Helium FIFO Buffers

Overview

FIFO buffers in Helium 2 are unidirectional data paths that allow data to flow from one thread to another. Only one thread may put data into a FIFO at any given time, and only one thread may take data out of it. As its name suggests the first data written to a FIFO by its registered writing thread will also be the first data read from that FIFO by the reading thread.

Definitions

FIFO

First In-First Out data structure. In this sort of data structure, the first data put in is the first data to be retrieved.

Implementation

Each FIFO consists of a base data structure containing general information about the FIFO and pointing to a singly-linked list of data structures containing the information to be transferred by the FIFO. References to the FIFO are made with the address of the base data structure. Members of the FIFO base data structure are listed in Table 1 below.

`THREAD *reader`	Thread that is allowed to read from the FIFO. Set to NULL to allow ISRs to read from the FIFO.
`THREAD *writer`	Thread that is allowed to write to the FIFO. Set to NULL to allow ISRs to write to the FIFO.
`FIFODATA *first`	First data element in the FIFO. This will be the next data element when the FIFO is read.
`FIFODATA *last`	Last data element in the FIFO. All new elements written to the FIFO will be inserted after this element.
`MUTEX *fMutex`	Mutex used by FIFO data access functions to ensure data integrity when reading or writing to the mutex.
`int fifoDepth`	Maximum number of data elements the FIFO may contain. This ensures that the FIFO does not take up all available pool memory

Table 1: Contents of a FIFO base data structure.

User Interface

Before a FIFO may be used, it must be created using the system call `fifoCreate`. This function populates elements of the base FIFO data structure (see Table 1) and registers the calling thread as either the reading thread or the writing thread. As previously stated, only one thread is allowed to read from a FIFO at a time. Similarly, only one thread is allowed to write to a FIFO at a time. Therefore, by passing either `FIFO_READ_MODE` or `FIFO_WRITE_MODE` as the mode parameter to `fifoCreate`,

the calling thread can define itself as either the registered reader or writer of the FIFO. It should subsequently register another thread to perform the complementary function by calling `fifoRegister` with the opposite data direction mode.

System Call Implementation

User threads may call system functions for accessing FIFOs using the TRAP 7 software interrupt. **Practically, these system calls should be made using the functions defined in `oscalls.c`** (see the list of functions in Table 2).

`fifoCreate`	Create a FIFO data structure.
`fifoRegister`	Register a thread as either a reader or a writer to a particular FIFO.
`fifoWrite`	Write data to a FIFO. Only the thread registered as the writer to that FIFO may call this function.
`fifoRead`	Read data from a FIFO. Only the thread registered as the reader of that FIFO may call this function.

Table 2: User functions to access FIFOs. Definitions of these functions are located in the file `oscalls.c`.

The functions listed in Table 2 populate the low data registers (D0-D6) with the parameters to be passed to the kernel-level system handler. D7 is then populated with a code corresponding to the function to be performed by Helium, and the TRAP 7 interrupt is called.