Estudis d'Informàtica i Multimèdia

Compressió de dades sobre FreeRTOS

Javier Rufas Rivas

Enginyeria en Informàtica

Consultor: Jordi Bécares Ferrés

15 de gener de 2012

Me gustaría dedicar este trabajo
a la memoria de mi ahijada Carla.

RESUM

En aquest projecte, es desenvolupa una aplicació FreeRTOS per a la mota LPC1769 connectada amb un mòdul WiFly. Aquesta aplicació agafa un fitxer d'Internet, el comprimeix i el desa un altre cop a Internet tot afegint les estadístiques bàsiques de temps i percentatge de compressió. El sistema és configurable i controlable remotament bé per un usuari o per una aplicació. Tot el procés inclou, a més del programari de la mota, diferents eines de suport basades en tecnologies de comunicació a través d'Internet que permetin el funcionament correcte del mateix. L'exemple que acompanya al projecte és un servei web REST desenvolupat en PHP.

S'ha posat molt d'èmfasi en la modularitat del sistema, ja que no és factible la seva utilització en un sistema de producció, però sí pot ser fàcilment reutilitzat per d'altres projectes que facin servir les eines que ofereix el sistema operatiu FreeRTOS. Tanmateix, enviar i rebre fitxers per Internet és un concepte prou divers i variable com per a tenir en compte la facilitat de l'actualització dels processos involucrats.

Finalment, aquest projecte serveix per il·lustrar la utilització del mòdul WiFly connectat a un sistema encastat.

Paraules clau: FreeRTOS, LPC1769, WiFly, compressió, gzip, sistemes encastats, embeeded systems, microcontrolador, Cortex M3, C, Linux, Apache, REST, PHP.

Aquest projecte representa el treball pràctic de l'assignatura Projecte Final de Carrera - Sistemes encastats dels estudis d'Enginyeria en Informàtica (segon cicle) a la Universitat Oberta de Catalunya.

Índex de continguts

Índex de figures

1 Introducció.

En aquest curs, a l'àrea de Sistemes encastats, s'estudia la mota LPC1769 d'NXP que és una placa que integra un microcontrolador ARM Cortex M3 amb 64 KB de memòria RAM 512 KB de memòria flash i divers maquinari per poder connectar-se a d'altres dispositius intel·ligents o no, com per exemple, ports sèrie, IC2, controladors de motors, sensors, etc... Adjunta a aquesta placa, ve una interfície JTAG per tal de poder-la programar i depurar a través del port USB de l'ordinador fent servir un IDE de l'empresa Code Red basat en Eclipse.

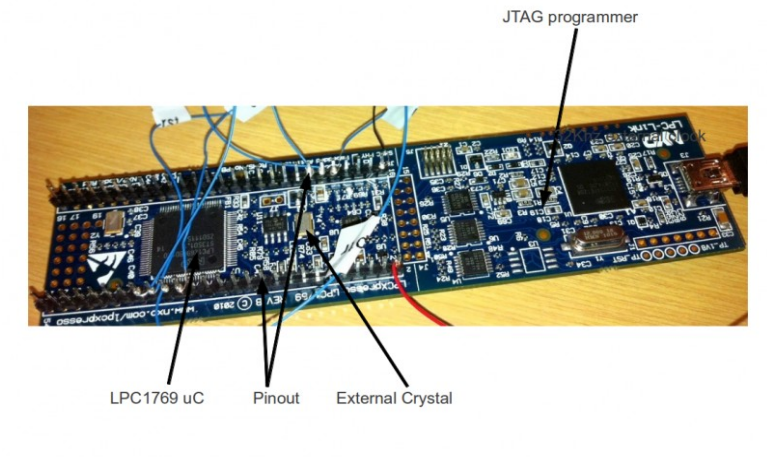

Figura 1: Mota LPC1769

El dispositiu complementari que s'estudia amb la mota és el mòdul de comunicació WiFly de Roving Networks, que és un dispositiu intel·ligent capaç de connectar-se a una xarxa WiFi i establir enllaços UDP i TCP. Tanmateix inclou prestacions addicionals que li permeten funcionar sense necessitat de cap maquinari addicional. Amb una configuració adient pot enviar, per exemple, valors de sensors que tingui connectats periòdicament a un servidor d'Internet.

Figura 2: Mòdul WiFly de Roving Networks

Hem practicat el desenvolupament de controladors per a la comunicació pel port de dades en sèrie (UART) i, especialment, la comunicació amb el mòdul Wifly quant a la connexió a la xarxa WiFi, i la transmissió de dades mitjançant el protocol HTTP cap a serveis web del tipus REST, que són molt recomanables en aquest entorn ja que precisen de molta menys quantitat d'informació que els serveis basats en SOAP.

Totes aquestes pràctiqueas s'han realitzat sobre la infraestructura que proporciona un Sistema Operatiu en Tems Real (RTOS) com és el FreeRTOS.

FreeRTOS és un programari que s'encarrega de la gestió de tasques que s'executen en el microcontrolador, proporciona processos de comunicació entre elles i assegura l'execució seguint la prioritat establerta per a cada una.

1.1 Justificació.

Per a la realització del Projecte Final de Carrera, vaig triar l'opció de Compressió i estudi, de la qual l'enunciat diu:

"El projecte consta, en que la mota (embedded 1769) descarregui un seguit de dades de Internet, les comprimeix-hi a la mota i després envii el fitxer comprimit i les estadístiques de la compressió (nivell de compressió, temps de compressió, memòria utilitzada, ...) a Internet.

El algoritme a aplicar es de lliure elecció per l'estudiant, com més eficient i òptim sigui, millor."

En una primera lectura, no sembla que una mota sigui el dispositiu més adient per a comprimir fitxers, però sí que és interessant el fet de disposar, d'una banda, d'un sistema de compressió de dades en aquest tipus de dispositiu per les possibles períodes en que no hi hagi connexió i, d'altra banda, de sistemes de transmissió de la informació amb d'altres equips remots mitjançant diversos protocols (HTTP, FTP, etc...).

Amb aquestes premisses, el projecte no s'ha enfocat tant a la eficiència de la compressió sinó a l desenvolupament d'un bastiment de tractament de fitxers (lectura i escriptura) d'Internet, mitjançant el mòdul WiFly, que sigui prou flexible com per a poder-la adaptar als diversos usos que se li poden donar en el món real.

1.2 Descripció.

El sistema el composa, principalment, el programari de la mota que espera una connexió entrant TCP a un port determinat. Quan es rep una petició d'aquest tipus bé mitjançant el programa TELNET o qualsevol de similar, el programa demana una de les tres ordres possibles: set, get i run. Amb això, l'usuari pot consultar i definir els paràmetres del procés: protocols i fitxers d'entrada i sortida, mètode i nivell de compressió, format de fitxer comprimit, etc... i llençar l'execució.

A l'altre extrem pot haver-hi diversos tipus de sistemes que s'hauran d'ajustar als requeriments de l'aplicació ja que els protocols poden ser diversos, tot i que en aquest projecte només hem abastat el protocol HTTP. Per això, s'han creat dos petits serveis web que s'han d'allotjar en un servidor que amb PHP i les configuracions de permisos adients per poder llegir i escriure els fitxers corresponents.

1.3 Objectius.

Els objectius que pretenc aconseguir amb aquest projecte són:

1. Treballar amb microcontroladors i obtenir un coneixement de les problemàtiques associades als mateixos.
2. Generar un bastiment per al tractament de fitxers d'Internet sobre FreeRTOS i el mòdul WiFly que sigui fàcil d'ampliar i configurar.
3. Generar una capa d'abstracció el més completa possible per a la gestió de les capacitats del mòdul WiFly que permeti al desenvolupador utilitzar el dispositiu de la forma més senzilla possible.
4. Proveir d'un sistema de comandament remot al conjunt mota-WiFly, permetent ser controlat per un usuari o per un programa.
5. Portar a FreeRTOS una llibreria de compressió de dades amb el mínim consum de memòria i qualitat acceptable.

1.4 Enfocament i mètode seguit.

En primer lloc, i desprès d'una fase de formació que va consistir en la realització de tres PAC's amb diferents objectius marcats pel consultor, vaig realitzar la selecció d'una de les propostes. És en aquest punt on es pot dir que va començar el projecte.

Per començar, vaig realitzar un anàlisi global de funcionament del sistema. D'aquest pas van sortir els casos d'ús i els equips implicats: un servidor d'Internet i un conjunt mota-WiFly amb connexió a Internet. Amb això hi havia suficient informació com per a determinar l'abast del projecte i les especificacions que havien de complir les diferents parts implicades.

En segon lloc, vaig escriure la planificació basant-me en l'experiència adquirida per poder valorar l'esforç necessari per al desenvolupament dels diferents programes. Com que aquesta experiència no era suficient, vaig fer una planificació de mínims per tal d'assegurar l'èxit del projecte i vaig tenir en compte els possibles riscos que podien determinar el fracàs del projecte.

A continuació, vaig començar l'etapa de desenvolupament de cada bloc per separat, provant el funcionament i comprovant que aquest era l'esperat. En aquesta etapa, els problemes sorgits es van poder resoldre amb relativa facilitat, omplint, al mateix temps els buits que l'anàlisi inicial i les especificacions no van omplir.

Finalment, vaig realitzar les proves d'integració on es va fer palesa la falta d'experiència en l'entorn de treball i vaig haver de prendre la decisió de prioritzar el disseny de l'aplicació en pro de la seva capacitat d'evolució i minvar la qualitat de la compressió de les dades. Tanmateix, aquest esforç suplementari va esgotar el temps que havia previst per a diverses ampliacions. Tot i així, el producte compleix les especificacions inicials.

1.5 Planificació.

La planificació del projecte abasta la definició de les fites i tasques, la temporització

1.5.1 Fites i tasques.

Les fites principals venien marcades pel Pla Docent de l'assignatura. Cada fita comporta la realització, dintre del termini, d'una sèrie de tasques.

1. Lliurament de codi font **2 de gener de 2013**

- **a)** Anàlisi de l'aplicació de la mota
- **b)** Anàlisi de l'aplicació del servidor
- **c)** Desenvolupament de l'aplicació de la mota.
  - Desenvolupament i prova de la Tasca de Control
  - Desenvolupament i prova de les tasques de Lectura i Escriptura
  - Desenvolupament i prova de la Tasca de Compressió
- **d)** Desenvolupament de l'aplicació al servidor
  - Instal·lació i configuració del servidor
  - Desenvolupament i prova del Servei de Lectura
  - Desenvolupament i prova del Servei d'Escriptura
- **e)** Proves d'integració

2. Lliurament de la Memòria **15 de gener de 2013**

- **a)** Recopilació dels resultats
- **b)** Redacció del document.

3. Lliurament de la Presentació **19 de gener de 2013**

- **a)** Generació de la presentació.
- **b)** Gravació del vídeo de la presentació.

1.5.2 Temporització

La duració i temporització de les fases del projecte és la següent:

		Nombre	Inicio	Terminado
1		**⊟Generació del codi**	**1/12/12 8:00**	**28/12/12 8:00**
2		Análisi de aplicació mota	1/12/12 8:00	2/12/12 8:00
3		Análisi aplicació servidor	2/12/12 8:00	3/12/12 8:00
4		**⊟Desenvolupament LPC**	**3/12/12 8:00**	**21/12/12 20:00**
5		Tasca Control	3/12/12 8:00	7/12/12 17:00
6		Tasca Lectura	8/12/12 17:00	11/12/12 2:00
7		Tasca Escriptura	11/12/12 2:00	13/12/12 11:00
8		Tasca Compressió	14/12/12 11:00	21/12/12 20:00
9		**⊟Desenvolupament servidor**	**22/12/12 20:00**	**26/12/12 8:00**
10		Servei lectura	22/12/12 20:00	23/12/12 20:00
11		Servei escriptura	24/12/12 20:00	26/12/12 8:00
12		**⊟Proves**	**7/12/12 17:00**	**28/12/12 8:00**
13		Prova Control	7/12/12 17:00	8/12/12 17:00
14		Prova Lectura Escriptura	13/12/12 11:00	14/12/12 11:00
15		Prova Compressió	21/12/12 20:00	22/12/12 20:00
16		Prova Servei lectura	23/12/12 20:00	24/12/12 20:00
17		Prova Servei escriptura	26/12/12 8:00	27/12/12 8:00
18		Prova d'integració	27/12/12 8:00	28/12/12 8:00
19		**⊟Memòria**	**1/12/12 9:00**	**15/01/13 17:00**
20		Recopilació dels resultats	1/12/12 9:00	26/12/12 18:00
21		Redacció del document	26/12/12 18:00	15/01/13 17:00
22		**⊟Presentació**	**15/01/13 8:00**	**19/01/13 17:00**
23		Realització de la presentació	15/01/13 8:00	17/01/13 17:00
24		Gravació del vídeo	18/01/13 8:00	19/01/13 17:00

Figura 3: Cronograma

En aquest cas, l'únic recurs humà és l'alumne i, per tant, no es pot preveure la realització en paral·lel de diverses tasques com es pot veure al diagrama de Gannt. En una altra situació, la part de desenvolupament es pot optimitzar d'una forma directament proporcional a les dimensions de l'equip.

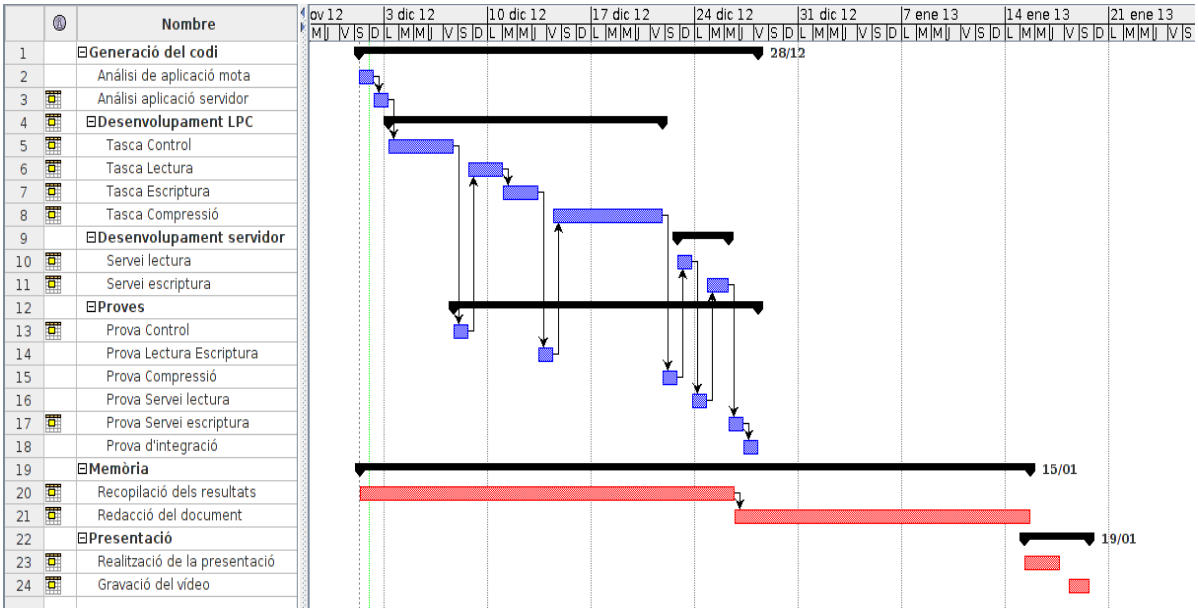

Figura 4: Diagrama de Gannt

1.5.3 Estudi de riscos.

Els possibles riscos que podem patir, amb la seva qualificació, temps i solució són:

1. Malaltia de l'alumne. Greu (indefinit).

   No hi podem actuar.

2. Desconeixement de la tecnologia. Greu (indefinit).

   Cerca de la informació i formació.

3. Avaria de la mota o WiFly. Greu (5 dies).

   El termini d'entrega és de 3 a 4 díes.

4. Avaria de ordinador. Lleu (1 dia).

   Si no es pot reparar, es canvia per el de reserva.

5. Avaria de la connexió. Lleu (1 hora).

   La connexió a Internet no és necessària. Si l'avaria és a l'encaminador, es pot canviar immediatament.

1.6 Recursos emprats.

Com a recursos de maquinari, es disposa d'una xarxa WiFi amb connexió a Internet, la mota LPC1769, el mòdul WiFly, l'adaptador CP2102 i un ordinador personal amb processador AMD Athlon X2, 4Gb de RAM i 600 Gb de disc dur amb sistema operatiu Linux Ubuntu 10.04. Tanmateix disposo d'un disc extern de 2 Tb. per a realitzar còpies de seguretat a nivell de dades com d'imatges de disc.

L'entorn de desenvolupament és el subministrat amb la mota: LPCXpresso. És un IDE basat en Eclipse que s'ha d'obtenir de la pàgina web del fabricant mitjançant el registre i l'obtenció posterior de la llicència, el que permet la depuració de codi que no arribi als 128 KB.

Com a equipament de reserva en cas d'avaria, puc disposar de tres ordinadors més, un dels quals té les mateixes característiques que el de treball. Com alternativa a la connexió a Internet, en cas que l'operador trigui més d'un dia a restablir el servei, disposo de connexió 3G i dos encaminadors de reserva.

1.7 Productes obtinguts.

Els productes obtinguts, representen a nivell de programació de la mota:

- Llibreria UART. És una capa d'abstracció de la gestió de la lectura i escriptura en els ports sèrie de la mota.
- Llibreria printf. És una implementació de la funció printf() amb sortida per la UART.
- Llibreria WiFly. És una capa d'abstracció de la gestió de les operacions del mòdul a través d'una UART.
- Llibreria utils. És la implementació d'algunes funcions que no existeixen a <string.h> de la llibreria proporcionada per Code Red.
- Llibreria WiFly per a FreeRTOS. És una capa d'abstracció del driver WiFly per a FreeRTOS, proveint de mecanismes d'exclusió mútua per a l'utilització del dispositiu per diferents tasques i lectura i escriptura en cues del sistema operatiu.
- Llibreria gzipLite per a FreeRTOS. És el resultat de portar la utilitat de compressió gzipLite [1] permetent-li llegir i escriure en cues del sistema operatiu.
- Aplicació PAC5App que és la que, fent servir les anteriors llibreries, implementa les funcionalitats definides a l'abast del projecte.
- Codi font modificat de gziplite per compilar-ho en un ordinador personal i poder descomprimir els fitxers resultants.

A nivell de programació del servidor:

- Servei web REST de lectura d'un bloc determinat d'un fitxer del servidor.
- Servei web REST d'escriptura d'un bloc determinat de dades a un fitxer del servidor.

1.8 Descripció dels capítols següents.

Al capítol 2 s'expliquen les investigacions realitzades per a conèixer l'estat de la tecnologia a l'àrea dels sistemes encastats.

Al capítol 3 es descriu el disseny de les parts que composen aquest projecte, quins han estat els criteris aplicats i perquè s'ha decidit.

Al capítol 4 es detallen els aspectes tècnics d'implementació.

Al capítol 5 es fa una reflexió sobre la viabilitat del sistema en un instal·lació real.

Y al capítol 6 es realitza una autoavaluació dels resultats.

1 http://sourceforge.net/projects/gziplite/

2 Antecedents.

En el món de la informàtica, el maquinari ha evolucionat en capacitat i prestacions segons les capacitats tecnològiques del moment, però sempre hi ha necessitats que no precisen d'una gran potència de càlcul ni grans quantitats de memòria ni emmagatzemament. Aquesta és la parcel·la on s'ubiquen els sistemes encastats. Són dispositius que, habitualment s'agrupen formant par d'un sistema més gran com, per exemple, un cotxe, que pot arribar a tenir una quantitat important d'aquests dispositius. També es poden trobar en electrodomèstics o, formant part del que s'anomena xarxes de sensors. En aquest últim cas podem trobar molts exemples als dispositius que formen les xarxes de les Smart Cities. Les motes funcionen individualment a tots els nivells, fins i tot d'alimentació elèctrica ja que sovint es troben sota el paviment, prenent dades de l'entorn, però es comuniquen amb d'altres nodes per tal d'enviar aquestes dades a un o més servidors centrals. És en aquest cas resulta evident que ha de primar el baix consum d'energia sobre les altres capacitats.

2.1 Estat de l'art.

Com ja hem vist, les aplicacions són molt variades, així com els dissenys de maquinari. Podem trobar molts tipus de microcontroladors i sistemes encastats que, habitualment, no són compatibles. És per això, i perquè cada fabricant pren les decisions que més li convenen, que no hi ha una estandardització com podem trobar a la informàtica d'ordinadors personals o servidors.

Tot i això, en el cas dels processador Cortex-M, hi ha una capa d'abstracció de les interfícies i els perifèrics que simplifica la reutilització del codi i redueix la corba d'aprenentatge. Aquesta capa s'anomena ARM Cortex Microcontroller Software Interface Standard i respon a les sigles CMSIS.

2.2 Estudi de mercat.

Com que aquest tipus de dispositius s'utilitzen fonamentalment en la indústria i es personalitza el programari per a la tasca prevista, no és fàcil trobar empreses que publiquin detalls de les tecnologies emprades, per tant o he trobat cap aplicació que faci servir una mota per a comprimir dades. Una altra cosa és el món de les persones que ho tenen per afició. Habitualment treballen amb sistemes de codi obert i comparteixen els seus desenvolupaments, tot i que no hi ha cap organització.

3 Descripció funcional.

3.1 Sistema total.

El projecte consisteix al desenvolupament d'un programari que funcioni sobre una mota LPC1769 amb un mòdul WiFly connectat i sigui capaç de rebre ordres de forma remota per tal de configurar la descàrrega, compressió i posterior enviament d'un fitxer a Internet.

Aquest programari s'ha de complementar amb un servei d'Internet que permeti obtenir una quantitat determinada de bytes d'un fitxer i un altre servei que permeti desar una quantitat determinada de bytes en una part determinada d'un altre fitxer.

El funcionament desitjat del tractament d'un fitxer seria que l'usuari, mitjançant una connexió telnet a la mota, pogués enviar un seguit d'ordres per definir els paràmetres de com obtenir, comprimir i desar el fitxer i poder, finalment, enviar l'ordre de compressió.

3.2 Mota.

El mòdul WiFly permet establit una connexió telnet per rebre informació i enviar-la a la UART. Tractarem aquesta informació com ordres, una per línia prenent com acabada quan es rebi qualsevol combinació de \r, \n per tal de garantir el funcionament des del major nombre de terminals.

A continuació, definim les comandes que ha d'entendre l'aplicació:

set <variable> <value>	Especifica el valor de la variable.
get <variable>	Retorna el valor de la variable, o la llista de totes si no s'especifica.
run	Comença el procés.

Les variables són:

wsInType	Tipus de servei de lectura (al menys “http”).
wsOutType	Tipus de servei d'escriptura (al menys “http”).
WsInUrl	URL del servei de lectura.
wsOutUrl	URL del servei d'escriptura.
wsInFile	Paràmetre del servei de lectura per determinar el nom del fitxer.
wsOutFile	Paràmetre del servei d'escriptura per determinar el nom del fitxer.
WsInBytes	Paràmetre del servei de lectura per determinar la quantitat de bytes a transferir.

wsIOutBytes Paràmetre del servei d'escriptura per determinar els bytes a escriure.

WsInOffset Paràmetre per determinar el començament de la lectura dins del fitxer.

wsOutOffset Paràmetre per determinar el començament de l'escriptura dins del fitxer.

wsOutTime Parámetre per enviar el temps de procesament del fitxer.

wsOutRatio Paràmetre per enviar el ratio de compressió obtingut.

cpOperation Operació a realitzar: “compress” o “uncompress” (al menys “compress”)

cpType Tipus de compressió (al menys “deflate”).

cpFormat Format del fitxer (al menys “gzip”).

cpLevel Nivell de compressió (0 a 9).

Quan es rebi la comanda “run”, la mota enviara la resposta: “Processing” encegarà el procés de compressió i, quan finalitzi, enviarà (si la connexió continua activa) les estadístiques del procés i la resposta “Done”.

Aquest programari s'ha d'implementar sobre FreeRTOS i ha de fer servir les eines que ens ofereix, separant els processos en tasques, utilitzant cues per al traspàs d'informació i semàfors per a l'ús dels dispositius amb la intenció de fer-ho més modular i fàcil de mantenir. El disseny ha de ser el suficientment flexible com per a permetre l'ampliació amb diverses operacions (descompressió, verificació, resums SHA, MD5, etc...), tipus de compressió (LZMA, etc...), tipus de fitxer (ZIP,7z, etc...) i serveis de pujada i baixada en diferents protocols (FTP, TCP, etc...).

Quant al programari de suport que ha de treballar juntament amb aquest dispositiu, realitzarem dos serveis web que siguin capaços de enviar i rebre parts determinades d'un fitxer. La implementació dels serveis esmentats es realitzarà sobre una màquina virtual amb un servidor web i un llenguatge adient (PHP, Java, etc...). D'aquesta forma es podrà lliurar juntament amb el programari.

3.3 Serveis REST.

Al **servei de lectura** del fitxer s'accedirà mitjançant una URL del tipus:

http://servidor/wsReadFileBlock?wsInFile=”nom del fitxer”&wsInOffset=”inici”&wsInBytes=”mida”

i ha de retornar els bytes del fitxer sense cap tipus de format afegit.

S'accepta el paràmetre addicional wsGetInfo. Si aquest paràmetre existeix i no és buit, el servei ha de retornar la informació del fitxer en format text. La primera línia conté la mida del fitxer en bytes.

El **servei d'escriptura** ha de tenir un funcionament similar amb una URL de la forma:

http://servidor/wsWriteFileBlock?
wsOutFile=”nom_del_fitxer”&wsOutOffset=”inici”&wsOutBytes=”bytes_hex”

On “bytes_hex” són les dades en hexadecimal. Com a paràmetres opcionals ha de tenir els relatius al resultat: wsOutTime (en mil·lisegons) i wsOutRatio com el percentatge de compressió. En el cas que existeixin, el servei desara en el fitxer demanat aquestes dades de resum en format text. El nom del fitxer no ha de ser el mateix que el de resultat ja que es sobreescriuria.

4 Descripció detallada.

Aquest capítol és, probablement el més important degut a que és on es troben els detalls més tècnics del desenvolupament i la implementació.

4.1 Sistema total.

Per obtenir el resultat desitjat, hem de comptar, com a mínim, amb la següent infraestructura de maquinari:

D'una banda tenim el conjunt mota-Wifly que es comuniquen per la UART 3 de la mota amb quatre cables segons l'esquema:

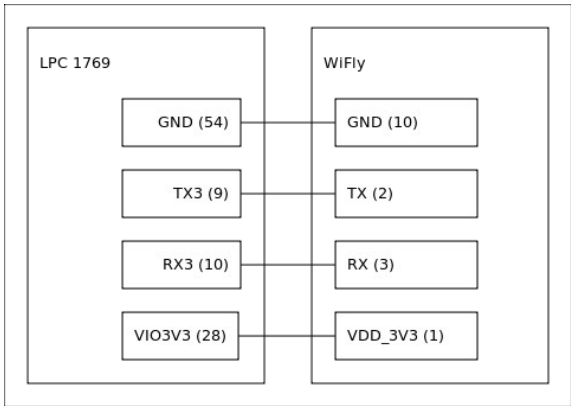

Figura 5: Connexió mota - Wifly

A l'entorn de desenvolupament i proves, tenim un conversor de UART a USB que connectarem per un costat a la mota i per l'altre a l'ordinador personal per poder visualitzar els missatges de la depuració mitjançant un programa de terminal com el CuteCom. Les connexions es realitzen com es pot veure en aquest esquema:

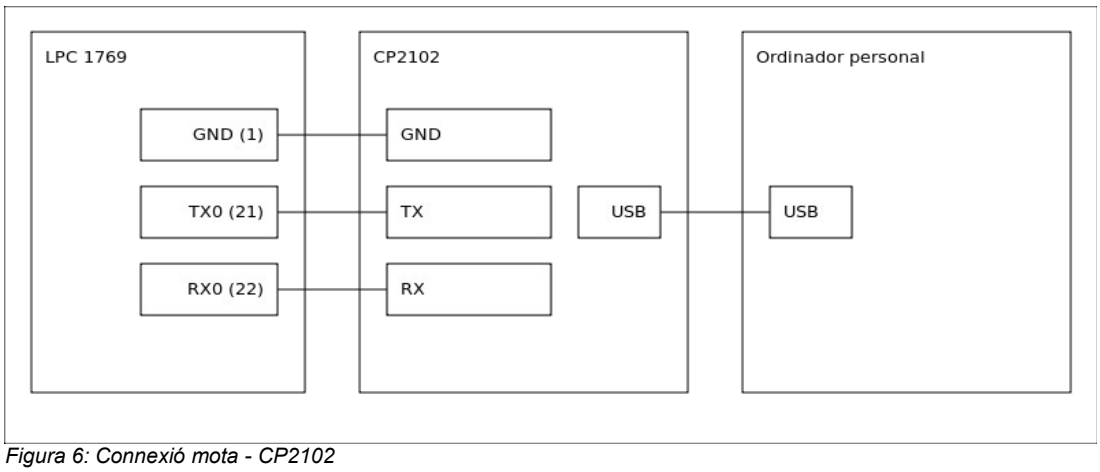

Figura 6: Connexió mota - CP2102

Aquest conjunt ha de trobar-se a una distància suficient d'un router WiFi amb connexió a Internet mitjançant la qual, el dispositiu serà accessible des de qualsevol terminal de la xarxa per tal de poder iniciar el procés que involucrarà a un o més servidors, que són les responsables de facilitar les dades d'entrada i desar les de sortida.

Podem visualitzar gràficament l'estructura de maquinari al següent diagrama:

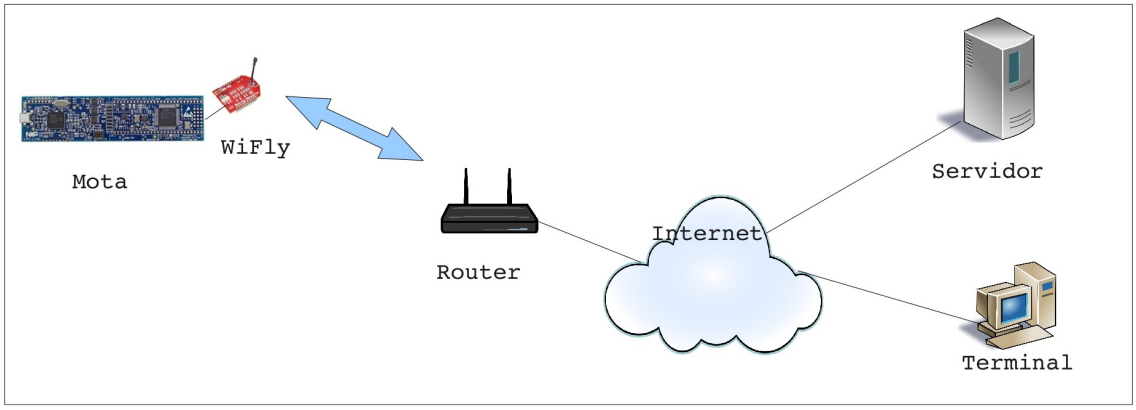

Figura 7: Diagrama de maquinari

En aquest procés, només hi ha un usuari possible que pot consultar i modificar el valor de les variables i executar el procés. Per tant el cas d'ús és el que es representa en el següent diagrama:

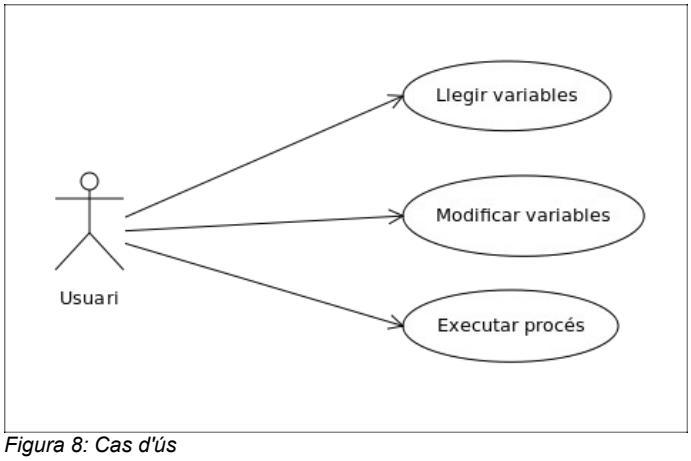

Figura 8: Cas d'ús

Per tenir una visió global del funcionament del programari he dibuixat el diagrama de blocs del programari on es poden apreciar les interaccions entre les tasques i els serveis.

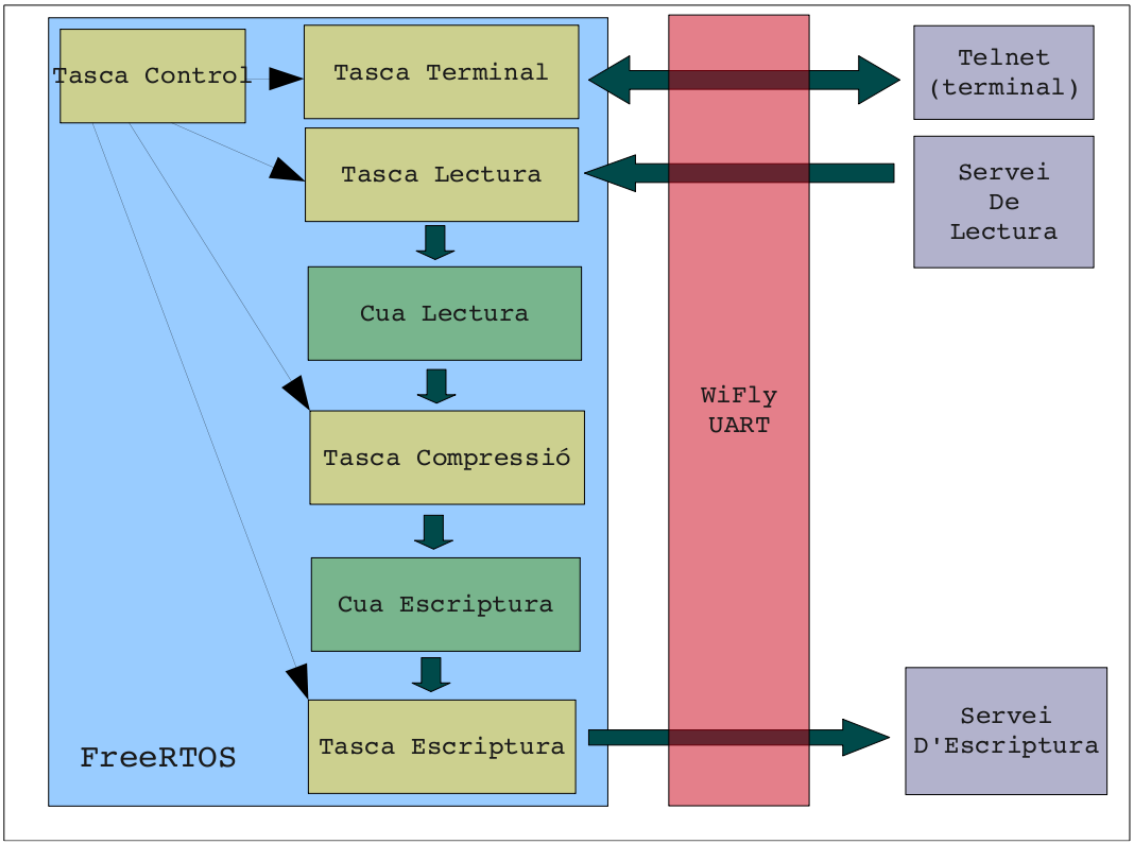

Figura 9: Diagrama de blocs del programari

Quan s'encén la mota, el programa inicialitza tots els valors i els dispositius i s'executa la tasca de control. Aquesta li demana al WiFly si està associat al router per enviar-li les dades en cas que no ho estigui, llavors posa al WiFly com a servidor TCP i treu pel dispositiu de debug la IP i el port on accepta connexions. En el moment que s'efectua una connexió, s'executa la tasca de terminal que és la que gestiona aquesta línia enviant i rebent els missatges d'un cantó a l'altre fins que es detecta la comanda RUN, on la tasca de control crea les tasques de lectura, compressió i escriptura depenent dels valors de les variable corresponents i es desactiva. En aquest moment, la tasca de lectura demana al servei de lectura les dades del fitxer i el contingut per blocs i el va escrivint a la cua de lectura fins que s'arriba al final, que s'envia un En Of File i es destrueix la tasca.

La tasca de compressió, llegeix les dades de la cua de lectura, els comprimeix i els va escrivint a la cua d'escriptura fins que s'acaba que es fa el mateix que a l'anterior: s'envia un EOF i es destrueix la tasca.

La tasca d'escriptura, llegeix de la cua d'escriptura i va omplit un petit buffer que es buida cap al servei d'escriptura fins que s'arriba al final, moment en que es genera el resum i s'envia al servei d'escriptura amb el nom del fitxer de sortida amb el text ".txt" afegit. A continuació, reactiva la tasca de control i es destrueix.

Finalment, la tasca de control torna al començament, iniciant de nou el procés.

4.2 Mota.

El conjunt del programari de la mota està composat pel codi del FreeRTOS, les llibreries de suport i les tasques del programa principal.

4.2.1 Llibreries de suport

Per al desenvolupament del programa de la mota s'han definit les capes d'abstracció o llibreries que s'han explicat al punt 1.7. D'aquestes, podem considerar a uart, wifly i wiflyFreeRTOS com a drivers de dispositiu, ja que abstrauen al programador del funcionament intern del dispositiu. En aquests casos, tot i que el llenguatge de programació és C, s'ha fet una aproximació a la programació orientada a l'objecte, creant una estructura que defineix els paràmetres bàsics del dispositiu i que equivaldria al conjunt de propietats de l'objecte. Els mètodes s'implementen com a funcions que reben un punter a aquesta estructura com a primer paràmetre permetent, d'aquesta forma, l'accés a les propietats des de qualsevol mètode.

4.2.1.1 UART

El driver de la UART s'ha desenvolupat a partir del driver original del fabricant, però s'ha modificar el buffer que passa a ser del tipus FIFO per tal de facilitar la gestió del mateix. L'estructura de propietats té els valors del port i baudrate i punters als paràmetres més importants de cada port. D'aquesta forma, no s'ha d'implementar la mateixa funció per a cada port com al codi original.

Els mètodes bàsics són :

UARTinit(), que inicialitza una UART.

UARTsendByte(), que envia un byte a la UART.

UARTgetByte(), que rep un byte de la UART.

UARTflush(), que buida el buffer de recepció.

UARTisAvailable(), que informa si hi han dades al buffer.

Basats en aquests mètodes, s'han afegit:

UARTsendBytes(), que envia un nombre determinat de bytes a la UART.

UARTgetBytes(), que rep un nombre determinat de bytes de la UART.

UARTreadString(), que llegeix bytes en un buffer fins que arriba el byte de End Of String.

UARTreadLine(), que llegeix bytes en un buffer fins que arriba el byte de End Of Line.

4.2.1.2 WiFly

Igual que l'anterior, el driver WiFly gestiona les capacitats del dispositiu. Aquesta llibreria és el resultat de portar la llibreria WiFlySerial (http://sourceforge.net/projects/arduinowifly/) escrita inicialment per la plataforma Arduino per Tom Waldock i distribuïda amb llicència LGPLv2. El propòsit de realitzar la portació a CMSIS en lloc de fer-ho directament a FreeRTOS, ha estat motivada per l'ànim de ampliar l'àmbit d'aplicació.

L'estructura principal és del tipus _wifly i conté les propietats de WiFlySerial a més d'una estructura _uart i punters a les funcions millis() i Debug(). Com que CMSIS no ofereix cap utilitat de mesura del temps, ha de ser el programador qui desenvolupi una funció que retorni el mil·lisegon actual des de l'inici de l'execució que dependrà de l'entorn per al que es desenvolupi. Tanmateix, s'ofereix la possibilitat de definir una funció de depuració per facilitar la detecció de problemes al funcionament del codi.

A més dels mètodes originals, s'han afegit els següents:

WiFlygetTxPower(), per obtenir la potència de la senyal.

WiFlysetTxPower(), per determinar la potència de la senyal.

WiFlysendPing(), per enviar un ping a una adreça IP o nom DNS de la xarxa.

WiFlysendHTTPGet(), per a enviar una petició GET senzilla del protocol HTTP a una URL determinada.

WiFlygetDataHTTPGet(), per rebre les dades d'una petició GET del protocol HTTP amb una gestió més avançada que el mètode anterior.

WiFlyDelayUntilNoData(), per esperar a rebre dades fins un temps determinat.

WiFlyDelayUntilReceivingData(), per esperar mentre es rebin dades encara que hi hagi retards.

WiFlyReceiveDataUntilDelay(), per rebre dades en un buffer tenint en compte que poden haver-hi retards.

Tanmateix he afegit algunes funcions per a facilitar el formateig dels missatges de depuració, i la possibilitat de traure apartir d'un nivell (FULL, INFO, CRITICAL o NONE).

4.2.1.3 WiFlyFreeRTOS

En un nivell superior d'abstracció es troba la gestió del WiFly des de FreeRTOS. Aquesta capa s'ha dissenyat com a genèrica, però s'han implementat només les utilitats específiques per a l'aplicació principal d'aquest projecte. Això no treu que sigui fàcilment ampliable mantenit la compatibilitat. D'altra banda, com que FreeRTOS ja disposa d'una gestió del temps, aquesta llibreria ja implementa la funció millis() necessària per al driver de WiFly.

L'estructura principal del tipus _wiflyFreeRTOS conté, òbviament una estructura del tipus _wifly i preveu una altra de tipus _uart per enviar els missatges de depuració. Ambdues tenen un semàfor associat per assegurar l'exclusió mútua en el cas d'utilitzar-se simultàniament per tasques diferents. Finalment, s'han afegit quatre cues, dos per a la gestió de terminal i altres dos per a la gestió de les dades rebudes per tal que les aplicacions no hagin de fer gaire més que llegir o escriure en aquestes.

Quat als mètodes, aquests són:

WiFlyFreeRTOSsetWiFlyUart(), assigna el dispositiu WiFly per la UART a la que està connectat.

WiFlyFreeRTOSsetUartDebug(), assigna una UART per depuració mitjançant el port, baudrate i nivell. Per defecte la depuració no està activa.

WiFlyFreeRTOSbegin(), inicialitza els dispositius un cop han estat assignats.

WiFlyFreeRTOSjoinAutoWPA(), utilitat per unir-se a una xarxa WiFi amb WPA indicant l'SSID i la contrasenya.

WiFlyFreeRTOSisAssociated(), retorna l'estat de associació de la connexió WiFi.

WiFlyFreeRTOSemptyQueues(), buida les cues.

WiFlyFreeRTOSterminalListen(), coloca el dispositiu en mode de recepció de dades (TELNET) i envia i rep fent servir les cues de terminal.

WiFlyFreeRTOSgetHTTPRequest(), envia una petició HTTP i rep les dades en un buffer.

Aquest últim mètode s'ha modificat hauria de rebre les dades de la petició i escriure-les en la cua de dades, però aquest comportament s'ha modificat degut a la variabilitat de les peticions HTTP des de l'aplicació i a la necessitat de reduir el consum de memòria del conjunt de l'aplicació. Tanmateix, es deixen les cues per possibles ampliacions.

4.2.1.4 gzipLiteFreeRTOS

Un dels punts més complicats va estar aconseguir una implementació del sistema de compressió deflate i del format de fitxer comprimit gzip. Van ser diferents els sistemes de compressió provats, sent candidats possibles:

miniz [1], va ser el primer candidat, però no complia amb els requeriments de memòria.

miniLZO [2], tot i que els requeriments de memòria són grans, es podia haver reduït i no ofereix un format de fitxer.

lzss-0.6.2 [3], compleix tots els requeriments, tret del format de fitxer, que podia haver estat gzip, però l'aplicació estàndard tampoc no l'hauria podit llegir.

gzipLite [4], compleix amb tots els requeriments inicials quant a necessitat de memòria i genera fitxers gzip que poden ser llegit d'un altre sistema.

Així doncs, vaig portar la llibreria gzipLite tot definint `ZIP_VERY_SMALL_MEM_USAGE` per tal de fer-ho el més petit possible, però sense perdre compatibilitat.

Per fer-ho integrable dins del FreeRTOS, vaig implementar les funcions:

readFromQueue(), per llegir les dades d'una cua.

writeToQueue(), per escriure les dades comprimides a una cua.

writeEofToQueue(), per escriure el caracter EOF un cop acabada la compressió.

Per a l'aplicació, vaig implementar les funcions:

setLevel(), per a definir el nivell de compressió.

getLevel(), per a obtenir el nivell de compressió.

Finalment, es van modificar les funcions read_buffer() i write_buffer() per a gestionar correctament aquest nou tipus de dispositiu.

1 http://code.google.com/p/miniz/

2 http://www.oberhumer.com/opensource/lzo/#minilzo

3 http://michael.dipperstein.com/lzss/

4 http://sourceforge.net/projects/gziplite/

4.2.1.5 util i printf

Per finalitzar aquest apartat de les llibreries, només vull afegir que, com al llarg del desenvolupament s'han fet paleses algunes mancances de l'entorn de desenvolupament (probablement degudes a la meva inexperiència), he hagut de desenvolupar diverses funcions que apareixen en d'altres implementacions de les llibreries de C i que no hi són disponibles en l'entorn en el que s'ha desenvolupat:

strupr(), converteix una cadena a majúscules.

strcmpi(), compara dues cadenes de text sense tenir en compte les majúscules i minúscules.

strncmpi(), igual que l'anterior, però amb un límit en el número de caràcters a comparar.

Com ja hem apuntat abans, la llibreria printf permet utilitzar la funció printf() sobre una UART. Es podria haver afegit a la llibreria UART ja que depèn d'ella, però es va decidir deixar-la perquè forma part del procés de formació, tot i que no es fa servir a l'aplicació final.

4.2.2 Tasques

El programari d'un sistema encastat segueix el patró d'inicialització i bucle infinit. El sistema es para quan es talla l'alimentació i ha de ser capaç de començar quan aquesta reaparegui. La aplicació de la mota segueix aquest patró.

Al començament, s'inicien totes les estructures i els valors per defecte. La primera d'aquestes és un array d'estructures tipus _property on les funcions de cerca per nom, retornen l'adreça de l valor d'una variable. D'aquesta forma, es por modificar fàcilment la quantitat i el nom de les variables en funció de les necessitats de cada aplicació concreta.

En segon lloc, s'especifiquen les tasques que han de realitzar una funció de lectura, escriptura i compressió i s'omplen els tres arrays de tipus _task on les funcions de cerca de tasques poden retornar els paràmetres necessaris per a crear-la. D'aquesta forma, es poden afegir funcions que maneguin diversos protocols i sistemes de compressió també en funció de les necessitats.

A continuació, s'arrenca la tasca de control.

4.2.2.1 Tasca de control

Tot i que a l'anàlisi es va preveure un funcionament de les tasques com s'ha explicat a l'apartat anterior, la implementació ha estat una mica diferent degut a que no vaig poder resoldre el problema de la gestió de la memòria per part del sistema operatiu, ja que a en aquesta versió, no es gestiona bé la destrucció d'objecte que no tenen la mateixa mida. És a dir, no es pot destruir una tasca o una cua i tornar a crear una altra de mida diferent ja que no hi ha els

mecanismes adients per fer-ho i el sistema es bloqueja per falta de memòria. Per això, la decisió que vaig prendre va ser modificar la tasca de control per permetre una primera configuració que es torna immutable fins el reinici de l'equip. Una altra opció possible era predeterminar les tasques sense cap possibilitat de configuració i, la darrera opció era modificar cada tasca per a que s'executés una funció o una altra depenent de la configuració triada, però aquesta opció era inviable per manca de temps i queda com a millora.

La tasca de control implementada, segueix aquest diagrama de flux.

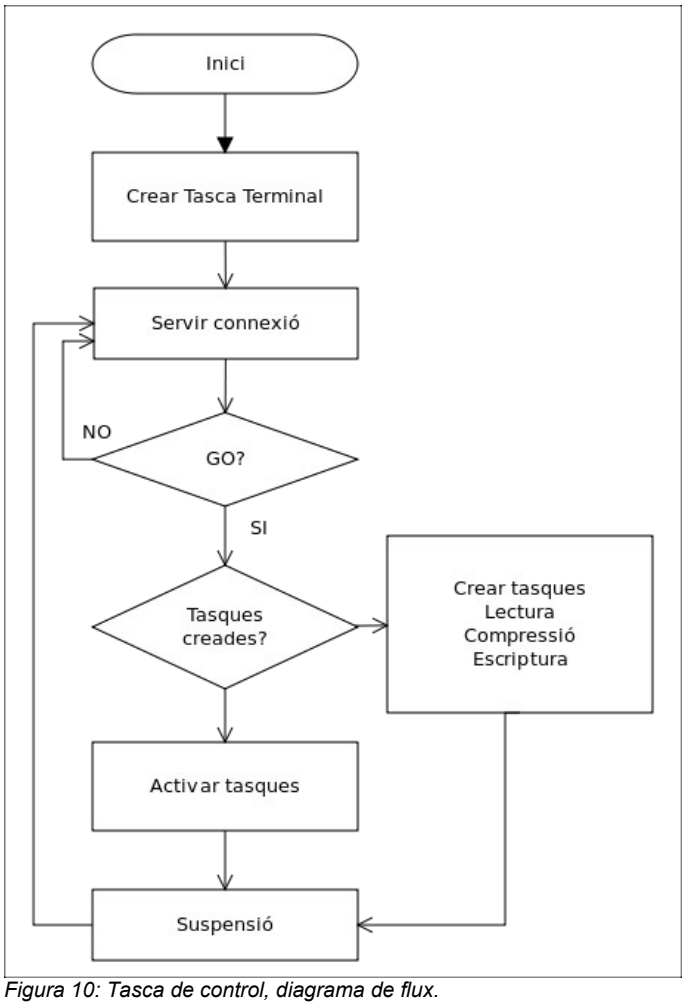

Figura 10: Tasca de control, diagrama de flux.

4.2.2.2 Tasca de terminal

La tasca de terminal té la funció de llegir les dades de la cua de terminal i comprovar que s'ha enviat una comanda correctament formada i tractar-la o, en cas contrari, enviar un missatge d'error.

El diagrama de flux és el següent:

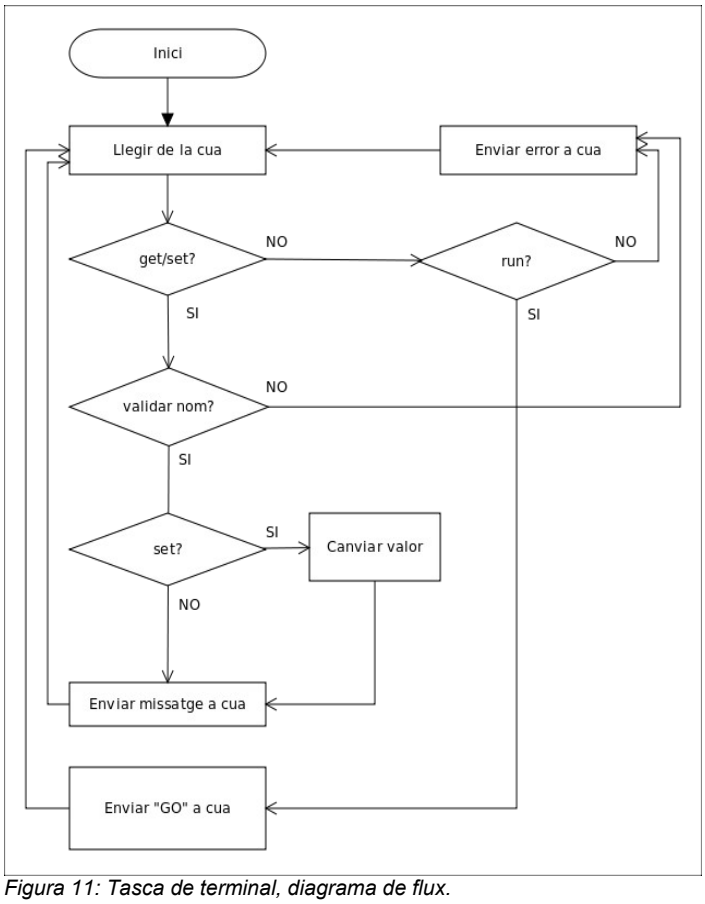

Figura 11: Tasca de terminal, diagrama de flux.

4.2.2.3 Tasca de lectura

La tasca de lectura implementada és la lectura HTTP cap a un servei web REST. Es troba en un bucle infinit on demana la informació del fitxer (mida en bytes) i llegeix el fitxer fins que arriba al final mitjançant la funció WiFlyFreeRTOSgetHTTPRequest() i escriu les dades a la cua de lectura. Per tal d'estalviar recursos, es fa servir la cua de la classe wiflyFreeRTOS. Això es una errada d'implementació, ja que l'aplicació no hauria d'utilitzar cap propietat d'un altre objecte, però com que aquesta funcionalitat correspon a la llibreria wiflyFreeRTOS, en el cas de poder integrar-la, no s'haurà de modificar el codi actual. És per això que es deixa documentat a l'espera de corregir aquests problemes.

El diagrama de flux de la tasca de lectura és:

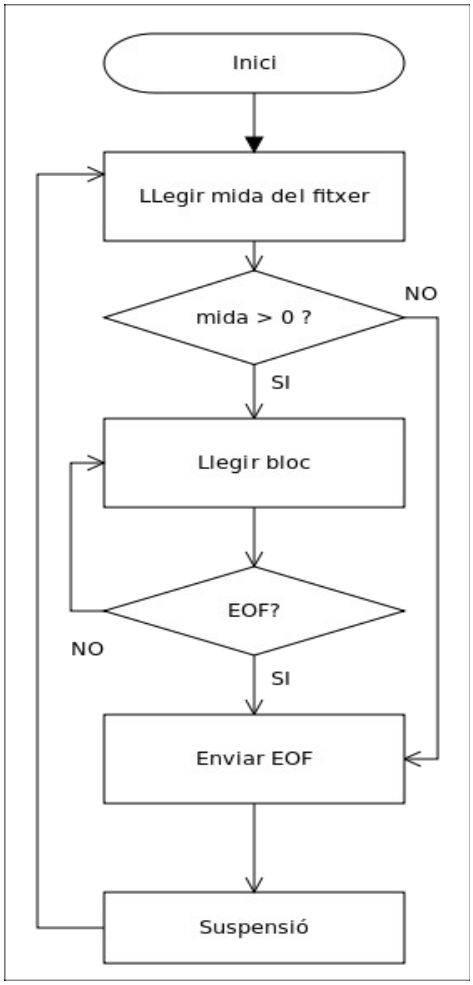

Figura 12: Tasca de lectura, diagrama de flux.

4.2.2.4 Tasca d'escriptura

Igual que la tasca de lectura, la d'escriptura llegeix de la cua corresponent, emmmagatzema en un buffer i, quan aquest s'ha omplert o bé s'ha rebut un EOF, s'envien les dades al servei d'escriptura.

El diagrama de flux és el següent:

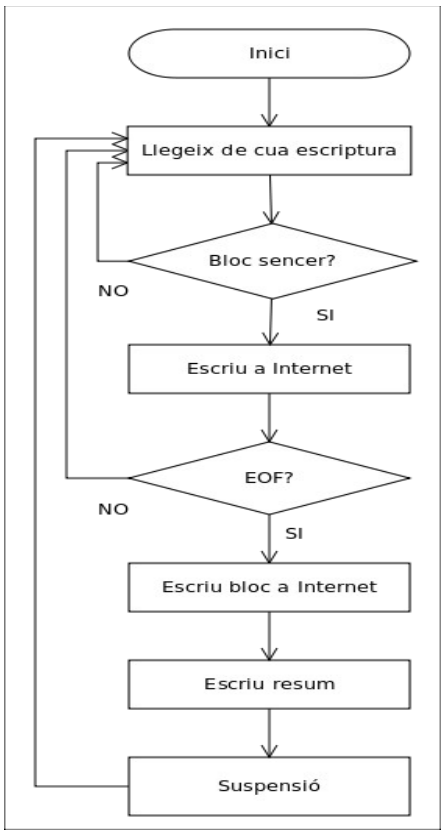

Figura 13: Tasca d'escriptura, diagrama de flux

4.2.2.5 Tasca de compressió

Tasques de compressió s'han implementat dues. La primera és una còpia de dades d'una cua a l'altra i es va desenvolupar per realitzar les proves de les altres tasques. Finalment es va deixar com a alternativa a una selecció de nivell de compressió 0.

L'altra és la que implementa la compressió deflate i el format gzip fent servir les funcions de la llibreria gzipLiteFreeRTOS, zip_init(), zip() i zip_stop().

El diagrama de flux que li correspon és el següent:

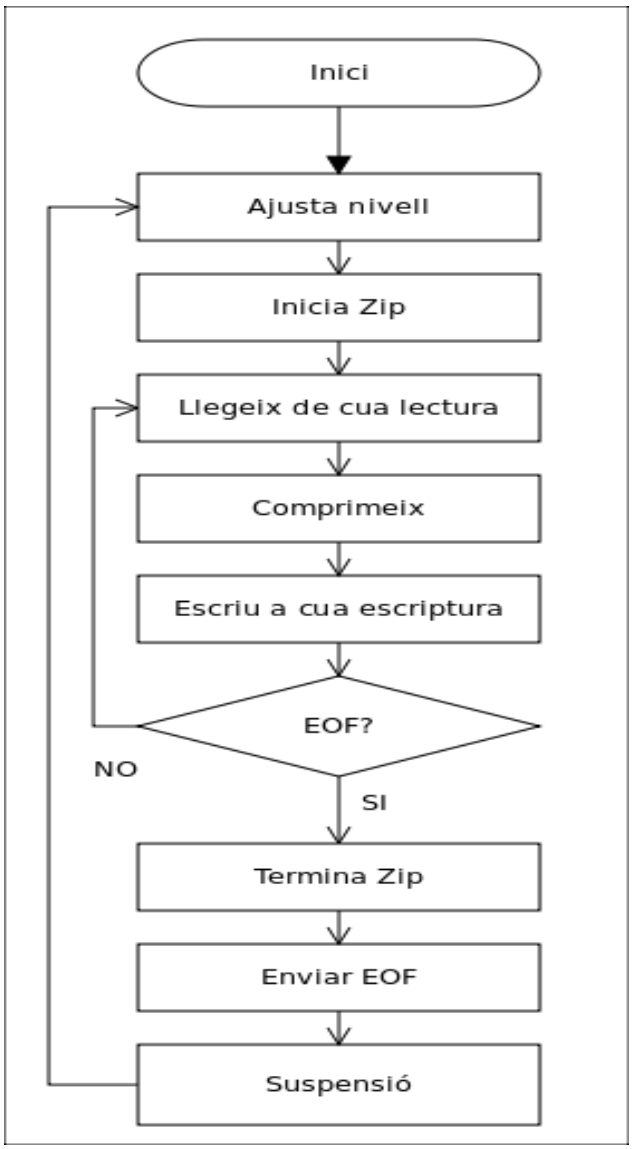

Figura 14: Tasca de compressió, diagrama de flux

4.3 Serveis REST.

Els serveis REST s'han implementat com a suport necessari ja que no hi ha serveis públics que compleixin els requeriments que es precisen.

En una primera implementació, vaig configurar una màquina virtual amb Lighttpd i PHP de les proporcionades per Turnkey Linux[1] que ofereixen un temps de posada en funcionament molt ràpid, a més d'estar completament preparades per a posar-les en producció amb tots els sistemes d'actualització necessaris. Amb aquesta màquina vaig desenvolupar els serveis, tot i la poca experiència que tinc en el llenguatge PHP. És per això que són serveis molt senzills. El fet de fer servir una xarxa domèstica sense fils per un projecte com aquest comporta un seguit

1 http://www.turnkeylinux.org/lighttpd-php-fastcgi

de problemes que una infraestructura més professional no hi dóna, ja que està més preparada per fer els encaminaments amb molta més facilitat, ja que té les eines per fer-ho. Per tal de solucionar l'inconvenient que representava tenir una màquina virtual a la que no podia accedir de la xarxa sense fils, vaig fer una cerca de proveïdors d'allotjament que disposessin de PHP i em vaig registrar a Hostinger[2] on em van assignar l'adreça http://jrufas.hol.es que és on es troba actualment la darrera versió del programari.

Passant per sobre de les consideracions de seguretat, a configuració és molt senzilla. Només s'ha de conèixer la ruta completa de la carpeta d'on es llegiran els fitxers i que el procés que dóna servei web tingui accés de lectura. Per una altra banda, s'ha de conèixer la ruta completa de la carpeta on s'escriuran els fitxers i tenir-ne accés d'escriptura. En el cas pràctic, la carpeta del servidor on s'allotja la pàgina principal és "/home/u146515205/public_html/" i aquí es van crear les carpetes "download" i "upload" com es pot veure a la figura següent.

Figura 15: Estructura de directoris del servidor REST

El **servei de lectura** comprova que el paràmetre wsInFile, que és obligatori no estigui buit i que existeix. En aquest cas, si els paràmetres wsInOffset i wsInBytes no hi són vàlids, es fixen a l'inici del fitxer, el primer i, a la mida del fitxer el segon. És a dir, si no s'especifiquen, es retornarà el fitxer sencer. A continuació es verifica la no existència del paràmetre wsGetInfo i es retorna el contingut sol·licitat com a application/octet-stream. En cas contrari, es retorna la mida del fitxer en format de text.

Si el fitxer demanat no existeix, o genera algun error, es retorna el missatge corresponent.

2 http://hostinger.es/

El **servei d'escriptura** té un funcionament similar quant al tractament dels paràmetres i el retorn de missatges d'error. Comprova si el fitxer existeix o si l'ha de crear i llavors, o bé converteix el paràmetre wsOutBytes de text en hexadecimal a un array de bytes i l'escriu al fitxer, o bé escriu el resum (wsOutTime i wsOutRatio) en format de text.

5 Viabilitat tècnica.

Actualment no és viable implementar el sistema en un entorn de producció. En primer lloc, perquè no té una utilitat com aplicació, ja que hi ha sistemes més senzills d'implementar i mantenir que donen més qualitat més ràpidament i, en segon lloc, perquè l'aplicació no ha estat testejada, i la memòria està pràcticament ocupada.

Una altra cosa és que pugui resultar interessant el poder controlar una mota des d'Internet i que el programari estigui desenvolupat i disponible. En aquest cas, es pot reutilitzar part d'aquest projecte per a complementar aquesta necessitat. D'altra banda, aquest projecte incorpora un bastiment de lectura, compressió i escriptura de dades que, degut a la seva flexibilitat pot ser adaptat a moltes situacions reals, tenint en compte l'ocupació de memòria que representa. Si es vol augmentar la qualitat de la compressió, s'haurà d'aconseguir estalviar memòria d'altres llocs que, probablement no siguin necessaris d'implementar.

Com a conclusió, podem afirmar que no és una aplicació útil, però que aporta unes solucions que es poden tenir en compte si es va a desenvolupar sobre un conjunt mota-WiFly i FreeRTOS.

6 Conclusions.

6.1 Conclusions.

A la vista del resultat es pot dir que s'han aconseguit els objectiu marcats, perquè he adquirit una experiència considerable en l'entorn dels sistemes encastats, i al llarg del projecte s'ha desenvolupat una aplicació que, tot i no tenir una utilitat pràctica, té una estructura que es pot reutilitzar en una amplia varietat de projectes, s'han portat i ampliat dues de les llibreries més completes que hi ha per a la gestió del mòdul WiFly, i para compressió de dades i s'ha desenvolupat un sistema de gestió remota per a FreeRTOS.

6.2 Proposta de millores.

Un cop desenvolupada i lliurada l'aplicació, es pot apreciar que no totes les decisions de canvis van estar encertades i que es podrien prendre d'una altra forma, tot i que aquesta afirmació no és objectivable, perquè dependrà, en cada cas de la part que interessi millorar.

De tota manera, la gestió remota s'hauria de realitzar mitjançant un altre mecanisme, perquè com s'utilitza la mateixa línia per comunicar-se amb l'usuari que per configurar el WiFly, aquest envia la mateixa informació per ambdues línies i resulta una mica confús, rebre els textos “exit” abans del missatge de petició de comandaments. Aquesta part hauria també que modificar-la perquè fossi més ràpida ja que la seqüència d'ordres prèvies fa que la resposta sigui lenta.

També s'hauria de millorar la gestió de l'estat del WiFly ja que, en situacions concretes no reconeix bé les ordres i triga més del compte en associar-se o connectarse.

Finalment, el que sí que s'ha de millorar és l'ampliació de protocols d'entrada i sortida, tipus de compressió i formats de fitxer.

6.3 Autoavaluació.

El redactor d'aquesta memòria es troba satisfet de la feina realitzada, i pensa que el treball té una qualitat acceptable, tot i les mancances que presenta. Creu que s'ha enriquit molt en un tema que havia estat molt de temps en el punt de mira del seu interès i que, tot que no pensa dedicar-se professionalment, sap que va a esdevenir en una de les seves aficions més importants.

7 Bibliografia.

IniciCortexM3

http://cv.uoc.edu/app/mediawiki14/wiki/IniciCortexM3

NXP

http://www.nxp.com/

LPCXpresso

http://lpcxpresso.code-red-tech.com/LPCXpresso/

Roving Networks

http://www.rovingnetworks.com/

Wifly Command Reference

http://www.rovingnetworks.com/resources/download/93/wifly_user_manual

FreeRTOS

http://www.freertos.org/

The FreeRTOS Reference Manual.

http://shop.freertos.org/FreeRTOS_API_and_Configuration_Reference_s/1822.htm

WiFlySerial

http://sourceforge.net/projects/arduinowifly/

miniz

http://code.google.com/p/miniz/

miniLZO

http://www.oberhumer.com/opensource/lzo/#minilzo

lzss

http://michael.dipperstein.com/lzss/

gzipLite

http://sourceforge.net/projects/gziplite/

Cómo configurar el módulo RN-XV

http://crlsgzblog.tumblr.com/

PHP

http://php.net/

PHP Classes

http://www.phpclasses.org/

Turnkey Linux

http://www.turnkeylinux.org/