AOS Network Stack

Jae Min Choi

SNUCSE

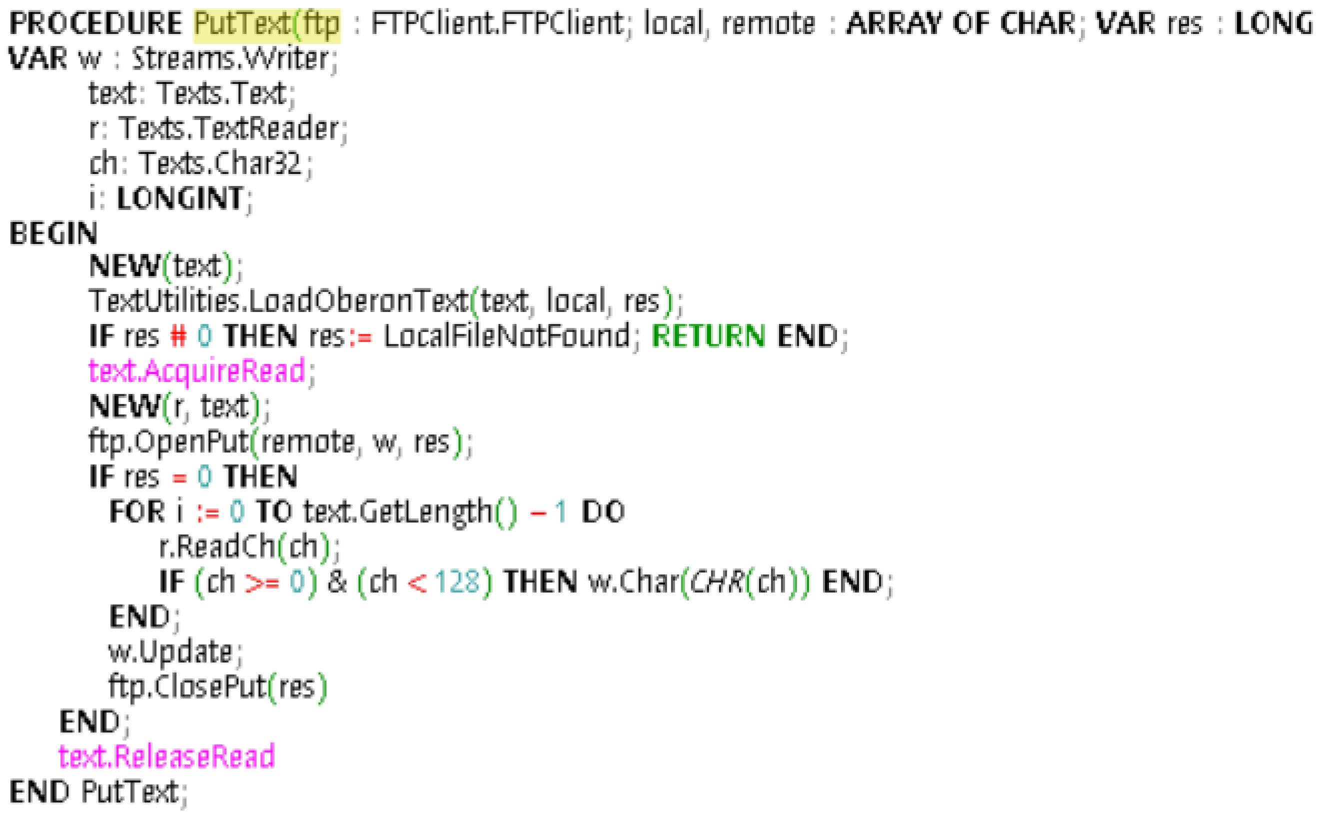
Function Call Procedure

```
FTP.PutText() in FTP.Mod {
        FTPClient.FTPClient.OpenPut() in FTPClient.Mod {
                FTPClient.FTPClient.GetDataConnection()
                FTPClient.FTPClient.OpenDataConnection()
                Streams.OpenWriter() in Streams.Mod
        }
        Streams.Writer.Char() in Streams.Mod {
                TCP.Connection.Send() in TCP.Mod {
                        Network.Copy() in I386.Network.Mod
                }
        }
        FTPClient.FTPClient.ClosePut() in FTPClient.Mod
}
```

FTP.Mod

- PutText()
  - Put a text file in a remote address

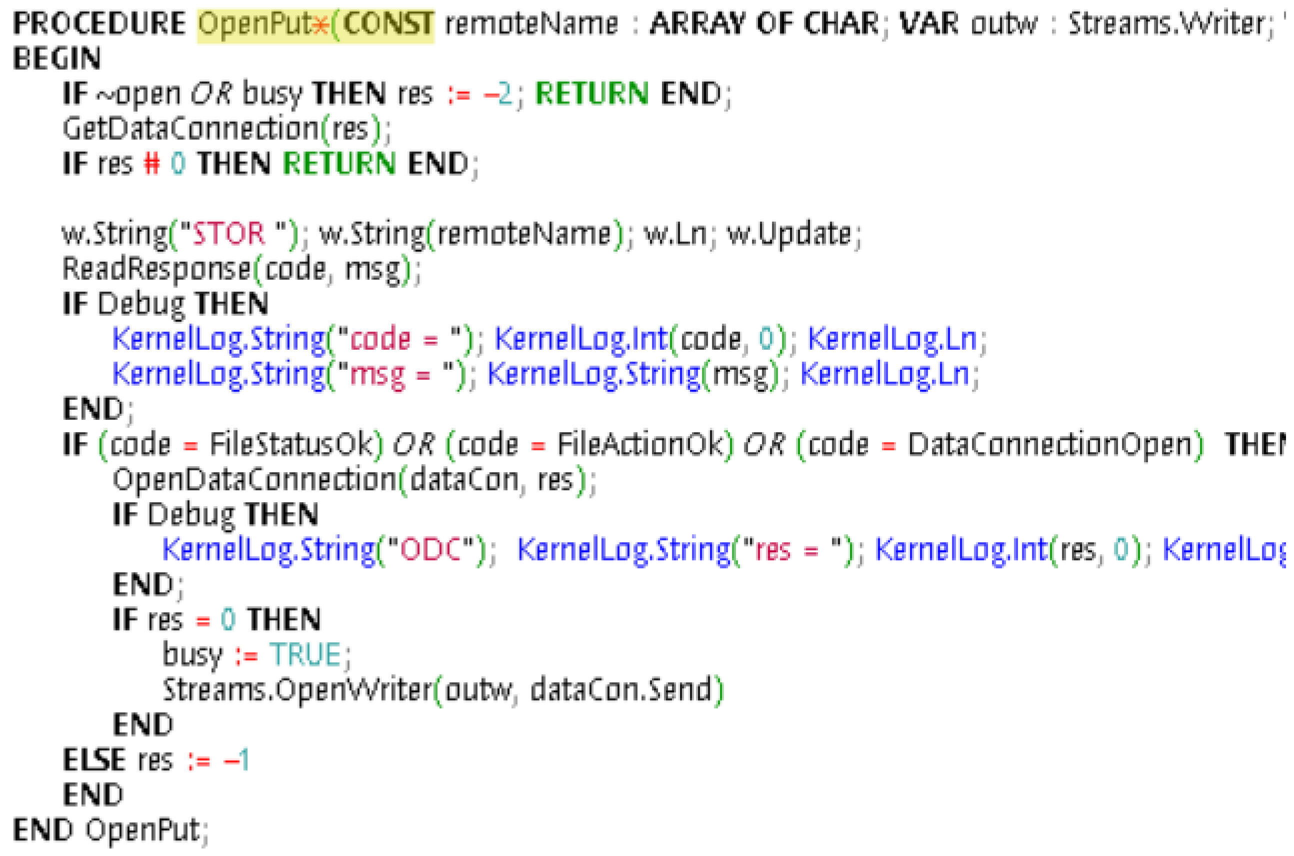
1. Load text
2. FTPClient.FTPClient.OpenPut()
   - Setup process that connects writer to remote address
3. Streams.Writer.Char()
   - Write characters
4. FTPClient.FTPClient.ClosePut()
   - Wrap up



```
PROCEDURE PutText(ftp : FTPClient.FTPClient; local, remote : ARRAY OF CHAR; VAR res : LONG
VAR w : Streams.Writer;
        text: Texts.Text;
        r: Texts.TextReader;
        ch: Texts.Char32;
        i: LONGINT;
BEGIN
        NEW(text);
        TextUtilities.LoadOberonText(text, local, res);
        IF res # 0 THEN res:= LocalFileNotFound; RETURN END;
        text.AcquireRead;
        NEW(r, text);
        ftp.OpenPut(remote, w, res);
        IF res = 0 THEN
          FOR i := 0 TO text.GetLength() - 1 DO
                r.ReadCh(ch);
                IF (ch >= 0) & (ch < 128) THEN w.Char(CHR(ch)) END;
          END;
          w.Update;
          ftp.ClosePut(res)
      END;
      text.ReleaseRead
END PutText;
```

FTPClient.Mod

- OpenPut()
  1. GetDataConnection()
     - Get available connection
  2. ReadResponse()
     - Poll for remote status
  3. OpenDataConnection()
     - Establish connection
  4. Streams.OpenWriter()
     - Link writer with data connection
- Now ready to copy text

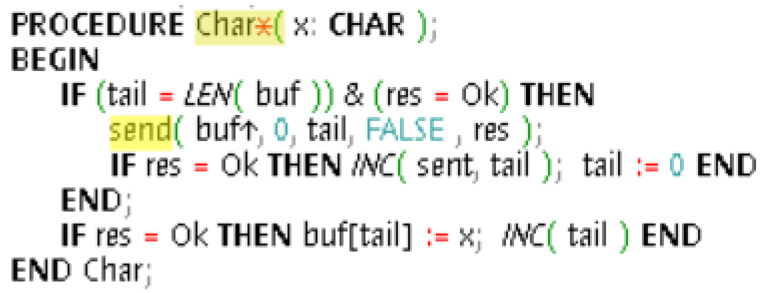


```
PROCEDURE OpenPut*(CONST remoteName : ARRAY OF CHAR; VAR outw : Streams.Writer; '
BEGIN
    IF ~open OR busy THEN res := -2; RETURN END;
    GetDataConnection(res);
    IF res # 0 THEN RETURN END;

    w.String("STOR "); w.String(remoteName); w.Ln; w.Update;
    ReadResponse(code, msg);
    IF Debug THEN
        KernelLog.String("code = "); KernelLog.Int(code, 0); KernelLog.Ln;
        KernelLog.String("msg = "); KernelLog.String(msg); KernelLog.Ln;
    END;
    IF (code = FileStatusOk) OR (code = FileActionOk) OR (code = DataConnectionOpen)  THE
        OpenDataConnection(dataCon, res);
        IF Debug THEN
            KernelLog.String("ODC");  KernelLog.String("res = "); KernelLog.Int(res, 0); KernelLo
        END;
        IF res = 0 THEN
            busy := TRUE;
            Streams.OpenWriter(outw, dataCon.Send)
        END
    ELSE res := -1
    END
END OpenPut;
```

Streams.Mod

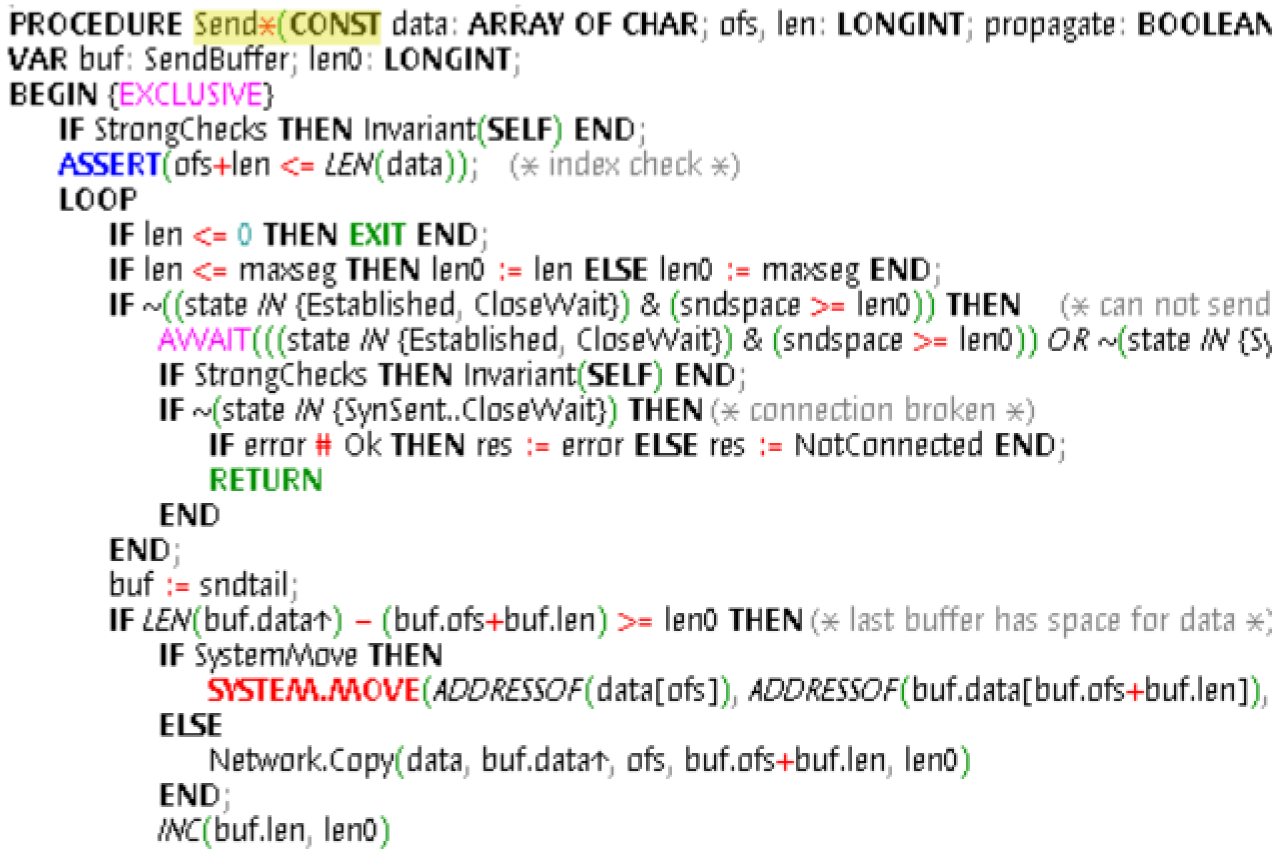
- Writer.Char()
  - Write a character
  1. send()
     - A delegate function
     - Calls Send() of the data connection



```
PROCEDURE Char*( x: CHAR );
BEGIN
    IF (tail = LEN( buf )) & (res = Ok) THEN
        send( buf↑, 0, tail, FALSE , res );
        IF res = Ok THEN INC( sent, tail );  tail := 0 END
    END;
    IF res = Ok THEN buf[tail] := x;  INC( tail ) END
END Char;
```

TCP.Mod

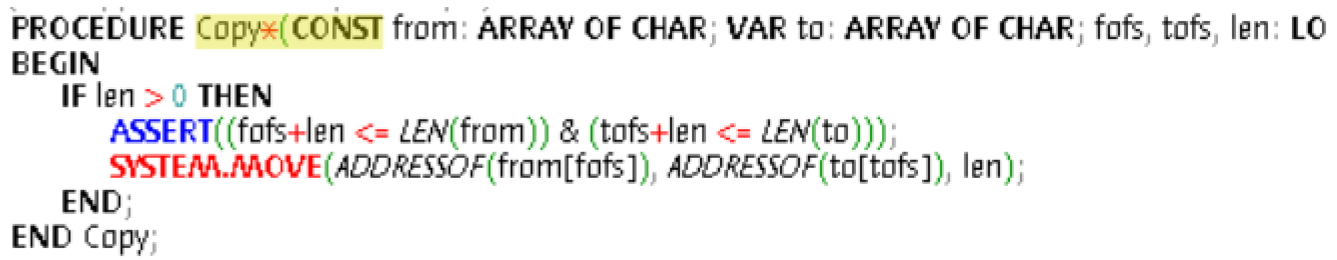
- Connection.Send()
  - With the data connection previously established, actually copy the data
  - Network.Copy()



```
PROCEDURE Send*(CONST data: ARRAY OF CHAR; ofs, len: LONGINT; propagate: BOOLEAN
VAR buf: SendBuffer; len0: LONGINT;
BEGIN {EXCLUSIVE}
    IF StrongChecks THEN Invariant(SELF) END;
    ASSERT(ofs+len <= LEN(data));   (* index check *)
    LOOP
        IF len <= 0 THEN EXIT END;
        IF len <= maxseg THEN len0 := len ELSE len0 := maxseg END;
        IF ~((state IN {Established, CloseWait}) & (sndspace >= len0)) THEN   (* can not send
            AWAIT(((state IN {Established, CloseWait}) & (sndspace >= len0)) OR ~(state IN {Sy
            IF StrongChecks THEN Invariant(SELF) END;
            IF ~(state IN {SynSent..CloseWait}) THEN (* connection broken *)
                IF error # Ok THEN res := error ELSE res := NotConnected END;
                RETURN
            END
        END;
        buf := sndtail;
        IF LEN(buf.data↑) – (buf.ofs+buf.len) >= len0 THEN (* last buffer has space for data *)
            IF SystemMove THEN
                SYSTEM.MOVE(ADDRESSOF(data[ofs]), ADDRESSOF(buf.data[buf.ofs+buf.len]),
            ELSE
                Network.Copy(data, buf.data↑, ofs, buf.ofs+buf.len, len0)
            END;
            INC(buf.len, len0)
```

I386.Network.Mod

- Copy()
  - End of call stack
  - Use SYSTEM.MOVE to copy the char
  - Target address is probably memory mapped



```
PROCEDURE Copy*(CONST from: ARRAY OF CHAR; VAR to: ARRAY OF CHAR; fofs, tofs, len: LO
BEGIN
    IF len > 0 THEN
        ASSERT((fofs+len <= LEN(from)) & (tofs+len <= LEN(to)));
        SYSTEM.MOVE(ADDRESSOF(from[fofs]), ADDRESSOF(to[tofs]), len);
    END;
END Copy;
```