APPLICATION PROTOCOL DESIGN

Protocol

- A protocol defines:
  - Message formats
  - Message sequences in communication
  - How to process a message
- Goals
  - Everyone must know
  - Everyone must agree
  - Unambiguous
  - Complete
- Questions are raised while designing an application protocol:
  - Is it to be stateful vs stateless?
  - Is the transport protocol reliable or unreliable?
  - Are replies needed?
  - Is it to be broadcast, multicast or unicast?
  - Are there multiple connections?

Example: POP session

```
C: <client connects to service port 110>
S: +OK POP3 server ready <1896.6971@mailgate.dobbs.org>
C: USER bob
S: +OK bob
C: PASS redqueen
S: +OK bob's maildrop has 2 messages (320 octets)
C: LIST
S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S: .
C: QUIT
S: +OK dewey POP3 server signing off (maildrop empty)
C: <client hangs u>
```

Example: FTP authentication

```
>   ftp 202.191.56.65
C: Connected to 202.91.56.65
S: 220 Servers identifying string
User: tungbt (C: USER tungbt)
S: 331 Password required for tungbt
Password:(C: PASS)
S: 530 Login incorrect
C: ls
S: 530 Please login with USER and PASS
C: USER tungbt
S: 331 Password required for tungbt
Password:(C: PASS)
S: 230 User tungbt logged in
```

Message format

- Two pieces of data
  - Header: contain message type, describing what type of data in payload
    - Distinguish different type messages.
  - Payload
    - Data
- Message type
  - Short and descriptive type
  - SHOULD has fix length
    - So we can parse the message and understand its type easily
  - Example 1: see POP session

Data Format of messages

- In byte format
  - The first part of the message is typically a byte to distinguish between message types.
  - Further bytes in the message would contain message content according to a pre-defined format
  - Advantages: compactness
  - Disadvantages: harder to process
  - Example: IP message (but IP is not application protocol)

Data Format of messages

- In character format
  - A message is a sequence of one or more lines
  - The start of the first line of the message is typically a word that represents the message type.
  - The rest of the first line and successive lines contain the data.
- Ex: HTTP message

Example: HTTP request

request line (GET, POST, HEAD commands)

header lines

```
GET /dccn/index.html HTTP/1.1
Host: www.it-hut.edu.vn
User-agent: Mozilla/4.0
Connection: close
Accept-language:en-us
```

CR, LF

(extra carriage return, line feed)

indicates end of message

Example: HTTP response

status line (protocol status code status phrase)

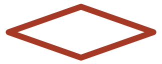
```
HTTP/1.1 200 OK
Connection close
Date: Tue, 16 Mar 2008 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 15 Mar 2008 …...
Content-Length: 8990
Content-Type: text/html

data data data data data ...
```

header lines

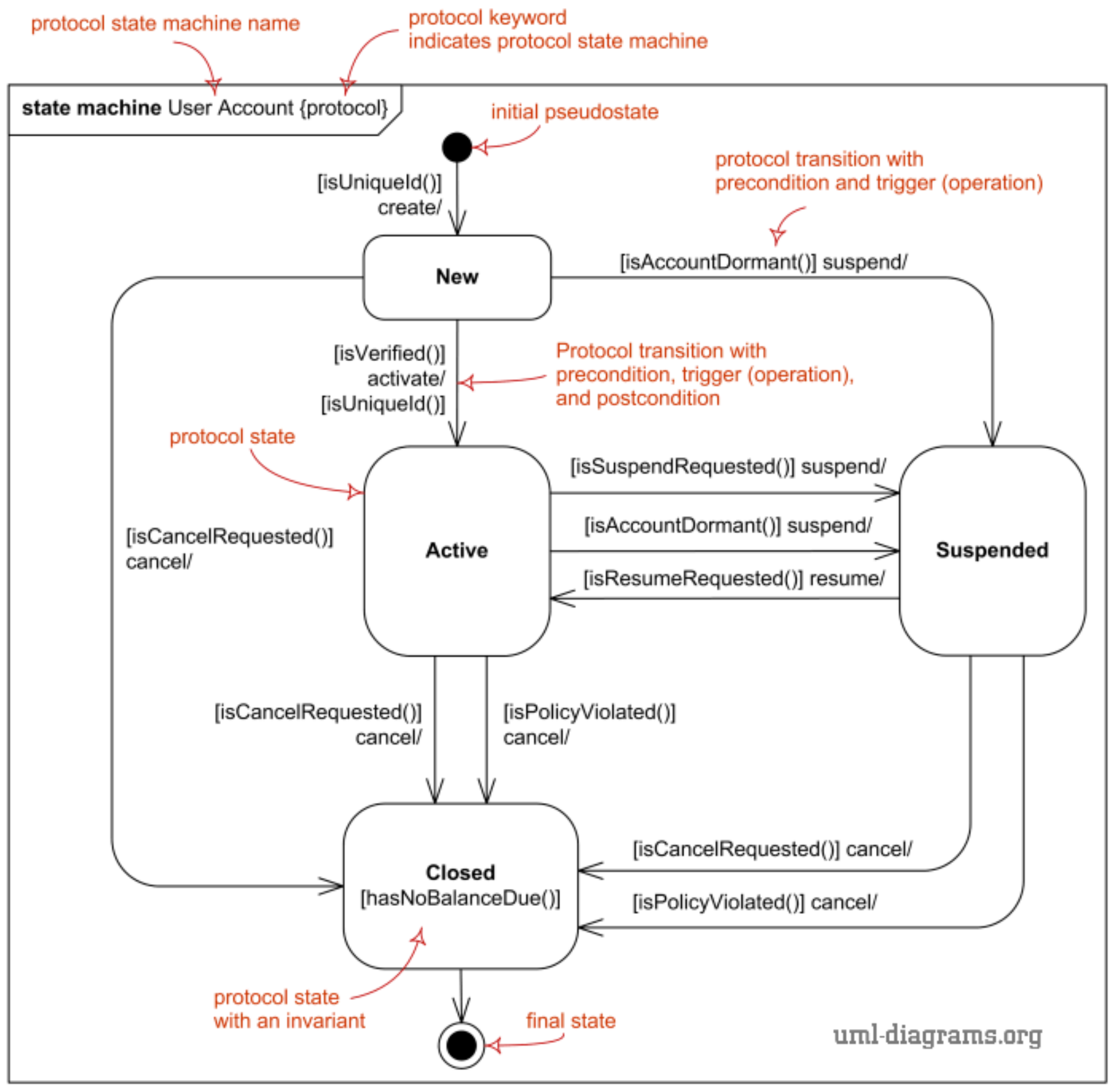
data, e.g., requested HTML file

Define message sequence using UML Protocol State Machine Diagram

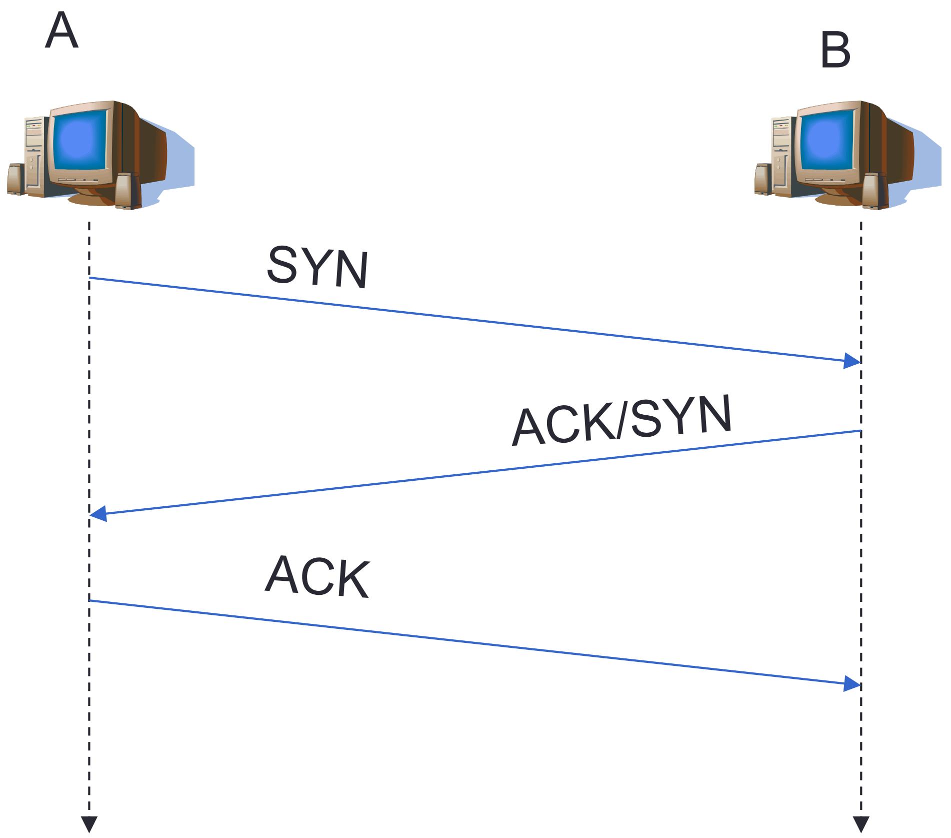
- State:
- Transaction: $\xrightarrow{\text{Trigger[Precondition]/[Effect]}}$
- Choose: 
- State Table

Current state	**Transaction**		**Next state**
	Receive	**Send**	

Define message sequence using UML Protocol State Machine Diagram


protocol state machine name
protocol keyword indicates protocol state machine
state machine User Account {protocol}
initial pseudostate
[isUniqueId()] create/
protocol transition with precondition and trigger (operation)
New
[isAccountDormant()] suspend/
[isVerified()] activate/ [isUniqueId()]
Protocol transition with precondition, trigger (operation), and postcondition
protocol state
[isCancelRequested()] cancel/
Active
[isSuspendRequested()] suspend/
[isAccountDormant()] suspend/
[isResumeRequested()] resume/
Suspended
[isCancelRequested()] cancel/
[isPolicyViolated()] cancel/
Closed [hasNoBalanceDue()]
[isCancelRequested()] cancel/
[isPolicyViolated()] cancel/
protocol state with an invariant
final state
uml-diagrams.org

Define message sequence using sequent diagram-Message flow



Protocol description by usecase

- Describe what happen on client side/server side for each usecase
  - Message to be sent
  - Message to be receive
  - How to handle message.
- Ex:
  - Usecase login:
    - What is the message client should send to server
    - What are possible reply message from server to client
    - What server and client should do when receiving one of above messages.

Suggested steps

- Define usecases
- Define message format and list of messages with detailed fields
- For each usecase:
  - Define message sequent diagram
  - Define corresponding message handling and processing at client/server

Exercise 1

- Define protocole for the application Monolog chat. The “monolog” chat works as follows:
  - Client can send to server either login name or some messages.
  - When client sends login then server accepts and remembers his login
  - When client sends to server a text message, server saves the message into a log file. Each client has a separate log file.

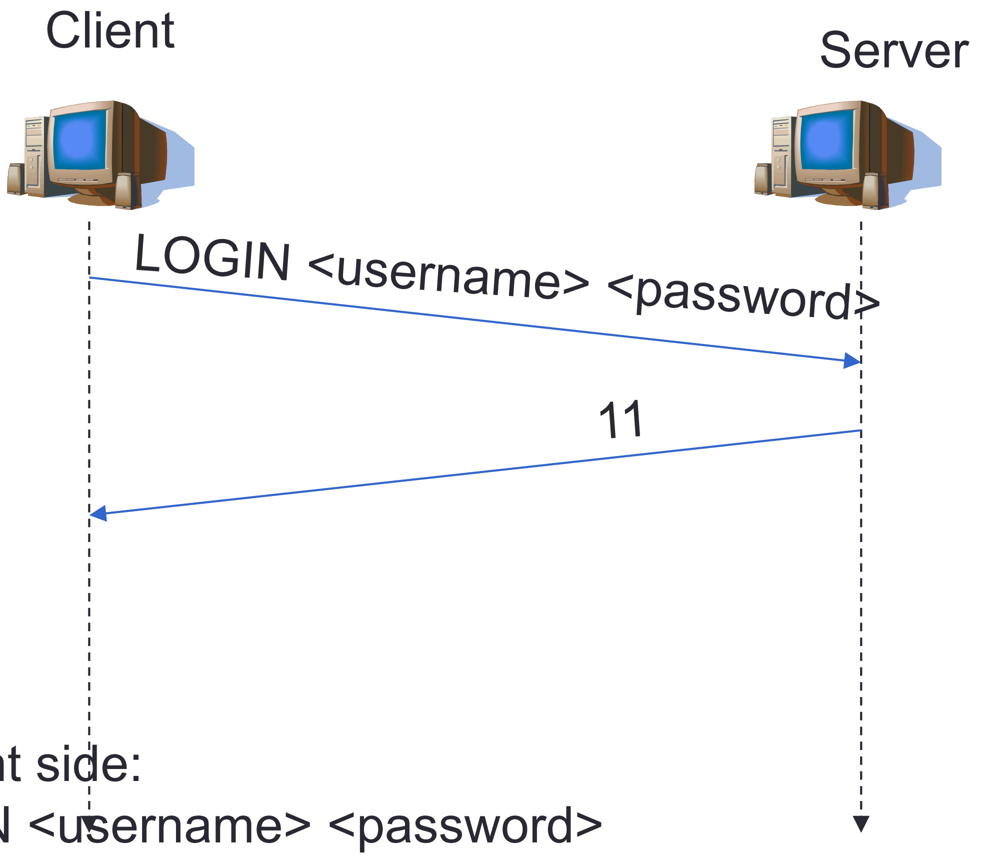
- Hints:
  - Need a message type for login, another one for text message
  - Define fields of each messages and their length/type.
  - Define how client/server processes each message.
  - Draw protocol state machine.
  - Then now you can write code!!!

Exercise 2

- Create a UDP/TCP Client/Server “dialog” chat program that work as follows:
  - Client can send to server either login name or text message
  - When client sends login then server accepts and remembers his login
  - When client sends server a text message for transfering to another client , server sends this message to the client. Server should tell also the receiver that from whom the message come.
- Hints:
  - Started from the protocol of exercice 1 and define new message formats

- Usecase: Login, Send text
- Login:
  - From Client- to server: LOGIN <username> <password>
  - From Server-Client:
    - LOGIN 1: means OK. -→ 11
    - LOGIN 2: Not ok. -->12
    - LOGIN 3: already login. -→ 13
- Send text:
  - From client- server: TEXT <dest-username> <Content>
  - From server-client: - TEXT 1: successful: 2 client logged, message delivered. → 21
    - - TEXT 2: user sending message not logged in → 22
    - - 3: dest-user not logged in (live chat). →23
    - - 4: dest-user does not existed (if you accept offline message) →24

- Login success



Processing on Client side:
- Client send LOGIN <username> <password>

Processing on Server side:
- Verify if username and password match according server database
- If match: send message: 11

Exercise 3

- Refine Student Schedule Management application.
- What was the application protocol using between client and server?
  - Usecase: Login, Request for single day schedule, Request for all week schedule, Quit.
  - Messages
  - Procedure for each functionality (usecase)
- Working in pair with your classmates.
  - Define messages and procedure for each functionality.
  - Revise the server and client accordingly. You revise server and your partner revise the client.
  - Server and client should print on terminal the messages they receive and send.
  - Test the application.

Exercise- sending complex data

- Reuse Echo server and Echo client
- Sending complex data from client to server
  - Send a struct
    - struct{
      - char username[];
      - char password[];
      - int count;

      }
  - Send an array
  - See if server receive well data when client and server are running on two machines

Project

- Take a project from the given project list
- Propose your own project→ get it validated by lecturer
- Requirement:
  - Client and Server use programming languages C, C++ or Java.
  - Have to handle sockets by yourself
  - Define your application protocol by yourself
  - Client and server must run on two separate PCs.

Working plan

- Week 1:
  - Team identification: students/team.
  - Topic identification: Pick a topic or propose an application over internet that you want to develop. (3 slides)
  - Topic Review with lecturer
- Week 21/10
  - Design review, by presentation (for half number of teams with smallest topics ID)
    - Application introduction
    - Architecture of the application: client/server, P2P, hybrid
    - Functionality
    - Protocol design (very important): message, state machine, message processing
    - ➔ get feedback and revise
- Week 28/10:
  - Design review for remaining teams

Working plan

- Weeks:
  - Progress update, by presentation
- Final test: Final presentation.
  - Brief introduction with slide about the application, the design
  - Demo
  - Each team has to submit a report (hard copy): including design and application evaluation.

Design presentation

- Application description
- Game rule (if any)
- Application architecture (figure)
- Functionality
  - Use case
- Working procedure for each functionality
  - Communication diagram between client/servers, or between clients
- Message design
  - Message formats
  - Message sequences in communication
  - How to process a message