ECES 338 Assignment #8

Due: April 21, 1999

Spring 99, Ozsoyoglu, G., 100 compulsory points; 100 optional points

In this assignment, you will implement the Printer Daemon Problem (from Assignments 4 and 7) using a socket connection and with/without Posix threads. Assume that four user processes (clients) run on different "client" machines. And, the "server" machine provides allocation/deallocation services for three printers by executing the procedures "GetPrinter" and "ReleasePrinter", requested using a socket connection.

Part I: Connection-Oriented (TCP-IP) Sockets and No Threads--Sequential Server

(*100 points; Compulsory!)*

A sequential server sequentializes the service requests, and serves one request at a time. For connection-oriented socket communication, the socket calls that the client issues are *socket*(), *connect*(), *write*() (to send a request), *read*() (to get the reply), and *close*() (to close the socket connection). The socket calls that the server issues are *socket*(), *bind*(), *listen*(), *accept*(), *read*() (to get the request), *write*() (to send the reply), and *close*() (to close the socket connection).

In this part, you are to create four concurrently running user (client) processes, each running on a different machine. A client process is in a loop "Compute-RequestaPrinter-ReleaseThePrinter", following the procedure listed below:

(a) ComputeSomething;

(b) Request the allocation of a printer using a socket connection;

(c) Wait for the server's response. If the server returns the message "Printer is not allocated"; then sleep some, and repeat (b) again.

(d) (*A printer is allocated*.) ComputeSomething;

(e) Release the printer by informing the server (over a socket connection).

The server's implementation of the requests "GetPrinter" and "ReleasePrinter" is quite straightforward, following the same semantics as in assignment #7.

In this implementation, clients have no priorities, and are not necessarily serviced on a first-come-first-serve basis--thus,a client may starve at steps (b)-(c).

Test your program with a long run (making sure that you have cases in which all four clients request a printer at about the same time), script and turn in the output as well as your source code. Make sure that you start the server (that runs procedures GetPrinter and ReleasePrinter) before starting the clients. Clients should print their actions with the machine name and the time attached.

To start remote processes, you can login to five different machines, and start (first) the server, and (then) the four clients. Or, you can use the remote shell command to start the server and the clients. A shell script to launch the server on the remote host cerne is

```
rsh -n cerne server &
```

Part II: (Connection-Oriented or Connectionless (UDP-IP) Sockets and Posix Threads--Concurrent Server

(*100 Points; Optional!)*

A concurrent server spawns a thread/process for each request. You are to use posix threads to provide concurrent services at the server side. For connectionless socket communication, the socket

calls that the client issues are *socket*(), *bind*(), *sendto*() (to send a request), and *recvfrom*() (to get the reply). The socket calls that the server issues are *socket*(), *bind*(), *recvfrom*() (to get the request), and *sendto*() (to send the reply).

In this part, you will again create four concurrently running user (client) processes, each running on a different machine. A client process is in a loop "Compute-RequestaPrinter-ReleaseThePrinter", as follows:

(a) ComputeSomething;
(b) Request the allocation of a printer using a socket connection;
(c) Wait for the server's allocation of the printer.
(d) (*A printer is allocated*.) ComputeSomething;
(e) Release the printer by informing the server (over a socket connection).

In the case of a connectionless socket implementation, after creating a socket and binding to it (and initializing Posix mutexes and condition variables), the server's implementation of the requests "GetPrinter" and "ReleasePrinter" is as follows:

(a) Block waiting for a request from client (i.e., execute recvfrom());
(b) (A request has now been received) Spawn a child thread (pthread_create() call) to service the request; detach from the spawned thread (pthread_detach() call) and repeat (a);

Please note that now, once the requested service is performed (i.e., the client "gets" a printer, or a printer is "released") the child thread needs to "reply" to the client that the requested service has been performed.

The child thread that services a "GetPrinter" request executes mutually exclusively (implemented by pthread_mutex_lock() and pthread_mutex_unlock() calls):

(a) If a printer is not available, (release mutex variable and) wait on a condition variable (i.e., the pthread_cond_wait() call) until a signal arrives;
(b) Reply to the client that a printer is now available, and exit;

The child thread that services a "ReleasePrinter" request executes mutually exclusively:

(a) Release the printer (i.e., do housekeeping);
(b) Reply to the client that a printer is now released;
(c) Signal a thread waiting on the condition variable (i.e., the pthread_cond_signal() call); exit;

In this implementation, clients have no priorities, and are serviced on a first-come-first-serve basis--thus,a client does not starve.

Test your program with a long run (making sure that you have cases in which all four clients request a printer at about the same time), script and turn in the output as well as your source code. Make sure that you start the server (that runs procedures GetPrinter and ReleasePrinter) before starting the clients. Clients should print their actions with the machine name and the time attached.