

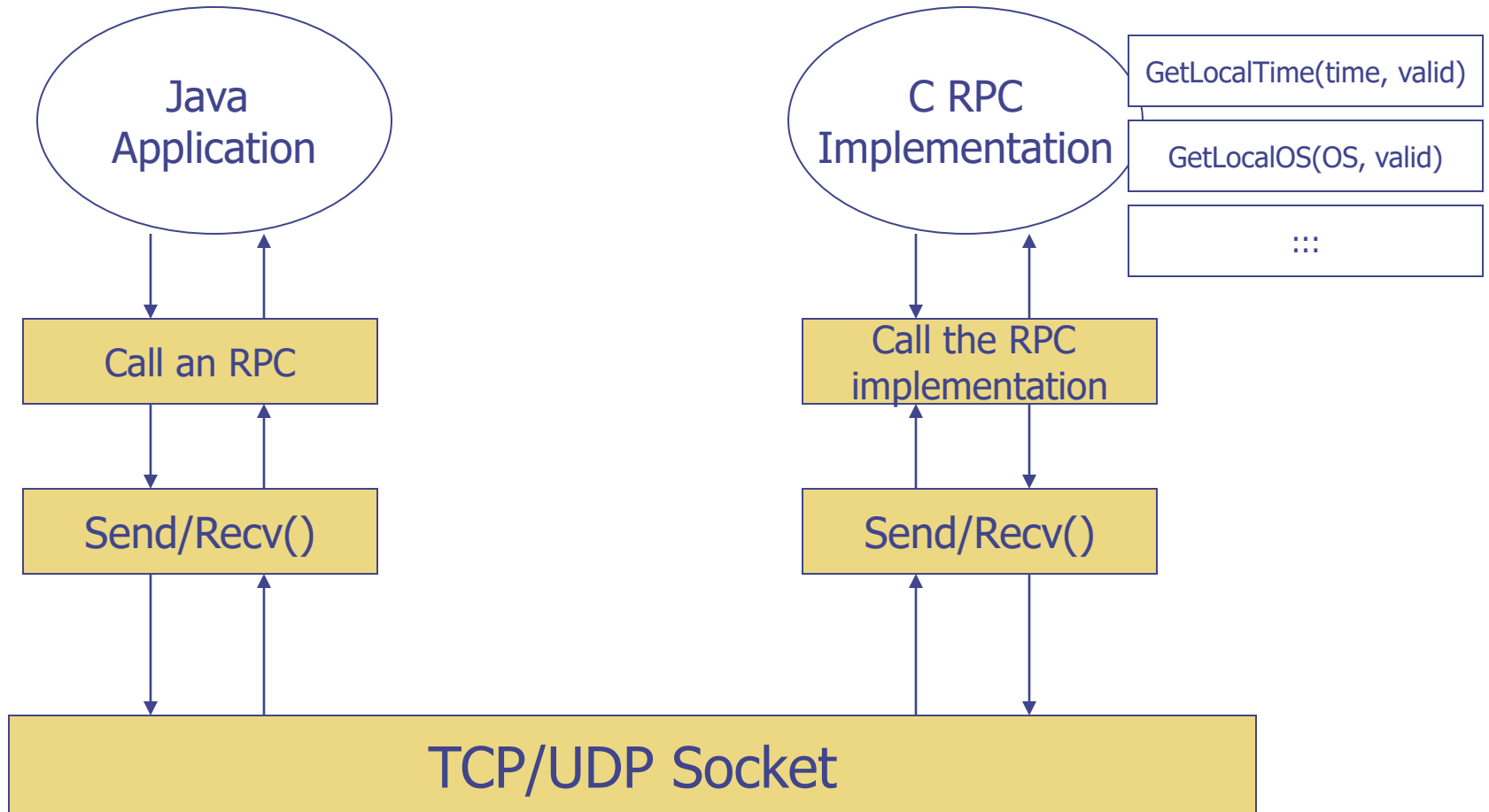
Remote Procedure Calls

CS587x Lecture
Department of Computer Science
Iowa State University

Remote Procedure Call

- What is RPC for?
 - Allowing programs to call procedures located on another machine transparently
- Scope of use
 - Distributed computing
 - ◆ Task and data partitioned environments
 - ◆ Task distribution
 - Front-end load-balances across functional back ends
 - Services
 - ◆ Client-server model
 - ◆ Mail servers, databases (transaction servers)

Java-to-C (J2C) RPC



Interface Design

- C Interface
 - How to call its C implementation?
- Java Interface
 - How to represent a C function in Java
 - How to set inputs
 - How to execute
 - How to get outputs

C Interface Design

- Every C function is implemented as `void CmdXYZ(char *buffer)`, where the interpretation of `buffer` is determined by `CmdXYZ`

```
typedef struct
{
    int    time;
    char   valid;
} GET_LOCAL_TIME;

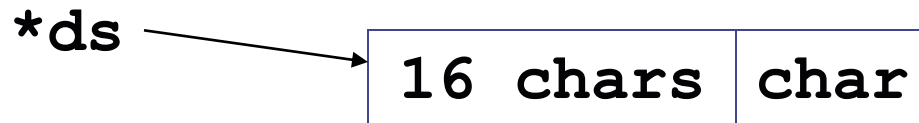
void GetLocalTime(GET_LOCAL_TIME *ds);
```



GetLocalOS(char *buffer)

```
typedef struct
{
    char  OS[16];
    char  valid;
} GET_LOCAL_OS;

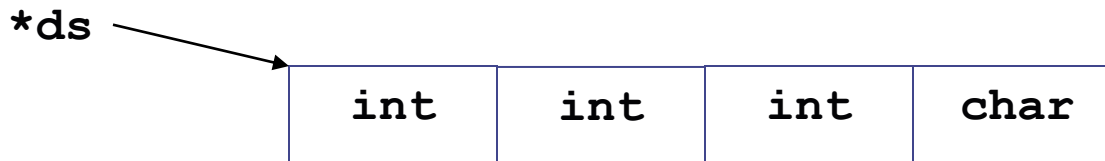
void GetLocalOS(GET_LOCAL_OS *ds) ;
```



GetDiskData (char *buffer)

```
typedef struct {  
    int DiskNumber;  
    int Cylinder;  
    int Sector;  
    char Status;  
} GET_DISK_DATA;
```

```
void GetDiskData(GET_DISK_DATA *ds) ;
```



Java Interface Design

- Each C function has a corresponding class

```
class GetLocaltime();
```

- Steps of making an RPC

1. Instantiate an RPC object

```
obj = new GetLocalTime();
```

2. Set inputs

```
obj.valid.setValue(FALSE);
```

3. Execute

```
obj.execute(IP, PORT);
```

4. Get outputs

```
int t = obj.time.getValue();
```


RPC Class of GetLocalTime()

```
class GetLocalTime {
    c_int    time;
    c_char    valid;

    public int execute(string IP, int port);
}

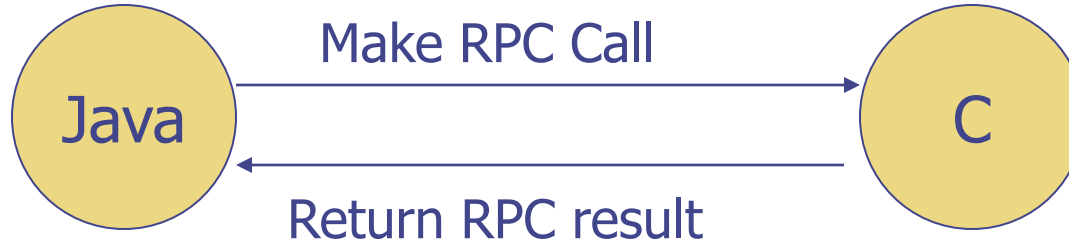
// the representation of a c integer in java
class c_int {
    byte[] buf = new byte[4]; // little endian

    public int getSize(); // the size of buf
    public int getValue(); // the int value represented by buf
    public void setValue(byte[] b); // copy the value in b into buf
    public void setValue(int v); // set buf according to v
    public byte[] toByte(); // return buf
}

// the representation of a C char (and other c types )in java {...}
```

Implementation of execute()

- Communication protocol



CmdID	CmdLength	CmdBuf
-------	-----------	--------

Header

- CmdID (100 bytes): the command ID
 - CmdLength (4 bytes): the length of CmdBuf
- CmdBuf (dynamic): the parameters to the command

Implementation of execute()

- Create a binary buffer

1. `int length = time.getsize()+valid.getsize();`
2. `byte[] buf = new byte[100+4+length];`

- Marshall parameters into the buffer

1. `buf[0, 99] = "GetLocalTime"; offset = 100;`
2. `buf[100, 103] = length; offset^^;`
3. `buf[offset, time.getsize()] = time.toByte(); offset^^`
4. `buf[offset, valid.getsize()] = valid.toByte();`

- Send/receive the buffer to/from the RPC server

1. `s = CreateSocket(IP, port);`
2. `SendPacket(s, buf, buf.length());`
3. `RecvPacket(s, buf, buf.length());`

- Set parameters according to the buffer

1. `time.setValue(buf, 104);`
2. `valid.setValue(buf, 104+time.getsize());`

C Implementation

- Wait for a connection
- When a connection arrived, launch a thread to process a command as follows:
- Receive a command
 1. `header = new byte[104]`
 2. `RecvPacket(header, 104);`
 3. `if (header[0-99] is NOT a valid command)exit;`
 4. `length = header[100-103];`
 5. `buf = new byte[length];`
 6. `RecvPacket(s, buf, length);`
- Execute the command
 1. `switch header[0-99] of`
 2. `case "GetLocalTime":`
 3. `{`
 4. `GetLocalTime(buf);`
 5. `break;`
 6. `}`
 7. `Case ".....":`
- Send the command back
 1. `SendPacket(s, header+buf, 104+length);`

What to submit (Java code)

- c_int.java
- c_char.java
- GetLocalTime.java:
- GetLocalIOS.java
- Test.java

```
//Test.java
main()
{
    //testing GetLocalTime
    obj = new GetLocalTime();
    obj.valid.setValue(FALSE);
    obj.execute(IP, PORT);
    int t = obj.time.getValue();
    //print out t and valid

    //testing GetLocalIOS
    :::
}
```

What to submit (C code)

- **Server.c**
 1. **Main**
 - Wait for socket connection
 - Upon receiving a connection request, launch a thread CmdProcessor to handle the request
 2. **CmdProcessor**
 - Receive CmdID (100 bytes)
 - Receive CmdLength (4 bytes)
 - Receive the CmdBuffer, the size of which is specified by CmdLength
 - Call the corresponding C function and update the data in CmdBuffer
 - Send CmdID, CmdLength, and CmdBuffer back

Think Further!!!

- A new command needs to be added?
- An existing command needs to be deleted?
- Some parameters to a command need to be changed?
 - Add a new field
 - Delete an existing field
 - Change the type of an existing field