Comp 4981 Computer Systems Technology February 2008.

Data Communication Option

Assignment #2

Due: February 12, 0930 hrs. This is an individual assignment.

Objective: To implement a **Client-Server** application using **Message Queues**.

Assignment:

(a). Write and test a C/C++ application that implements a **Client** and a **Server**. The client reads a filename from the standard input and writes it to the IPC channel. The server reads this filename from the IPC channel and tries to open the file for reading. If the server can open the file, it responds by reading the file and writing it to the IPC channel, otherwise the server responds with an ASCII error message. Your implementation must have the following functions (in addition to the main functions):

- Function '**client**' as described above.

- Function '**server**' as described above.

- Two functions to send and receive messages; '**mesg_recv**' and '**mesg_send**'. These two functions will be used by the client and server functions.

(b). The second part of this assignment requires you to implement a client-server with **multiplexed messages** using M**essage Queues**. This is essentially a server with multiple clients.

Consider our simple example of a server process and a single client process. With a message queue, a single queue can be used having the type of each message signify if the message is from the client to the server, or vice versa.

In a server with multiple clients we can use **a type 1**, say, to indicate a message from any **client to the server** and a **type 2** to indicate messages from **server to client**. If the client passes its PID as part of the message, the server can send its messages to the client processes, using the client's PID as the message type. Each client process specifies the **msgtype** argument to **msgrcv** as its PID.

Another feature provided by the type attribute of messages is the ability of the receiver to read the messages in an order other than first-in, first-out. We can, in essence, assign **priorities** to the messages by associating a priority to a type, or a range of types.

Furthermore, we can call ***msgrcv*** with the **IPC_NOWAIT** flag to read any messages of a given type from the queue, but return immediately if there are no messages of the specified type.

Constraints:

- You are required to implement a client-server example using the type 1 and type message convention explained above.

- You are required to implement a priority system as outlined above.

- There are times when a process wants to impose some structure on the data being transferred. This can happen when the data consists of variable-length messages and it is required that the reader know where the message boundaries are so that it knows when a single message has been read.

– More structured messages can also be built, and this is what the UNIX message queue form of IPC does. More structure can also be added to either a pipe or FIFO. Define a message in a '**mesg.h**' header file as:

```
#define MAXMESSAGEDATA (4096-16)  /* don't want sizeof(Mesg) > 4096 */
#define MESGHDRSIZE      (sizeof(Mesg) - MAXMESGDATA)
                         /* length of mesg_len and mesg_type */

typedef struct
{
  int mesg_len;       /* #bytes in mesg_data */
  long mesg_type; /* message type */
  char mesg_data [MAXMESGDATA];
} Mesg;
```

Each message has a type which we define as a long integer whose value must be greater than zero. The purpose of having a type field is to allow multiple processes to multiplex messages into a single queue (used only for the message queue implementation).

We also specify a length for each message, and allow the length to be zero. If we use the stream format for messages, with newlines separating each message, the length is implied by the newline character at the end of each message.

What we are doing with the **Mesg** definition is to precede each message with its length, instead of using newlines to separate the messages. The only advantage this has over the newline separation is if binary messages are being exchanged - with the length preceding the message the actual content of the message does not have to be scanned to find the end.

- It is a requirement that the server create a separate process to service each client request.

- Within each client process you are required to use **at least** one **thread**. You can assign any client task to the thread that you wish.

To be Submitted:

- You are required to hand in complete well documented design work and printed listings for each implementation (two in total).
- Ensure that you clearly explain testing procedures for your programs and provide test programs if necessary.
- It is a requirement that you demonstrate your working program in the lab.
- Your are required to hand in all your code on a CD.

Evaluation

(1). **Design Work**:	/ 20	
(2). **Code Quality**:	/ 10	
(3). **Documentation**/**Testing**:		/ 15
(4). **Working Program**:		
Message Queues:	/ 55	
Total:	/ 100	