14 Interrupt Migration
Interrupt Migration
Introduction

At boot time, the HP-UX kernel allocates external card interrupts to CPUs in a round-robin method. The kernel is not aware of card traffic patterns at that time. The boot allocation scheme could allow two heavily loaded cards to be mapping interrupts to the same CPU. This might lead to system performance degradation.

Interrupt Migration provides a flexible mechanism for managing CPU interrupt assignments by the movement of external I/O device interrupts from one CPU to another.

The /usr/contrib/bin/intctl command allows the user to display the interrupt configuration of the system and to migrate the interrupts between CPUs.
Overview of Interrupt Migration Impact on Driver

The interrupt migration operation is a transparent one to applications. The associated cards need not be quiesced during migration between processors. The applications utilizing the cards are not affected by these operations.

Drivers must be designed to spend minimal time in their ISR routines. They may need to quiesce the card interrupts during a migration operation, but quiescing the card DMA engines is not necessary.

In the HP-UX environment, drivers can use Line Based Interrupts (LBIs) or Transaction Based Interrupts (TBIs).

**LBI Drivers**

Most drivers using LBIs do not need to know which CPU is handling the interrupt. For the exceptions, WSIO provides new interrupt migration related events and informs the drivers of their use. If a driver using LBI has not registered for these interrupt event flags with WSIO, the interrupts of the associated card will be migrated to a different processor without informing the driver.

**TBI Drivers**

Drivers using TBIs spread the interrupt load across CPUs in the system and could interrupt more than one CPU. The drivers need to program their cards with the interrupt specifics (CPU address and data vector), so any TBI interrupt migration will be processed by the driver. The TBI drivers must first register with WSIO to utilize the new events.

WSIO provides new event masks for the drivers; they will be notified when a new CPU is enabled for interrupts in the system.

**Interrupt Line Sharing**

According to PCI bus specifications, a device can share an interrupt line with another device, port or function. All devices sharing the same interrupt line interrupt the same CPU. Migration of any of the interrupts sharing a line results in all interrupts on that line being migrated.
Interrupt Migration Event Masks and Registration with WSIO

WSIO defines three new event flags as part of interrupt migration. Two are for TBI based drivers, and one is for LBI based drivers.

LBI Event Flags

An optional new event flag, WSIO_EVENT_LBI_MIGR, upon registration, notifies the driver when the specified Line Based Interrupt is migrated to a new processor.

WSIO_EVENT_LBI_INTR_MIGR

Drivers of cards using LBIs may register for this event flag. They do this with the wsio_reg_drv_capability_mask() call. A driver registers for events in its _attach() routine after claiming the card it controls by calling isc_claim(). The driver also needs to register for an event handler in its _install() routine. Registration for this event flag is needed only if the driver has cached the interrupt-CPU. With an interrupt migration operation initiated, the interrupting CPU changes to a different CPU. If the driver has registered for this event flag, it is notified of this CPU change through the event handler mechanism.

Example of a driver registering for WSIO_EVENT_LBI_INTR_MIGR event:

driver_attach(....) {
    wsio_event_mask_t newmask;

    .
    isc_claim(isc);
    newmask = oldmask | WSIO_EVENT_LBI_INTR_MIGR
    ret = wsio_reg_drv_capability_mask( isc, newmask);
    ..
   ..
}

The `wsio_reg_drv_capability_mask()` call is described in the Device Driver Reference (DDR). If the driver has registered for the event flag, its event handler is invoked twice with the `WSIO_EVENT_LBI_INTR_MIGR` flag by WSIO. The first invocation notifies the associated card that the interrupt is migrating to a different CPU. The second invocation notifies the associated card that the migration is complete. The two invocations can be differentiated by the `wsio_generic_event_t` arg structure passed to the driver event handler. The element `wsio_intr_migr_info_t` inside the `wsio_generic_event_t` structure differentiates the two notifications.

```c
typedef struct wsio_generic_event {
    wsio_event_t event; // WSIO_EVENT_LBI_INTR_MIGR
    wsio_event_id_t event_id; // wsio provided event_id
    struct isc_table_type *isc; // pointer to isc
    generic_complete_callback_t wsio_completion_cb; // wsio callback
    void *arg; // pointer to wsio_intr_migr_t
} wsio_generic_event_t;

typedef struct wsio_intr_migr {
    void *intr_object; // Associated interrupt object
    spu_t new_spu_id; // ‘-1’ or new CPU id
                        // (WSIO_LBI_INTR_MIGR_NOTIFY
                        // or WSIO_LBI_INTR_MIGR_COMPLETE)
    wsio_intr_migr_info_t info; // WSIO_LBI_INTR_MIGR_NOTIFY or
                                 // WSIO_LBI_INTR_MIGR_COMPLETE
    wsio_intr_ret_t ret; // return value set by the driver
    void *resvd; // reserved field.
} wsio_intr_migr_t;
```

Once the driver handles the event, it invokes the WSIO provided completion callback routine to indicate completion status of the event. The code for the completion callback routine is as follows:
typedef int (*generic_complete_callback_t)(struct isc_table_type *,
    wsio_event_id_t, void *);

(*wsio_completion_callback) (isc, event_id,
    wsio_intr_migr_p);

The parameters are defined as:

isc
    Pointer to the ISC structure
event_id
    event id
wsio_intr_migr_p
    Pointer to the wsio_intr_migr that was passed to the driver event handler through the arg field of the wsio_generic_event_t structure.
Interrupt Migration
Interrupt Migration Event Masks and Registration with WSIO

Figure 14-1  LBI Interrupt Migration

The only permissible return value from the driver for this event flag is **WSIO_OK**.

---

**NOTE**

The driver event handler should **not** register for a timeout call to invoke the WSIO provided callback routine.

---

The general flow of events is shown in Figure 14-1, LBI Interrupt Migration, and the specific steps are:

1. WSIO invokes the driver’s event handler routine with the event flag as **WSIO_EVENT_LBI_INTR_MIGR**. The *arg* field of the *wsio_generic_event_t* structure contains the interrupt object for which this event is being performed. The event *info* flag is set to **WSIO_LBI_INTR_MIGR_NOTIFY**.
2. The driver returns the completion status by invoking the `wsio_completion_cb` function pointer.

3. Once the driver returns, WSIO invokes low level routines (Machine dependent layer) to migrate the line based interrupt to the new CPU.

4. The migration causes a spurious interrupt on the CPU to which the interrupt of this card has been migrated. Drivers **must** be capable of handling spurious interrupts.

5. When migration is complete, WSIO invokes the driver’s event handler with the event flag as `WSIO_EVENT_LBI_INTR_MIGR`. The `arg` field of the `wsio_generic_event_t` structure contains the interrupt object for which this event is being performed. The event `info` flag is set to `WSIO_LBI_INTR_MIGR_COMPLETE`. The `new_spu_id` field in the `arg` structure contains the value of the new CPU.

6. The driver returns the completion status by invoking the `wsio_completion_cb` function pointer.

**TBI EVENT FLAGS**

Two new flags are introduced as part of interrupt migration for drivers using TBIs. The first flag, `WSIO_EVENT_OFFLINE_CPU`, is a mandatory event, and upon registration notifies the driver when the concerned Transaction Based Interrupt is migrated to a new processor.

The second flag, `WSIO_EVENT_ONLINE_CPU`, is an optional event, and upon registration notifies the driver of any processor being enabled for interrupts.

**WSIO_EVENT_OFFLINE_CPU**

This event flag is for drivers using TBIs for their cards. All drivers using transaction based interrupts must register for the `WSIO_EVENT_OFFLINE_CPU` event flag. Drivers register for this event flag with the `wsio_reg_drv_capability_mask()` call. A driver registers for events it its `_attach` routine after claiming the card it controls by calling `isc_claim()`. The driver also must register for an event handler in its `_install()` routine.
Example of a driver registering for WSIO_EVENT_OFFLINE_CPU event:

```c
    driver_attach(....) {
        wsio_event_mask_t newmask;
        ...
        isc_claim(isc);
        newmask = oldmask | WSIO_EVENT_OFFLINE_CPU;
        ret = wsio_reg_drv_capability_mask(isc, newmask);
        ...
    }
```

Refer to the Device Driver Reference for a description of the wsio_reg_drv_capability_mask() call.

The interrupt objects are allocated in the init() routine. If a driver attempts to allocate a TBI and it has not registered for the WSIO_EVENT_OFFLINE_CPU event flag, WSIO fails the TBI allocation call with WSIO_ERROR.

The WSIO subsystem invokes the driver handler with the WSIO_EVENT_OFFLINE_CPU for the following scenarios:

- A CPU is being disabled or reserved for interrupts.
- The card interrupt is being bound to a different CPU

The `wsio_generic_event_t` argument for the event handler is stated as follows:

```c
typedef struct wsio_generic_event {
    wsio_event_t event;      // WSIO_EVENT_OFFLINE_CPU
    wsio_event_id_t event_id;// wsio provided event_id
    struct isc_table_type *isc;// pointer to isc
    generic_complete_callback_t wsio_completion_cb;
        // wsio callback
    void *arg;               // pointer to wsio_intr_migr_t
} wsio_generic_event_t;
```

```c
typedef struct wsio_intr_migr {
    void *intr_object;   // Associated interrupt object
    spu_t new_spu_id;    // '-1' or new CPU id
    wsio_intr_migr_info_t info; // NULL for this event
} wsio_intr_migr_t;
```
Interrupt Migration

Interrupt Migration Event Masks and Registration with WSIO

```c
wsio_intr_ret_t ret;  // return value set by the driver
void *resvd;         // reserved field.
} wsio_intr_migr_t;
```

Once the driver handles the event, it invokes the WSIO provided completion callback routine to indicate completion status of the event. The code for the completion callback routine is:

```c
typedef int (*generic_complete_callback_t) (isc_table_type *,
    wsio_event_id_t, void *);
(*wsio_completion_callback) (isc,
    event_id, wsio_intr_migr_p);
```

isc Pointer to the ISC structure

event_id Event identification

wsio_intr_migr_p Pointer to the wsio_intr_migr that was passed to the driver event handler through the arg field of the wsio_generic_event_t structure.

Upon return from the driver event handler, status is set in the ret field inside the wsio_intr_migr_t structure. The permissible return values from the driver event handler are:

- WSIO_E_DRV_DEACTIVATE_CALL The deactivate call failed
- WSIO_E_DRV_SET_CPU_SPEC The wsio_set_cpu_spec call failed
- WSIO_E_DRV_ACTIVATE_CALL The activate call failed
- WSIO_OK Everything is OK
NOTE

The driver event handler should **not** register for a timeout call to invoke the WSIO provided callback routine.

The general flow of events is presented in Figure 14-2, TBI Interrupt Migration, and the specific steps are:
1. WSIO invokes the driver’s event handler using the
   WSIO_EVENT_OFFLINE_CPU flag.

2. The driver performs the following tasks to migrate the interrupt
   from the CPU.
   a. Disable the card interrupts; **only** the interrupt corresponding to
      the interrupt object passed in the wsio_generic_event_t is
      disabled. It does **not** have to disable all other interrupts
      associated with this card.
   b. Invoke the wsio_intr_deactivate() routine for the interrupt
      object passed in wsio_generic_event_t structure. This disables
      the interrupt at the higher levels. For example, the driver’s ISR
      will be removed from the CPU’s interrupt switch table. This can
      cause pending interrupts to be lost. To nullify these effects, a
      spurious interrupt is generated when the driver calls
      wsio_intr_enable() (described later in this sequence).
   c. Invoke the wsio_intr_set_cpu_spec() routine. This migrates
      the interrupt object sent in the wsio_generic_event_t structure to
      the new CPU. See the wsio_intr_set_cpu_spec() call for further
      details.
   d. The drivers detect the new CPU to which the interrupt has been
      migrated with the wsio_intr_get_assigned_cpu() call, and
      determine the transaction address and the data with the
      wsio_intr_get_txn_info() call.
   e. With the interrupt migrated to a new CPU, the driver invokes
      the wsio_intr_enable() routine to enable the interrupt object
      passed in the wsio_generic_event_t structure. This interface
      results in a spurious interrupt being fired off to the migrated
      CPU. Drivers **must** be capable of handling these spurious
      interrupts.
   f. Program the card to use the new transaction address and
      transaction data obtained in step d.
   g. Return the completion status (WSIO_OK) with the
      wsio_completion_cb() routine.
Interrupt Migration
Interrupt Migration Event Masks and Registration with WSIO

**WSIO_EVENT_ONLINE_CPU**

This event flag is for drivers using TBIs. If the driver has registered for this event, WSIO invokes the driver event handler with the **WSIO_EVENT_ONLINE_CPU** flag when it is notified that new CPUs are enabled for interrupt processing.

Drivers register for this event flag with the `wsio_reg_drv_capability_mask()` call. A driver registers for events in its _attach_() routine after claiming the card it controls by calling `isc_claim()`. The driver also must register for an event handler in its _install_() routine.

Example of a driver registering for a **WSIO_EVENT_ONLINE_CPU** event:

```c
driver_attach(....)
{
    wsio_event_mask_t newmask;

    isc_claim(isc); newmask = oldmask | WSIO_EVENT_OFFLINE_CPU.
    ret = wsio_reg_drv_capability_mask(isc, newmask);
    ..
    ..
}
```

See the DDR for a description of the `wsio_reg_drv_capability_mask()` call.

The `wsio_generic_event_t` argument for the event handler is written as follows:

```c
typedef struct wsio_generic_event {
    wsio_event_t event;      // WSIO_EVENT_ONLINE_CPU
    wsio_event_id_t event_id; // wsio provided event_id
    struct isc_table_type *isc; // pointer to isc
    generic_complete_callback_t wsio_completion_cb; // wsio callback
    void *arg;               // Number of processors enabled
} wsio_generic_event_t;
```
Interrupt Migration Flow

The HP-UX environment interrupts could be either LBIs or TBIs. From WSIO perspective, the events that can lead to interrupt migration operation are:

- Changing the interrupt state of a CPU to RESERVED or DISABLED.

  When changing the interrupt state of a CPU from ENABLED to RESERVED or DISABLED, all the card interrupts associated with the CPU are redistributed in a round-robin method to the other interrupt ENABLED CPUs.

- Changing the interrupt state of a CPU to ENABLED.

  When a CPU state is changed to interrupt ENABLED, all the drivers registered for this event are informed.

- Migration of an interrupt from one CPU to another CPU through the user level command /usr/contrib/bin/intctl.

For CPU interrupt state change details refer to subsystems such as Processor Sets, ICOD (Instant Capacity on Demand), or RTE (Real Time Extension).

The following are the steps involved in migrating an LBI and a TBI.
Interrupt Migration

Interrupt Migration Flow

- **LBI processing**

  For every Line Based Interrupt:

<table>
<thead>
<tr>
<th><strong>WSIO Layer</strong></th>
<th><strong>Driver Layer</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO Layer</td>
<td>Driver Layer</td>
</tr>
<tr>
<td>When the driver is registered for WSIO_EVENT_LBI_INTR_MIGR, the driver event handler is invoked with the event information by WSIO_LBI_INTR_MIGR_NOTIFY.</td>
<td></td>
</tr>
</tbody>
</table>

  The driver handles the NOTIFY event and returns WSIO_OK to WSIO.

  WSIO calls the low level machine dependant routines to migrate the interrupt to the new CPU. Later, WSIO invokes the driver event handler to notify the completion event information as WSIO_LBI_INTR_MIGR_COMPLETE event.

  The driver handles the COMPLETE event and returns WSIO_OK to WSIO.

- **TBI processing**

  For every Transaction Based Interrupt:

<table>
<thead>
<tr>
<th><strong>WSIO Layer</strong></th>
<th><strong>Driver Layer</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>WSIO Layer</td>
<td>Driver Layer</td>
</tr>
<tr>
<td>WSIO invokes the driver event handler with an event flag as WSIO_EVENT_CPU_OFFLINE.</td>
<td></td>
</tr>
</tbody>
</table>

  Since this is a TBI, the driver performs the necessary steps for reprogramming the card with the new CPU value. Later the driver returns WSIO_OK to WSIO.
Assumptions and Dependencies

- The driver **must** handle spurious interrupts. The interrupts can happen even if the card is suspended or powered off.

- The routines `isrlink()`, `isrunlink()`, `wsio_intr_alloc()` and `wsio_intr_free()` are only invoked by the driver as part of handling events initiated from WSIO. This includes card online addition and removal, ioscan, DLKM load or unload, and interrupt migration. Drivers must not invoke these routines while handling non-WSIO events such as card reset. The `wsio_intr_set_cpu_spec()` and `wsio_intr_set_irq_line()` functions are not invoked for non-WSIO events.

- The routines `wsio_intr_activate()` and `wsio_intr_deactivate()` behave in different ways.

  If drivers using LBI invoke routines `wsio_intr_activate()` or `wsio_intr_deactivate()` while interrupt migration is in progress, the routines return failure.

  If interrupt migration is invoked and driver invocation with the above calls is in progress, interrupt migration will wait for the calls to complete.

  If the driver is using a TBI, the driver carries out the migration process. Part of the migration process is disabling and later enabling card interrupts. This can involve invocation of `wsio_intr_activate()`, `wsio_intr_deactivate()` and `wsio_intr_set_cpu_spec()` calls. Since the driver initiates all the processing, the drivers ensure there are no other threads calling these routines and the interrupt migration thread at the same time. WSIO does not be ensure this synchronization.

- When invoking the `wsio_intr_get_assigned_cpu()`, `wsio_intr_get_txn_info()` and `wsio_intr_get_irq_line()` routines, and interrupt migration is in progress, it is not guaranteed the values returned by these routines are the same as those assigned when interrupt migration is complete.

- Only one high availability event, such as card OLAR/D, error handling, processor OL*, ioscan, or interrupt migration will be in progress in the system at any given point in time.