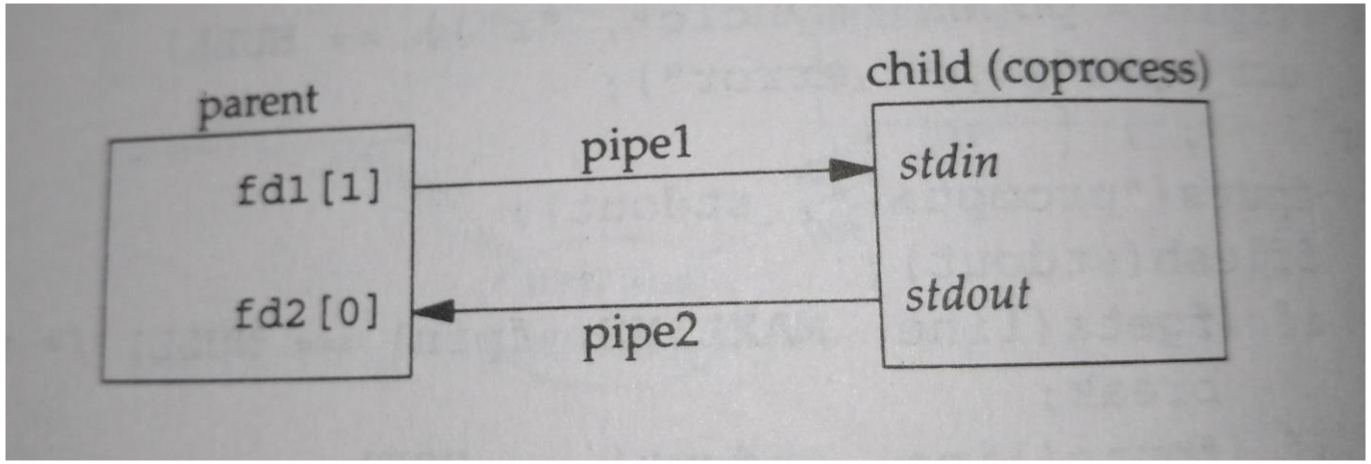
CLIENT-SERVER PROPERTIES

CLIENT-SERVER PROPERTIES

- Some of the properties of clients and servers that are affected by the various types of IPC used between them are listed below.
- The simplest type of relationship is to have the client *fork* and *exec* the desired server.
- Two half-duplex pipes can be created before the *fork* to allow data to be transferred in both directions



- The server that is executed can be a set-user-ID program, giving it special privileges.
- Also, the server can determine the real identity of the client by looking at its real user ID.

CLIENT-SERVER PROPERTIES

- With this arrangement, we can build an open server (Section 17.5)
- It opens files for the client instead of the client calling the *open* function.
- This way, additional permission checking can be added, above and beyond the normal UNIX system user/group/other permissions.
- We assume that the server is a set-user-ID program, giving it additional permissions (like root access).
- The server uses the real user ID of the client to determine whether to give it access to the requested file.
- This way, we can build a server that allows certain users permissions that they don't normally have.
- In this example, since the server is a child of the parent(i.e., parent associated with a particular user), all the server can do is pass back the contents of the file to the parent. It cannot open the file and send back file descriptors.

CLIENT-SERVER PROPERTIES

- Another case is when the server is a daemon process that is contacted using some form of IPC by all clients.
- We can't use pipes for this type of client-server.
- A form of named IPC is required, such as FIFOs or message queues.
- With FIFOs, we saw that an individual per client FIFO is also required if the server is to send data back to the client.
- If the client-server application sends data only from the client to the server, a single well-known FIFO suffices

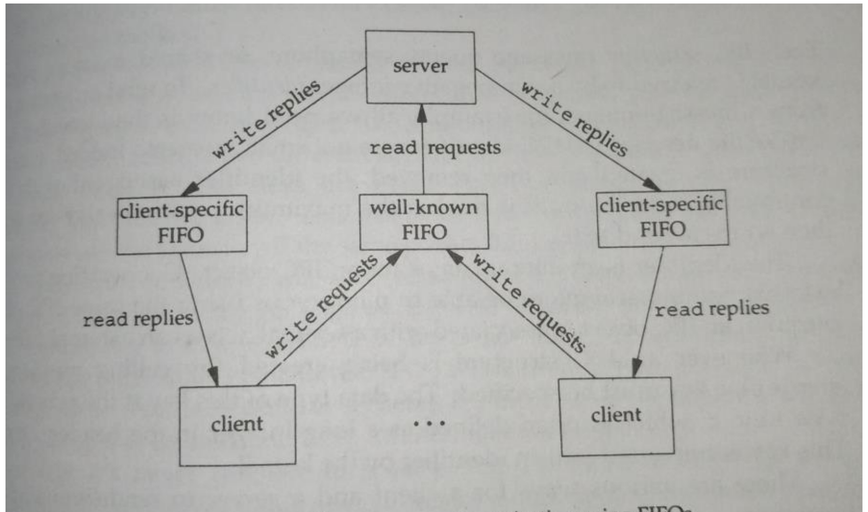


Figure 15.23 Client–server communication using FIFOs

CLIENT-SERVER PROPERTIES

- Multiple possibilities exist with message queues –

  – A single queue can be used between the server and all the clients, using the type field of each message to indicate the message recipient.

  For example, the clients can send their requests with a type field of 1. Included in the request must be the client's process ID (PID).

  The server then sends the response with the type field set to the client's PID.

  The server receives only the messages with a type field of 1, and the clients receive only the messages with a type field equal to their PIDs.

  – Alternatively, an individual message queue can be used for each client.

  Before sending the first request to a server, each client creates its own message queue with a key of IPC_PRIVATE. The server also has its own queue, with a key or identifier known to all clients.

  The client sends its first request to the server's well-known queue, and this request must contain the message queue ID of the client's queue.

  The server sends its first response to the client's queue, and all future requests and responses are exchanged on this queue.

CLIENT-SERVER PROPERTIES

(One problem with this technique is the wasteful use of a limited systemwide resource (a message queue), and a FIFO can be used instead. Another problem is that server has to read messages from multiple queues. Neither select nor poll works with message queues)

- Either of these two techniques using message queues can be implemented using shared memory segments and a synchronization method (a semaphore or record locking).
- The problem with this type of client-server relationship (the client and the server being unrelated processes) is for the server to identify the client accurately.
- Unless the server is performing a non-privileged operation, it is essential that the server know who the client is. This is required, for example, if the server is a set-user-ID program.
- Although all these forms of IPC go through the kernel, there is no facility provided by them to have the kernel identify the sender.
- With message queues, if a single queue is used between the client and the server (so that only a single message is on the queue at a time, for example), the *msg_lspid* of the queue contains the process ID of the other process.
- But when writing the server, we want the effective user ID of the client, not its process ID. There is no portable way to obtain the effective user ID, given the process ID.
- Naturally, the kernel maintains both values in the process table entry, but other than rummaging around through the kernel's memory, we can't obtain one, given the other.

CLIENT-SERVER PROPERTIES

- Another technique can be used to allow the server to identify the client.(Section 17.3)
- The client must create its own FIFO and set the file access permissions of the FIFO so that only user-read and user-write are on.
- We assume that the server has superuser privileges (or else it probably wouldn't care about the client's true identity), so the server can still read and write to this FIFO.
- When the server receives the client's first request on the server's well-known FIFO which must contain the identity of the client-specific FIFO, the server calls either *stat* or *fstat* on the client-specific FIFO.
- The server assumes that the effective user ID of the client is the owner of the FIFO (the *st_uid* field of the *stat* structure).
- The server verifies that only the user-read and user-write permissions are enabled.
- As another check, the server should also look at the three times associated with the FIFO (the *st_atime*, *st_mtime*, and *st_ctime* fields of the *stat* structure) to verify that they are recent (no older than 15 or 30 seconds, for example).
- If a malicious client can create a FIFO with someone else as the owner and set the file's permission bits to user-read and user-write only, then the system has other fundamental security problems.

THANK YOU

By,

Ashok Aravind S N

1BG18IS007