Lab 8 – Programming with Processes

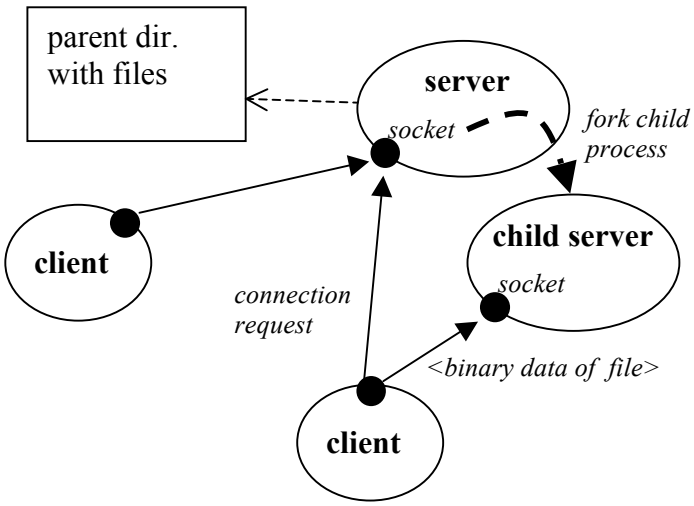
A file server using TCP socket communication

You need to implement an Internet file server that allows uploading files by any client connected to the Internet. For example, sensor node clients upload sensor data files to the sensor gateway running the file server. TCP sockets are used for the communication between server and client. We refer to the course "Data communication and computer networks" for more information on TCP and socket +calls. For testing purposes, the local loopback network 127.0.0.1 is preferred.

A multi-processing single port file server

A building has typically many sensor nodes and some of them might start uploading their data at exactly the same time. Therefore, the file server should be able to handle multiple connection requests at the same time. The server can solve this problem by implementing the manager/worker multi-tasking design pattern. The main or manager process listens on one port for incoming client requests and creates a new worker process for each new connection request to handle the file uploading using the socket returned by the accept() call.

The creation of the worker processes must be done dynamically, i.e. do not work with a fixed set and static pool of processes. In advance you don't know how many clients are going to connect! The server must be truly multi-processing: a new request must be handled even when some file upload from a client is on-going.



Keep in mind that the parent should 'wait' on the 'exit' of its children to avoid the creation of zombie-processes. This means that the parent has to keep track of the PIDs of all children and check from time to time (without blocking on-going file downloads and new incoming connection requests) if one of the children already exited. Use 'waitpid()' to do this and check out the 'WNOHANGUP' option of this command.

The server is started up as a terminal application with a command-line argument to define the port number on which the server has to listen for incoming connections, e.g.:

> fserver 1234

start a file server listening on the port 1234.

The client is also started up as a terminal application with the IP address and port number of the server as a command line argument, e.g. > fclient 127.0.0.1 1234.

The client connects to the file server. Once connected, the client uploads the file to the file server using the open TCP connection.

The directory where the server runs is called the parent directory. You may assume that clients can only upload files in the parent directory. As an extension to this exercise, you may implement a server that allows uploading files in subdirectories of the parent directory.

The available source code shows an example of a server and client using TCP/IP socket communication. First start the server, next start the client. The client will send a text message to the server. The server will ‘echo’ this message back to the client (this is called an echo server and is useful for testing purposes). You can start from this code and modify it. In the given code, the server however supports only one client connection at the same time.