Assignment 4: Implementing RPC/RMI

Operating Systems

Goal

- Implement RPC/RMI software to obtain a better understanding of the underlying issues
- RPC
  - C/C++
  - Communication module + client-side & server-side stubs + dispatcher
- RMI
  - Your implementation of Java RMI
  - Communication module + client-side & server-side stubs + dispatcher + Remote Reference module
**RPC**

- **You have to implement**
  1. **Communication module**
     - Implements protocol for providing at most once RPC semantics
     - API used by client & server stubs & dispatcher
       
       ```
       doOperation(InetAddress serverhost, int serverport, 
                   int methodId, byte[] arguments)
       byte[] getRequest();
       void sendReply(byte[] reply, InetAddress clienthost, 
                      int clientport)
       ```
  2. **Client & server stubs**
     - Handcrafted for arithmetic server of Assignment 3
     - Marshall/Unmarshall arguments & results
  3. **Client & server**
     - Already implemented by you for Assignment 3 but you will have to modify your code to work with your client & server stubs

---

**Role of client and server stub procedures in RPC**

![Diagram showing the role of client and server stub procedures in RPC](image-url)
RPC project notes

- Communication module
  - Use Request-Reply-Ack protocol for implementing at most once semantics
  - Use `select` system call for implementing timeouts
  - Your code should "drop messages" to demonstrate the correct working of your RRA protocol
    - Print enough debug messages so that grader can see your protocol at work
- Stubs
  - Convert data types into canonical format and vice versa
    - ` htonl() & ntohl()` for integers
    - Flat array of bytes containing arguments
- Client
  - Server hostname & port are command-line arguments, I.e. no need for a binder/registry

Example: using `select()` for timeouts

```c
#include <sys/time.h>
/* use select to test whether there is any input on descriptor s */

int anyThingThere(int s) {
    unsigned long read_mask;
    struct timeval timeout;
    int n;

    timeout.tv_sec = 10; /*seconds wait*/
    timeout.tv_usec = 0; /* micro seconds*/
    read_mask = (1<<s);
    if((n = select(32, (fd_set *)&read_mask, 0, 0, &timeout))<0)
        perror("Select fail:\n");
    else printf("n = %d\n");
    return n;
}
```
**RMI project**

- You have to implement
  1. Communication module
      - Use TCP streams as transport
  2. Client stub class
  3. Server skeleton class
  4. Generic server & Dispatcher
  5. Remote Reference Module
      - Maps Remote Object References to local references
  6. Client
      - Already implemented for Assignment 3, but will need to be changed slightly to work with Registry (provided to you) and your RMI implementation

Outline of code for classes implementing items 4 and 5 are given to you

---

**The role of proxy and skeleton in RMI**

![Diagram of RMI role of proxy and skeleton](image)

- Client
  - object A proxy for B
  - Remote reference module
  - Communication module

- Server
  - Remote object B
  - Request
  - Reply
  - Remote reference module
  - Communication module
  - skeleton & dispatcher for B’s class
Remote Reference Module

- **Class RemoteObjRef**
  - Implement Remote Object References (ROR)
  - Two methods
    - Constructor
    - localise()
      - Called by client to create a proxy for remote object whose reference was obtained from Registry

- **Class RORtable**
  - maintains a mapping between RORs and objects in the server process

---

Generic Server (**yourRMI**)

- Pass name of class implementing remote object ("servant") as command line argument
- Creates classes corresponding to servant and skeleton
- Creates instance of servant, creates ROR, adds object to RORtable, registers ROR with registry
- Code for communication module & generic dispatcher
**RMI project notes**

- **Marshalling/Unmarshalling**
  - Can use Java serialization for flattening arguments
  - Use Java serialization API

- **Generic server**
  - Use classes in java.lang package for generic server & dispatcher (see code outline provided to you)
  - Can use Java reflection for creating generic skeleton (but don't need to do this, i.e. you can have a specific skeleton)