/* CRITICAL.IH: Management of critical regions and interrupts

INTERFACE

Fernando
Updated: July 14, 1984

Interface Specification for Critical Sections (version of 7/14/84)
Fernando Pereira
Quintus CONFIDENTIAL

The critical section mechanism allows C or Prolog code to block until explicitly unblocked the occurrence of selected classes of events. The main concepts are that of an event, wich is any kind of abrupt bump to the flow of execution of system code, and that of an event type, which is a class of events that have boradly the same kind of effect. For example, events that one way or the other make the system jump to the top level, such as EV_ABORT, have all the event type ABORT_EVtype. It is possible to define an arbitrary number of event types; the classification of events in types is left to those people who define new events: they will know (I hope) what kind of behavior their new events cause.

Events are blocked and unblocked by a pair of operations, which for C are actually expanded in-line for efficiency. A critical section is the stretch of code between a block/unblock pair. Blocking of event types is stacked, that is if an an event type is blocked several times without being unblocked, the same number of unblockings is required to allow events of that type to take action again. This way, functions that call other other functions within critical sections need not know whether the called functions have their own critical sections for the same type.

The two operations are:

BlockEvents(type) [block_events(Type) from Prolog]

> Block all events of type 'type'. If an event of this type is called for while the type is blocked, the event number is recorded for later execution. If several events occur while their type is blocked, only the last one is remembered.

UnblockEvents(type) [unblock_events(Type) from Prolog]

> Unblock all events of type 'type'. If events of the blocked type occured while the type was blocked, the last one is fielded now,

unless other blockings of the same type are still active.

Advice on the Use of Critical Sections

Critical sections should be used whenever the occurrence of an event of a given type might leave some critical data in a corrupted state. For example, when new clauses are being added to the clause chain for a procedure, there might be an interval in which the pointers are not consistent. That interval should be a critical section for events of the ABORT_EVtype type, otherwise the code area might be corrupted.

Currently, there are four event types: ABORT_EVtype already discussed, NULL_EVtype for events that are never blocked, EMUL_EVtype for events that need a tidy emulator state (this must be defined by someone) and CONT_EVtype for events that will jsut continue the execution, maybe changing some atomic flags.

*/

```
/* Number of different event types */

&define EV_TYPES 4

/* Current event types */

&define NULL_EVtype     0       /* nothing at all -- never block these! */
&define ABORT_EVtype    1       /* events that bump us right to the top */
&define EMUL_EVtype     2       /* events that need a tidy emulator state */
&define CONT_EVtype     3       /* events that just continue */

/* ------------------------------------------------------------------
   Block event types
   ------------------------------------------------------------------ */

/* Block one type of event. Expanded in-line. */

extern int ev_blocked[];

&define BlockEvents(T)  (ev_blocked[T]++)

/* Out-of-line version to be called from Prolog */

extern void block_events(/* int */);

/* ------------------------------------------------------------------
   Unblock event types
   ------------------------------------------------------------------ */

/* Unblock one type of event */

extern int field_event(/* int */);

/* An arithmetic test rather than a if is used so that the
   whole test will be an expression, making C syntax problems
   on macro expansion less likely.
```

```
*/

&define UnblockEvents(T) ((--ev_blocked[T] == 0 &&
                           ev_happened[T] != EV_NULL) ? field_event(T) : 0)

/* Out-of-line version to be called from Prolog. */

extern void unblock_events(/* int */);

/* ------------------------------------------------------------------------
   Test for blocked event
   ------------------------------------------------------------------------ */

extern Bool EventBlocked(/* int */);

/* ========================================================================
                          FINIS
   ======================================================================== */
```

```
/* CRITICAL.C: Management of critical regions and interrupts

                                                    Fernando
                                                    Updated: July 14, 1984

        +=========================================================+
        |                                                         |
        | WARNING: This material is CONFIDENTIAL and proprietary  |
        |          to Quintus Computer Systems, Inc.              |
        |                                                         |
        |                                                         |
        |  Copyright (C) 1984,  Quintus Computer Systems, Inc.    |
        |  All rights reserved.                                   |
        |                                                         |
        +=========================================================+

*/

#include "critical.ih"

/* Level of event type blocking, suspended events. For simplicity,
   no more than one suspended event per type. */

public int ev_blocked[EV_TYPES], ev_happened[EV_TYPES];

/* Event type table */

private int ev_type[] = {
    NULL_EVtype,                /* EV_NULL -- no event at all */
    NULL_EVtype,                /* EV_NOTHING */
    NULL_EVtype,                /* EV_START (must check this...) */
    NULL_EVtype,                /* EV_DIE (never blocked) */
    ABORT_EVtype,               /* EV_IOERR */
    ABORT_EVtype,               /* EV_ARITH */
    ABORT_EVtype,               /* EV_EOF */
    ABORT_EVtype,               /* EV_SIGNAL */
    ABORT_EVtype,               /* EV_ABORT */
    NULL_EVtype,                /* unused */
    NULL_EVtype,                /* unused */
    EMUL_EVtype,                /* EV_OVSTACK */
    EMUL_EVtype,                /* EV_OVHEAP */
    EMUL_EVtype,                /* EV_OVTRAIL */
    EMUL_EVtype,                /* EV_OVPDL */
    EMUL_EVtype,                /* EV_OVSYMATOM */
    EMUL_EVtype,                /* EV_OVSYMPROC */
    EMUL_EVtype                 /* EV_OVCODE */
};

#define EV_BLOCKABLE EV_OVCODE  /* the last event in the table */

/* ------------------------------------------------------------------
   Test for blocked event
   ------------------------------------------------------------------ */

public Bool EventBlocked(ev)
int ev;
```

```
{
    int evtype;

    if (ev < EV_NULL || ev > EV_BLOCKABLE) return false;
    evtype = ev_type[ev];
    if (ev_blocked[evtype] > 0) {
        ev_happened[evtype] = ev;
        return true;
    }
    return false;
}

/* ---------------------------------------------------------------------
   Field the saved event for an event type just unblocked
   --------------------------------------------------------------------- */

/* This function returns an int to typecheck with the funny
   inline code of UnblockEvents (critical.ih)
*/

public int field_event(evtype)
int evtype;
{
    int event = ev_happened[evtype];
    ev_happened[evtype] = EV_NULL;
    if (event != EV_NULL) PrologEvent(event);
    return event;
}

/* ---------------------------------------------------------------------
   block_events and unblock_events
   --------------------------------------------------------------------- */

public void block_events(evtype)
int evetype;
{
    BlockEvents(evtype);
}

public void unblock_events(evtype)
int evtype;
{
    unblock_events(evtype);
}

/* =====================================================================
                          FINIS
   ===================================================================== */
```