Larry Bush		Readme file for Homework 4
email	:	bushl2@rpi.edu Lawrence_Bush@dps.state.ny.us
Student ID	:	660 220 742
Home Work 4	:	Chat Server
Class	:	Operating Systems
Instructor	:	Susan Bonner
Language	:	C++

My program does the following:

Main Assignment -	Implements a chat server. Handles many clients all talking at once.
Extra Credit -	Changes security parameters (using the security objects) to allow different uses log on. Implements impersonation.

Known Problems: None.

Description of Program Files

File ----	Purpose -------
main.cpp	Initializes the chat server. Calls functions to accept new client connections and handle them.
Server.h	Declares the Server class, which manages a group of client connections for a chat service and broadcasts (to other clients) all of the messages that it receives.
Server.cpp	Implementation of Server member functions.
Client.h	Declares the Client class, which represents one connection to a chat service.
Client.cpp	Implements of Client member functions.

Program Design

My program uses the starter code design.

Server Semantics

My server echos all of the output to the server window.
It also echos when a user disconnects.
It also sends error messages to the server window.

It also checks for many errors. (see below)

All access to the client list is mutexed. For example, when we insert, delete, assign or iterate through the client list, the mutex is invoked to prevent other conflicting operations on the list.

Mutex : Initialized in the server constructor.
Closed in the server destructor.

Resource management

- The server and client object ~destructors close all handles they have open and delete all directly allocated memory. The client ~destructor also flushes and disconnects the pipe, close the handle to the pipe and reverts to self(Impersonation cleanup). The server ~destructor also closes the mutex that the server's constructor initialized.

- My server object cleans up all clients before closing a server object (even though the server object is never closed).

- My program deletes and cleans up clients when they disconnect.

- My program cleans up clients if they are accidentally disconnected.

- My program cleans up clients if the thread cannot be started.
This is done directly (right where the error is detected) because the throwback point is in the main program. This is because, when a thread is spawned, if the tread cannot be created, the client is left in the list but not running. Therefore, we must remove it then and there. Since we cannot start the thread, we cannot use the thread catch mechanism to do this cleanup for us.

Command Line Error Catching

None. Dan and spec. said that the chatclient does this for us.

SysCall Error Catching

I check for errors on:

WaitForSingleObject
ReleaseMutex
GetUserName
CreateMutex
CloseHandle
_beginthreadex (already done is starter code)
InitializeSecurityDescriptor
SetSecurityDescriptorDacl
CreateNamedPipe
ConnectNamedPipe

FlushFileBuffers
DisconnectNamedPipe
ReadFile
WriteFile
ImpersonateNamedPipeClient

My program throws the errors where appropriate.

For example: If an error occurs in the client initialization,
the program allows the program to continue so that the error will be
caught in the server and thrown back so that the client will be deleted
and removed from the list (cleaned up).

Impersonation Error checking

Note to grader: On some computers, ImpersonateNamedPipeClient returns an
error if you try to connect from the same computer that the server is running on.
This may not occur on your machine. Dan tested it and was allowed to
impersonate himself. However, this occured on every computer I have run it
on (VCC and Library).

This is not an error in the program, this is caused my Mircrosoft not
allowing you to impersonate yourself in combination with the fact that
my program checks for these errors and responds appropriately.

If the function fails, the return value is zero. I report this error,
and close the handle so that the client will not be served.

If the return value (of ImpersonateNamedPipeClient) indicates that the
function call failed, then no client requests should be executed.
However, if you want to be able to connect to and test the server from
the same machine, and are unable to, then uncomment the #define line
in server.cpp.

```
//////////////////////////////////////////////////////////////////////////////////
// Un-comment the following line to allow the user to logon from the same computer
//
//                              #define ALLOW_LOGON_FROM_SAME_COMPUTER
//
//////////////////////////////////////////////////////////////////////////////////
```

This causes the program to report this error, but take no action.
Then, the program processes the client requests as usual.

Note however, that uncommenting this could lead to a security problem under a
different situation.

MSDN says that if the ImpersonateNamedPipeClient function fails, the client is not impersonated,
and all subsequent client requests are made in the security context of the process that called
the function. If the calling process is running as a privileged account, it can perform
actions that the client would not be allowed to perform. To avoid security risks,
the calling process should always check the return value. If the return value indicates that
the function call failed, no client requests should be executed.

Sample Output

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\bushl2.WIN\Desktop>chatclient LISPC-08 larry3
Welcome, LARRY3!
LARRY1: Hello?
Hello.
LARRY1: Hello!
Hello ;-)
LARRY1: Bye?
Bye.
LARRY1: Bye bye.
Bye :-(
QUIT


C:\Documents and Settings\bushl2.WIN\Desktop>
```

_