KULGRAM REVERSING.ID

Bare Metal C++

Pemateri : Wijaya adhisurya

Bare Metal C++ artinya adalah program C++ yang berjalan di atas prosesor langsung

Tanpa ada OS sama sekali.

Umumnya bare metal program adalah embedded program.

Karena latar belakang saya adalah embedded software engineer, saya akan khusus berbicara soal bare metal di embedded system.

Kalo misalkan di tengah-tengah ada pertanyaan, dipersilakan.

Saya akan menjawab sesuai dengan yang saya tahu

Menurut pengalaman saya, kebanyakan embedded system software ditulis dengan menggunakan bahasa C.

Ada juga yang mengaku bahwa dia menulis dengan C++, tapi banyak dari fitur2 C++ tidak dipakai

Sebagai contoh ketika seseorang memakai wiring library, sebenarnya dia sedikit memakai C++, tapi jelas programming model dengan satu fungsi panjang itu sebenarnya adalah gaya pemrograman C

Dalam artian, belum banyak memanfaatkan fitur2 berorientasi objek dari C++

Implementasi dari wiring library yang paling terkenal pastinya sudah banyak yang tahu.

Dia dipakai di Arduino dan RPi

Di RPi, namanya adalah wiringpi

http://wiringpi.com/

Ada juga implementasi khusus untuk STM32

Namanya STM32duino

Kita pernah bikin prototipe CAN to USB adapter dengan stm32duino,

tapi diganti memakai HAL standar (STM32 Cube) karena masih kurang cepat

(belum bisa mencapai 1 MBit /s)

Ok, kembali soal C vs C++

Mengapa harus memakai C++?

Tujuan utamanya adalah code organization

Untuk memisahkan bagian2 berbeda dari embedded software

Bisakah kita memakai C saja tanpa C++?

Tentu saja bisa...

Namun, jika kita sudah memahami pentingnya mengelompokkan kode, lama-lama kode kita akan mengarah ke object oriented

Dengan demikian, daripada mengemulasi object oriented programming menggunakan bahasa C, lebih baik kita langsung memakai C++

Namun, ada perkecualian di mana kita tidak perlu menerapkan object orientation sama sekali

Misal, ketika kode kita di bawah 500 baris

Malah jadi ribet.

Sampai di sini ada yang ditanyakan?

ok, saya lanjut ya...

kemudian, batasan embedded system itu seperti apa sih?

Sebenarnya, ini kadang sering jadi perdebatan, karena memang tidak ada definisi resmi apa itu embedded system

Embedded System itu sangat luas

dimulai dari 8 bit prosesor dengan 512 bytes RAM, sampai RPi juga masih sering ada yang menyebutnya sebagai embedded system

embedded system kalo menurut saya, karakter utamanya adalah hard realtime.

Jadi, dia tidak harus cepat

Tapi timing-nya sangat strict

Karena itu, biasanya ketika sudah memakai general purpose OS semacam Linux, constraint untuk hard realtime jadi semakin turun, alias makin tidak akurat

Sebagai contoh, di motor control yang kita kembangkan, deadline-nya adalah 100 us.

Kalo telat memproses, ada control loop yang terlewati.

Yah, boleh lah kalo RAM 8 MB, prosesor 500MHz masih dianggap ke dalam embedded system. Tapi itu sebenarnya udah terlalu besar.

Karakteristik kedua dari embedded system adalah akses langsung ke hardware

Kita bisa langsung mengakses sensor, motor, LED, atau apapun itu tanpa harus melalui perantara yang cukup panjang

Karena itu, embedded system punya beberapa peripheral khusus yang bisa diakses langsung dari program kita

Ciri lain dari embedded system adalah, standard library dihilangkan ataupun kalau ada di-strip down habis-habisan

Misal, tidak ada yang namanya printf, cin, dan cout

Karena tidak ada terminal yang dipakai untuk menuliskan ke layar.

sampai di sini ada yang ditanyakan?

ok, kalau begitu saya lanjut lagi

Tantangan apa saja ketika kita memprogram embedded system, bahkan ketika kita sudah punya development kit sekalipun?

Tantangan 1: Pasar hardware embedded system sangat fragmented

Ada ARM yang cukup besar, ada PIC, Ada AVR, Ada TI DSP yang saya pakai, ...

Masing2 adalah keluarga prosesor yang berbeda-beda

Masing2 punya karakteristik yang beda2 dalam cara memprogramnya

Bahkan sesama ARM dari beda manufaktur pun punya cara yang berbeda-beda dalam cara mengimplementasikan protokol yang sama

Contoh, CAN peripheral

Nah, bahkan dari keluarga prosesor sama, dari manufaktur yang sama, kadang beda prosesor beda sama sekali cara mengaksesnya

Karena itu, masing2 keluarga prosesor punya toolchain sendiri-sendiri

Memang ada secercah harapan dari GCC dan CLANG

Mereka sudah support MSP430 dan ARM Cortex-M

Tapi, kayak platform populer semacam ESP32 belum disupport sama GCC dan Clang

Kita harus pakai toolchain official dari mereka

Nah, karena compiler dan toolchain ini ada banyak, kita juga tidak bisa sembarangan memakai C++ modern di sini

Setau saya, baru ARM Cortex-M yang bisa support C++ modern

Itupun gara2 GCC

Tantangan lain lagi adalah, dukungan SDK dari pabrikan sendiri tidak selalu bagus

Jangan heran kalau suatu saat kalian menemukan compiler bug

Jangan heran juga, kalau SDK dari pabrikan itu ternyata tidak rapi dan tidak portable

Maksudnya portable di sini bisa dipindah ke prosesor baru dari keluarga yang sama dengan gampang

Kadang, saking malesnya pabrikan, mereka hanya bikin struct 2 yang ngarah ke register, memakai bitfield.

Dalam hal ini, saya cukup salut dengan "Atmel Software Framework" dan STM Cube, yang paling tidak mau merapikan SDK-nya sehingga bisa membantu kita para embedded software engineer

Nah, tantangan dari embedded software ini sangat banyak, mengapa tidak memakai Arduino saja yang gampang?

Saya setuju kalo kita memang harus memasyarakatkan Arduino.

Arduino ini menurut saya bukan mainan untuk para software engineer yang pengen belajar hardware

Melainkan sebaliknya.

Mainan para electronics engineer yang gak mau ribet soal programming

Ngoprek hardware, bikin shield, tancep di arduino, jalankan...

If it works for you, then use it...

Namun, ketika kita membutuhkan untuk hard realtime dan tight deadline, Arduino tidak menyelesaikan masalah

Apalagi, Arduino API-nya untuk mengakses komunikasi kebanyakan adalah blocking.

Dan I/O itu secara umum lambat

Di kantor saya, saya mengimplementasikan sendiri pemrosesan data komunikasi secara asynchonous, karena itu menghemat waktu prosesor banyak sekali

1.3 ms ilang, prosesor di busy loop, padahal seharusnya bisa untuk melakukan yang lain

I/O yang cepet ada juga sih, ADC sama GPIO misalnya...

nanti kita bahas soal itu.

Soal Arduino ini, sayangnya ada lembah curam antara setelah ngoprek Arduino, terus kemudian tiba2 langsung pake AVR-nya

Kalo yang sudah biasa dengan yang lebih susah sih, tidak masalah. Justru Arduino mempermudah pekerjaan kita.

Pertanyaan!

Satria Ady Pradana
ESP32 pake xtensa kan ya?

Wijaya Adhisurya
Yap,

Satria Ady Pradana
kalo gak salah pas oprek Arduino IDE dengan ESP, aku liat toolchain yang dipake Xtensa elf dari GCC, berarti itu bukan di-maintain sama community?

Wijaya Adhisurya
wait, ada ya?

Satria Ady Pradana
ada

Wijaya Adhisurya
Syukurlah kalo sudah ada.

Clang yang belum ada...

Satria Ady Pradana
tripletnya sih xtensa-lx106-elf

Wijaya Adhisurya
Kalo sudah ada kan, lumayan, bisa bermimpi coding Kotlin di ESP32 :grin:

Satria Ady Pradana
cuman gak tau apakah itu community ato dari vendor

Wijaya Adhisurya
Nice, ESP32 masuk daftar must buy artinya

Wijaya Adhisurya
ok...

selanjutnya, kita masuk ke bagian general embedded system architecture

Nah, apapun keluarga prosesor-nya, biasanya tetap memakai komponen2 ini.

Tantangannya memang akan beda di cara memprogram masing-masingnya

Ada embedded system peripherals itu kira-kira bisa dibagi menjadi empat bagian:
1. Actuators
2. Sensors
3. Communication
4. Others

actuators sebenarnya adalah nama lain dari output...

kita menulis sesuatu di register prosesor, maka prosesor akan melakukan sesuatu

Ada 3 macam actuators yang sering dipakai...

GPIO, yang meneruskan logic high atau logic low keluar. Misal, ketika digunakan untuk menyalakan dan mematikan LED

PWM, atau pulse width modulation, yang mengeluarkan pulsa periodik dengan lebar tertentu.

DAC, atau Digital to Analog Converter, yang meneruskan voltage analog keluar...

PWM ini sering disebut sebagai poor man's DAC, karena dia bisa dipakai untuk menggerakkan motor misalnya

Syaratnya, load-nya capacitive...

Atau, jika frekuensi PWM-nya tinggi, bisa saja seolah-olah seperti DAC...

Seperti misalkan LED

itu sebenarnya bukan redup, tapi kedip terus

kalo pas terang, baru itu nyala terus

http://www.waitingforfriday.com/?p=404

Sementara untuk DAC, voltage yang dikeluarkan adalah analog dan dia datar, bukan berbentuk pulsa periodik seperti PWM

Peripherals jenis kedua adalah sensors, di mana embeded software membaca masukan dari dunia luar untuk diterjemahkan sebagai bilangan digital

Sensor digital (on/off) bisa dengan memakai GPIO

GPIO bisa dipakai sebagai masukan atau keluaran, tapi pas pertama kali program dimulai, harus ditentukan apakah kita mau memakainya sebagai keluaran atau masukan

Setelah GPIO, ada lagi yang namanya input capture

Ini tidak terlalu populer, tapi sering ada di mikrokontroler yang agak kompleks

ini kebalikan dari PWM

Dia menangkap sinyal serupa PWM

Dan yang ketiga, guess what ...

ADC, Analog to Digital Converter :grin:

Kebalikan dari DAC

Dia menangkap sinyal analog untuk diterjemahkan menjadi bilangan digital

Peripherals jenis ketiga adalah communications

Kalo hobbyist, yang paling umum dipakai adalah UART, I2C, dan SPI

Kadang-kadang ada CAN juga, tapi chip yang ada CAN biasanya lebih mahal

communications ini untuk meneruskan data dari prosesor keluar prosesor, biasanya ke device lain atau ke external peripheral

kalo misalnya di sistem kita butuh semacam "hard disk", kita bisa pakai EEPROM.

Nah EEPROM ini adanya di luar prosesor, bisa pakai I2C atau SPI

Bahkan, sering juga sensor interface-nya bukan analog tapi digital melalui komunikasi

Misal, melalui SPI atau I2C

Detail tentang keempat interface komunikasi ini bisa dibaca sendiri ya, kalo dijelaskan di sini terlalu panjang. :grin:

Pada intinya, keempatnya adalah serial communication protocol

Serial maksudnya, data line yang dipakai hanya satu wire saja...

Jadi misalnya ngirim satu byte dengan cara memberikan pulsa hi/lo dengan format tertentu ke satu jalur wire/kabel.

khusus untuk UART, dia memiliki jalur sendiri antara kirim dan terima, sehingga bisa saling mengirim tanpa harus bergantian

kelemahannya, dia cuma bisa komunikasi antara kedua belah pihak

sementara SPI, I2C, dan CAN, node2 yang bisa berkomunikasi bisa banyak, tapi harus bergantian

Peripheral jenis terakhir, others atau lain-lain

Salah satu contohnya adalah DMA...

Tugas DMA adalah memindahkan data dari satu alamat memori / peripheral ke alamat memori / peripheral yang lain, tanpa campur tangan prosesor

halo @adipginting

Atau misalkan koprosesor untuk kriptografi...

Karena biasanya kriptografi itu mahal, biasanya dibuatkan sirkuit logic sendiri di luar prosesor

Nah, selain peripheral, ada juga core dari prosesor itu sendiri

Programming model di embedded system itu gampang banget

Kita ndak perlu mikir apakah itu userspace, kernelspace

virtual memory segala macem...

Karena biasanya, semua alamat memori langsung bisa directly addressable

Jadi, address space ini biasanya yang sering kita sentuh ada 3: flash, RAM, sama peripherals

Semuanya cuma tentang membaca dan menulis ke alamat memori

Di mana perlakuan2 khusus ada ketika kita mau mengontrol peripheral.

karena peripheral register ini bisa berubah sendiri nilainya, maka harus selalu diakses dengan volatile...

oke, sebelumnya saya minta maaf, karena C++-nya ternyata cuma 25%

malah banyak embedded-nya

Marilah kita beranjak ke C++-nya...

Coding style di embedded C++ itu seperti apa...

Seperti saya singgung sedikit tadi, tidak ada cin dan cout di embedded...

Kalopun ada, itu biasanya dibungkus, diarahkan ke UART...

Beruntunglah para umat Arduino yang punya Serial.print

Nah, kadangkala, kita harus mengimplementasikan data struktur kita sendiri...

Kayak queue misalnya

Suatu saat saya butuh priority_queue untuk bikin timer sequencing

ternyata, kita saya tarik priority_queue dari STL itu ke program saya, ukuran programnya membengkak bertambah 20KB...

Ketika saya bikin sendiri priority_queue pakai linked list, ukurannya jadi jauh lebih kecil

jadi, hati-hati pake STL di embedded

Terus, kesalahan umum yang terjadi adalah biasanya tidak suka memakai dynamic polymorphism/inheritance di embedded

Ya, memang ada yang namanya indirection ketika kita memanggil vtable, tapi itu nggak seberapa

Karena, saya pernah mengganti dynamic polymorphism dengan static polymorphism memakai template

ternyata "cuma" lebih lambat 5-7%

Demi kalo ada orang baru tidak garuk2 kepala liat kode template, lebih baik pake inheritance

Terus, saya juga pernah ingin menghilangkan pointer ke register dengan memakai alamat register jadi template parameter.

Saya pengen kode high-level saya sama seperti hand-coded assembly

Hasilnya adalah, kodenya jadi membengkak juga

Untuk tiap alamat register, harus ada kopian dari fungsi yang mengakses register tersebut

huge space loss for marginal execution time gain

Nah, template tidak selamanya buruk...

Kalo anda kebetulan bertemu dengan penulis SDK yang kampret dan pengen bikin register map sendiri sesuai dengan datasheet, hal ini bisa dilakukan

Ini adalah cara untuk mengakses register secara type safe dan portable

https://yogiken.files.wordpress.com/2010/02/c-register-access.pdf
30 KB

Jadi, asyik-nya metode ini, dia memisahkan antara write only, read only, dan read write access...

Jadi, ketika kita menulis ke alamat yang read only, langsung compile error

Lalu, bagaimana dengan exception?

Exception adalah hal yang tidak disukai di embedded system, dan di banyak compiler tidak di-support

Alternatif dari exception memang kurang menarik sih, kita harus pakai error code, dan setiap fungsi harus kita cek error code-nya...

Terakhir, embedded system cenderung tidak menyukai dynamic memory allocation, walaupun kadang2 diperlukan

Ini karena dynamic memory allocation menyulitkan static analysis

Ketika program kita di-disassembly, sulit untuk nyari variabel apa alamatnya di mana

juga, ketika kita semuanya pake static allocation, kita bakalan tau saat program kita ukurannya terlalu besar

stack sudah dikasih kapling sendiri, jadi ketika dicompile gak bertabrakan dengan variabel global

di tempat kerja saya yang dulu, kita bahkan mengganti semua new dengan memory pool...

Akibatnya jadi ribet

karena semua klas harus dikasih new operator khusus

another tradeoff in convenience for marginal gain...

Tidak apa2 memakai new kalo memang perlu.

Cuma, kalo bisa dihindari lebih baik dihindari...

Ya tapi tidak perlu sampai bikin memory pool sendiri juga, kecuali memang kondisinya sangat khusus

misal, kita takut akan banyak terjadi fragmentasi

sehingga bisa cepet habis heap-nya

Sekian dari saya...

Ada yang ingin ditanyakan?

kalo tidak ada, saya kembalikan ke @xathrya , terima kasih