

Testes de integração contínua em um dispositivo embarcado baseado em Linux

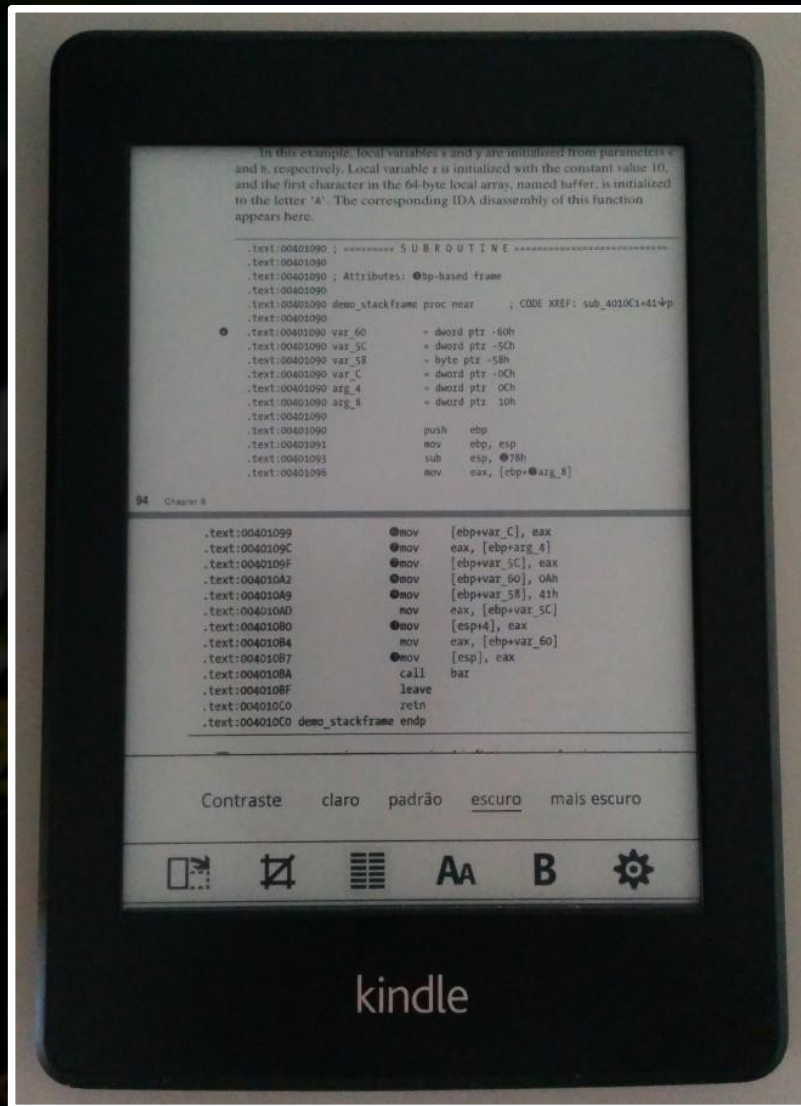
Mesailde Souza de Oliveira Matias

Orientador: Prof. Me. Ramon Rosa Maia Vieira Júnior

Curso de Pós-Graduação em Engenharia de Sistemas

Escola Superior Aberta do Brasil

Definição do Tema



Dispositivo embarcado:
Kindle (leitor de livros eletrônicos)

Software embarcado:
KOReader (projeto de software livre)

Problema:
Executar testes de integração
contínua do KOReader no próprio
Kindle, em vez de um ambiente
simulado.

Justificativa

Relevância para a sociedade

- Leitores com tela *e-ink* proporcionam maior **conforto visual**.
- Entretanto, o **software padrão** desses leitores é muito **limitado**.
 - Principalmente com formatos PDF e DJVU, muito adotados para livros técnicos e científicos.
- Existe **software livre** para suprir essas demandas.
 - Aplicação de técnicas de engenharia de software tem potencial de melhorar sua qualidade e estabilidade.

Justificativa

Relevância para o curso de pós-graduação

- Aplicação de conceitos estudados no curso (**Teste de Software**) em um ambiente que não é abordado no mesmo (**sistema embarcado**).

Relevância para a área do conhecimento

- Avaliação experimental da hipótese “*é importante executar testes no próprio dispositivo embarcado*” em um novo sistema.

Problema de pesquisa: Dificuldade em executar testes de integração contínua em sistemas embarcados.

Objetivo geral: Desenvolver uma solução capaz de monitorar a linha principal do repositório de código fonte do KOREader e, quando ocorrerem mudanças, automaticamente compilar o projeto e executar os testes em um dispositivo Kindle.

Objetivos específicos:

- Monitorar continuamente alterações do repositório de código fonte do projeto.
- Compilar o software quando a linha principal do repositório sofrer alterações.
- Mapear no dispositivo alvo, via rede, um diretório do computador hospedeiro contendo o software compilado.
- Adaptar o *framework* de testes utilizado pelo projeto para ser executado diretamente na plataforma embarcada.
- Executar remotamente o *framework* de testes, coletando os resultados.

Metodologia

Pesquisa Experimental

- Coleta de resultados (**falha** ou **sucesso**) dos testes, dentro do ambiente desenvolvido neste trabalho.
- Coleta de **tempos** de execução dos testes.
- Estudo dos **motivos** pelos quais alguns testes falharam.
- **Correção** de erros e **contribuição** para o projeto:



Fundamentação Teórica

- Broekman e Notenboom (2003): Importância do **Teste de Software** no contexto de **Sistemas Embarcados**.
- Fowler e Foemell (2006): Conceito de **Integração Contínua**.
- Greening (2007): **Desafios** na realização de testes em Sistemas Embarcados.
- Karlesky et al. (2007); Greening (2007): Limitações da técnica de *mocking* em sistemas embarcados.
- Holck e Jørgensen (2007); Lanubile (2009): Importância da Integração Contínua em projetos **colaborativos e livres**.

Metodologia Detalhada

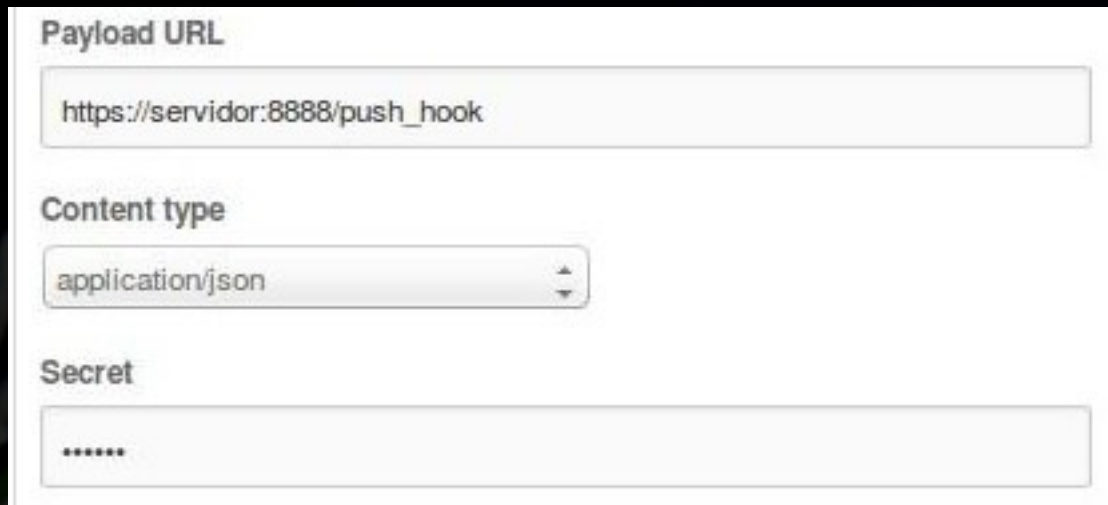
Ambiente e Ferramentas

- **Dispositivo embarcado:** Kindle PaperWhite (i.MX50 800MHz – ARM Cortex-A8, 256MB RAM, 2GB Flash)
- **Controle de versões do projeto:** Git (hospedado no GitHub)
- **Linguagens utilizadas:** Lua, Shell Script (bash)
- **Comunicação via rede:** USBNetwork / SSH
- ***Framework* de testes:** Busted
- **Depurador de memória:** Valgrind
- **Depurador de código:** gdb

Metodologia Detalhada

Técnicas

- *Web-hooks* para monitoramento do repositório.



A screenshot of a web hook configuration form. It contains three fields: 'Payload URL' with the value 'https://servidor:8888/push_hook', 'Content type' set to 'application/json' in a dropdown menu, and 'Secret' with a masked value represented by six dots.

Payload URL
<input type="text" value="https://servidor:8888/push_hook"/>
Content type
<input type="text" value="application/json"/>
Secret
<input type="text" value="....."/>

- Compilação cruzada com opções de depuração ativas.
- Mapeamento via rede do diretório do software para evitar desgaste desnecessário da memória Flash.

Metodologia Detalhada

Coleta e tratamento de dados

- Informações de depuração (erros de execução)
 - Coleta: gdb remoto (gdbserver).
 - Tratamento: análise de *backtrace* e análise comparativa com o código fonte.
- Tempos de execução
 - Coleta: por meio do próprio *framework* Busted.
 - Tratamento: comparação direta entre tempos (computador versus Kindle) e construção de histogramas.

Principais Resultados

Identificação de um erro de SIGILL e de sua causa

The screenshot shows a debugger window with assembly code. The code is as follows:

```
R0, R6
R6, =(aInvalidSeriali - 0x2F6C)
R2, R9
R3, #0x1B9
R1, PC, R6 ; "Invalid serialize stream %d (lin
lua_error
loc_2E64 ; problema aqui - R5 desalinhado
```

Below this, a comment indicates the cause of the error: "problema aqui - R5 desalinhado". The assembly code continues with:

```
loc_2E64 ; problema aqui - R5 desalinhado
ULDR S0, [R5]
UCUT.F64.S32 D0, S0
ADD SP, SP, #0xC
LDMFD SP!, {R4-R9,PC}
```

```
437: case 4: {
438:     int n = 0;
439:     int * pn = rb_read(rb,&n,4);
440:     if (pn == NULL)
441:         _invalid_stream(L,rb);
442:     return *pn;
443: }
```


Principais Resultados

Contribuição de correções para o projeto

18		serialize.c	
		@@ -148,7 +148,8 @@ rb_read(struct read_block *rb, void *buffer, int sz) {	
148	148		int ptr = rb->ptr;
149	149		rb->ptr += sz;
150	150		rb->len -= sz;
151		-	return rb->buffer + ptr;
	151	+	memcpy(buffer, rb->buffer + ptr, sz);
	152	+	return buffer;
152	153		}
153	154		
154	155		if (rb->ptr == BLOCK_SIZE) {
		@@ -161,10 +162,10 @@ rb_read(struct read_block *rb, void *buffer, int sz) {	
161	162		int copy = BLOCK_SIZE - rb->ptr;
162	163		
163	164		if (sz <= copy) {
164		-	void * ret = rb->current->buffer + rb->ptr;
	165	+	memcpy(buffer, rb->current->buffer + rb->ptr, sz);
165	166		rb->ptr += sz;
166	167		rb->len -= sz;
167		-	return ret;
	168	+	return buffer;

Principais Resultados

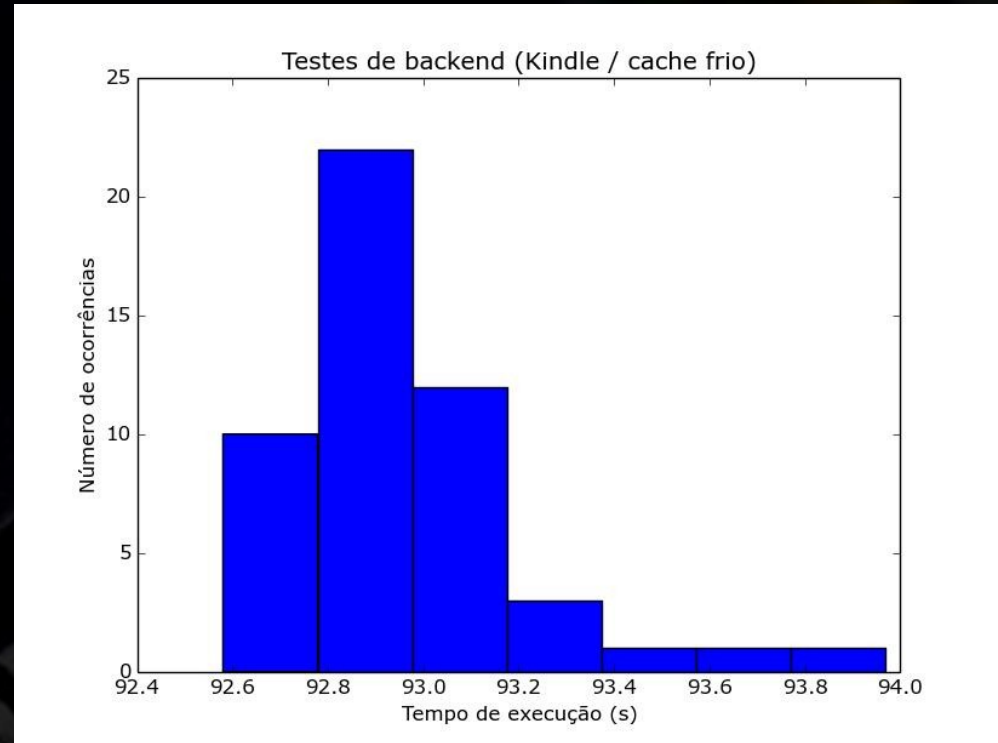
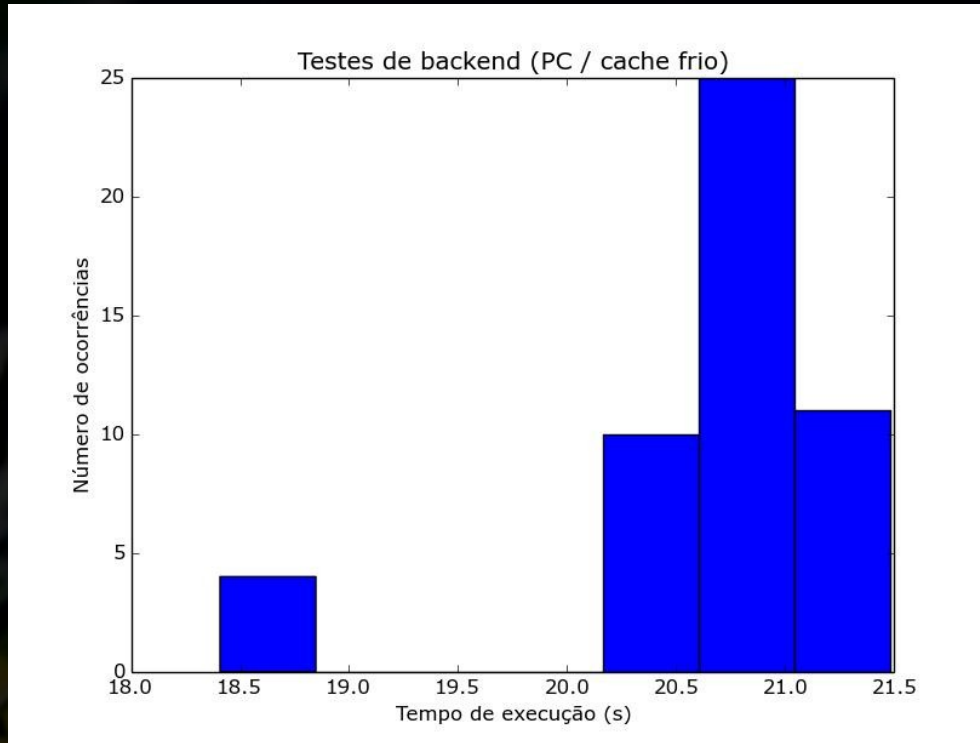
Tempos de execução

Suíte de testes	Cache	Dispositivo	Tempo médio (s)	Desvio padrão (s)
<i>Backend</i>	Frio	Kindle	92,96	0,24
<i>Frontend</i>	Frio	Kindle	20,46	0,05
<i>Frontend</i>	Quente	Kindle	5,77	0,07
<i>Backend</i>	Frio	Computador (PC)	20,65	0,68
<i>Frontend</i>	Frio	Computador (PC)	3,00	0,01
<i>Frontend</i>	Quente	Computador (PC)	1,04	0,01

- Existe diferença de tempos entre Kindle e computador, porém é desprezível frente ao tempo de compilação (dezenas de minutos).

Principais Resultados

Variabilidade dos tempos de execução



- Fato interessante: menor desvio padrão no Kindle que no computador.
- Hipótese: menos tarefas em execução no sistema embarcado?

Conclusão

- Em um caso prático, alguns tipos de erros de programação **escapam** aos testes automatizados realizados de forma **simulada** em um computador, **concordando** com Grenning (2007) e outros autores.
- Resultados dos testes automatizados efetuados no dispositivo foram úteis, permitindo enviar **contribuições** ao projeto.
- É **viável** a adoção do processo: o aumento do tempo de execução com relação aos testes realizados em ambiente simulado é plenamente **tolerável**, por não ser o gargalo do processo de integração contínua.

Limitações

Possibilidades para trabalhos futuros

- Método proposto não é capaz de lidar com testes em código de origem não confiável.
 - Permitiria aos desenvolvedores verificar o resultado dos testes automatizados ao receber uma contribuição de terceiros desconhecidos, antes mesmo de ler o código enviado pelo contribuidor.
- É um grande e interessante desafio implementar em um sistema embarcado os mecanismos de isolamento necessários a esse tipo de recurso.