University of Guelph, CIS*3210 Yuhang Xu
Request for Comments: 32108 SEP.30.2019
Category: Informational

Online Book Catalogue -- Protocol 1.0

Abstract

The Online Book Catalogue (OBC) is a console based protocol application that allows multiple users to transmit book data to server. The protocol was designed in traditional client-server form. In which client and server are both console applications and It would accept command line argument as input. The protocol was designed to be robusted, it resolved the problem of synchronization which could prevent crashes and overloading. The OBC protocol is also capable of resolving some edge cases which improves the usability.

Table of Contents

Introduction

1.1-Purpose

This protocol is a client-server application designed using TCP socket in C. The purpose of the protocol is to allow user to effective manage book information without doing tedious work. The protocol would be easy to use and would effectively solve the issues of Process Synchronization and bad argument.

1.2-Terminology

Connection: A transport layer virtual circuit established between two application programs for the purpose of communication.

Message: The basic unit of OBC communication. A message must be specified in order for client and server to know what action to perform.

Client: An application program that establishes connections for the purpose of sending requests.

Server: An application program that accepts connections in order to service requests by sending back responses.

Request: An OBC request message

Response: An OBC response message

Description of protocol

OBC messages consist of request from client to server and responses from server to client. In this protocol, the client would be responsible for requesting a connection with the server and send three types of messages(SUBMIT, GET, REMOVE) through the connection. The server should accept the connection from one user or multiple user and builds a linked list structure to hold the data.

Client---------------Request--------------------->Server

Client ←----------------Response----------------Server

<u>The three types of Messages</u>

1. SUBMIT messages containing book description
2. GET messages containing requests for a particular book
3. REMOVE messages containing requests to remove a book from catalogue

<u>Basic Structure of Request and Response</u>

OBC-message = Request

| Response

A book description would have the following components: TITLE, AUTHOR, LOCATION.

Request = Method

| TITLE-Header

| AUTHOR-Header

| LOCATION-Header

LF

Response = Status-Line

Status-Line = Status-Code SP (Search_Result) LF

Format of Client and Server messages

<u>3.1-Request:</u>

A request message from a client to a server includes, within the first line of that message, indicating the ways of how a client can use the online catalogue. For example, you can Submit a new book Check the availability of a book Remove a book from catalogue

Method

The method is case-sensitive.

```
                    Methods = "SUBMIT"
                               |   "GET"
                               |  "REMOVE"
```

The listed method are acceptable. If there are unspecified resources, the server side would return 501, indicating that a method is not implemented or unrecognized.

Method Definition(MD)

MD.1, SUBMIT

The SUBMIT method accepts book descriptions and makes a request to the destination server. A valid input of TITLE, AUTHOR and LOCATION is required in the transmitting process.

Example:

```
SUBMIT
TITLE Intro to R
AUTHOR writerJoe
LOCATION USA
```

MD.2, GET

The GET method means retrieve whatever information is given. It should accept headers like TITLE or AUTHOR as search condition and make a request to the end server.

Example:

```
GET
TITLE Intro to R          // expected to receive records of all books in catalogue called "Intro to R"
```

MD.3, REMOVE

The REMOVE method removes book information based on the information given.

Example:

```
REMOVE
TITLE Intro to R
AUTHOR writerJoe
LOCATION USA
```

Header field Definition(HD)

HD.1, TITLE

TITLE is the name of the book, the client must enter the name of the book in order for transmission process to start. The title should be a valid string input.

HD.2, AUTHOR

AUTHOR is the author of the book, the client must enter the author of the book in order for transmission process to start. If author name is unknown, then user should input "unknown" in AUTHOR. author should be a valid string input.

HD.3, LOCATION

LOCATION is where the book was originally published, the client must enter the book location information in order for transmission process to start. If location is unknown, then user should input "unknown" in Location. Location should be a valid string input.

3.2-Response

Eg1:**Request Message:**

```
SUBMIT
TITLE Intro to R
AUTHOR writerJoe
LOCATION USA
```

Example response Message:

```
202 OK (Book submitted correctly)
Date: Sep/30/2019 12:30pm    // System date(Today, right now)
```

Eg2: **Request Message:**

```
GET
TITLE Intro to R // expected to receive records of all books in catalogue called "Intro to R"
```

Example response Message:

```
204 OK (Searched book is available and the search result is returned)

Book Title: Intro to R
Book Author: writerJoe
Publish Location: USA
Date: Sep/30/2019 12:30pm    // Data entry time

Book Title: Intro to R(Intermediate)
Book Author: writerJoe
```

A “SUCCESS” response message would be just status code with date and time. However, when the GET command is used, it would print out the search result. One thing to keep in mind is that an error code return would not display the date and time. It could be simply like 400 Bad Request.

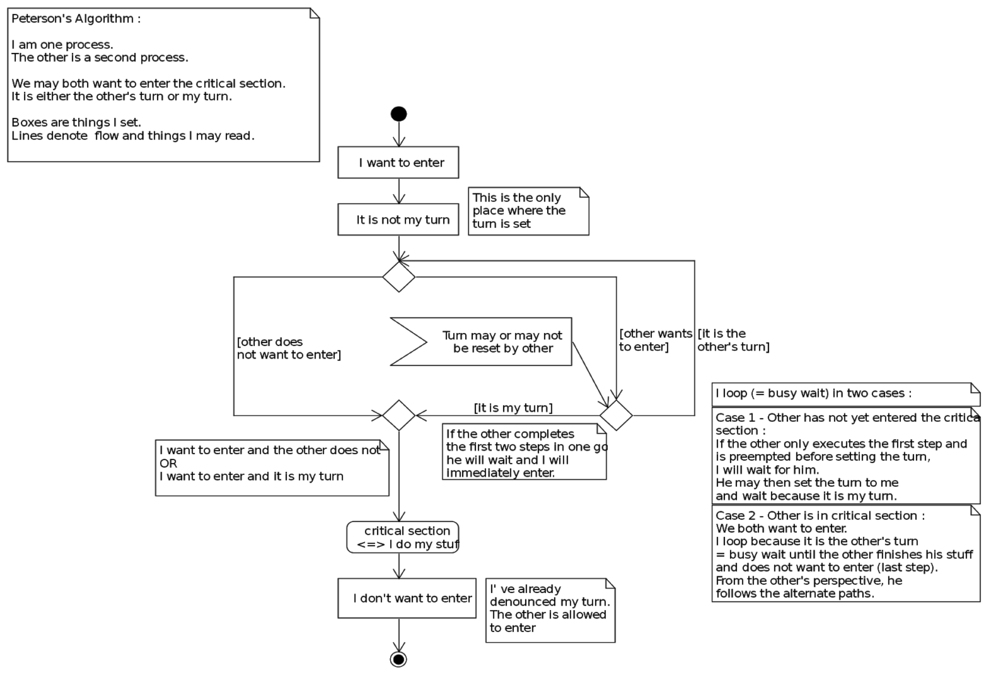
Status Code Definition

```
        Status-Code =
                        “202”       //OK (Book submitted correctly)
                    |   “204”       //OK (Searched book is available and the search result is
    returned)
                    |   “206”       //OK ( Book deleted)
                    |   “208”       //No content
                    |   “400”       //Bad request
                    |   “401”       //Unauthorized
                    |   “403”       //Forbidden
                    |   “500”       //Internal server error
                    |   “501”       //Unrecognized or not implemented
                    |   “503”       //Service unavailable
```

A status code of 2** indicate that the request was executed successfully, the status code of 4** indicate that the request failed with client side error. The status code of 5** indicates that the server side error.

Synchronization Policies

Synchronization is a major issue when multiple clients want to connect to an online book catalogue server for book updating and search. This is a mutex problem and requires an algorithm to solve it. In OBC, we implemented the peterson’s algorithm to solve the issue. The idea of the algorithm is that it allows two or more processes to share a single-use resource. However they can never be in the same section at the same time. The logic of the algorithm is outlined below:


Peterson's Algorithm :
I am one process.
The other is a second process.
We may both want to enter the critical section.
It is either the other's turn or my turn.
Boxes are things I set.
Lines denote flow and things I may read.
I want to enter
It is not my turn
This is the only place where the turn is set
[other does not want to enter]
Turn may or may not be reset by other
[other wants to enter]
[it is the other's turn]
[it is my turn]
I loop (= busy wait) in two cases :
Case 1 - Other has not yet entered the critica section :
If the other only executes the first step and is preempted before setting the turn,
I will wait for him.
He may then set the turn to me and wait because it is my turn.
Case 2 - Other is in critical section :
We both want to enter.
I loop because it is the other's turn
= busy wait until the other finishes his stuff and does not want to enter (last step).
From the other's perspective, he follows the alternate paths.
If the other completes the first two steps in one go he will wait and I will immediately enter.
I want to enter and the other does not
OR
I want to enter and it is my turn
critical section <=> I do my stuf
I don't want to enter
I' ve already denounced my turn. The other is allowed to enter

Reaction of Client and server to errors

When an error occurs, the protocol should specify the error and return the correct error code to client. It should also provide the reason of the failure.There are two types of error code in status code that used to represent client error and server error.

Error Code Description

Client Error------------- 4xx

A client error indicate that the request made by clients contains bad syntax and request cannot be completed by the information given.

400 Bad request:

The request could not be understood by the server due to bad syntax.

401 Unauthorized:

The request requires user authentication, User doesn’t have the right privilege to access the data or modify it.

403 Forbidden:

The request has been rejected by the server. It does not need to provide a reason why it was rejected.

Server Error------------- 5xx

A Server error indicates that the server has failed to respond to a valid request("at least it should seem valid").

500 Internal server error:
The server encountered an unexpected error which prevented it from responding to the client.

501 Unrecognized or not implemented:
The request made by the user is not in the design of the protocol. Therefore server side would not execute such command.

503 Service unavailable:
The server is currently unable to handle the request. It may be due to high numbers of accessing, overloading or server maintenance.

Border-Cases Behaviour

Currently, the border-cases behaviour includes duplicated data and incomplete request. The protocol have designed solutions to handle them.

1, For duplicate data, the protocol would inform the user that what they are trying to submit is already in the Server and cannot be submitted.

2, An incomplete request should receive an error code (see Error Code Description). If client does not enter a valid book information, for example blank TITLE, the protocol would catch that error and ask client to correct the invalid field.

Reference

https://en.wikipedia.org/wiki/Peterson%27s_algorithm#/media/File:Peterson's_Algorithm.svg

https://tools.ietf.org/html/rfc1945