {
            CSP_chanInCopy (clist[selected], text, -100);
            printf (text);
        }
        else
        {
            printf ("-1\n");
        }
    }
    CSP_altClose (&alt);
}
```

Alternation (3)

```
int main (int argc, char** argv)
{
    Process * w1, * w2, * b;

    if (argc > 1) { delay = atoi (argv[1]); }

    chan0 = CSP_chanAlloc (CSP_ONE2ONE_CHANNEL, 0);
    chan1 = CSP_chanAlloc (CSP_ONE2ONE_CHANNEL, 0);

    w1 = ProcAlloc (hello1, 0, 0);
    w2 = ProcAlloc (hello2, 0, 0);
    b = ProcAlloc (world, 0, 0);

    ProcPar (w1, w2, b, NULL);

    ProcAllocClean (w1);
    ProcAllocClean (w2);
    ProcAllocClean (b);

    CSP_chanDestroy (chan0);
    CSP_chanDestroy (chan1);
    return 0;
}
```