- To initiate a transfer, the requestor calls the procedure `link`. This procedure sends the message to the server process and queues the LCB on its *message queue*. At this point the server process has not been involved, but the dispatcher awakes the process with an `LREQ` (*link request*) event.

  On the requestor side, the call to `link` returns immediately with information to identify the request; the requestor does not need to wait for the server to process the request.
- At some time the server process sees the `LREQ` event and calls `listen`, which removes the first LCB from the message queue.
- If a buffer is associated with the LCB and includes data to be passed to the server, the server calls `readlink` to read in the data.
- The server then performs whatever processing is necessary, and next calls `writelink` to reply to the message, again possibly with a data buffer. This wakes the requestor on `LDONE`.
- The requestor sees the `LDONE` event, examines the results, and terminates the exchange by calling `breaklink`, which frees the associated resources.

Only other parts of the kernel use the message system directly; the *file system* uses it to communicate with the I/O devices and other processes. Interprocess communication is handled almost identically to I/O, and it also is used for maintaining fault-tolerant *process pairs*.

This approach is inherently asynchronous and multithreaded: after calling `link`, the requestor continues its operations. Many requestors can send requests to the same server, even when it's not actively processing requests. The server does not need to respond to the link request immediately. When it replies, the requestor does not need to acknowledge the reply immediately. Instead, in each case the process is woken on an event that it can process when it is ready.

Process Pairs, Revisited

One of the requirements of fault tolerance is that a single failure must not bring the system down. We've seen that the I/O processes solve this by using process pairs, and it's clear that this is a general way to handle the failure of a CPU. Guardian therefore provides for creation of user-level process pairs.

All process pairs run as a *primary* and a *backup* process. The primary process performs the processing, while the backup process is in a "hot standby" state. From time to time the primary process updates the memory image of the backup process, a process called *checkpointing*. If the primary fails or voluntarily gives up control, the backup process continues from the state of the last checkpoint. A number of procedures implement checkpointing, which is performed by the message system:

- The backup process calls `checkmonitor` to wait for checkpoint messages from the primary process. It stays in `checkmonitor` until the primary process goes away or relinquishes control.