Figure 0.3 C program for server procedures in Sun RPC.

```
/* File S.c - server procedures for the FileReadWrite service */
#include <stdio.h>
#include <rpc/rpc.h>
#include"FileReadWrite.h"

void * write_2(writeargs *a)
{
 /* do the writing to the file */
}

Data * read_2(readargs * a)
{
        static Data result;     /* must be static */
        result.buffer  =  ...   /* do the reading from the file */
        result.length  =  ...   /* amount read from the file */
        return &result;
}
```

Server program ◊ The implementor uses the C function prototypes in the header file created by *rpcgen* as a basis for the implementation of the service as shown in Figure 0.3. The server procedure names are the names given in the interface definition converted to lower case and with an underscore and the version number appended. The argument of each server procedure is a pointer to a single argument or to a structure containing all the arguments. Similarly the value returned is a pointer to a single result or to a structure that contains the results. The latter must be declared as *static*.

The server program consists of the server procedures, supported by the *main*, the despatcher and the marshalling procedures, all of which are output by *rpcgen*. The *main* procedure of a server program creates a socket for receiving client request messages and then exports the service interface by informing the local port mapper of the program number, version number and the port identifier of the server.

When a server receives an RPC request message, the despatcher checks the program and version numbers, unmarshals the arguments and then calls the server procedure corresponding to the procedure number specified in the RPC request message. When the server procedure returns the results, it marshals them and transmits the reply message to the client.

Binding ◊ We have already noted that Sun RPC operates without a network-wide binding service. Therefore clients must specify the hostname of the server when they import a service interface. The port mapper enables clients to locate the port number part of the socket address used by a particular server. This is a local binding service – it runs at a well-known port number on every host and is used to record the mapping from program number and version number to port number of the services running on that host. When a server starts up it registers its program number, version number and port number with the local port mapper. When a client starts up, it finds out the server's port by making a remote request to the port mapper at the server's host, specifying the program number and version number. This means that servers need not run at well-known ports.

When a service has multiple instances running on different computers, the instances may use different port numbers for receiving client requests. Recall that broadcast datagrams are sent to the same port number on every computer and can be received by processes via that port number. If a client needs to multicast a request to all the instances of a service that are using different port numbers, it cannot use a direct broadcast message for this purpose. The solution is that clients make multicast remote procedure calls by broadcasting them to all the port mappers, specifying the program and version number. Each port mapper forwards all such calls to the appropriate local service program, if there is one.