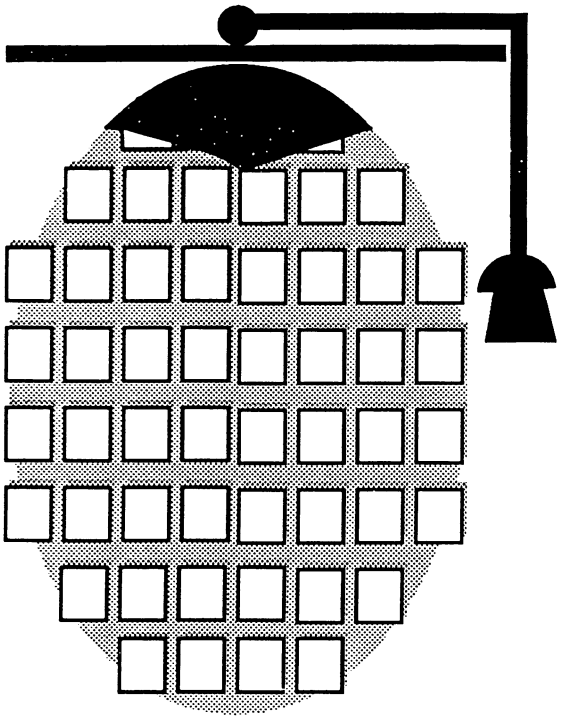


INMOS Transputer and Occam Courses

Mixed language programming

SGS-THOMSON MICROELECTRONICS

INMOS is a member of the SGS-THOMSON Microelectronics group.

inmos

Toolset software

Overview

Type 1 processes

Type 2 processes

Type 3 processes

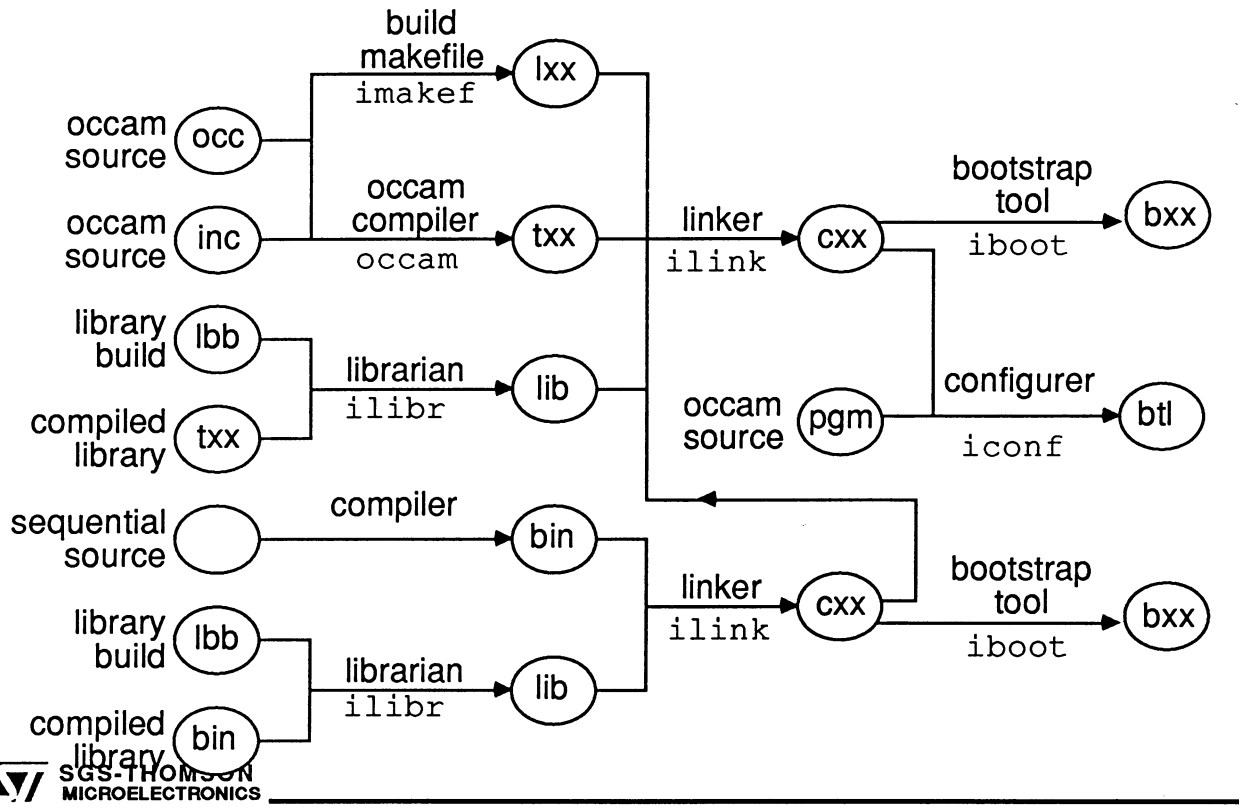
Hello world practical part 2

inmos

Overview

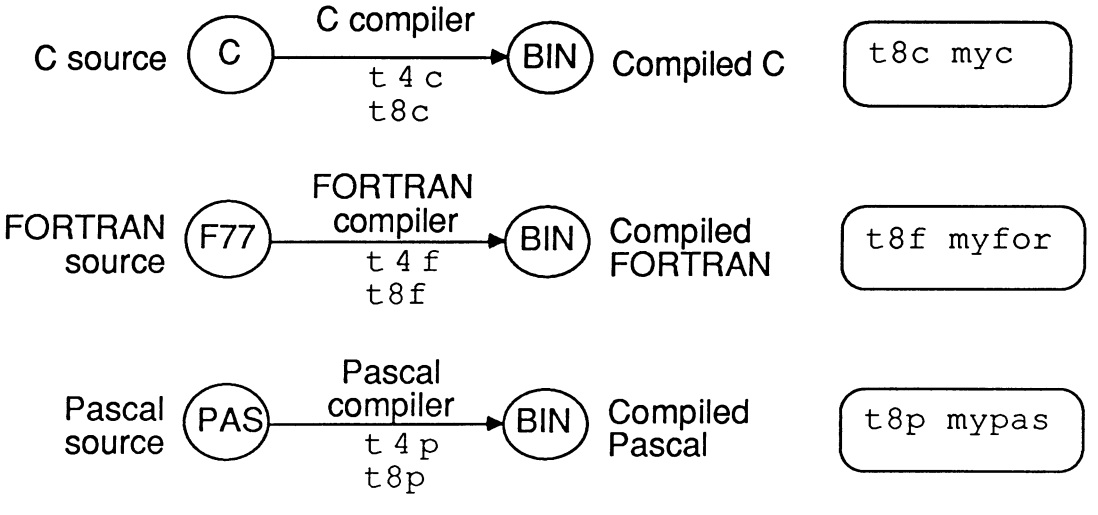
inmos

Compilation steps

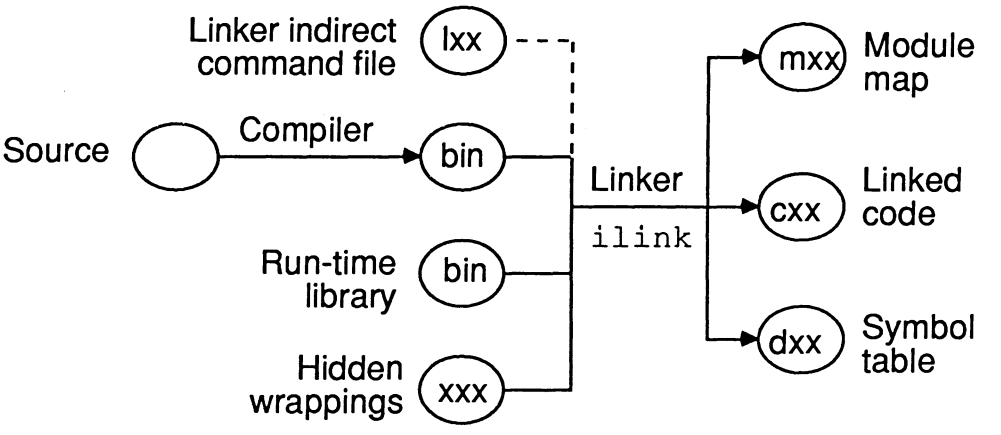


inmos

Sequential language compilers



Linking




```
ilink /f myprog.l4h /o myprog.c4h
```

Linker command must use output file option.

Indirect linker file

Used by linker.

Lists files to be linked together.

Gives procedure name for calling code.

Extension lxx.

To link a type 2 C process for T414 in halt mode:

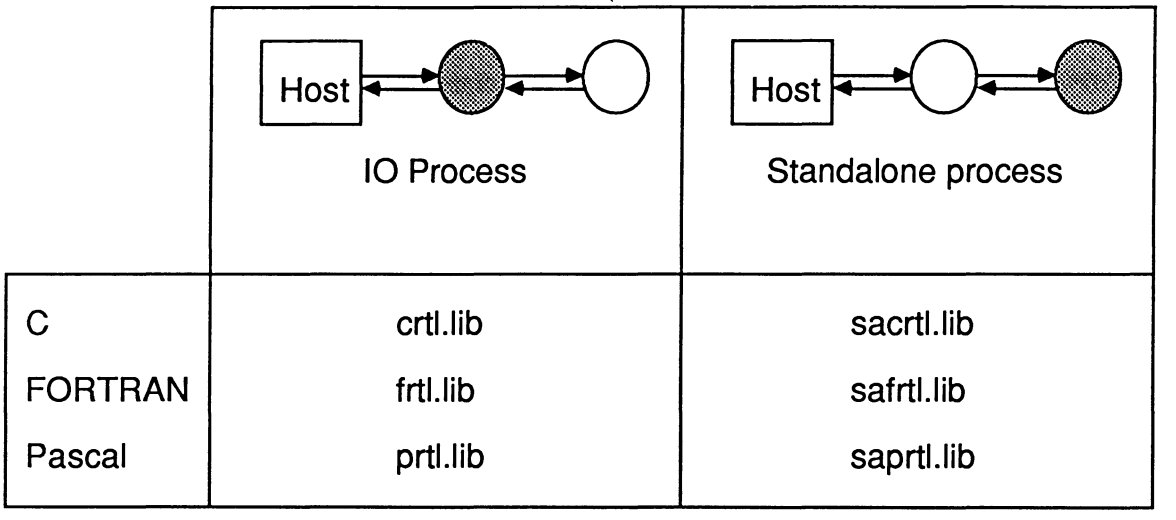
File: cprog.l4h

```
cproc=procent.c4h cprog.bin crtl.lib
```

cproc is the name of the procedure called by the harness.

cprog.c is the name of the file containing the C source.

Run-time libraries



C	crtl.lib	sacrtl.lib
FORTRAN	frtl.lib	safrtl.lib
Pascal	prtl.lib	saprtl.lib

Make file

Used by Make.

Gives file dependencies and building commands.

To make a linked type 2 C process for T414 in halt mode:

File: nonoccam

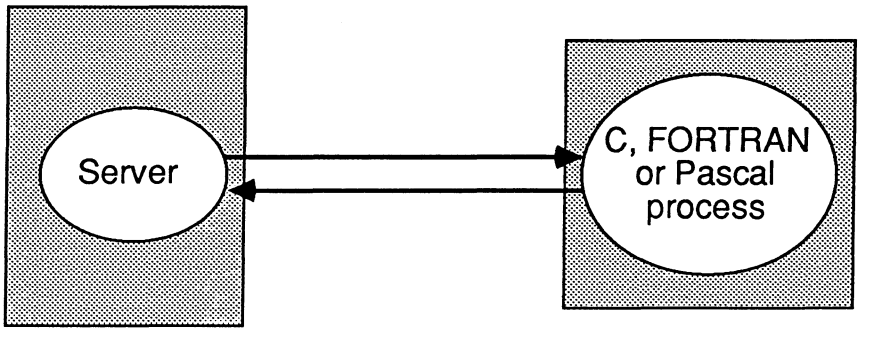
```
cprog.c4h:◊cprog.bin  cprog.l4h
◊ilink /f cprog.l4h /o cprog.c4h

cprog.bin:◊cprog.c
◊t4c cprog
```

Use tabs where shown as ◊.

```
make -f nonoccam
```

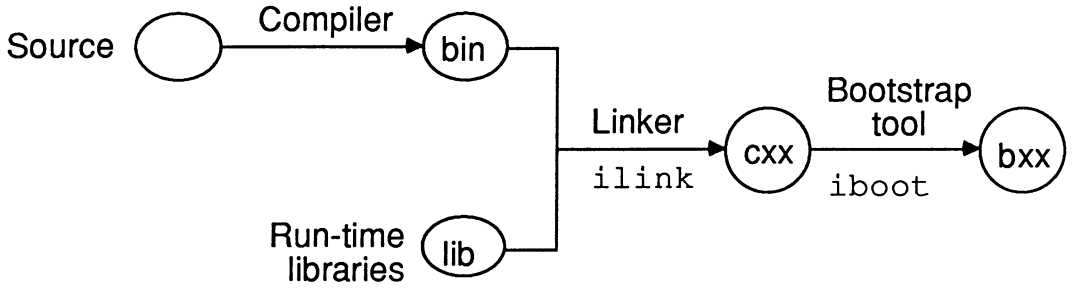
Type 1 processes

Single process code



A process that communicates only with the host is a type 1 process.

Compiling type 1 processes



Makefile and linker file are built by hand.

Make file

File: cprog

```
cprog.b4h:◊cprog.c4h
◊iboot cprog.c4h

cprog.c4h:◊cprog.14h cprog.bin
◊ilink /f cprog.14h /o cprog.c4h

cprog.bin:◊cprog.c
◊t4c cprog
```

```
make -f cprog
```

Linker indirect file

There is no calling code, so no procedure name.

Host interface provided by mainent.cxx.

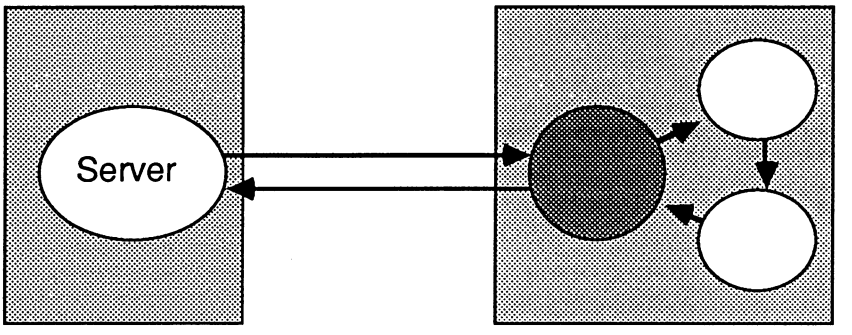
Use full i/o run-time libraries.

File: cprog.l4h	File: forprog.l4h	File: pasprog.l4h

`mainent.c4h`
`cprog.bin`
`crtl.lib`
	`mainent.c4h`
	`forprog.bin`
	`frtl.lib`
		`mainent.c4h`
		`pasprog.bin`
		`prtl.lib`

Type 2 processes

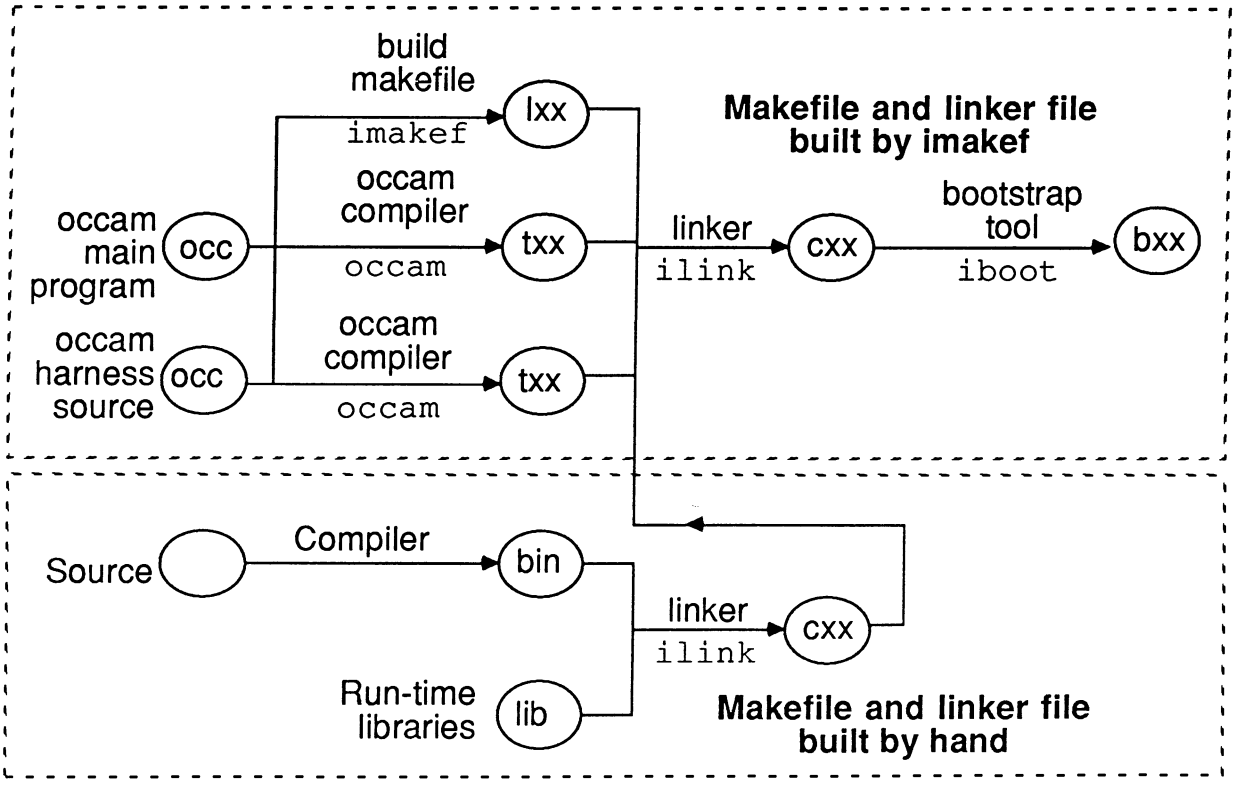
Host interface



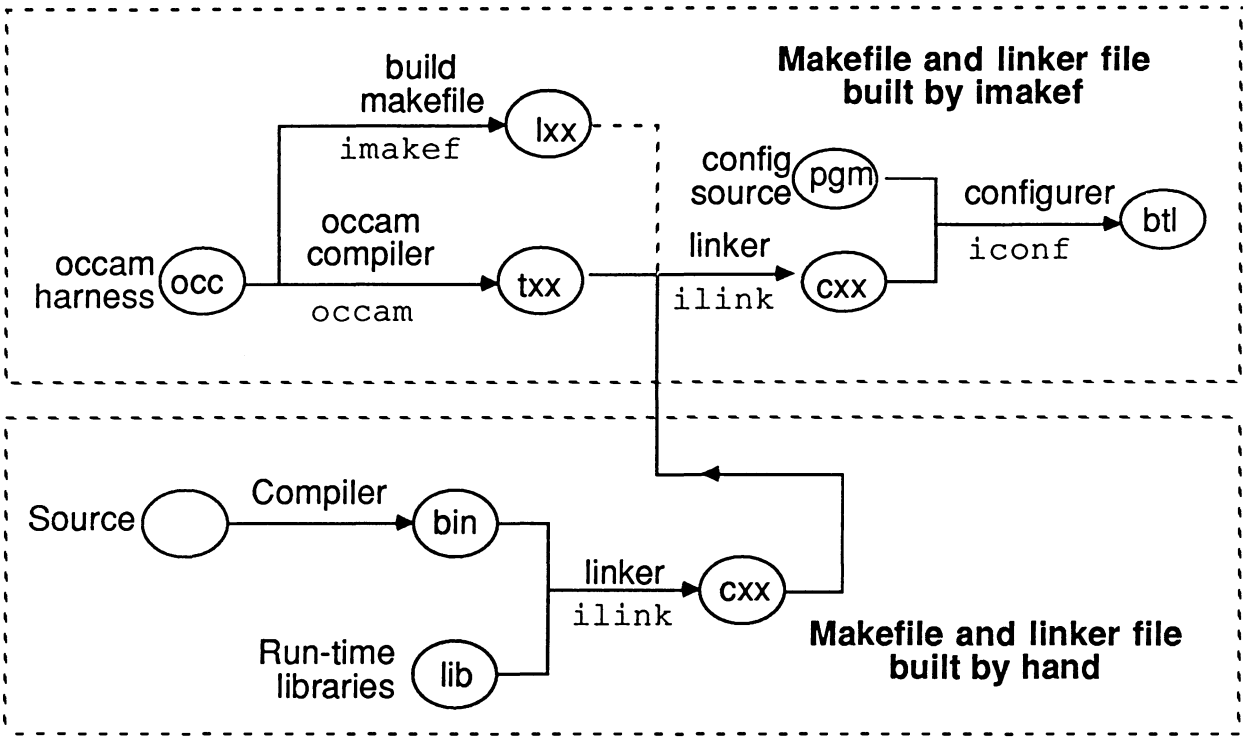
A process that communicates with the host and with other processes is a type 2 process.

A type 2 processes must have an occam harness.

Compiling for a single transputer



Compiling for multiple transputers



Make file

File: nonoccam

```
cprog.c4h:◊cprog.bin  cprog.l4h
◊ilink /f cprog.l4h /o cprog.c4h

cprog.bin:◊cprog.c
◊t4c cprog
```

```
make -f nonoccam
```

Indirect linker file

Procedure name is given before equals sign (=).

Host interface is provided by procent.cxx.

Type 2 processes always use i/o run-time libraries:

crtl.lib, frtl.lib or prtl.lib.

File: cprog.l4h

```
cproc=procent.c4h cprog.bin crtl.lib
```

File: forprog.l4h

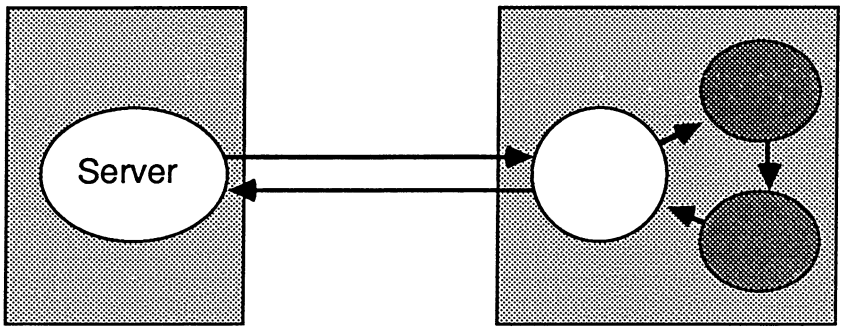
```
forproc=procent.c4h forprog.bin frtl.lib
```

File: pasprog.l4h

```
pasproc=procent.c4h pasprog.bin prtl.lib
```


Type 3 processes

Standalone processes



A process that communicates only with other processes

is a type 3 process.

A type 3 process must have an occam harness.

Compiling

Compilation steps are the same as for type 2 processes.

Make file

File: nonoccam

```
cprog.c4h:◊cprog.bin  cprog.l4h
◊ilink /f cprog.l4h /o cprog.c4h

cprog.bin:◊cprog.c
◊t4c cprog
```

```
make -f nonoccam
```


Indirect linker file

Procedure name is given before equals sign (=).

Host interface is provided by procent.cxx.

Type 3 processes use standalone run-time libraries:

sacrtl.lib, safrtl.lib or saprtl.lib.

File: cprog.l4h

```
cproc=procentc.t4h cprog.bin sacrtl.lib
```

File: forprog.l4h

```
forproc=procentf.t4h forprog.bin safrtl.lib
```

File: pasprog.l4h

```
pasproc=procentp.t4h pasprog.bin saprtl.lib
```

Hello world practical

Write a C, FORTRAN or Pascal program to display a simple message on the screen.

Construct an indirect linker file and makefile for your code as a type 1 process, to compile for an IMS T800.

Use make to compile it.

Hello World - possible C solution

File: hello.c

```
main  ()
{
    printf ("Hello World");
}
```

File: hello.l8h

```
mainent.c8h
hello.bin
crtl.lib
```

File: hello

```
hello.b8h:◊hello.c8h
◊iboot hello.c8h
hello.c8h:◊hello.l8h hello.bin
◊ilink /f hello.l8h /o hello.c8h
hello.bin:◊hello.c
◊t8c hello
```

```
make -f hello
```

inmos

Hello World - possible FORTRAN solution

File: hello.f77

```
      PRINT*,'Hello World'
      STOP
      END
```

File: hello.l8h

```
mainent.c8h
hello.bin
frtl.lib
```

File: hello

```
hello.b8h:◊hello.c8h
◊iboot hello.c8h
hello.c8h:◊hello.l8h hello.bin
◊ilink /f hello.l8h /o hello.c8h
hello.bin:◊hello.f77
◊t8f hello
```

```
make -f hello
```


inmos

Hello World - possible Pascal solution

File: hello.pas

```
program hello (output);
begin
  writeln ('Hello World');
end.
```

File: hello.l4h

```
mainent.c4h
hello.bin
prtl.lib
```

File: hello

```
hello.b8h:◊hello.c8h
◊iboot hello.c8h
hello.c8h:◊hello.l8h hello.bin
◊ilink /f hello.l8h /o hello.c8h
hello.bin:◊hello.pas
◊t8p hello
```

```
make -f hello
```

inmos

Harnesses and wrappings

Overview

Calling and #IMPORT

Channels

Wrappings

Harness practical

inmos

Overview

Harness function

Occam interface.

Defines channels and creates pointers.

Defines workspace.

Calls occam and sequential language procedures.

Main process for linking and configuration.

Hides details.

Harness structure

```
PROC harness (CHAN OF SP fs, ts,
              []CHAN OF DATA in, out)

  ...  #IMPORT directives
  ...  workspace definition

  SEQ
    ...  create channel pointers
    ...  procedure calls
:
```

Calling and #IMPORT

#IMPORT

```
#IMPORT "cfpprog.c4h"
```

cfpprog.c4h is the linked file name.

#IMPORT has the same effect as #USE

but is ignored by imakef.

Calling procedures

```
cfp.type2 (fs, ts, flag, ws1, ws2, in.ptr, out.ptr)

cfp.type3 (flag, ws1, ws2, in.ptr, out.ptr)
```

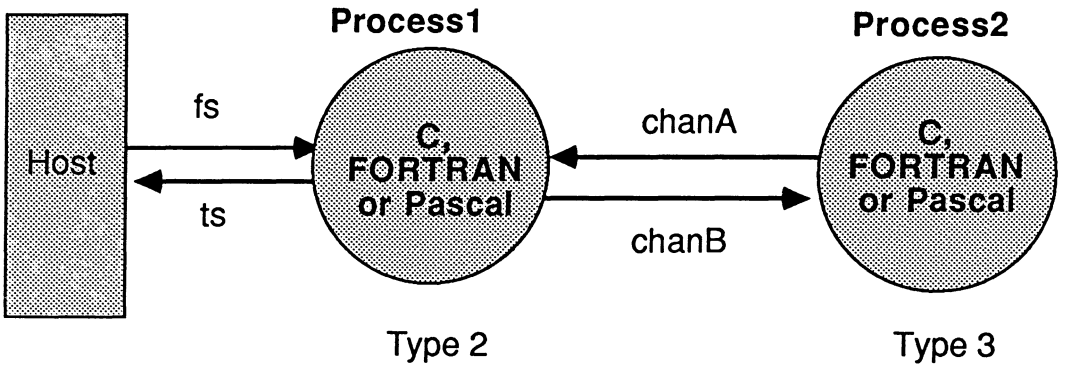
Procedure name is defined in linker options

Non-occam call parameters are fixed.

fs	CHAN OF SP	channel from host (type 2 only)
ts	CHAN OF SP	channel to host (type 2 only)
flag	VAL INT	workspace type
ws1	[]INT	first workspace
ws2	[]INT	second workspace
in.ptr	[]INT	pointers to input channels
out.ptr	[]INT	pointers to output channels

Channels

Multi-process program



Channel declarations

All user channels must be declared in the harness.

```
CHAN OF SP fs, ts:

CHAN OF DATA chanA, chanB:
```

Reserved channels

	IO process	Subscripts	Standalone process	Subscripts
C	2 input, 2 output	0, 1	1 input, 1 output	0
FORTRAN	2 input, 2 output	1, 2	1 input, 1 output	1
Pascal	2 input, 2 output	0, 1	-	

Purpose of reserved channels

	IO process	C or FORTRAN standalone process
resvd.in[0]	unused	unused
resvd.out[0]	debug	debug
resvd.in[1]	standard input	
resvd.out[1]	standard output	

Channel pointer procedures

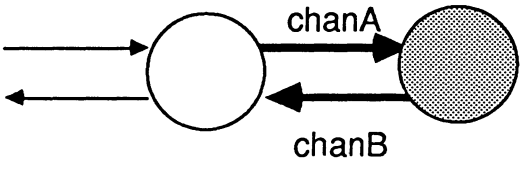
To create pointers to channels,

which can be passed to non-occam language processes.

```
LOAD.INPUT.CHANNEL (pointer, channel)

LOAD.OUTPUT.CHANNEL (pointer, channel)
```

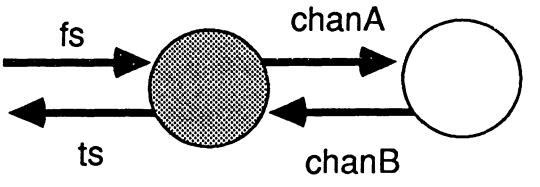
The pointer must be an INT.

Standalone process channels




```
[2]INT in1.pointer, out1.pointer:  -- pointers

SEQ
  LOAD.INPUT.CHANNEL  (in1.pointer[1],  chanA)
  LOAD.OUTPUT.CHANNEL (out1.pointer[1], chanB)
```

Input and output process




```
[3]INT in0.pointer, out0.pointer:  -- pointers
SEQ
  LOAD.INPUT.CHANNEL  (in0.pointer[2],  chanB)
  LOAD.OUTPUT.CHANNEL (out0.pointer[2], chanA)
```

Host channels are passed directly and do not need pointers.

Wrappings

Standalone wrappings

Type 3 process

```
PROC process2 (CHAN OF DATA in, out)
  #IMPORT "pasprog.c4h"
  [2]INT in.ptr, out.ptr:
  SEQ

    -------------------------- set up pointers
    LOAD.INPUT.CHANNEL (in.ptr[1], in)
    LOAD.OUTPUT.CHANNEL (out.ptr[1], out)

    ---------------------------- call pasprog
    VAL comb.ws IS 1:
    [1]INT ws2:
    [4000]INT ws1:
    pasprog (comb.ws, ws1, ws2, in.ptr, out.ptr)
:
```

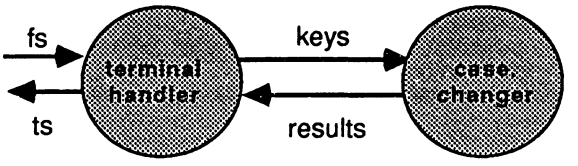

IO process wrappings

Type 2 process

```
#INCLUDE "hostio.inc"
PROC process1 (CHAN OF SP fs, ts,
               CHAN OF DATA from.app, to.app)
  #IMPORT "cfprog1.c4h"
  #USE "hostio.lib"
  [3]INT in.ptr, out.ptr:
  SEQ
    ------------------------------ set up pointers
    LOAD.INPUT.CHANNEL (in.ptr[2], from.app)
    LOAD.OUTPUT.CHANNEL (out.ptr[2], to.app)
    --------------------------- -- call cfp.process
    VAL comb.ws IS 1:
    [1]INT ws2:
    [4000]INT ws1:
    cfp.process (fs, ts,
                 comb.ws, ws1, ws2, in.ptr, out.ptr)
    so.exit (fs, ts, sps.success)
:
```

Harness practical

Write harnesses for the processes shown below.



The terminal.handler will be a type 2 process and the case.changer will be a type 3 process.
Use CHAN OF BYTE for results.
Use the protocol

```
PROTOCOL DATA
  CASE
    data; BYTE
    terminate
:
```

on channel keys.

Write linker indirect files to link the terminal.handler with the i/o libraries and the case.changer with the standalone libraries.
Write a make file for all the non occam code to compile for an IMS T800.
Write an occam main program and use imakef to construct a make file for all the occam.

inmos

Workspace

Setting up workspace

Workspace allocation

Workspace practical

inmos

Setting up workspace

Workspace parameters

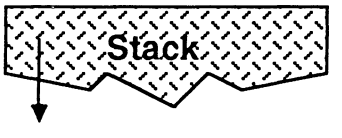
Allocated by the occam harness.

Passed to sequential language

subprograms as array parameters.

```
[size1]INT ws1:
[size2]INT ws2:
seq.proc (ws.flag, ws1, ws2, in, out)
```


Data areas



Subprogram parameters
and return addresses.
No check on overflow -
result unpredictable.

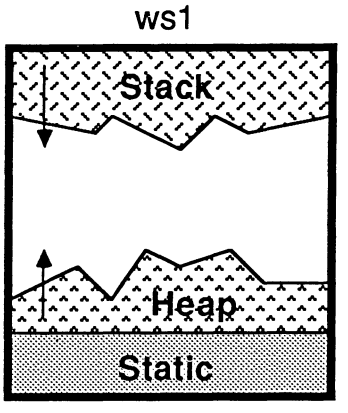
Ordinary variables and arrays.
Fixed workspace.

Variable length data structures,
eg strings.
Overflow checked.

Minimum stack - 400 words

Minimum static + heap - 4000 words

Combined workspace



Saves memory.

Reduces risk of overflow.

Used for program development.

Set flag = 1

Set size of ws2 to 1 word.

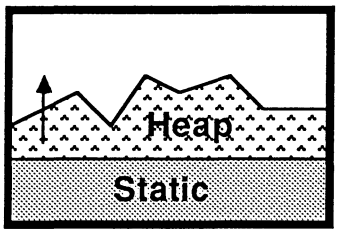
Defining combined workspace

```
[1]INT  min:
[5000]INT  stack.and.heap:

seq.proc (1, stack.and.heap, min, in.ptr, out.ptr)
```

Separate workspace

ws1

ws2



Stack can be placed on fast on-chip RAM.

May give faster performance.

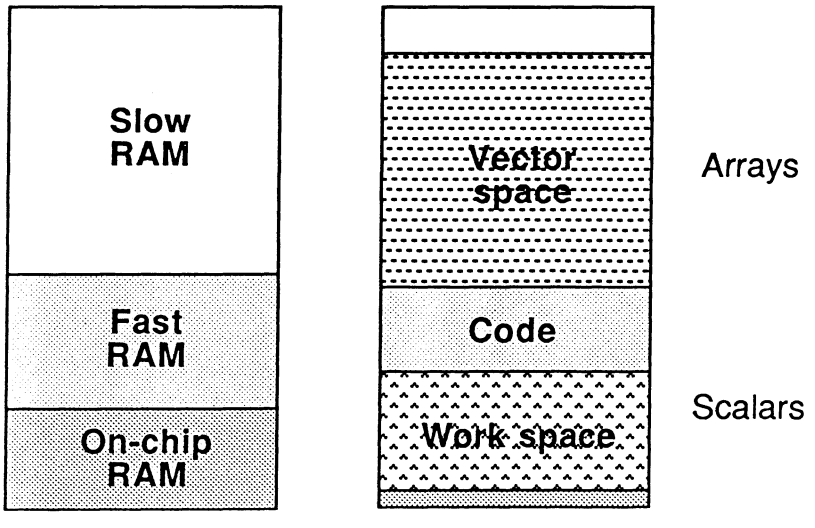
Set flag = 0

Defining separate workspace

```
[4500]INT heap:
[512]INT stack:

seq.proc (0, stack, heap, in.ptr, out.ptr)
```

Workspace allocation

Occam workspace

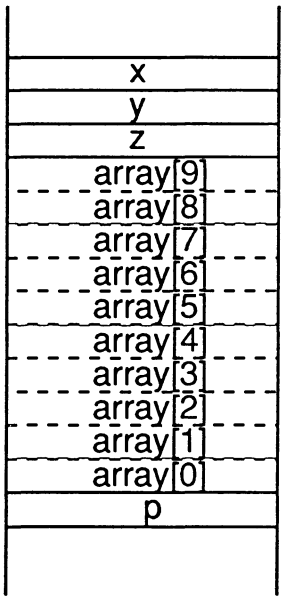


Occam variables

All workspace is allocated by occam. compiler.

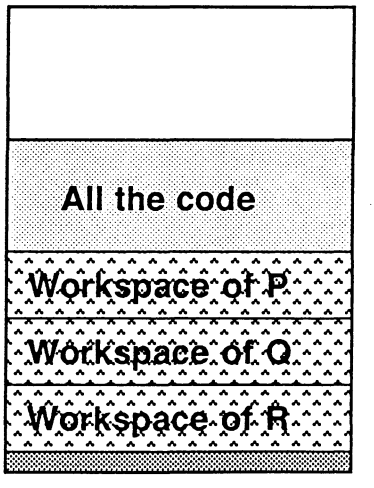
```
INT x, y, z:
[10]INT array:
PLACE array IN WORKSPACE:
INT p:
```

Variables declared last are in fastest RAM.



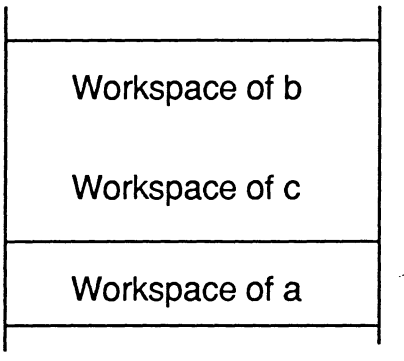
Workspace in a PAR

```
PAR
  ...  P
  ...  Q
  ...  R
```



Workspace for procedures

```
PROC a ()
  ...
:
SEQ
  ...  b
  a ()
  ...  c
```

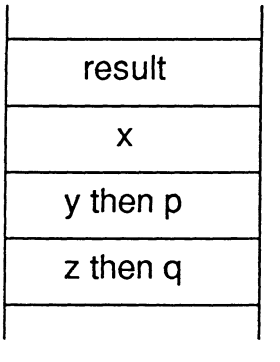


Local variables

Save space in a SEQ.

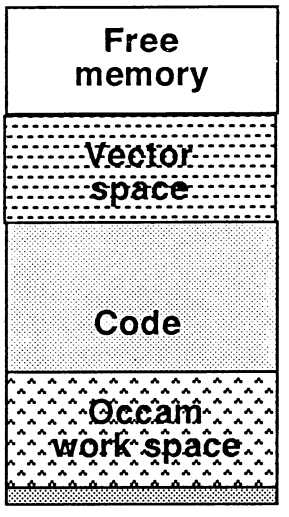
May run faster.

```
INT result:
SEQ
  INT x, y, z:
  SEQ
    ...  code using x, y, z
  INT p, q:
  SEQ
    ...  code using p, q
```



Declarations require no run time.

Free memory



Parameters of outer level occam

on a single transputer:

1 channel from host

2 channel to host

3 array of INT free memory

```
PROC myprog (CHAN OF SP fs,ts,
             []INT free.memory)
```


Using free memory

```
#INCLUDE "hostio.inc"
PROC harness (CHAN OF SP fs,ts,[]INT free.mem)
  #IMPORT "seqproc.c4h"
  #USE "hostio.lib"
  #INCLUDE "iohandlr.inc"
  [3]CHAN OF DATA in, out:
  [3]INT in.ptr, out.ptr:
  PAR
    io.handler (fs, ts, out, in)
    ------------------------ call seq.proc
    VAL sep.ws IS 0:
    [512]INT stack:
    SEQ
      ...  set up channel pointers
      seq.proc (fs,ts,sep.ws,stack,free.mem,
                                 in.ptr,out.ptr)
      ...  terminate server
:
```

Estimating workspace needs

In theory can be calculated for programs with no recursion.

Trial run after setting all memory to a recognisable value.

Allow generous margin for exceptional cases.

Workspace practical

Time a process with different workspaces.

Write code in C, FORTRAN or Pascal to increment a variable 10,000 times.
Write a type 3 harness to time the process and print the result.
    Use the free.memory parameter to pass
    as much space as possible to your code as combined workspace.
    Make the harness terminate the server after writing the results.
Compile and run it.

Modify your harness so that your code has separate workspace
    as low in the memory space as possible. Time the process again.
Switch the order of declaration of the stack and heap and time it again.

inmos

Channels and protocols

Channels

Data types

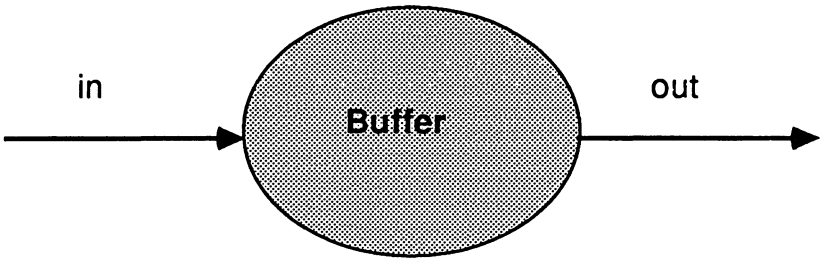
Protocols

Practical

inmos

Channels

Buffer



Message:

letter of size 1 byte

integer len of size 1 word

string of length len bytes

Occam buffer

```
PROC buffer (CHAN OF ANY in, out)
  BYTE letter:
  [80]BYTE string:
  INT length:

  WHILE TRUE
    SEQ
      in ? letter
      out ! letter

      in ? length :: string
      out ! length :: string
:
```

C buffer

```
#include <chan.h>

buffer (in, out)
  CHAN *in, *out;
{
  char letter, string[80];
  int len;
  for (;;)
  {
    chan_in_byte (&letter, in);
    chan_out_byte (letter, out);

    chan_in_word (&len, in);
    chan_in_message (len, string, in);
    chan_out_word (len, out);
    chan_out_message (len, string, out);
  }
}
```


Parameters of main

```
#include <chan.h>

main (argc, argv, envp, in, inlen, out, outlen)
   char *argv[];          /* token strings */
   int argc;              /* no of tokens in argv */
   char *envp[];         /* NULL */
   CHAN *in[], *out[];   /* channel pointer arrays */
   int inlen, outlen;    /* sizes of channel pointer arrays */

{
   int value, i;
   chan_out_word (value, out[i]);
}
```

C channel i/o functions

#include <chan.h>

Function	Parameters	Parameter type
chan_in_message	no of bytes buffer channel pointer	int array or pointer pointer
chan_out_message	no of bytes buffer channel pointer	int array or pointer pointer
chan_out_word	data channel pointer	word, eg int or float pointer
chan_out_byte	data channel pointer	byte, eg char pointer

FORTRAN buffer

```
      SUBROUTINE BUFFER (IN, OUT)

      INTEGER IN, OUT, LEN
      CHARACTER LETTER, STRING*80
      DO 10 WHILE (.TRUE.)
        CALL F77_CHAN_IN_BYTE (LETTER, IN)
        CALL F77_CHAN_OUT_BYTE (LETTER, OUT)
        CALL F77_CHAN_IN_WORD (LEN, IN)
        CALL F77_CHAN_IN_MESSAGE (LEN, STRING, IN)
        CALL F77_CHAN_OUT_WORD (LEN, OUT)
10      CALL F77_CHAN_OUT_MESSAGE (LEN, STRING, OUT)

      RETURN
      END
```

FORTRAN channel i/o subroutines

INCLUDE 'CHAN.INC'

Subroutine	Arguments	Argument type
F77_CHAN_IN_MESSAGE	no of bytes buffer channel address	INTEGER array INTEGER
F77_CHAN_OUT_MESSAGE	no of bytes data channel address	INTEGER array INTEGER
F77_CHAN_OUT_BYTE	data channel address	word, eg INTEGER or REAL INTEGER
F77_CHAN_OUT_WORD	data channel address	byte, eg CHARACTER INTEGER

Pascal buffer

```
procedure buffer (in, out : integer);

$include '\tp1v2\channels.inc'
var
    letter : char;
    len    : integer;
    string : packed array[1..80] of char;

begin
    while true do
        begin
            inmess (in, letter, 1);
            outbyte (letter, out);
            inmess (in, len, 4);
           inmess (in, string, len);
            outword (len, out);
            outmess (out, string, len);
        end;
end;
```

Pascal channel i/o procedures

$include '\tp1v2\channels.inc'

Procedure	Parameters	Parameter types
inmess	channel subscript buffer no of bytes	integer array integer
outmess	channel subscript buffer no of bytes	integer array integer
outword	data channel subscript	word, eg integer or real integer
outbyte	data channel subscript	byte, eg char or boolean integer

Data type implementation

Occam types

Sizes of variables and constants in bytes.		16 bit transputer	32 bit transputer
BOOL	Boolean	1	1
BYTE	8 bit unsigned integer	1	1
INT	Integer	2	4
INT16	16 bit integer	2	2
INT32	32 bit integer	4	4
INT64	64 bit integer	8	8
REAL32	32 bit floating point	4	4
REAL64	64 bit floating point	8	8
pointer	address	2	4

Arrays, variables and constants are word aligned

Simple C types

Sizes of variables and constants in bytes.		32 bit transputer
char	character	1
register	32 bit integer	4
int	32 bit integer	4
unsigned int	16 bit unsigned integer	4
short	32 bit integer	4
long	32 bit integer	4
float	32 bit floating point	4
double	64 bit floating point	8
pointer	address	4

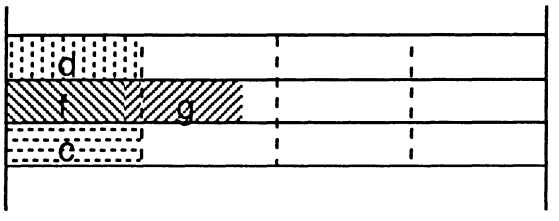
All types except char are word aligned.

Other C types

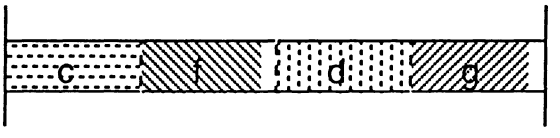
struct and union types are rounded up to whole words.

Each sequence in a struct is byte aligned. Sequences of more than one byte are word aligned.

```
struct s {char c; int f:7, g:7; char d}
```




```
struct s {char c; int f:7; char d; int g:7}
```



enum is not implemented.

FORTRAN 77 types

Sizes of variables and constants in bytes.		32 bit transputer
LOGICAL	Boolean	4
CHARACTER	character	1 per char
INTEGER	32 bit integer	4
COMPLEX	2 x 32 bit integer	8
REAL	32 bit floating point	4
DOUBLE PRECISION	64 bit floating point	8
COMPLEX	2 x 32 bit floating point	8

Arays, variables and constants are word aligned.

Pascal types

Sizes of variables and constants in bytes.		32 bit transputer
boolean	Boolean	1
char	Character	1
integer	32 bit integer	4
shortreal	32 bit floating point	4
real	64 bit floating point	8
set		4
enumerated (≤256 items)		1
enumerated (>256 items)		4
pointer	address	4

Arays, variables and constants are word aligned.

Protocols

Protocol security

Unmatched protocols cause deadlock.

Only occam checks protocols.

In C, FORTRAN and Pascal the programmer is responsible.

Use good programming practice to avoid errors:

- put input and output in library subprograms.
- use meaningful names.

Simple protocols

```
CHAN OF INT from.keys, echo:
CHAN OF REAL64 data1, data2:
CHAN OF BOOL test.results:
CHAN OF ANY to.screen:
CHAN OF [23]INT blocks:
```

Counted arrays in occam

```
CHAN OF INT::[]BYTE string:
[42]BYTE message:

string ! SIZE message::message
```

```
CHAN OF BYTE::[]REAL32 real.array:
[max.array]REAL32 buffer:
BYTE byte:

real.array ? byte::buffer
```


Counted arrays in C

```
outarray(out, message)
   CHAN *out;
   char message[];
{
   chan_out_word(sizeof(message), out)
   chan_out_message(sizeof(message), message, out)
}
```

```
inrealarray(in, buffer, length)
   CHAN *in;
   float buffer[];
   char length;
{
   chan_in_byte(&length, in)
   chan_in_message((int) length, buffer, in)
}
```

Named sequential protocols

```
PROTOCOL RECORD IS INT; REAL32; BYTE:
CHAN OF RECORD data.in, data.out:

INT  n:
REAL32  x:
BYTE  char:
data.in ? n; x; char
```

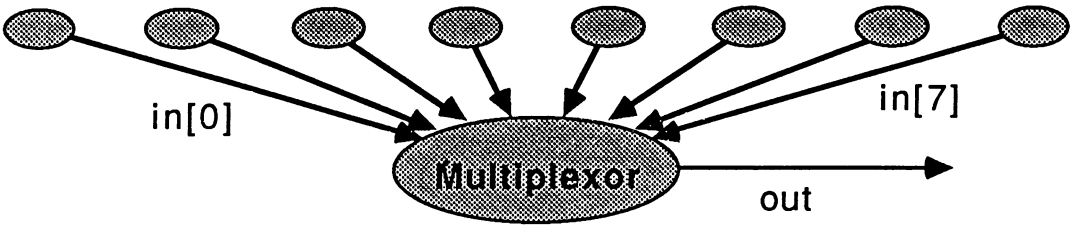
```
PROTOCOL COMMAND IS [3]BYTE; INT; INT:
CHAN OF COMMAND in:

INT arg1, arg2:
[3]BYTE command:
in ? command; arg1; arg2
```


Sequential protocols in C

```
struct record {int n; float x; char ch};
struct record inrecord(in)
   channel in;
{
   struct record record1;
   chan_in_word (&record1.n, in);
   chan_in_word (&record1.x, in);
   chan_in_byte (&record1.ch, in);
   return(record1);
}
```

```
struct command {char cmd[3];int arg1;int arg2};
outcommand(out, comm)
   channel out;
   struct command comm;
{
   chan_out_message (comm.cmd, out);
   chan_out_word (comm.arg1, out);
   chan_out_word (comm.arg2, out);
}
```

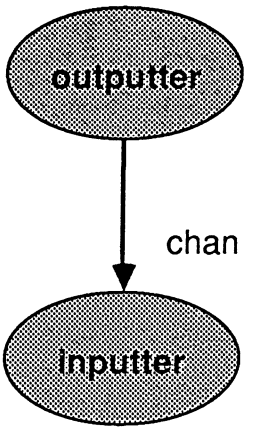
Multiplexor




```
[8]CHAN OF REAL32 in:

PROTOCOL MULTIPLEX IS INT; REAL32:
CHAN OF MULTIPLEX out:

WHILE going
  ALT i=0 FOR 8
    in[i] ? x
      out ! i; x
```


Using variant protocols in occam

```
PROTOCOL QUANTITY
  CASE
    q.no; INT
    q.weight; REAL64
:
CHAN OF NUMBER chan:
PAR

  SEQ
    chan ! q.weight; x+y
    chan ! q.no; 3

  chan ? CASE
    q.no; n
      ... process n
    q.weight; x
      ... process x
```



Tags are bytes

starting from 0

Using variant protocols in C

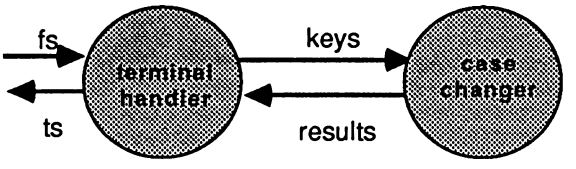
```
#define q_no      (char)0
#define q_weight  (char)1

union quantity
{
   int no;
   double weight
}

outweight (out, w)
   channel out;
   double w;
{
   chan_out_byte (q_weight, out);
   chan_out_message (8, w, out);
}
```


Inputting variant protocol

```
#define q_no      (char)0
#define q_weight  (char)1

union quantity {int no; double weight}

inquantity (in, qty, q_tag)
   channel in:
   union quantity qty;
   char q_tag;
{
   chan_in_byte (&q_tag, in);
   if (q_tag == q_no)
     chan_in_word (&qty.no, in);
   else if (q_tag == q_weight);
     chan_in_message (8, &qty.weight, in);
    else
     error();
}
```

Change case practical - part 3

Write a case changing program in C, FORTRAN or Pascal consisting of two parallel processes.



The terminal handler alternately polls the keyboard and the results channel. It passes on all characters from the keyboard to the case.changer. When it receives any character from the results channel it displays it on the screen. When it recieves a '%' character it terminates the server and itself and passes on the terminate message to the case changer.

Use the variant protocol DATA on channel keys.

The case.changer process changes upper case letters to lower case and lower case to upper and sends the result back down the results channel. Any other characters are returned unchanged. When it receives a terminate tag it terminates itself.

Use the main program, DATA protocol and occam wrappings from the harness practical.

Compile all the code using make and run it.

inmos

Configuration

Configuration

The configurer

Filter part 3 practical

inmos

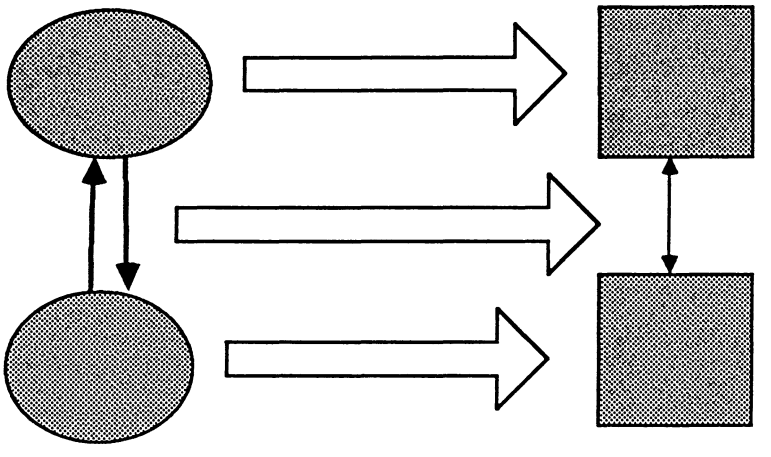
Configuration

inmos

Configuration

Placing:

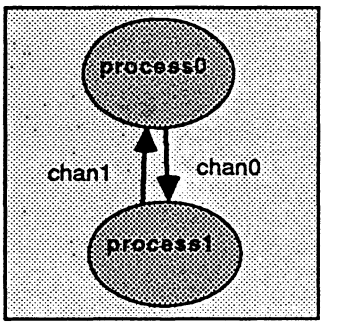
processes	onto	processors,
channels	onto	links.



inmos

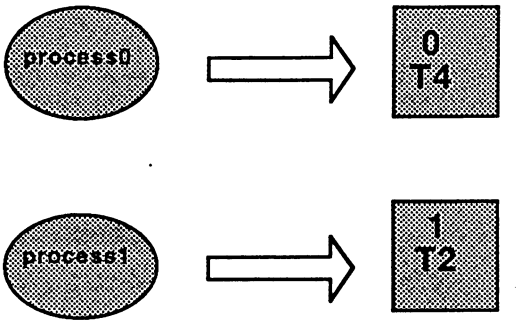
Single transputer code

```
#SC "proc0"
#SC "proc1"
CHAN OF ANY chan0, chan1:

PAR
  process0 (chan1, chan0)
  process1 (chan0, chan1)
```



Placing processes

```
#SC "proc0"
#SC "proc1"
PLACED PAR
  PROCESSOR 0 T4
    process0 ()
  PROCESSOR 1 T2
    process1 ()
```

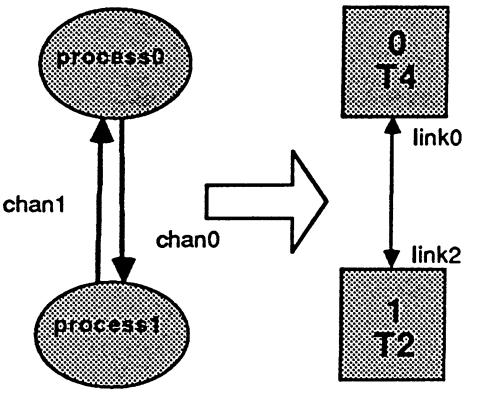


The first processor listed must be the root transputer

ie connected directly to the host.

Placing channels

```
CHAN OF ANY chan0, chan1:

PLACED PAR
  PROCESSOR 0 T4
    PLACE chan0 AT linkout0:
    PLACE chan1 AT linkin0:
    process0 (chan1, chan0)
  PROCESSOR 1 T2
    PLACE chan0 AT linkin2:
    PLACE chan1 AT linkout2:
    process1 (chan0, chan1)
```



inmos

Link addresses

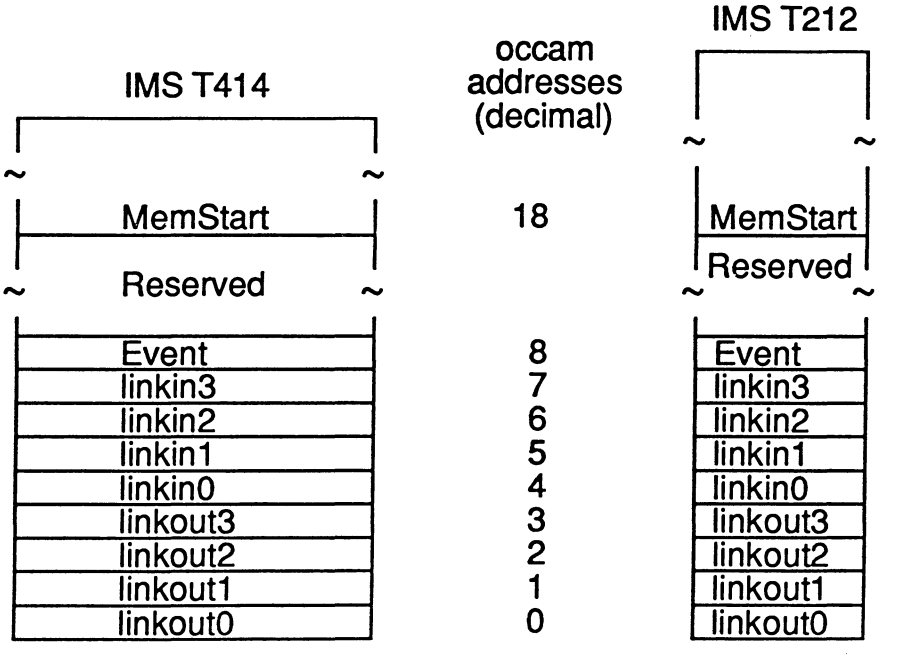
Links are at the following occam addresses:

```
VAL linkout0 IS 0:
VAL linkout1 IS 1:
VAL linkout2 IS 2:
VAL linkout3 IS 3:
VAL linkin0  IS 4:
VAL linkin1  IS 5:
VAL linkin2  IS 6:
VAL linkin3  IS 7:
```

These constants may be held in a #INCLUDE file.

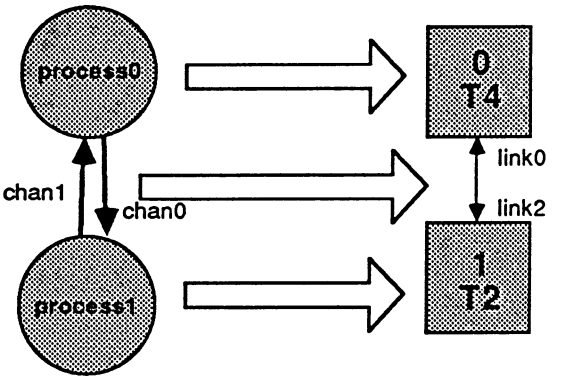
inmos

The occam addressing system



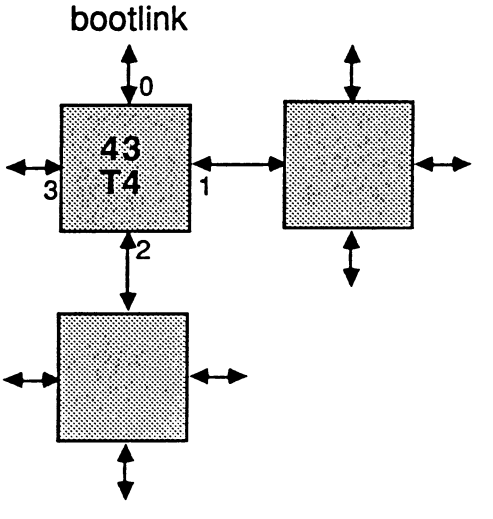
Complete configuration

```
#INCLUDE "linkadds"
#SC "proc0"
#SC "proc1"
CHAN OF ANY chan0, chan1:
PLACED PAR
  PROCESSOR 0 T4
    PLACE chan0 AT linkout0:
    PLACE chan1 AT linkin0:
    process0 (chan1, chan0)
  PROCESSOR 1 T2
    PLACE chan0 AT linkin2:
    PLACE chan1 AT linkout2:
    process1 (chan0, chan1)
```

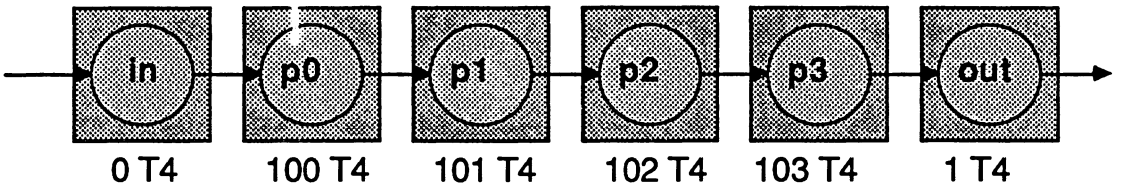


Loading a program

```
PLACED PAR
  ...  other processors
  PROCESSOR 31 T4
    PLACE chan0 AT linkin1:
    PLACE chan3 AT linkout2:
    proc31 (chan0, chan3)
  PROCESSOR 43 T4
    PLACE chan0 AT linkout1:
    PLACE chan1 AT linkout3:
    PLACE chan2 AT linkin2:
    proc43 (chan0, chan1, chan2)
  PROCESSOR 26 T8
    PLACE chan2 AT linkout0:
    proc26 (chan2)
  ... more processors
```



Replicated placed PAR

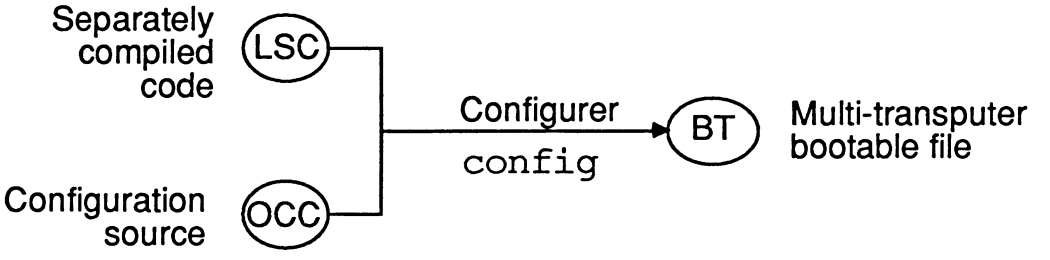



```
#SC "in"
#SC "out"
#SC "p"
[7]CHAN OF ANY pipe:
PLACED PAR
  PROCESSOR 0 T4
    ... place channels
    in (pipe[0], pipe[1])
  PLACED PAR i = 0 FOR 4
    PROCESSOR i+100 T4
      ... place channels
      p (pipe[i+1], pipe[i+2])
  PROCESSOR 1 T4
    ... place channels
    out (pipe[5], pipe[6])
```


The configurer

The configurer

To map processes onto processors.




```
config myconfig
```


Options

i	information - extra diagnostic messages
m	map - produces .cmp configuration map instead of bootable file
r	ROM - root processor boots from ROM

Filter practical - part 3

Copy and modify your filter code to filter 4 letters, configured to run on a target processor of one T4 and four T2 transputers.

The terminal handler should be on the T4. Each filter should be on a T2.

Draw a diagram showing processes, processors, channels and link numbers. Use the PC motherboard schematic (foil SL0.26) to help work out the internal link numbers. The terminal handler talks to the PC down link 0.

Write the configuration and run the configurer. Put the link addresses in an include file. Use a replicated PLACED PAR for the filters.

Connect the appropriate links on the PC add-in board to the links on the PC motherboard using the coloured link cables. Connect the PC add-in board Down port to the PC motherboard Up port using a grey reset cable.

Harness - possible solution page 1

File: main.occ

```
#INCLUDE "hostio.inc"
PROC change.case (CHAN OF SP fs, ts, []INT mem)
  #INCLUDE "protocol.inc"
  CHAN OF DATA keys:
  CHAN OF BYTE results:
  #USE "termharn.t8h"
  #USE "caseharn.t8h"

  PAR
    case.changer (keys, results)
    terminal.handler (fs, ts, results, keys)
:
```

File: protocol.inc

```
PROTOCOL DATA
  CASE
    data; BYTE
    terminate
:
```


Harness - possible solution page 2

File: termharn.occ

```
#INCLUDE "hostio.inc"
#INCLUDE "protocol.inc"
PROC terminal.handler (CHAN OF SP fs, ts,
          CHAN OF BYTE results, CHAN OF DATA keys)
  #USE "hostio.inc"
  #IMPORT "termhand.c8h"
  [3]INT in.ptr, out.ptr:
  SEQ
    LOAD.INPUT.CHANNEL (in.ptr[2], results)
    LOAD.OUTPUT.CHANNEL (out.ptr[2], keys)
    VAL comb.ws IS 1:
    [1]INT ws2:
    [5000]INT ws1:
    terminal.handler.c (fs, ts, comb.ws, ws1, ws2,
                in.ptr, out.ptr)
    so.exit (fs, ts, sps.success)
:
```

Harness - possible solution page 3

File: caseharn.occ

```
#INCLUDE "protocol.inc"
PROC case.changer (CHAN OF DATA keys,
                   CHAN OF BYTE results)
  #IMPORT "casech.c8h"
  [2]INT in.ptr, out.ptr:
  SEQ
    LOAD.INPUT.CHANNEL (in.ptr[1], keys)
    LOAD.OUTPUT.CHANNEL (out.ptr[1], results)
    VAL comb.ws IS 1:
    [1]INT ws2:
    [5000]INT ws1:
    case.changer.c(comb.ws, ws1, ws2, in.ptr, out.ptr)
:
```


Harness - possible solution page 4

File: termhand.l8h

```
terminal.handler.c=procent.c8h termhand.bin crtl.lib
```

File: casech.l8h

```
case.changer.c=procentc.t8h casech.bin sacrtl.lib
```

File: termhand

```
termhand.c8h:  termhand.bin   termhand.l8h
  ilink /f termhand.l8h /o termhand.c8h

termhand.bin:  termhand.c
  t8c termhand
```

File: casech

```
casech.c8h:  casech.bin   casech.l8h
  ilink /f casech.l8h /o casech.c8h

casech.bin:  casech.c
  t8c casech
```

Workspace - possible solution page 1

File: wkspace1.occ

```
#INCLUDE "hostio.inc"
PROC wkspace.1 (CHAN OF SP fs, ts, []INT free.mem)
  #USE "hostio.lib"
  #IMPORT "test.c8h"
  VAL comb.ws IS 1:
  VAL sep.ws  IS 0:
  [1]INT in.ptr, out.ptr:
  TIMER clock:
  [1]INT ws2:
  INT start, end:
  SEQ
    clock ? start
    test (comb.ws, free.mem, ws2, in.ptr, out.ptr)
    clock ? end
    ...  display result
    so.exit (fs, ts, sps.success)
:
```


Workspace - possibleC solution page 2

File: test.c

```
main ()
{
     int i;
     for (i = 0; i < 10000; i++);
}
```

File: test.l8h -- linker file

```
test=procentc.t8h test.bin sacrtl.lib
```

File: test -- make file

```
test.c8h:◊test.bin test.l8h
◊ilink /f test.l8h /o test.c4h
test.bin:◊test.c
◊t8c test
```

Workspace - possible FORTRAN solution page 3

File test.f77

```
      DO 10 I = 0, 10000
10    CONTINUE
      STOP
      END
```

File: test.8h -- linker file

```
test=procentf.t8h test.bin safrtl.lib
```

File: test -- make file

```
test.c8h:◊test.bin test.l8h
◊ilink /f test.l8h /o test.c8h
test.bin:◊test.c
◊t8f test
```


Workspace - possible Pascal solution page 4

File: test.pas

```
program test;
var
   i: int;
begin
   for i := 0 to 10000 do
    begin
    end;
end.
```

File: test.l8h -- linker file

```
test=procentp.t8h test.bin saprtl.lib
```

File: test -- make file

```
test.c8h:◊test.bin test.l8h
◊ilink /f test.l4h /o test.c8h
test.bin:◊test.c
◊t8p test
```

Workspace - possible solution page 5

File: harness.occ -- for separate workspace

```
#INCLUDE "hostio.inc"
PROC harness (CHAN OF SP fs, ts, []INT free.mem)
  #USE "hostio.lib"
  #IMPORT "test.c4h"
  VAL comb.ws IS 1:
  VAL sep.ws  IS 0:
  [1]INT in.ptr, out.ptr:
  TIMER clock:
  INT start, end:
  SEQ
    clock ? start
    [4000]INT heap:
    [512]INT stack:
    PLACE stack IN WORKSPACE:
    test (sep.ws, stack, heap, in.ptr, out.ptr)
    clock ? end
    ...  display result
    so.exit (fs, ts, sps.success)
:
```

File: casech.c

```
#include <chan.h>
#include <ctype.h>
#define keys       in[1]
#define results    out[1]
#define data       (char) 0
#define terminate  (char) 1

main (argc, argv, envp, in, inlen, out, outlen)
  char *argv[], *envp[];
  int argc, inlen, outlen;
  CHAN *in[], *out[];
{
  char tag, ch;
  int going = 1;
  while (going == 1)
    {
      chan_in_byte (&tag, keys);
      if (tag == terminate)
        going = 0;
      else
        {
          chan_in_byte (&ch, keys);
          chan_out_byte (chcase(ch), results);
        }
    }
}
```

Change case 3 possible solution page 1

```
char chcase (ch)
  char ch;
{
  if (isupper(ch))
    return (tolower(ch));
  else if (islower(ch))
    return (toupper(ch));
  else
    return (ch);
}
```


Change case 3 - possible solution page 2

File: termhand.c

```
#include <chan.h>
#include <stdio.h>
#define keys         out[2]
#define results      in[2]
#define data         (char) 0
#define terminate    (char) 1
#define terminatech '%'

main (argc, argv, envp, in, inlen, out, outlen)
  char *argv[], *envp[];
  int argc, inlen, outlen;
  CHAN *in[], *out[];
{
  char ch;
  int going = 1;
  while (going == 1)
   {
     ch = getchar();
     if (ch == terminatech)
       {
         going = 0;
         chan_out_byte (terminate, keys);
       }
     else
       {
         chan_out_byte (data, keys);
         chan_out_byte (ch, keys);
         chan_in_byte(&ch, results);
         putchar(ch);
       }
   }
}
```

Filter part 3 - possible solution - page 1

File: main.occ

```
#INCLUDE "linkadds"
#INCLUDE "constant"
#SC "termhand"
#SC "filter"
CHAN OF ANY from.host, to.host:
[n+1]CHAN OF ANY pipe:
PLACED PAR
  PROCESSOR n T4
    PLACE from.server AT linkin0:
    PLACE to.server   AT linkout0:
    PLACE pipe[0]     AT linkout2:
    PLACE pipe[n]     AT linkin1:
    terminal.handler (from.server, to.server, pipe[n], pipe[0])
  PLACED PAR i = 0 FOR n
    PROCESSOR i T2
      PLACE pipe[i]   AT linkin1:
      PLACE pipe[i+1] AT linkout2:
      filter (pipe[i], pipe[i+1])
```

File: linkadds.occ

```
VAL linkout0 IS 0:
VAL linkout1 IS 1:
VAL linkout2 IS 2:
VAL linkout3 IS 3:
VAL linkin0  IS 4:
VAL linkin1  IS 5:
VAL linkin2  IS 6:
VAL linkin3  IS 7:
```

File:constant.occ

```
VAL n              IS 4:
VAL terminate.ch IS '%':
```


Filter part 3 - possible solution - page 2

```
fdep termhand
tobat termhand
termhand
linkt @termhand.lnk,termhand.lsc /c/f/"terminal.handler"
fdep filter
tobat filter
filter
linkt @filter.lnk, filter.lsc /c /f /"filter"
config main
afserver  -:e -:b main.bt
```