

TIMING SUITE FOR REAL-TIME SYSTEMS T1.stack

Introduction

Static stack analysis with T1.stack

COST-EFFECTIVE, DETAILED AND ACCURATE STACK ANALYSIS

Based on the binary (the ELF file), **T1.stack** performs a static code analysis: the binary is disassembled, function calls are extracted and the call tree is reconstructed. At the same time the stack consumption for each function is determined. The call tree and the stack consumption per function are combined into the comprehensive and powerful **T1.stack** view.

yatem:	Core0 • 2	Summary	Show Annotations					Stack 0	Stack	
Naplayin	g full call tree						Stack 1		CSAUM	
Function I	Names	Atributes	Status Address	in content (D)	self (C)	machable (0)	in context (1)	sef (1)	reachable (1)	#Callers Source Path A
-OS_Se	cheduleCore2 [RecDepth 1]	7.20	0x80002455	160	160	312	64	64	768	0 2 \Downloads\S
i-int_app	plproc2	7.00	0x8000CF20	0	0	152	64	64	832	0 Z \Downloads\S
int_app	piproc1	7.20	0x8000CDF0	0	0	152	64	64	\$32	0 Z \Downloads\S
i-int_app	plprsc0	7: 00	9x80005080	0	0	152	64	64	895	0 ZhDownloade\S
i na	in_apploroc0	73 D	0x80000000	0	D	152	125	64	002	1 Z:\Downloads\S
-	TA_OS_Hendler	7.80	0x80003FA4	0	0	101	123	0	762	3 Z.\Downloads\S
	B-OS_Handler	- 二十二	0x80002AAC	0	0		128	64		832 ZhDownloade\S
	OS_Schedule [RecDepth:1]	7.000	0x80002A98	0	0		120			E 10040 00000 0
	- OS_ScheduleCom1 (RecDepth 1, Ituncation)	4-970	0x80002200	n/a	72	40	n/a	54	7	0 ZhDeenleade\S
	OS_ScheduleCore2 (RecDepth 1, Truncation) GOS_ScheduleCore0 [RecDepth 1]	-86	NUMBER	OF BYTE	S OI	N THE	T.			кімим ѕтаск с
	G-Core0_tmsTask (RecDepth 1)	7.50	STAC	CK CONS	UM			S S	UMPTIC	ON POSSIBLE, A
	-TA_OS_Schedule [RecDepth 1]						55			
	@-T1_AppHander	-12	REVIOUS F	UNCTIO	12 20	JFAR	256	2		OM THE CURRE
	Gree0_tmsRunnable0	-	0x900035A2	135	0	Ű.	256	C	ONTEX.	T (THIS POSITIO
	🗟 calindrects (indCals 2)	5	0x80005105	130	n		145			
	-funC	ET	0x8(NU	MBER O	F BY	TES ON	J THE		HE CAL	LIKEE)
т т			0x8(0	0	1 Z\Downloads\S
		6	Dx80	STACK	. CO	NSUME	DBY	0	.0	1 Z\Dewnloads\S
	H-TA_Button_Read		0.8(THE E	UNC	TION I		. 0	0	2 Z.\Downloads\S
	E-T1_DelayRoutinePC		0-8					64	123	158
	Ifii-Engine_RPMTick		0x80004952	136	0	0	256		0	1 Z \Downloads\S
	TI_RxWheetSpeed	A MARCHINE	0x80003EBE	136	0	0	320		64	1 Z \Downloads\S
	Gre0_50msTask [RecDepth 1]	71.00.040	0x8000306C	136	D	8	256	Statement and the	384	1 Z \Downloads\S
	I) T1_TraceEvertNoSusp		0-C0000000	136	0	0	192		0	51 Z:\Downloads\S
	 Core0_vatableTask 		0x80003492	106	0	0	192		192	1 Z-\Downloads\S
_	Core0_isrTask		6x800030E8	136	0	0	192			2 Z-Downloads\S
	B Core0_25msTask [RecDepth:1]	77 2 D D	0x80003004	136	D	Ū	256		256	1 Z\Downloads\S
	E Core0_BackgroundTask	たこの	Dx800030A8	0	0	152	256	- 64	704	1 Z\Downloads\S v

Indirect function calls

For any static code analysis there are limitations with respect to resolving indirect function calls. Such calls typically use function pointers and it is essential to know all call-targets (functions) which can possibly be called at run-time. **T1.stack** allows to complete any gaps in the static analysis **through annotation**. Three kinds of annotation are supported: manual annotation, import of generated annotation files and annotation through T1.flex measurements. Simply **measuring** call-targets is unique and a highlight of **T1.stack**. Such measurements can also be used to cross-check and verify annotations from other sources.

Unresolved indirect function call:

-Core0_1msRunnable0		0x800035A2	136	0	8	256	64	192
– callIndirects [IndCalls:2]	2	0x8000510E	136	0	8	256	64	0
-funC	6	0x80005106	144	8	8	192	0	0
te⊢TA_Button_Read		0x8000400C	136	0	0	256	0	0
teller T1_DelayRoutinePC		0x80008F8E	136	0	0	320	64	128
Engine_RPMTick		0x80004952	136	0	0	256	0	0

Resolved indirect function call by dynamic T1.flex measurement:

Core0_1msRunnable0	0x800035A2	136	0	8	256	64	192
🖨 callIndirects [IndCalls:2]	0x8000510E	136	0	8	256	64	64
-funC	0x80005106	144	8	8	192	0	0
-funB	0x800050F4	136	0	0	256	0	C
-funA	0x800050E2	136	0	0	256	0	(
- TA_Button_Read	0x8000400C	136	0	0	256	0	(
te⊢T1_DelayRoutinePC	0x80008F8E	136	0	0	320	64	128
<u> </u>	0x80004952	136	0	0	256	0	(

T1.stack offers the advantage of detailed analysis. It detects not only of the amount of used stack but also how and why it is used. Deep understanding of stack consumption allows successful optimization of stack usage and detection of unintended or purposeless use of the stack. Using less stack often helps to improve runtime performance.

When using a high level language it is not possible to predict the stack usage from even a detailed knowledge of the C source code. With auto-generated code, the problem is even worse. Using **T1.stack**, stack consumption can be continually tracked so that the effects of coding and compiler flags can be monitored and understood.

The accurate and detailed analysis of total stack usage combined with validation allows stacks to be reliably dimensioned with T1.stack and thus avoids the waste of allocating unnecessary memory. What's more: stack-overflows can be avoided.

T1.stack (C	ore0) T1.	cont Ta	ble View	runnables.c (0	Core0) ii
Source	Merged	D	isassembl	у + 1 - 1	D] G
	ma warni				
	* TASE			nt a) = 1;	1
funC:	UNUSED (VOIA	cite ii	((a) - 1)	
F00x800051	06.	32 1f		mov	\$d15
A0x800051				sub.a	
A 0x800051				st.w	
A0x800051				ret	
#ifdef	TASKING	3			(
# pra	igma warn	ning	default	t	1
<pre>#endif /</pre>	* _TASI	KING_	*/		
}					
/*					(
	lIndire	ets (void)		
{					
callIndi		17 / 1	indire	ect call */	
		91 00	00 fb	movh.a	\$a15.0
				ld.a	
				calli	
				call using	
A 0x800051	1a: 9	91 10	00 f8	movh.a	%a15,0
A 0x800051	le: 9	99 ff	00 ac	ld.a	%a15,[
A0x800051	22: 0	ic Of		ii	%a15

Key benefits include:

- Static analysis based on the binary file
- 3rd party code can be **analyzed without the source code**
- Compiler effects (e.g. optimizations) are also taken into account
- Measurement assisted resolving of indirect function calls (function pointers)
- Extreme fast analysis (e.g. a 150MB ELF file of an engine management ECU could be analyzed in less than two minutes on a regular PC)
- Call tree offers additional insights into the software structure
- Built-In source code- and disassembly-viewer

Technical data

Supported CPU architectures:

- Infineon TriCore (*)
- ARM, ARM Thumb
- PowerPC, PowerPC VLE (*)
- RH850
- x86

(*) enhanced static analysis engine

gliwa.com

fon +49 - 881 - 13 85 22 - 0 fax +49 - 881 - 13 85 22 - 99 mail: info@gliwa.com