POLYTECH
CLERMONT-FERRAND

Application note

Stream of minimalist web server design

Implementation of a minimalist web server on a microcontroller

Author : MONTES Florent GE5

2013-2014

Summary

Figures:

Introduction:

This application note presents the stream of minimalist web server design. The first part presents how minimalist web server works. The second part shows how to install it on a microcontroller program. And to finish the last part will be used to improve some functionality of the minimalist web server.

Feature of this minimalist web server:

the web server implement in this application note has the following functionality:

- get response
- inclusion of html header
- communication between html request and hardware output

Untreated in this application note:

This application note only concerns the implementation of the web server, the implementation of TPC/IP stack is untreated. A TCP/IP stack is required to run the application. You can chose a free TCP/IP stack or use the function of your own application and match it with the minimalist web server of this application note.

How minimalist web server works:

The figure 1 below shows the functioning of the web server. When a request arrives, the request is analyzed. This analysis phase detects if the request complies, and detects the parent access files. The second step detects the type of request, in this web-server only the get request is treated. The third step allows to search the requested page. And to finish the page is send.

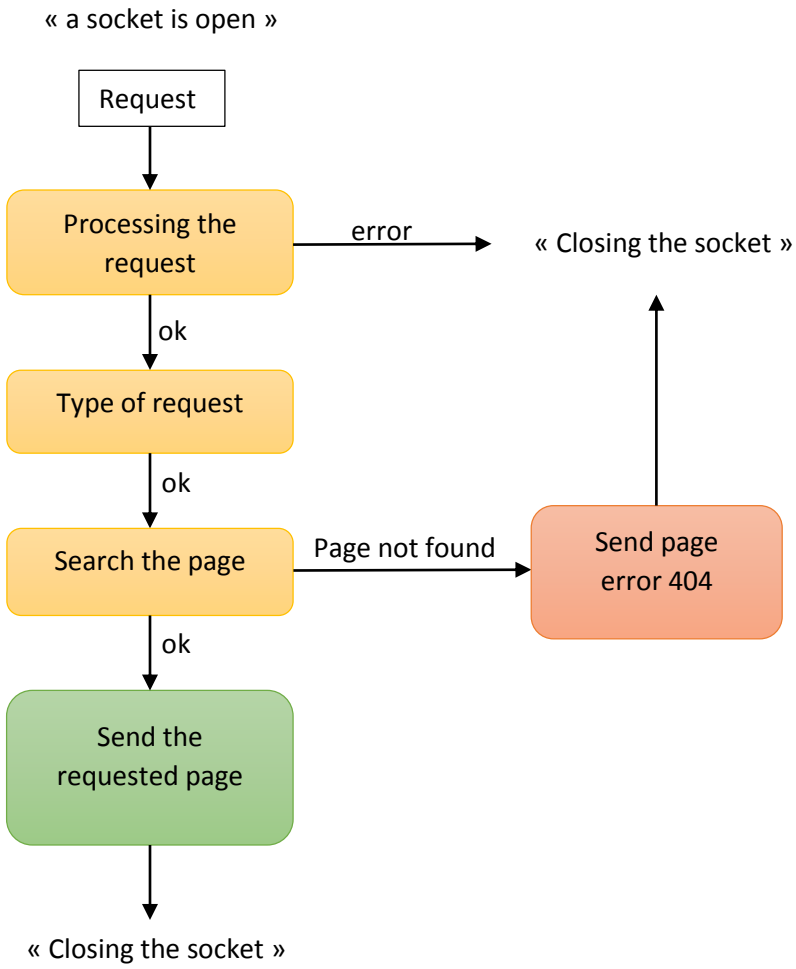

Figure 1: functioning of the web server

Sending a page:

To send a page you must use the function of your system, but before sending a page you must include the headers of the page to send. The sending headers are the following :

- HTTP/1.0 200 OK
- Content-Type: “mime type of the page or the file to send”

For the minimalist web-server we'll send only the headers above (figure 2), but there are other optional.

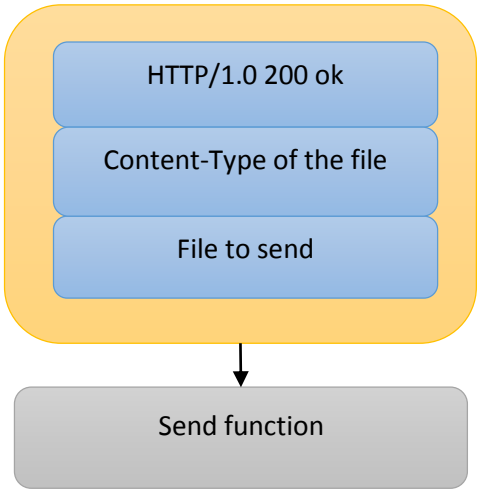

Figure 2: send a page

Send error 404

To send the error 404 page, the procedure is the same as sending a normal page. But the first header change (figure 3).

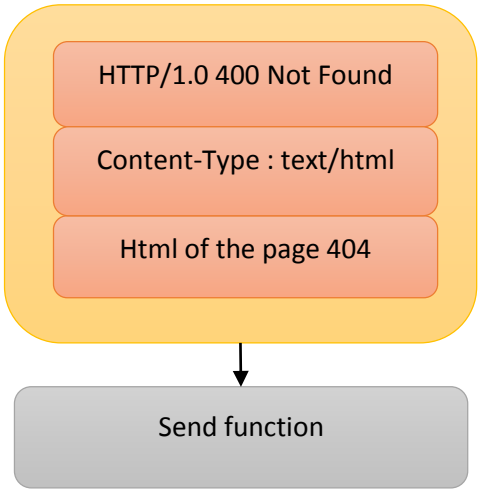

Figure 3: 404 page

The pages are stored in the memory of the microcontroller as a byte stream, they are sent by the sending function directly in this form. (see below in “Structure of pages”).

Install the minimalist web-server in your microcontroller software:

This section will explain one way to insert the minimalist web-server in a microcontroller program. Reading, sending, opening, and closing sockets function of the system will be used. The figure 4 shows how the Web-server must be implanted to work in your own application.

The application have two steps, the first is the initialization of the system and the second step is an infinite loop.

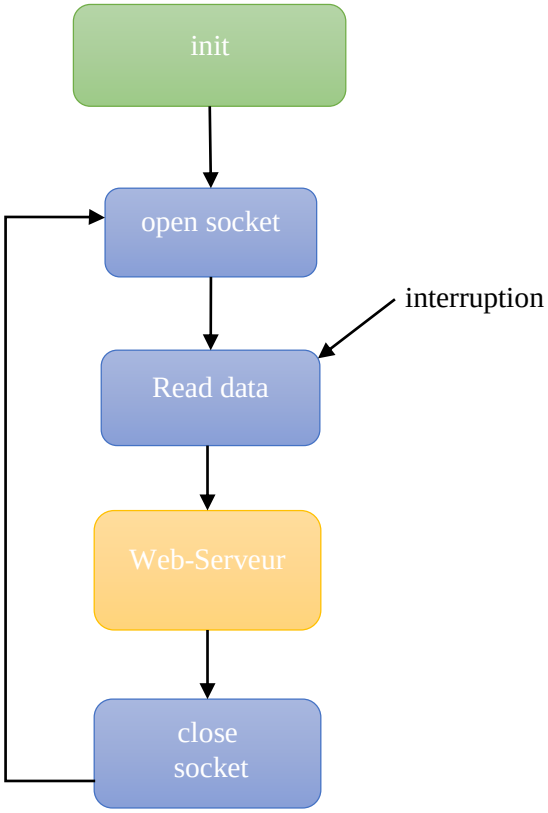

Figure 4: microcontroller program

Initialization:

The first step is the initialization of your system. You can configure you microcontroller and initialize the TCP/IP stack.

Open socket :

Next a socket on the port that you want communicate must be open. This socket will receive and send the data.

Read data :

On this implementation the data is reading whit interruption. When an interrupt is received we will read the data on the socket.

Web-Server:

This is the minimalist web server application.

Close socket :

When the web server terminate this function, we close the socket to terminate the connation. After this a new socket is open and the loop begins again.

C code of minimalist web-server:

This program in C language is the code of the minimalist Web-server, you can re-use this in your own application if you change the necessary functions.

Main program:

This first part is the main program of the Minimalist Web server. To use this in your application you must change reading and sending functions.

```
#define BUFSIZE 1460

extern  const FilesList file_list;
extern const unsigned char page_404_html_data[];

int j, file_fd, buflen, len;
long i, ret;
char * fstr;
static char buffer[BUFSIZE+1];
static char control_led[50];
static char nom_page[50];
unsigned char *page ;
int         retval = 0;
int         index =0;

/* read length data receive */
buflen = ptrStrRecvArgs->recvBufLen; // use your custom function to read the length of the data

/* copy data in buffer */
memcpy(buffer,ptrStrRecvArgs->ptrRecvBuf,buflen); // use your custom function to read the data


/* compare data  */
if( strncmp(buffer,"GET ",4) && strncmp(buffer,"get ",4) )
{
            return send_page_404();
}


/* detecting parent file access */
for(j=0;j<i-1;j++)
            if(buffer[j] == '.' && buffer[j+1] == '.')
            {
                        return send_page_404();
            }

/* convert no filename to index file */
if( !strncmp(&buffer[0],"GET /\0",6) || !strncmp(&buffer[0],"get /\0",6) )
            strcpy(buffer,"GET /index.html");


/* search index of the page */
index = indexOfFile(&buffer[5]); // this function is used to find the page in a list of files

if(index>= 0)
{
        retval=send_page((int *)file_list.files[index]->data, (int *)file_list.files[index]->type, file_list.files[index]->size );
}
else retval = send_page_404();

return 1;
}
```

Function indexOfFile :

This function allows to search for the requested page in a structure, you can define your own structure.

```
/* return the index of the file */
int indexOfFile(unsigned char *name)
{
	int i;
	for(i=0; i<file_list.count; i++)
	{
		if(strcmp(name, file_list.files[i]->name)==0) return i;
	}
	return -1;
}
```

Function send_page:

This item allows to format the data and then sending it. Careful you should replace the function by sending the features available on your system

```
int send_page(int *data, int *type, int data_size )
{
	int buffer1[1460];
	int *buff=buffer1;
	int header_size;
	int retval;
	int len_ldata ;
	int footer = 2;
	int taille_restante_buffer;
	int taille_a_copier;

	len_ldata = data_size;


	/* standard log message */
	strcpy(buffer1, "HTTP/1.0 200 OK\r\nContent-Type: ");
	buff+=31;
	strcpy(buff, type);
	buff+=strlen(type);
	strcpy(buff, "\r\n\r\n");
	buff+=4;
	header_size = buff - buffer1;
	taille_restante_buffer = 1460 - header_size;

	while (len_ldata > 0)
	{
		if(len_ldata > taille_restante_buffer)
			taille_a_copier = taille_restante_buffer; else taille_a_copier =  len_ldata;

		memcpy(buff, data, taille_a_copier);
		len_ldata -= taille_a_copier;
		taille_restante_buffer -= taille_a_copier;
		data+=taille_a_copier;
		buff += taille_a_copier;

		if(taille_restante_buffer>0)
		{
			strcpy(buff, "\n\0");
			taille_restante_buffer-=footer;
		}
```

```
                retval = rsi_send_data(1, (int *)buffer1, 1460 - taille_restante_buffer, RSI_PROTOCOL_TCP);
                // the rsi_send_data function must be changeby your own sending function
                 buff = buffer1;
                taille_restante_buffer = 1460;
        }



        return 0;
}
```

Function send_page_404:

The sending page 404 function is used when the web server did not find the requested page.

```
/* send error 404 */
int send_page_404()
{
        int buffer[707];
        int *buff=buffer;

        strcpy(buffer, "HTTP/1.0 404 Not Found\r\nContent-Type: text/html\r\n\r\n");  // header 404
        buff+=51;
        memcpy(buff, page_404_html_data, 656);
        buff+=656;
        return   rsi_send_data(1, (int *)buffer, 707, RSI_PROTOCOL_TCP);  // the rsi_send_data function must be change
                                                                          by your own sending function
}
```

Structure of pages :

The code below is the structure used for the page in the minimalist web-server program

```
typedef struct
{
        unsigned char *name;
        unsigned char *type;
        unsigned char *data;
        unsigned int size;
} File;

typedef struct
{
        File **files;
        unsigned int count;
} FilesList;
 // -> temperature.html
const unsigned char temperature_html_name[] = "temperature.html";
const unsigned char temperature_html_type[] = "text/html";
const unsigned char temperature_html_data[] =
{  0x3c, [...] ,0x6d };
const File temperature_html = {temperature_html_name, temperature_html_type, temperature_html_data, 3308};
```

Communication between the web page and the outputs of the microcontroller

When a web server is located on a microcontroller it is often useful to establish a connection between the web pages and the outputs of the microcontroller. To do this you must modify the application of the web-server to read the orders send by the web page.

For example you page web can do this get request :

http://www.led.html?led1=On&led2=On&led3=On&led4=On

For interact with the micro-controller you need to recover information after the question mark, in this case is **led1=On&led2=On&led3=On&led4=On**. To do this we will modify the minimalist web-server.

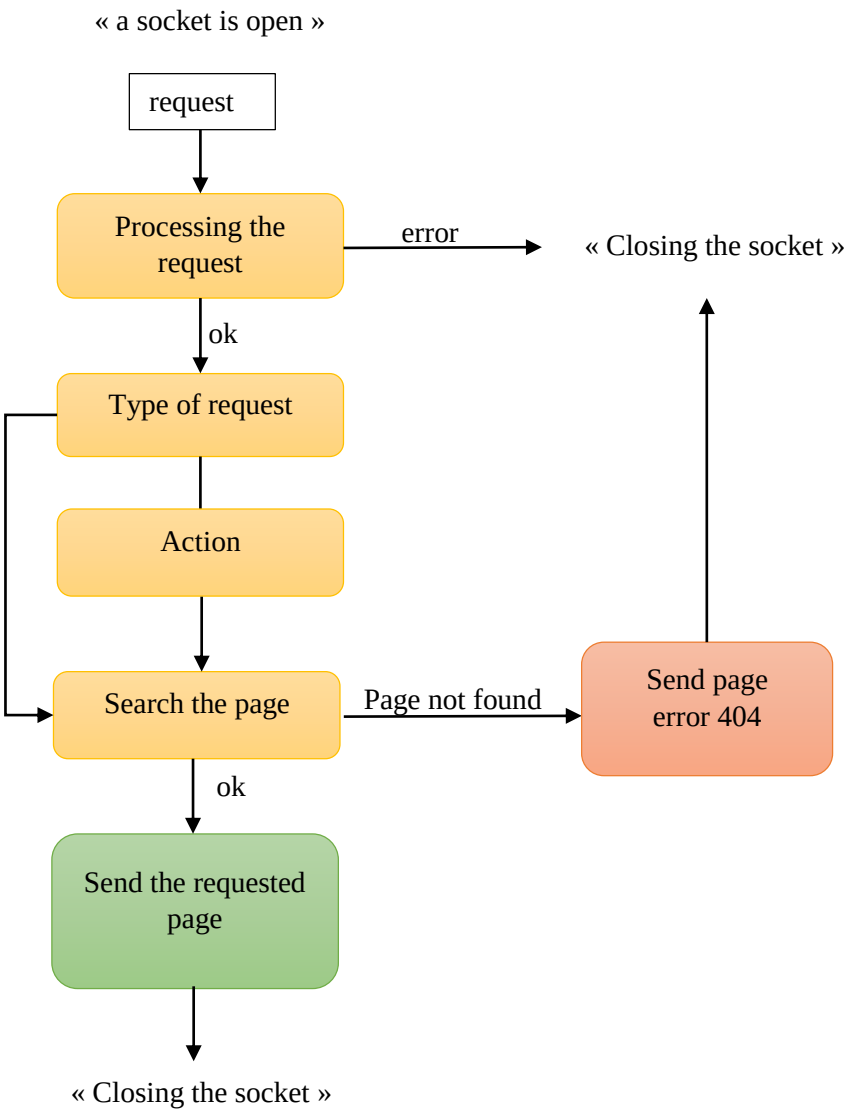

Figure 5: modification of the web-server

The change is made after the detection of the type of request. When a command is detected the action is realized and after the requested page is send.

New main program:

```
#define BUFSIZE 1460

extern  const FilesList file_list;
extern const unsigned char page_404_html_data[];

int j, file_fd, buflen, len;
long i, ret;
char * fstr;
static char buffer[BUFSIZE+1];
static char control_led[50];
static char nom_page[50];
unsigned char *page ;
int         retval = 0;
int         index =0;

/* read length data receive */
buflen = ptrStrRecvArgs->recvBufLen; // use your custom function to read the length of the data

/* copy data in buffer */
memcpy(buffer,ptrStrRecvArgs->ptrRecvBuf,buflen); // use your custom function to read the data


/* compare data  */
if( strncmp(buffer,"GET ",4) && strncmp(buffer,"get ",4) )
{
            return send_page_404();
}


/* detecting parent file access */
for(j=0;j<i-1;j++)
            if(buffer[j] == '.' && buffer[j+1] == '.')
            {
                        return send_page_404();
            }

/* convert no filename to index file */
if( !strncmp(&buffer[0],"GET /\0",6) || !strncmp(&buffer[0],"get /\0",6) )
            strcpy(buffer,"GET /index.html");

/* extract informatin */
            if (comp(buffer,control_led))
            {
            retval= send_page (1, (int* )file_list.files[3]->data,file_list.files[3]->size,RSI_PROTOCOL_TCP);
            process_led(control_led); // this is the action you can put your custom action here
            return 1;
            }
/* search index of the page */
index = indexOfFile(&buffer[5]); // this function is used to find the page in a list of files

if(index>= 0)
{
        retval=send_page((int *)file_list.files[index]->data, (int *)file_list.files[index]->type, file_list.files[index]->size );
}
else retval = send_page_404();

return 1;
}
```

The function below extracts the information after the question mark.

```
/* extract information after get */
int comp(int *trame, int *commande)
{
	int i=0;
	int j=0;

	while(trame[i] != '\0')
	{
		if (trame[i] == '?')
		{
			while (trame[i] != '\0')
			{
				commande[j]= trame [i];
				j++;
				i++;
			}
			while (j < 50)
			{
				commande[j]= ' ';
				j++;
			}
			return 1;
		}
		i++;
	}
return 0;
}
```

Conclusion:

This application gives a way to implement a web server in a microcontroller but there are many other methods, some better suited to one type of application than another.

This Application Note is based on the web-server of this website http://www.ingelibre.fr/?p=314

Contact: florent.montes@etudiant.univ-bpclermont.fr