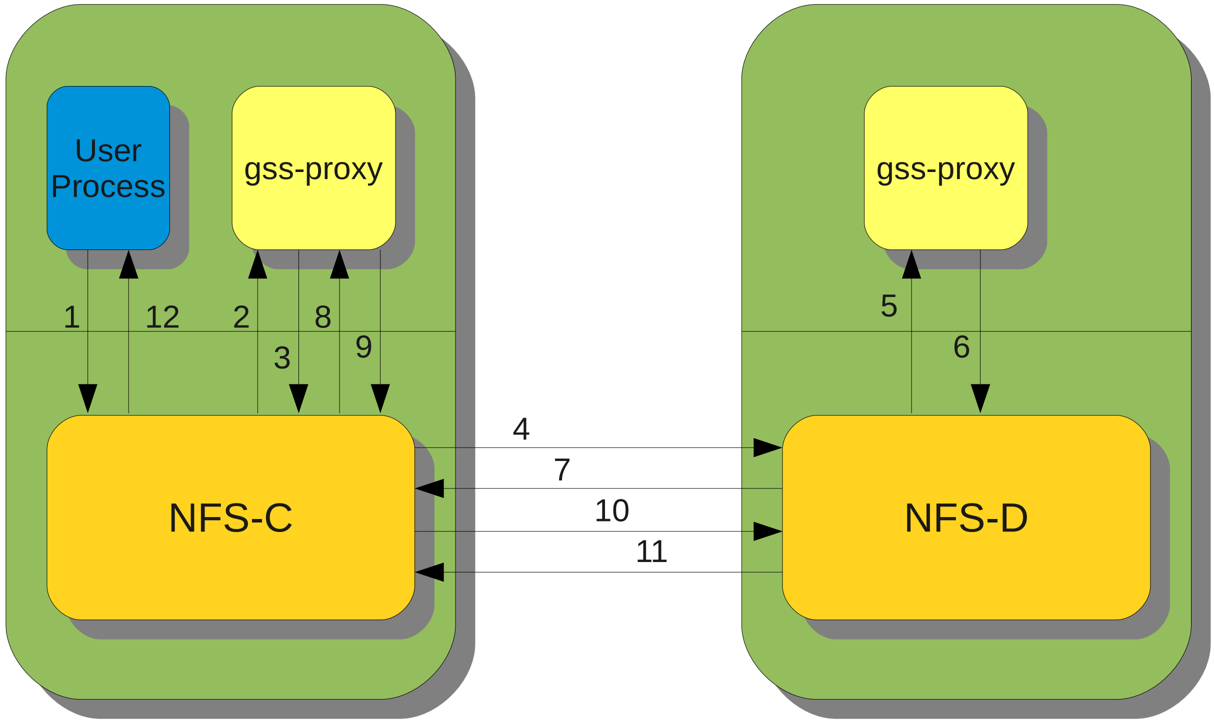

1. User process does initial operation requiring an NFS object which requires rpcsec_gss authentication.
2. Kernel code discovers it has no cached context for this user/server combination and does an upcall to obtain a security context.
   The request is done by pumping a gssx_arg_init_sec_context RPC request up to a file like it is done now for rpc.svcgssd (except this use a manually crafted protocol).
   gss-proxy assumes the user has a credential cache and a valid krbtgt.
   If a valid ccache is found for the user the gss-proxy calls the actual gss_init_sec_context() GSSAPI call and eventually acquires a ticket for the remote server.
3. gss-proxy sends down a gssx_res_init_sec_context reply containing a context reference and aa token to send to the server.
4. The NFS client sends a NULL RPC call to the server.
5. The server performs a gssx_arg_accept_sec_context RPC call to the server's gss-proxy.
   It performs an actual GSSAPI gss_accept_sec_context() call using the NFS keytab, and completes the negotiation.
6. The gss-proxy returns a gss_res_accept_sec_context RPC reply to the NFS server which contains a lucid context, a set of credentials, and an output token.
7. The NFS server returns the token to the client
8. The client takes the token and makes a second gssx_arg_init_sec_context() call to gss-proxy
9. The gss-proxy complete the init context and returns a lucid context to the kernel in a gss_res_init_sec_context reply.
10. The original operation can now be performed using the security context cached by the kernel.
11. Response to the original operation
12. Results are returned to the user process