

Projet Linux C++ embarqué

[git@github.com](https://github.com/caytro/LinuxCppEmbarque.git) :caytro/LinuxCppEmbarque.git Configuration locale :

QT Creator Qmake

Config : Qmake.pro

```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle
CONFIG -= qt
LIBS += -L/usr/lib/x86_64-linux-gnu/ -lcurl
SOURCES += \
    main.cpp
```

Installation composants :

- curl et libcurl (apt)
- boost (apt) pour json parser

Configuration qemu

make list-defconfigs

make qemu_aarch64_virt_defconfig → !! réinitialise le .config !!

make → machine de base

start :

hostfwd=tcp::5555-:22 transfère le port 5555 de localhost sur le port 22 de qemu

```
qemu-system-aarch64 -M virt \
-cpu cortex-a57 \
-nographic \
-smp 1 \
-kernel output/images/Image \
-append "root=/dev/vda console=ttyAMA0" \
-netdev user,id=eth0,hostfwd=tcp::5555-:22 -device virtio-net-
device,netdev=eth0 \
-drive file=output/images/rootfs.ext4,if=none,format=raw,id=hd0 \
-device virtio-blk-device,drive=hd0
```

make xconfig : (make et .start à chaque étape)

- **Enable C++ support** (BR2_TOOLCHAIN_BUILDROOT_CXX)
- **libcurl** (BR2_PACKAGE_LIBCURL)
- **Path to the users tables** (BR2_ROOTFS_USERS_TABLES)

- myConfig/users.txt
 - sylvain 1000 sylvain 1000 =password /home/sylvain /bin/sh - User
- **Root filesystem overlay directories** (BR2_ROOTFS_OVERLAY)
 - overlay/
 - /home/user et /root (remplacer \$ par #):
 - .profile
 - export PS1="\u@qemu:\W/\$"
 - exemples :

```
etc home root usr var

./etc:
azerty.kmap  init.d

./etc/init.d:
S70kmapFrench  S80Cron

./home:
sylvain

./home/sylvain:

./root:

./usr:
bin  share

./usr/bin:
progarm

./usr/share:
zoneinfo

./usr/share/zoneinfo:
Europe

./usr/share/zoneinfo/Europe:
Paris

./var:
spool

./var/spool:
cron

./var/spool/cron:
crontabs

./var/spool/cron/crontabs:
root
```

- **openssh** (BR2_PACKAGE_OPENSSH) → tout cocher
- **json-for-modern-cpp** (BR2_PACKAGE_JSON_FOR_MODERN_CPP)
 - nlohmann
- **boost-filesystem**

Utilisation de curl (C) pour requete HTTP GET :

```
#include <iostream>
#include <curl/curl.h>
#include <memory.h>
using namespace std;
struct response {
    char *memory;
    size_t size;
};

static size_t
mem_cb(void *contents, size_t size, size_t nmemb, void *userp)
{
    size_t realsize = size * nmemb;
    struct response *mem = (struct response *)userp;
    char *ptr = (char *)realloc(mem->memory, mem->size + realsize + 1);
    if(!ptr) {
        /* out of memory! */
        printf("not enough memory (realloc returned NULL)\n");
        return 0;
    }
    mem->memory = ptr;
    memcpy(&(mem->memory[mem->size]), contents, realsize);
    mem->size += realsize;
    mem->memory[mem->size] = 0;
    return realsize;
}

int main()
{
    cout << "Hello World!" << endl;
    struct response chunk = {.memory = (char *)malloc(1),
                             .size = 0};
    CURL *hnd = curl_easy_init();
    curl_easy_setopt(hnd, CURLOPT_CUSTOMREQUEST, "GET");
    curl_easy_setopt(hnd, CURLOPT_URL, "https://api.ambeedata.com/latest/by-lat-lng?lat=43.560537&lng=1.404690");
    struct curl_slist *headers = NULL;
    headers = curl_slist_append(headers, "x-api-key: b83fcfd7137ff81d96b92a34d3488506b7d3976bda58077cab133e94efd0a240");
    headers = curl_slist_append(headers, "Content-type: application/json");
    curl_easy_setopt(hnd, CURLOPT_HTTPHEADER, headers);
    curl_easy_setopt(hnd, CURLOPT_WRITEFUNCTION, mem_cb);
    curl_easy_setopt(hnd, CURLOPT_WRITEDATA, (void *)&chunk);
    printf("Before : \n");
    CURLcode ret = curl_easy_perform(hnd);
    cout << "Retour";
    printf("Retour : %d\n", ret);
    printf("Chunk : %s\n", chunk.memory);
    free(chunk.memory);
}
```

Sortie :

```
Retour : 0
Chunk : {"message":"success","stations":
[{"CO":0.234,"NO2":16.543,"OZONE":12.586,"PM10":12.853,"PM25":2.6,"SO2":0.661,"
city":"Toulouse","countryCode":"FR","division":"Haute-
Garonne","lat":43.556374,"lng":1.403964,"placeName":"Avenue de
Larrieu","postalCode":"31100","state":"Occitanie","updatedAt":"2023-09-
14T07:00:00.000Z","AQI":16,"aqiInfo":
{"pollutant":"NO2","concentration":16.543,"category":"Good"}}]}
```

Création d'une classe myCurl pour gérer les accès à l'API.

Utilisation de Nlohmann/json.hpp pour parsing

Doc : <https://github.com/nlohmann/json> ou
https://json.nlohmann.me/api/basic_json/

Utilisation très simple et intuitive.

Création d'une classe myParse basée sur cette lib.

Compilation croisée qemu

Dans le dossier devel/AirQualityWatch

```
.sylvain@server-bureau:~/ajc/formation/LinuxEmbarque/Projet/devel/
AirQualityWatch$ cat crossCompil
#!/bin/sh

../../buildroot/buildroot-work/output/host/bin/aarch64-buildroot-linux-gnu-g++
-o airQualityWatch-qemu main.cpp mycurl.cpp myparsing.cpp -lcurl

cp ./airQualityWatch-qemu ../../buildroot/buildroot-work/overlay/root/bin
```

Problèmes :

g++ :

Lors de l'installation de jsonpp, erreur : Manque le fichier
output/host/bin/aarch64-buildroot-linux-gnu-g++

Il faut compiler le compilateur g++.

Tentative : **Force the building of host dependencies**

(BR2_FORCE_HOST_BUILD) → toujours pas de fichier output/host/bin/aarch64-
buildroot-linux-gnu-g++'

Solution : Redémarrer à partir de 0 en ajoutant support C++ avant la première
compil. Semble fonctionner, fichier présent → **Fixed**

Persistence des dossiers de l'arborescence de la machine virtuelle qemu :

Les datas stockées dans /tmp ont été perdues après poweroff et start. Après tests, les données stockées en cours d'utilisation dans /home persistent après poweroff. Modification du source C++ afin de stocker les datas téléchargées dans /home/sylvain/datas

<filesystem> lib n'est pas dans buildroot :

Utilisation de la lib boost/filesystem. Préférée à LUA car il semble que boost propose plus de libraries, en particulier une lib graphique.

TODO

- Choix d'un lib Graphique
- Configurer cron dans buildroot pour récupération data sur l'API, sauvegarde et m à j des graphiques
- Stocker API-KEY dans un fichier pour pouvoir la mettre à jour (doit être renouvelée tous les 15 jours)
- Choisir et paramétrer un server HTTP permettant d'afficher les graphs dans une page web

Classes

myCurl

Utilise la lib <curl/curl.h> pour gérer les requêtes HTTP GET à la source des données.

Nécessite -lcurl à la compilation.

myParsing

Utilise la lib <nlohmann/json.hpp> pour gérer le parsing, les lectures et écritures disque des datas au format Json

myGraphics

Utilise les libs

- `<gd.h>`
- `<gdfontt.h>`
- `<gdfonts.h>`
- `<gdfontmb.h>`
- `<gdfontl.h>`
- `<gdfontg.h>`

pour générer les graphiques.

Nécessite `-lgd` à la compilation.

myOptions

Permet de modifier le paramétrage de l'application à partir d'un fichier json modifiable dans le dossier `overlay/root/AIQWA.config` de buildroot.

```
{
  "curlopts":{
    "url": "https://api.ambeedata.com/latest/by-lat-lng",
    "urlParams":{
      "lat":"43.560537",
      "lng":"1.404690"
    }
  },
  "api-key": "x-api-key:b83fcfd7137ff81d96b92a34d3488506b7d3976bda58077cab133e94efd0a240",
  "filesOwner":sylvain",
  "logPath":"/home/sylvain/logs",
  "logFileName":"AIQWA.log",
  "dataPath":"/home/sylvain/datas",
  "dataFileName":"AirQualityWatch",
  "chartPath":"/home/sylvain/charts",
  "curveChartFileName":"curve.png",
  "histoChartFileName":"histo.png",
  "pieChartFileName":"pie.png",
}
```