



MODULAR PROGRAMME ASSESSMENT SPECIFICATION

Module Details

Module Code UFCEHX-20-2	Run 09SEP/1 AY	Module Title Computer Networks and O/S
Module Leader Ian Johnson	Module Tutors Ian Johnson, John Counsell, Laurence O'Brien	
Component and Element Number B2		Weighting: (% of the Module's assessment) 25%
Element Description Coursework - 2		<u>Total Assignment time</u> 12 hours + lab time

Dates

Date Issued to Students To be supplied	Date to be Returned to Students 21st May 2010
Submission Place PROJECT ROOM - 2Q30 (Help Desk open 9.00 - 6.00pm)	Submission Date 22nd April 2010
	Submission Time 2.00 pm

Deliverables

As per attached specification

Module Leader Signature

Ian Johnson

Section 1 MUST be completed Individually.

Section 2 MAY be done in Pairs.

Where work is done in pairs then a single hand-in containing both solo and collaborative work should be submitted with a group cover sheet showing both student IDs.

The quality of the documentation **SHALL** be taken into consideration for this assignment. As you will be providing documentation that is to support and clarify the design of a network protocol it is important that it is well laid out, spell-checked and that **ALL** non-original diagrams or other material **MUST** be correctly referenced.

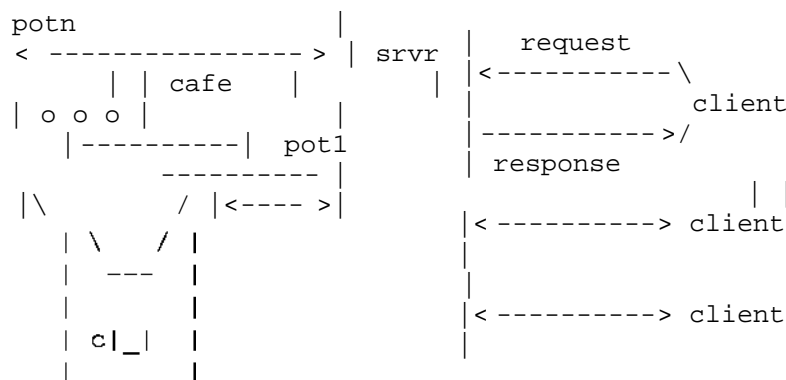
The following criteria **MUST** be adhered to

- Font size 12
- Student ID(s) on each page (as a header or footer)
- Page number (header or footer)
- Title page to include module title and student number(s)
- All sign-off sheets to be included as an appendix.
- All diagrams and tables to be titled

Requirements :

The design and development of an RFC2324 compliant Coffee Pot Server.

Most modern applications include network or Internet connectivity. In order to gain a better understanding of the role of an operating system and networking software in supporting such applications, you are to study the design and implementation of a standards compliant server and a client with which to test the server. The client and server will support HTCPCP, the Hyper-Text Coffee-Pot Control Protocol. This protocol is designed to allow the remote control of coffee brewing machines via the internet. An HTCPCP server accepts requests from remote clients and uses these requests to manage the coffee machine. The server provides responses based on the status of the coffee brewing machine(s). HTCPCP is a modified subset of HTTP



The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in RFC 2119

You **MUST** also understand the meaning of the term "**DEPRECATED**".

Deliverables:

Solo 60%

- 0) A signed and dated form for the worksheets 1 - 3. Worksheet 1 = 5%, worksheet 2 = 10% worksheet 3 = 10%. (total 25%)
- 1) A short (< 1000 words) description in your own words explaining the Coffee Pot Control Protocol. This should include the corrected BNF for your protocol. Any changes that you wish to make to the protocol must be clearly identified along with the justification for the changes. A 'cut and paste' from the original RFC is NOT acceptable. 25%. It will require a very well presented and reasoned document to obtain full marks in this section.
- 2) Signed documentation describing the messages and responses of your system included correctly formed examples. 10%

Pairs 40%

The emphasis may be on design or on coding. Excellent design and weak code will gain as many marks as weak design and excellent [code. ie](#) strength in one will compensate for weakness in the other.

- 3) Your design(s) and/or your code for the client and the server. This must be signed and dated by your lab tutor to whom it must have been demonstrated/explained successfully. The lab tutor **MUST** indicate the degree to which s/he considers it to have met the requirements. 0% to 40%

You should demonstrate, or explain in the case of a design, the following stages as soon as they are completed:-

- i) A stand-alone client with user input from the keyboard and output to the screen.
- ii) A stand-alone server with input from either the keyboard or a file and output to the screen or to a file.
- iii) The combination of your client and server. The server should print out the request message and the client should print out the response message.

Note that the overall quality of the documentation that you hand in will affect the mark allocated. Spelling, layout, code comments, references etc are all important.

Any failure to comply with the requirements for presentation or sign-off as specified in this assignment **SHALL** result in a reduction of marks.

Section 1 (solo) :-

In order to complete this assignment you will need to :-

- A) Complete the network worksheets 1, 2 & 3. These **MUST be completed and SIGNED OFF** within the first 4 lab sessions of semester 2. These will provide the technical underpinning for the high level protocol that you will develop in the rest of the assignment. The worksheets can be found via links on the module web page http://www.cems.uwe.ac.uk/ngunton/net_os/cnos.html
- B) **Understand** the role of BNF¹ in defining a grammar and UML State Charts in defining and designing network protocols. An explanation of the BNF used is to be found in RFC 2616 section 2. There are numerous explanations of State Charts/diagrams on the web. Two good starting points are
- <http://en.wikipedia.org/wiki/Statechart> http://www.ertin.com/pr_s_tate_diagrams.html
- C) **Read** RFC2324. This provides an introduction to RFCs (Request For Comments). RFCs are the standards documents of the Internet, all key protocols are described and defined in them. Networking software is required to comply with the defined standards if it is to interact correctly with other software using the same protocols. Note that, as with all protocols during their development phase, there are some ambiguities in RFC2324. It is up to you to recognize and interpret these ambiguities and omissions in such a way that reflects your understanding of the intent of RFC2324. This should be documented as the first part of the deliverables. Credit will be given for the quality of decisions made at this stage.
- D) **Study** the BNF description given at the end of this document along with the written descriptions shown in RFC2324. As a start you should consider :
- The difference between a `URL` and a `URI`
 - The structure of a `request-message` and a `response-message`
 - What goes in the `method` and `URL` fields of the request message?
 - How to request additions to your coffee?
 - How to find out what additions are available?
 - How the server should respond to the previous two questions?
- E) **Clarify and Correct** the BNF in RFC2324 and then develop a set of typical request/response exchanges. Decide which of this set of messages and responses should be generated by the client and which by the server. Your decisions and any modifications should be documented and again credit will be given for the standard of your documentation. You **MUST** get this phase completed by week 31 and approved (signed off) by your lab tutor. You **SHOULD NOT** develop the design or the code without understanding the structure of the messages.

¹Backus-Naur Form http://en.wikipedia.org/wiki/Backus-Naur_Form

Section 2 (in pairs or solo)

- F) **Develop** a full design for both client-side and server-side parsing. You will be expected to use a recognized design methodology such as those provided within UML. It is recommended that you consider designing both the server and the client as state machines, (statechart diagrams in UML). Credit will be given for the quality of the design and can be offset against implementation, although at least a basic implementation of the client will be expected. Your design **MUST** clearly show how you under-take the parsing of messages and user input. Note that the most complex areas are
- Parsing user input in the client and constructing a valid request.
 - Parsing the request message in the server and building an appropriate response.
- G) You **MAY** implement the design using a language and platform of your choice but you are advised that lab support might not be available for languages or platforms other than for C and GNU/Linux. You **MAY** use the code examples provided as a starting point, it will be up to you to decide which of the examples are the most appropriate. There are additional examples available via the links on Ian Johnsons web-pages ... <http://www.cems.uwe.ac.uk/~irjohnso> . Note that these pages are only available on-site. The majority of examples are in C. If you use code from other sources then it **MUST** be clearly identified as such and you **MUST** be able to explain its functionality to your lab tutor. You **MUST** get your lab tutor to initial the sign-off sheet to confirm that you have explained any derived code adequately.
Note that in the absence of a coffee machine you will have to simulate it by either
- a) implementing variables in the server to represent the coffee machine status
or
 - b) reading and writing to a file holding the coffee machine status
or
 - c) implementing a coffee machine which is controlled by the server

NOTE:

Your coffee-pot server is not required to provide all additions but must recognize the additions list and respond with an appropriate message.

```

coffee-url = "coffee" ":" [ "//" host ] [ "/" pot-designator ] [ "?" additions-list ]

pot-designator = "pot-" integer ; for machines with multiple pots
additions-list = #( addition )

HTCPCP-message = Request | Response ; HTCPCP/0.1 messages

generic-message = start-line
                  * (message-header CRLF)
                  CRLF
                  [ message-body ]
start-line       = Request-Line | Status-Line

message-header = field-name ":" [ field-value ]
field-name     = token
field-value    = * ( field-content | LWS ) field-
content = <the OCTETs making up the field-value
and consisting of either * TEXT or combinations
of token,                separators, and quoted-string>

message-body = entity-body

Method = "PROPFIND" ; Section 2.1.3
       | "GET" ; Section 2.1.2
       | "BREW" ; Section 2.1.1
       | "POST" ; Section 2.1.1
       | "WHEN" ; Section 2.1.4
       | extension-method

extension-method = token

Request = Request-Line
         * ((request-header
            | entity-header ) CRLF)
         CRLF
         [ message-body ]

Request-Line = Method SP Request-URI SP HTCPCP-Version CRLF

Request-URI = "*" | absoluteURI | abs_path | authority

request-header = Accept-Additions ; Section 2.2.2.1
                | Safe-Condition ; Section 2.2.1.1

```

Server responses

```
Response          = Status-Line
                   * ((response-header
                       | entity-header ) CRLF)
                   CRLF
                   [ message-body ]
```

```
Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
```

The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. There are 5 values for the first digit:

- o 1xx: Informational - Request received, continuing process
- o 2xx: Success - The action was successfully received, understood, and accepted
- o 3xx: Redirection - Further action must be taken in order to complete the request
- o 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
- o 5xx: Server Error - The server failed to fulfill an apparently valid request

```
Status-Code      = "100" | ; Continue
                  "200" ; OK
                  | "400" ; Bad Request
                  | "401" ; Unauthorized
                  | "402" ; Payment Required
                  | "403" ; Forbidden ; Not
                  | "404" ; Not Found
                  | "406" ; Not Acceptable
                  | "410" ; Gone
                  | "501" ; Not Implemented
                  | "503" ; Service Unavailable
```

```
response-header = Age
                 | Retry-After
                 | Safe
```

```
entity-header = Accept-Additions
```

```
Retry-After = "Retry-After" ":" ( HTTP-date | delta-seconds )
Safe        = "Safe" ":" safe-nature
safe-nature = "yes" | "no" | conditionally-safe
conditionally-safe = "if-" safe-condition
safe-condition = "user-awake" | token
```

Entity Body

```
entity-body := Content-Encoding( Content-Type( data ) )
```

The entity body of a POST or BREW request MUST be of Content-Type "message/coffeepot". Since most of the information for controlling the coffee pot is conveyed by the additional headers, the content of "message/coffeepot" contains only a coffee-message-body:

```
coffee-message-body = "start" | "stop"
```

Accept-Additions header field

```
Accept-Additions = "Accept-Additions" ":"  
                  "#" (addition-type [ accept-params])  
addition-type = ( "*" |  
                 | milk-type |  
                 | syrup-type |  
                 | sweetener-type |  
                 | spice-type |  
                 | alcohol-type ) * ( ";" parameter )  
milk-type      = ( "Cream" | "Half-and-half" |  
                 | "Whole-milk" | "Part-skim" |  
                 | "Skim" | "Non-dairy" )  
syrup-type     = ( "Vanilla" | "Almond" | "Raspberry" )  
sweetener-type = ( "White-sugar" | "Sweetener" ( |  
spice-type     = "Cinnamon" | "Cardamon" ) ( "Brandy"  
alcohol-type   = | "Rum" | "Whiskey"  
               | "Aquavit" | "Kahlua" )  
parameter number number | volume  
volume        = ( "1" | "2" | "3" | "4" | "5" )  
               = ( "dash" | "splash" | "little" | "medium" | "lots" )
```

Safe-Condition header field

```
Safe-Condition = ("user-awake" | token )
```