

2nd session

* virtualization Vs emulation *

Hardware * ملحوظة virtualization هي تخصيص الـ resources على أكمل وجه من operating systems، بحيث يتصرف operating systems كـ guest OS مثل كـ host OS ليكون حاسماً في virtualization بتحمل.

* ببسالة السؤال المهم، هل guest OS يقدر بـ run مثل instructions على x86 Assembly؟ الإجابة لا، كل Guest architecture instruction set يلتزم بالـ main device أو الـ host الذي يديره.

* حوار تخصيص resources بين guest و host هو hypervisor الذي يحول ذلك ما يحوله.

* baremetal hypervisor على الحدودية، وهو شكل من hypervisor الذي يخلي guest OS من interaction directly hardware، وهو الذي يخلي guest OS من interaction hardware resources.

* guest run على baremetal hypervisor، أما بيجي أي instruction run على hardware، فهو instruction على guest OS، ببسالة من تخصيص طول.

* النوع الثاني هو type 2 hypervisor، وهو زكي الذي يشحنه في guest OS مثل host operating system like VirtualBox.

* حملة emulation دى بقى بتحمل لو انت عايز تRun
بسكل لاد يعني كالhardware، فبتجيبي Emulator، hardware لhardware من حاكاة ذات

* كمثال لو انت عايز تRun ARM instruction على x86
زى ال SVC اللي بتديله رقم ويكمله interrupt SW، والـ handler بيتحدد على حسب الرقم ده.

* او انت عايز تحاكى hardware لسى من موجود في الحياة اتس

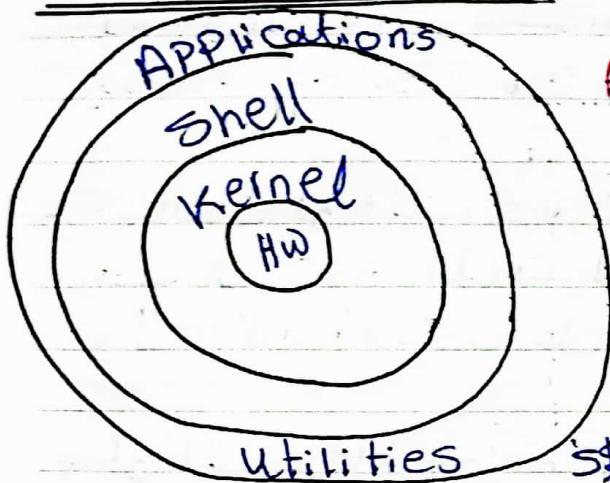
* فهو بيعاكى بين اتنين حاس، سوا يحمل assembly mapping او يحالف حاس اكده موجود instructions

* اشهر Emulator موجود هو Q-emulator open source وهو يدعم virtualization support

* emulators بيردو خلى بالك بيأخذ وقت كبير، ويكون supported by HW من لازم بقى بالعكس ملعن دعوة بالhardware

* بالنسبة لل Linux history ابغى ~~30:40~~ لحد الدقيقة 30:40 في اول فيديو في بلاي ليست الشهور من YouTube او

Linux Architecture :-



A-Kernel :-

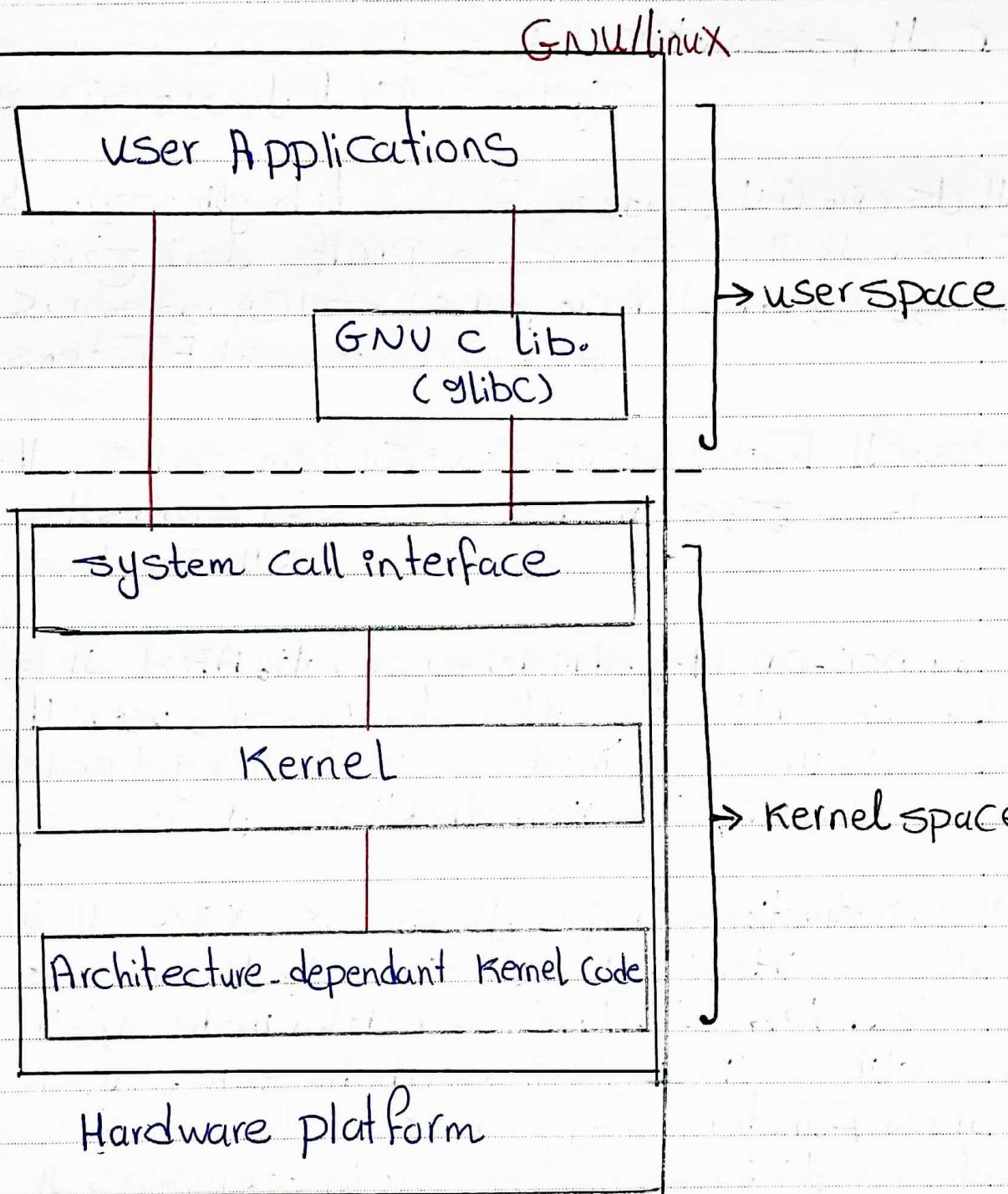
* الـ Kernel موجودة خوفاً على Hw ، كل طول ، فـ bios يبقى فيها hardware dependants.

* بِرْد و بِيكُون فِي جَزْء hardware sِcheduling لِـ independent فِي Algorithm CS أَو LS o - Algorithm Architecture لِـ SIC/S

* زع مایت فاکر، ال RTOS فی (autosar OS) autosar بیکوون فی میت ۷۰٪ او ۹۰٪ ده کود ثابتے عفیش ای تکنیس بیحمل علیه، والباقي بیتکنیس بی تحین ال MC لانک بیتکنیس بقی تکنیل Porting لـ OS علی الـ MC ره

* ال Kernel ڪسنا اقدر انعامل هئا، بڪل ال Kernel و هڪن اوچيلو directly کارئ، وکل ما هو داخل Kernel Space یا Kernel ما هو خارج، user space یا

Kernel Space Vs User Space :-



User Space:-

* اى حد يجيء يتكامل مع kernel يستخدم user space و يوصل لل system calls

* طبعاً lib glibc موجودة لأن من كل الناس يستخدم printf و scanf من الأول فما كان كده موجودين ليه أحكام من الأول، ففيها حيث انت ممكن تستخدمها functions

* ال processor الـ Function calls دى من user space

في الحقيقة يكون عند برامجاتي ~~modes~~ برامجاتي ~~privileges~~ برامجاتي ~~resources~~

* في الـ ARM هنالك mode، ويند فيه non-privileged mode، ويند فيه user ويند فيه modes كل mode بيدليه صلاحيات مختلفة

فانت بتدخل جوا kernel mode في الـ PAL بس

* في الـ x86 كانت بتدخل بـ trap instructions، في الـ ARM كانت بتدخل بالـ SVC interrupt

تحتاج لـ DAL و PAL من mode، فتحتاج لـ abstraction من glibc و تتحمّل responsibility من application

تحتاج لـ rubbers functions التي انت تحتاجها عشان تدخل في الـ kernel operations mode

Note:-

اوقات يحتاج ارجح لل System call glibc و مروجت لل Kernel او حاجات سطح فمشن
لو انا قيمل متلا String compare او حاجات Kernel محتاج ادخلها

Look @ slide #40:-

ال Kernel موجود جوايا بعده Subsystems بعدها

Linux distributions:-

* رى تجميئه بقى Linux Kernel و System libraries (ls) Ubuntu الى زى device drivers (ls) Yocto او Fedora او Linux mint او

* طبق اىاس متوفى ال Embedded Linux يكملوا اىس؟ حوار انك
تجتمع ال Linux distributions ده مقرف شويفت بس ان
ممكن يقع في bug في كيبيت لـ Kernel متلا
Combatable من glibc الى version او او
Yocto لها automated tool ، فعن kernel
مدمولته بال Python بتمالى الحوار ره و في tool
Yocto buildroot بس
اـ حل منها شويفت اـ لها
اـ أكثر استخاداً لأن فيها تفاصيل كثيرة و ممكن تكمل بها
ـ لـ ubuntu او OS

Memory

revision on build process & ~~data~~ segments:-

* بعدين أنا أعرف إنك من ذاكر الكلام ده كويسي بس تناهى
نراجع عليه شوية كتناي محتاجه لمن وجهته نظر
(embedded)

data memory segments (RAM) :-

A - Memory mapped peripherals reg :-

* روى ال MC معمولها registers لـ RAM
Accessing it، بحيث إن الـ MC بـ mapping
ـ RAM بـ bus، والتي يمكنها الوصول
ـ MC will copy data to RAM

RAM
Memory mapped
peripherals
registers

• data

B - .data segments:-

initialized و initialized global أو static

• bss

• heap
↓
• stack

C - .bss segment :-

* رو فيديو الـ uninitialized global أو zero
uninitialized أو zero انت و نميسلي، والـ garbage
ـ static الـ zero ديمى ديمى

* وكمان فيديو الـ static و global أو zero

D - .heap :-

Dynamic allocation

E - .Stack :-

temp data - local var - Func. arguments - CPU context switching

Note :-

في الـ stack والـ heap الـ PC في الماوسين ،
في Local memory | heap في بعض الأحيان يُسمى Local
كثير من ES في

Program memory :-

A - Interrupt vector table segments :-

Addresses of ISRs & exists at the beginning of the program memory (mostly)

Interrupt vector table segment

.text(code)

.rodata

.data

B - .text (code) segment :-

Contains the code ~~inst~~ & it is the biggest segment

C - .rodata segment :-

Contains the Constant global variables ROM

D - .data segment :-

The initialization of static & global variables
→ initial value

#Build process

لما يجي ايدل بيلد لكود ال C بعدى على 4-Blocks أمست اللى دا
→ Preprocess → Compiler → Assembler → Linker.

1 Preprocessor -

ـ **Block** المسؤول عن كل الحالات التي في **handle**
ـ **الى اسماها pre-processor directive**
ـ **Text replacement** **Text preprocessing** المسؤول عن كل

حال output اولی بیخروج من بعد مرحله ال pure C code preprocessor کو کور مفهومیت ای hash # و انت مفهوم المفروض کل ال h-File هی عباره عت برروی Text replacement یعنی لاما نتکار فی اول کور

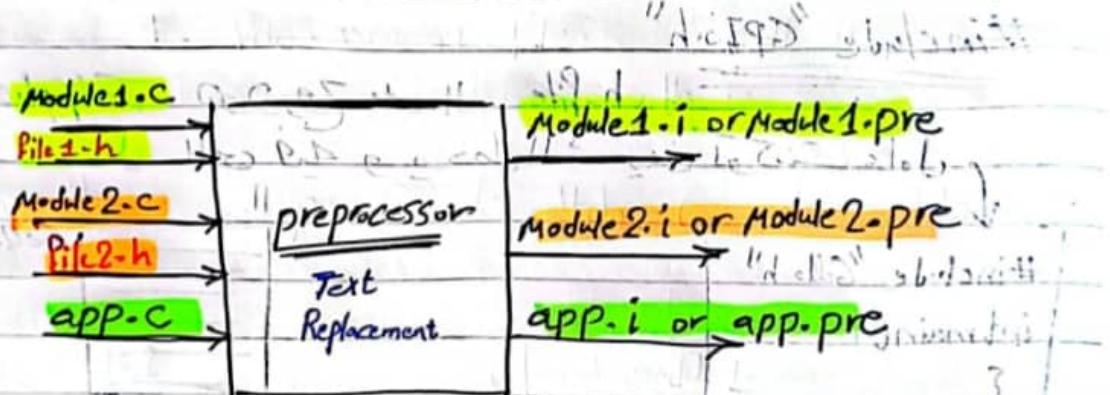
```
#include "GPIO.h"
```

ج. ترجمة المفاهيم في المكتبات المترابطة (Turtleback) h-file Header Turtleback

البراجماتي \rightarrow preprocessor \rightarrow المدخلات

* If job submit to ~~single~~ and then will bring all
if off go ~~single~~ int main() > off class catalog item number;
else if you want to do it multiple times multiple will bring all
this is how it is. so if ~~multiple~~ we have to do

لوحة الـ prototypes و declarations في file.c يحد فحصاً في file.h او ما يحده في file.h



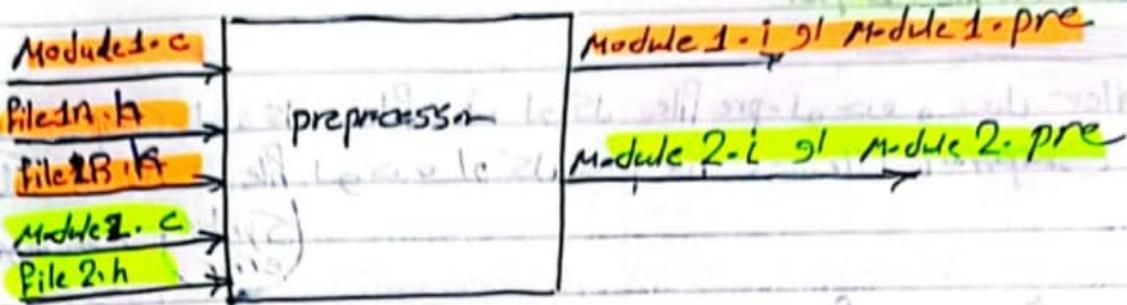
فیلد دا مکد "Module1.C" کان کامل include "File1.h" و replacement files Module1-C راج هات کل.

~~Tool~~ ~~#include "File1.h"~~ .i او .pre ~~Tool~~ خرج file.i جدید است. حسب اول الگوی است سعال بسیما برخی جملات file.pre او file.i

* ۱۹ جل سیکھا، دنیا اسے Module 1-6 پر حسب الیاتی انتہا کیا جائے۔ pre او replacement مفہوم اسی کیوں # PureC file تو مخفوم اسی

← ملحوظة بروگریسسور بیشیل ال Comments عنوان ال Compiler

ثاني لوكات بروكدا



ويمثل File1B.h, File1A.h و #include في المدخلات Module1.c حيث replacement يتم بـ Comments csl او #.csl

Preprocessor

- It is the 1st stage of build process.
- the input to this stage is .c and .h files.
- the output of this stage .i or .pre

* Preprocessor

is responsible for

① Strip out (remove) Comments

② replace text directive

③ produce C Code without any preprocessor directive

معلومة لو في error لأن "no error" لأن #define SIZE 5; هو error

~~no error~~ ~~#define~~ ~~SIZE 5;~~ ~~error~~ ~~preprocessor~~ ~~no error~~

① #include "file.h"

② #define (宏)

③ #if

④ #else

⑤ #endif

⑥ #for

⑦ #while

⑧ #do

⑨ #end

2nd Block is the Compiler

الCompiler يأخذ كل file .pre و يبدل him إلى file .i . او كل file .i . او كل file .asm .
 ستقدر ان كل file .i . او كل file .asm .
 مفهومات لاي (Syntax error)

سيعرف ال Compiler ببعض الملفات كلها مع بعض لا ده يفعل
 الكل file .i . ويتأكد ان كل file .i . خالي من Syntax errors

True or False

- Compiler Compile all files together (X)
- Compiler Compile each file Separately (✓)

فال Compiler يأخذ كل file .pre و يترجمه ويعرف كل file .asm او file .S .
 ويرجعونك الى file .S .
 كل ما يحصل هنا يخمنه على ال Assembly

Assembly الى pure C Code حول ال Compiler .

Instructions (promises) (①) ←
 with their descriptions (②) ←

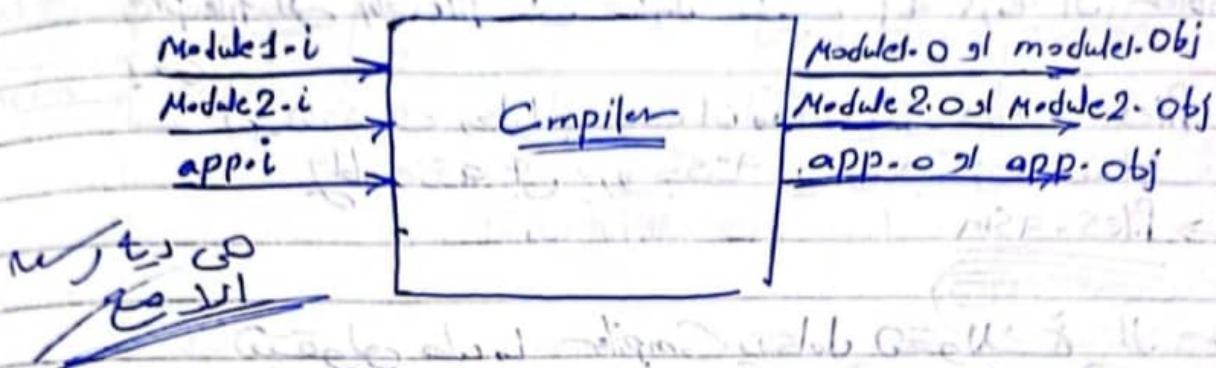


يصنف في حقيقة الكلمة هو قديم دلوقتي ال Compiler (modern Compiler)

ال Compiler كان مع يغدو مع assembler و Compiler

file .O [file .asm] file .S file .object

يخرج file.o الى دونيما طبقاً  فالاصح نادى رسمه دا



فديو قوي مع الـ file.asm دو لأن خلاصنا Modern Compiler

Assembly و Compiler بعث كلانها في نفس الـ Compiler وبالتالي الـ preprocessor او output نادى ياخ file.obj و file Binary و file.h

* ولاحظ كل File لوحدة يعني خـ Module1.i حوله Module1.o و Module2.o حوله Module2.pre

فلو كان: عندك اعماق في Syntax الكور ميت اللي هيترض ادتو preprocessor و لو عندك خطأ في error replacement او replacement فيها ميت اللي هيترض

True, false values in the file assembly file

→ Modern C Compiler Convert file.i to file.asm (X)

→ Modern C Compiler Convert file.i to file.o (✓) 
Binary file

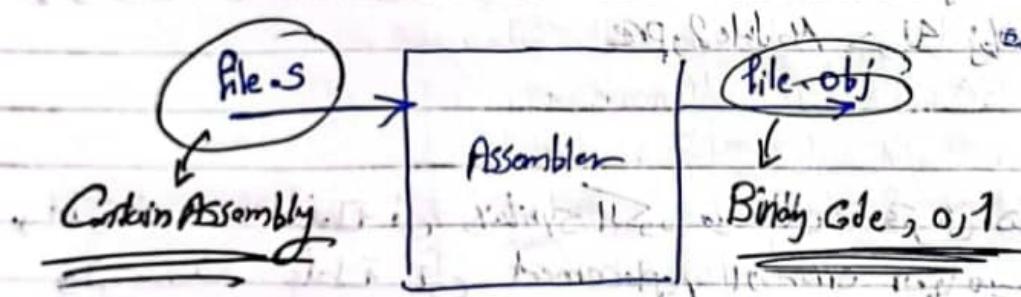
3rd-Block Assembler

ـ تستعمل طيب ده ميقات موجود !! مع modern C compiler يقتصر تحول من file to file على صنف طيب file Assembly to Assembly

لہ کہتے تو فہرست مکانیکیں اسے files یا assembly files کہا جاتا ہے۔

ستقولن طبعاً Compiler يتعامل معه لغة الـ File .C بتحول من pure C Code إلى File .o الذي هو Object File إلى File .a الذي هو Library لكن انت معاك دلوقتى File .S وليس .C . اللي هو Assembly file فيه تحتاج Assembler هو اللي ياخد هنا File assembly file ويعمل Object file إلى File assembly file

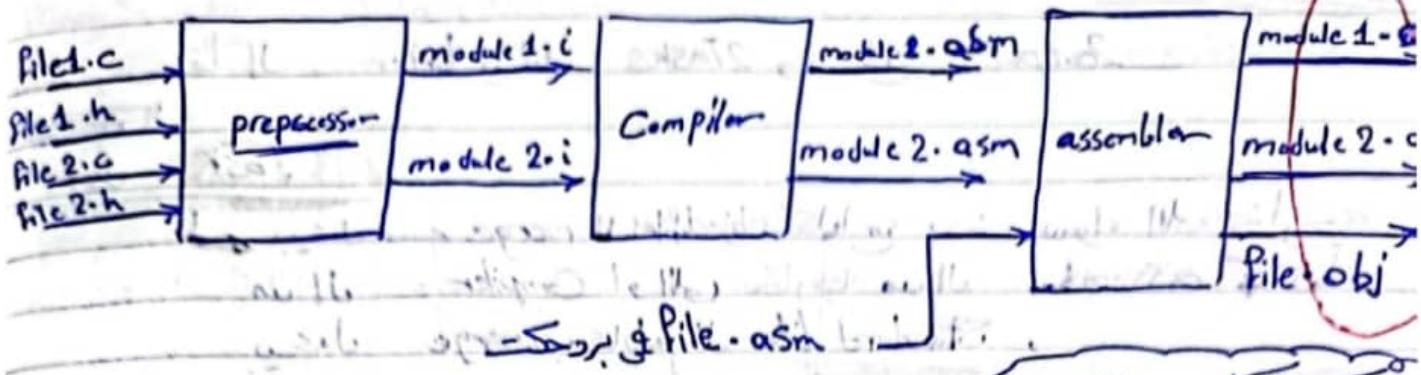
لیه اور Compiler میا خواون ہیات Compiler عائز و لاتے جاک Assembler پر file.asm



Compilers کان assemblers یا خذ fileasm الکان بیطابعها ۱۱ Compiler کان و بحولهه دو ای fileo کد ناو Compiler بیعا fileo میتمد علی fileasm ای دادن لد fileo

پروجکٹ غیر کدا میں دحتاجہ file.asm فیلم assembler لفونڈی دلوقتی داحتا جا دے.

Build process فرمان کانتے کانتے



وبد کا کل دی ریختن علی
Linker کا مسیر



file.asm
file.o
file.exe

بروجکت یا

بروز رسانی کرنے کے لئے اپنے کام کرنے کے لئے

بروز رسانی کرنے کے لئے اپنے کام کرنے کے لئے

بروز رسانی کرنے کے لئے اپنے کام کرنے کے لئے

بروز رسانی کرنے کے لئے اپنے کام کرنے کے لئے

بروز رسانی کرنے کے لئے اپنے کام کرنے کے لئے

بروز رسانی کرنے کے لئے اپنے کام کرنے کے لئے

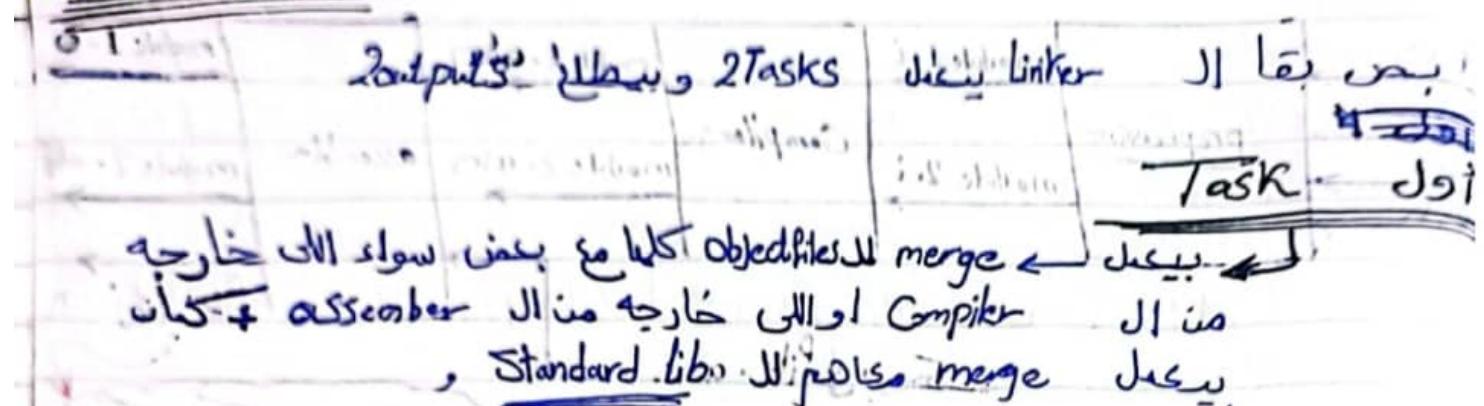
بروز رسانی کرنے کے لئے اپنے کام کرنے کے لئے

بروز رسانی کرنے کے لئے اپنے کام کرنے کے لئے

بروز رسانی کرنے کے لئے اپنے کام کرنے کے لئے

بروز رسانی کرنے کے لئے اپنے کام کرنے کے لئے

4th Block - Linker



داننا يا صريقي ال Libraries بتحفظ كأيضا object او binary كده
عالي طول اطلب ليك يستعمل دا - يحصل انت هندا "لما تروح تحمل
ده فيه prototypes ومنلا الكور بناء الـ `#include <String.h>`

في حقيقة أنا بس علها بس فليه تعلمها Build ونخلي على
خطوات (Build process) كلها وهو تزور وفكرة ال Build كل
شيء ضمني وهي اصل "Sure" وعموماً لاي خطأ Syntax error دى
Library حاضر انت بتستخرجها بس وعموماً هرم ادعىها
على كل Build process كل مره مع انها كود ثابت من غير تغير
فكتابه كذا ال Standard Library اللي هي بتكون كذا
→ #include <avr/io.h>

```
#include <String.h>
```

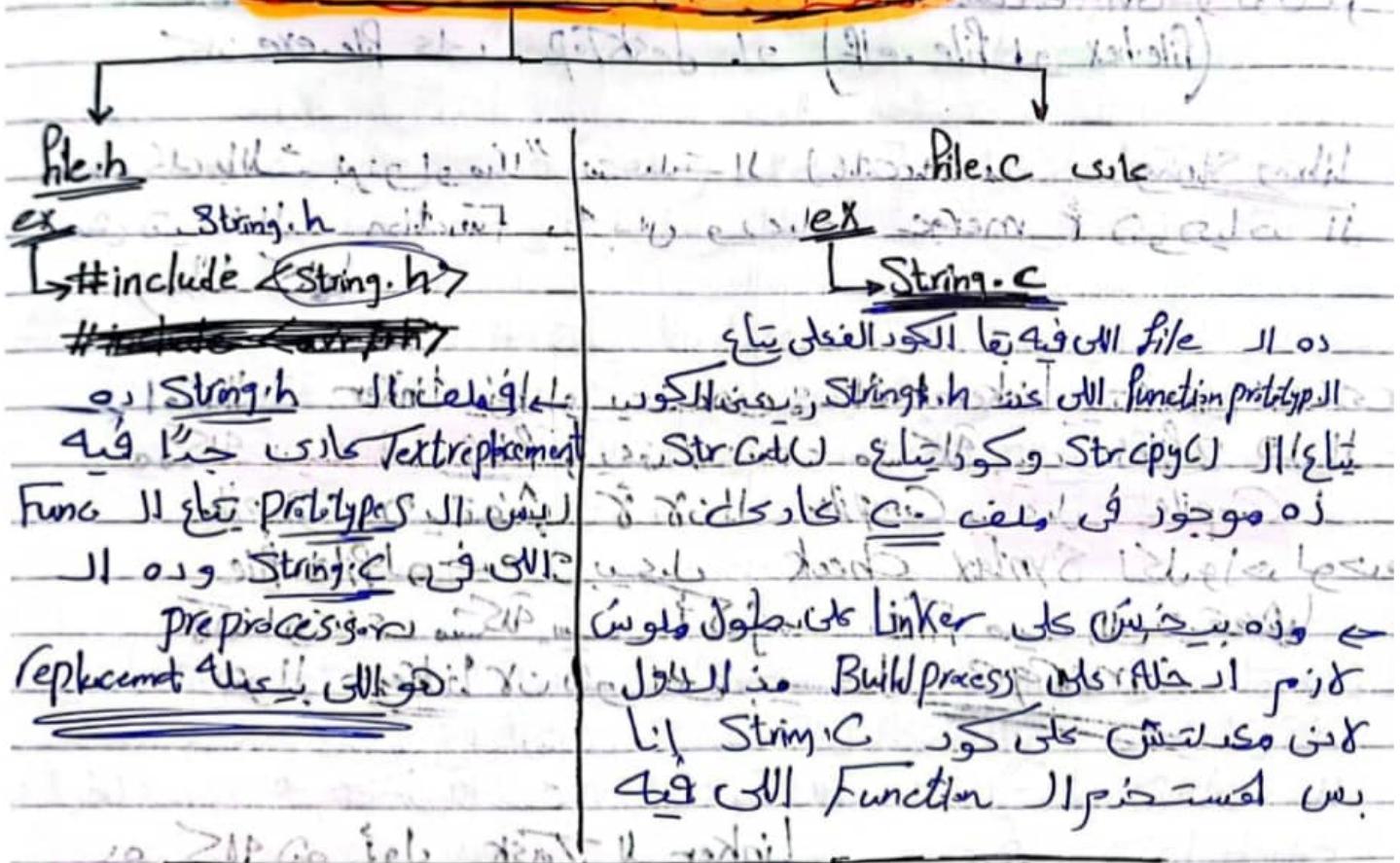
هـ دو فہریا Functions String.C پیغام میں "اللہ کو فیض" کے کل کوہہ من الدوال Build اسے Strcat(), Strcpy() کر

فی بناء دا الکوڈ بناء ال Standard Library مبنی بیخس معالک Binay S obj کے Linker کے طول کے file کاری

ملحوظہ رکز #include <String.h> میں دا ال Lib کو درکز preprocessor is replace ہے محرر Text replacement یعنی hash ہے (header) دا String.h یعنی file بناء ال ہے اس کے عادی رکز

و String.h دا محرر فيه Prototype ہے اس کے عادی String.c کے کوڈ بتایا گئی میں اس کے عادی رکز strcpy()

Standard Lib یعنی اسی 2 files میں موجود ہے



فکان الیاری انکے کاری Standard Lib دا بناء ال String.c کے Linker کے preprocessor کو درکز کوڈ حافظ و مظبوط و مدرس عمل علیہ فالکی بیحمل ان ال C دا بناء ال Standard Lib یکوں ہے کے Binay S کو درکز کے طول کے file کاری

کے Linker کے merge کے بعد رکز

ـ ملحوظة بس سريعة ال File.C شغال StandardLib من ٣٠٠٠ دينار
كتبة بحثاً هو يكتب file.lib او file.a بعدهم هو نفس الدليل
هو فيه المكتوب الفعلى زبارة ت، كما

ـ فكما ال files obj كل ال merge بيدل Linker. سوكه تباعت
زبط اللى جي من Assembler او اللى جي من Compiler و كان مع
StandLib.Lib.

ـ فتاتن ال Linker بيعمل merge لكل ال obj مع الakan ال libraries
اللى انت استعملتها و يطلع ال executable file اللى هو لوفا كر
كان (file.hes) او (file.elf) او (deskTop file.exe)

ـ و خلي يالك يبردو لو ميل "استعملت ال Struct" من String
ش هيأخذ ال function دي بس و يصلها merge هو عيادة ال Library
كلها من Struct.

ـ امني ال Linker بيرجع او يستيقظ من sleep ال files من نوعي
مسكلة بين ال files و يعني لو عيادي مسلكة بيت 2 files بدل ال 1
مسكلة Compile هيتحققوا 2 files لأن ال Compiler بيعمل Compiler
لوحدة file file Syntax check لكل واحد لو حده
لوكن لو عيادي مسلكة بيت 1 لكن من file مبيت اللى دقيقها
هو ال Linker لأن هو اللى بيعمل merge بعد linker Object Binary

ـ Linker Task هو أول Task لل Linker

ال Task الثانية ساعه ال [Linker]

memory allocation ال Linker هو المسئول عن حلية تانية غير ال merge ، الحاجة ديه يغ يحمل allocat لكل Variable وكل Function في الكور في اذ Segment الصع بتاعتتها هي ديه وظيفة ال Linker الثانية

فلو عملت initialized Global Var يتبعوا مكان في dataSegment في ال RAM وفدور ال Linker هو ال memory alloc. يعني لو وجد في كور بتاع ال Initialized Global Var هيدا يه address في dataSegment ال في RAM وطبلاو linker لعى في واحد من زطه files ال Static Var مش مكتوب initialized يروح ال linker مدرينه address في bss Segment ال في RAM ولو انت عمل المكتوب ال في كور يعني ال Linker يدخلها address او مكان Function في Const Global Var ، ولو عيدها Text Segment ال "rodata Segment" في address يديله

فال memory allocation هو مسئول عن كل حاجة في مكانها المطلوب اوطا [memory allocation]

بس خلى بالثانية هي ال Linker يكمل ال Task بتاعت ال memory يحتاج مساعدة خارجية ، واما المساعدة ديها بتجي بالكم ال Linker file او Fill file او embedded app files او ال Linker Script

اي ده حوار بين Linker Script - Linker file - ديه اسلوب اسلوب Assembly و C و C++ معندهش نتساشر فراخ وكل tool عندها ال Syntax بتاعتتها لحد ما ال الكام ، ومن انت اللي بتحب كتب انت اخر دل تحدلك او تزورها فيه

١١. اللینکر لاینر

طب ال Linkerfile يكون موجود في آيه ٥ :- ال Linkerfile يكون موجود في تو صيغة ال Segment memory يعني الذي يدخل كل Segment من فيت إلى افيت ، بلديك start address (addressStart) وال size (size) وال end address (endAddress) .
تابع كل Segment كل Segment

يعني ال file Linker يقول لك ال dataSegment ال start address بتبدأ من address وال size .
يعني ال dataSegment هو ال عنوان الذي يدخل في ال dataSegment .
300 هو ما يدخل في ال dataSegment .
فال Linker يحتاج لتقسيم ال memory عيّان يخفي كل Segments من كادر إلى عنوان كما يعرف بعمل ال memory allocation و يدخل الكل حاجة لها إلى متى في ال Segment .
و هنا "Segments" دليل على ال segments .

و عيّان كا في بروجكت وانت بتحتار نوع ال (micro Controller)
عيّان كل RAM memory مختلف يعني RAM
في 16-32 atmega حجمها 1KB في ال RAM كا 32KB مختلف في أحجام ال Segments من روى بعض وبالتالي كانوا يتطلب كل Segment من روى يعني في ال atmega-32 .
من 16 و كذلك يمكن حجم ال ROM في atmega-32 32KB و حجم ROM في atmega 16 أو 16KB .
احجام Segment تابع ال ROM من روى بعضها بربور

لذلك يعني مثلاً كان ال target atmega-16 .
يتطلب project بـ 25,000Byte .
فقال هناك Code Segment .
غيرها عنوان في Code Segment راح دربعها 27,500 .
أن تكون من ليس تحمل اصل انت ستكل على mega-16 .
حجم program كذا 16 KB وال Linker من عنوان
لل زعل كلها من اول 25,000 ليصبح الفيل الكورد و مسخن
يشكل على at-mega16

نوار Linker file ده هو اللى فيها توصيف ال memory وكل MC ١٢٧

انت مفروض سطبيت حاجة اسمها Assembler or Compiler or WinAVR merge بتابع او AVR كل وفعارى نفس او Linker عرف يعدل memory ل رطه بلى MC بس لوكل Controller مختلف فيه او لازم يكون عندك Linkerfile جديد له

ال Linkerfile ده Software file يعني انتا ممك افتحه و اعدل فيه بمعنى ان تقسيمه ال memory وال Segment يتابعوا ادوي تقسيمه في Software.

ال memory دى عباره عن Locations من Array، انتا اقدر اقول dataSegment من قيست ال فى فنت وال heap من قيست ال فى فنت وبقدر اقدر فى Linkerfile واخذه تقسيمه براحه.

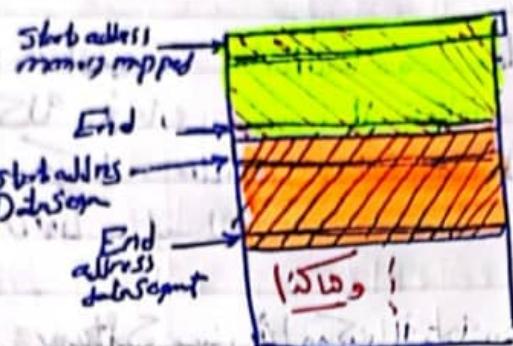
يعنى مثل لو في شغل لفيفات، مكتوبين global var او local var لكنه انتا ممك افتح Linkerfile واحد وال Segments كلها في مكان واحد الحين رول اسهمهم GlibsSegment كامب و اقول لا Var او Var الدا ننزلوا فين و راحطوا فين براحه كل ده في Software و تستوفى الحوار ده مع بعض ان شاء الله

فتقسى ال memory ما هي الا تقسيم في Linkerfile، فانا اقدر اللعب في Linkerfile براحه

ال Linker file بتحشلوا ال memory بقول عيّان يقون تقسيم memory و يبدأ بعدها ال Linker بعد دل المémory alloc وهو بيردود اخدله

الكور اللى دو رطه يدرثقا يقول كل حاجة تنزل فين

• فايت حرف تقسيم ال Segment اهم حاجة ميحصلن على الـ Start و End تابع كل Segment كذا مثلاً



~~فـ ٥٥١ الـ بـ يـ خـ دـ الـ رـ بـ كـ لـ اـ وـ حـ مـ هـ ئـ~~ ~~binder~~ ~~يـ خـ دـ الـ رـ بـ كـ لـ اـ وـ حـ مـ هـ ئـ~~ ~~hecta وـ الـ دـ الـ Rـ e~able file~~ ~~وـ دـ لـ دـ وـ merge~~

~~الخطابات المكتوبة~~ ~~Tasks~~ ~~الخطابات المكتوبة~~

~~الآن نحن في المراحل الأولى من المهمة~~

~~→ Ebenfalls kommt es bei dieser Art von Zellen zu einer Spaltung des Zellkerns.~~

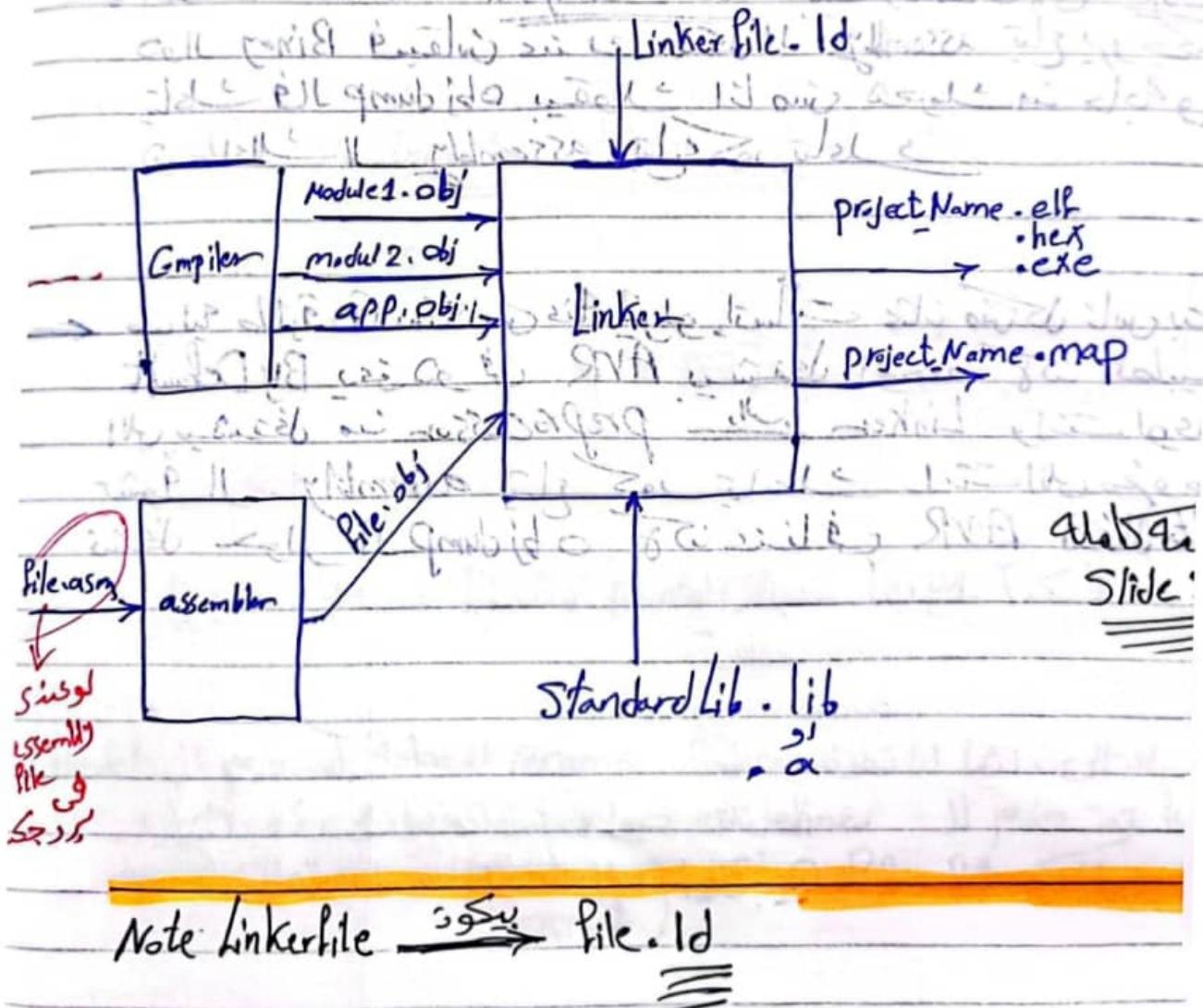
\rightarrow II (second) stage of life (adult)

نیز رائٹنگ کے ملکاہی لفظ اُبی

وكان الخلاصة - الـ 5 Tasks Linker

يُعمل merge لكل الكائنات التي عندك مع الـ Standard Lib و يُطلق على ذلك executable file .elf أو file .hex أو file .hex المترافق مع المبرمج Final Binary file بناءً على

لما و كذلك يحيل الـ memory allocation لكل Var وكل Func وكل حاجة في كود عندك بمساعدة الـ Linker File وبذلك حاجة اسها mapfile . الـ mapfile يصنف مساحات الـ memory على segments فيقول لك أنا مخلي كل حاجة فيت بمعنى الـ mapfile بيقول لك كل Variable هى تنزلت في اين address وain segment وهذا والـ mapfile يحيل ليه ايه ولا حاجة الى لوحيت تبص على اماكن الحاجة في memory من اذكر وبذلك ما تكتب اار English ياري يعني بخصوص الـ Var العلاني تنزل عندئنون كذا



في حاجة زياده تكتبوا بس صارق قال متفتحش سيرتها غير لو الرجال
الى بيعدل مقال Interview سالك فيها و بمعرف لوسلك كود الـ
اـرـاي بـلـيـكـلـ Build لـ Linkerـicـ و حـلـدـصـ من اـولـ الـ
ـpreprocessـ لـحدـ الـLinkerـ و اـتـفـتحـتـ لـكـتـهـ لـوـسـلـكـ اـتـحـرفـ لـيـهـ
ـعـنـ الـBlockـ تـاعـ objdump دـتـقـولـ الـتـالـيـ

ـالـBlockـ5ـ هوـالـ "objdump" دـهـ يـعـدـ dumpـ للـobjectـ الـكـوـدـ الـهـوـاـيـهـ
ـبـصـ دـهـ الـيـخـاـدـ الـB~inary~P~ileـ تـيـاعـكـ الـD~e~f~u~l~t~ اوـ P~i~l~e~ وـهـ يـطـلـعـ الـ
ـوـهـ يـكـوـنـ اـنـكـا~o~f~ او~ S~t~ او~ A~s~ وـهـ يـكـوـنـ خـيـهـ الـ
ـبـلـيـكـلـ الـt~y~ الـk~o~d~ الـc~o~d~ وـهـ مـسـتـقـلـ تـقـيلـ كـدـ اـلـوـعـاـيـزـ تـعـدـ
ـالـd~e~l~a~y~P~u~n~c~ دـهـ مـاـمـهـ اـنـخـوـتـ لـكـامـ
ـاسـطـرـ الـa~s~s~m~b~y~ دـهـ دـوـاهـ الـP~i~l~e~ الـيـخـاـدـ الـt~y~ الـk~o~d~ وـهـ مـاـمـهـ
ـجـانـ الـa~s~s~m~b~y~ دـيـاعـ كـوـرـ تـيـاعـ الـt~y~ الـk~o~d~

ـاـصـلـ اـنـتـ عـنـكـ modern Compiler سـحـولـ منـ Cـ إـلـيـ object مـلـلـ
ـهـوـالـ B~in~y~ فـيـقـاسـ عـنـ دـهـ تـكـلـ الـa~s~s~m~b~y~ دـهـ يـرـجـعـتـ
ـتـيـاعـ فـالـobjdumpـ بـيـقـولـ لـتـاـمـسـ هـجـرـلـكـ مـنـ حـاجـةـ وـ
ـهـيـصـلـكـ الـa~s~s~m~b~y~ الـk~o~d~

ـطبـ لـ طـارـقـ قـوـالـ مـكـلـمـيـنـ هـنـيـهـ صـرـلوـ اـتـسـالـتـ عـنـ مـسـكـلـ تـاسـ بـتـشـكـلـ
ـالـBy Defaultـ يـعـقـدـ دـهـ فيـ AVRـ بـتـشـكـلـ الـوـحدـهـ هـذـهـ لـالـطـبـيـعـيـ
ـالـL~i~n~k~er~ وـهـ لـوـعـاـيـزـ مـسـتـقـلـ مـنـ preprocessorـ دـالـيـهـ
ـشـوـفـ الـa~s~s~m~b~y~ دـيـاعـ كـوـرـ تـيـاعـ لـتـاـمـسـ لـهـ مـعـروـضـ
ـتـشـكـلـ حـوـارـ لـ objdumpـ لـكـ عـنـ نـافـ AVRـ هـذـهـ كـلـ

ـL~i~n~k~er~.

ـL~i~n~k~er~