



Requisitos

Sistema Operacional: Linux Debian-flavor

Ferramenta: linguagem de programação preferida

Objetivo

- Apresentar códigos de detecção de erros em palavras (SED - Bit de Paridade e (MED - CheckSum)
- Apresentar códigos de detecção e correção de erros em palavras (SEC-DED - Hamming)

Tarefa

Elaborar um simulador de armazenamento seguro (utilizando SEC-DED) de arquivos binários:

Funcionamento para gravação:

a) O arquivo para armazenamento deve ser passado via parâmetro no prompt/terminal

a1) ou ser possível de selecionar nos simuladores com interface gráfica

b) Deve ser mostrado os principais passos para produção da palavra síndrome + bit G ($M = 8$)

c) Deve ser salvo um novo arquivo com SEC-DED no formato da tabela 4

Exemplo:

```
$ ./hamming arquivoTeste.qqcoisa -w
```

-w para salvar o arquivo com integridade hamming

Sugestão de extensão para o arquivo com integridade: arquivoTeste.qqcoisa.wham

Funcionamento para leitura:

d) O arquivo para leitura deve ser passado via parâmetro no prompt/terminal

d1) ou ser possível de selecionar nos simuladores com interface gráfica

e) Deve ser mostrado os principais passos para verificação de integridade do arquivo

f) Em caso de falha, executar os procedimentos necessários para correção

f.1) Salvar o arquivo corrigido

g) Finalizar

Exemplo:

```
$ ./hamming arquivoTeste.qqcoisa.wham -r
```

-r para ler o arquivo com integridade hamming, verificar e recuperar (se necessário)

Entrega

via tarefa Teams

Assunto (IMPORTANTE): [OAC pratica 02] NomeSobrenome1 NomeSobrenome2

Anexo: arquivo em formato .zip com o nome idêntico do assunto do e-mail.

- contendo os códigos fontes utilizados.

+ arquivo HASH do arquivo zip.

Prazo: ao final do 7º dia após a apresentação desta prática.

Defesas ocorrem na próxima aula prática.

Introdução - ERROS em palavras:

Falha Permanente: Defeito físico que afeta células de memória ou (sinais de transmissão) de modo a alterar seu funcionamento, tornando-as não-confiáveis para armazenamento.

Erro não Permanente: É um evento aleatório e não destrutivo que altera o valor lógico de uma célula de memória (ou sinal de transmissão) sem causar dano físico.

As falhas e Erros são indesejáveis, mas acontecem frequentemente. Assim, códigos para detecção de erros (EDC) e códigos de detecção e correção de erros (ECC) são implementados nos mais diversos sistemas.

Tipo	Nome	Descrição
SED	Paridade	Apenas 1 bit extra
MED	Checksum	Algoritmos Hash de Segurança
SEC-DED	Hamming	Paridade de conjuntos de bits da palavra

SED (Single Error Detector) - Bit de Paridade:

Utiliza apenas um bit **R** de redundância que é composto pela função lógica OU_Exclusivo aplicado em todos os bits da palavra **P**. O bit **R** é gravado (ou transmitida) juntamente com a palavra **P**.

Exemplo: $P = \text{msb } 01101101$

Assim: $R = 0 \text{ xor } 1 \text{ xor } 1 \text{ xor } 0 \text{ xor } 1 \text{ xor } 1 \text{ xor } 0 \text{ xor } 1 = 1$

Palavra Gravada/Transmitida: $\text{msb } 101101101$

MED (Multiple Error Detector) - Checksum:

Mais conhecidos como Algoritmos Hash de Segurança, tem o objetivo de detectar eventuais alterações, propositais ou oriundas de falhas de sistemas, nas redes de comunicação e armazenamento e é o método mais aceito na internet. Existem diversos algoritmos: como o MD5 (Message-Digest algorithm 5) e o **SHA-2** (Secure Hash Algorithm 2), sendo este último, um algoritmo que gera um HASH de 256 bits.

Normalmente, servidores de arquivos disponibilizam o resultado hash em arquivos com formatação do tipo:

<HASH> <nomeDoArquivo>

Onde cada linha representa um HASH e um arquivo.

Para gerar um arquivo HASH com base em arquivos existentes, use:



```
$:> sha256sum <nomeArquivo1> <nomeArquivo_n> >> <nomeArquivoHASH>
```

Para verificar a integridade dos arquivos, adicione no mesmo diretório o arquivo HASH e use:

```
$:> sha256sum -c <nomeArquivoHASH>
```

Será apresentado uma lista de SUCESSO ou FALHA para todos os arquivos definidos no arquivo HASH.

Exemplo de conteúdo de arquivo HASH para os arquivos:

Nome	Tamanho
 A202 - Entrada e Saída.pdf	3,8 MB
 A301 - Hierarquia de Memórias.pdf	633,4 kB

comando:

```
$:> sha256sum A202\ -\ Entrada\ e\ Saída.pdf A301\ -\ Hierarquia\ de\ Memórias.pdf >> sha256sum.txt
```

Arquivo gerado:

```
28efe1f8aac9f9752da689a9f4fd894641e9878dbfd5ccbc14e2e9d822b8de7 A202 - Entrada e Saída.pdf
1d04aa57235f83b14dc2de2695446a0cdbac0e82395626ad014f8de849452595 A301 - Hierarquia de Memórias.pdf
```

Verificação (exemplos com sucesso):

comando:

```
$:> sha256sum -c sha256sum.txt
```

saída:

```
A202 - Entrada e Saída.pdf: SUCESSO
```

```
A301 - Hierarquia de Memórias.pdf: SUCESSO
```

Verificação (exemplo para arquivo A301 danificado/corrompido)

comando:

```
$:> sha256sum -c sha256sum.txt
```

saída:

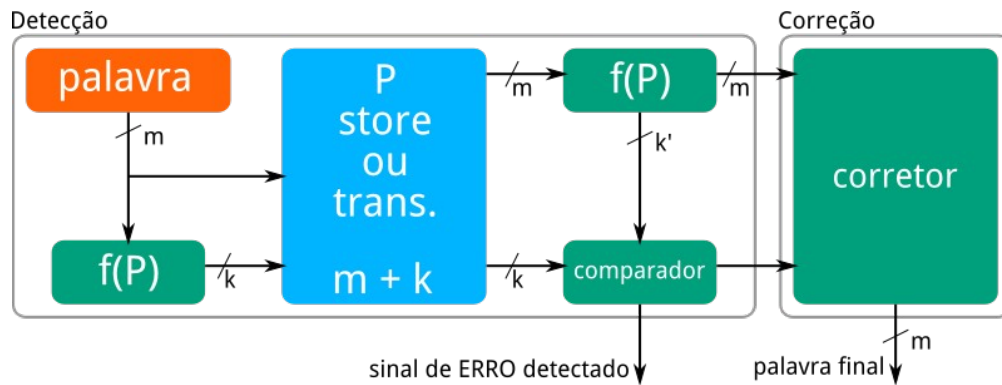
```
A202 - Entrada e Saída.pdf: SUCESSO
```

```
A301 - Hierarquia de Memórias.pdf: FALHOU
```

```
sha256sum: AVISO: 1 soma de verificação calculada NÃO coincide
```

indicando que o arquivo foi alterado ou está com problemas.

Funcionamento do processo de detecção e correção de erros:



Hamming SEC – Single Error Correction:

Em uma palavra de **8** bits precisa-se de **4** bits para detecção de erros, totalizando 12 bits para a palavra final:

Determinação do número de bits é dado por:

$$2^k - 1 \geq m + k$$

Onde:

k = Tamanho da palavra de correção

m = Tamanho da palavra

Desta forma:

$$2^4 - 1 \geq 8 + 4 \rightarrow 15 \geq 12$$

Disposição dos bits para palavra de 8 bits + ECC:

- Dados: **M**

- Teste: **C**

Tabela 1: Disposição dos bits de Teste **C** e Dados **M**.

12	11	10	9	8	7	6	5	4	3	2	1
M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1
1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001

Bits de Dados **M** válidos para cálculo dos bits de teste **C**:

Tabela 2: Bits de Teste **C** e seus respectivos bits de Dados **M**.

C1		M7		M5	M4		M2	M1
C2		M7	M6		M4	M3		M1
C4	M8				M4	M3	M2	
C8	M8	M7	M6	M5				

Nota-se que, cada bit teste **C** calcula a paridade (xor) dos demais bits cuja posição contenha o valor 1 correspondente a este:

Tabela 3: Exemplo de equivalência entre o bit de teste **C1** e os bits de Dados **M**.

C1	M1	M2	M3	M4	M5	M6	M7	M8
0001	0011	0101	0110	0111	1001	1010	1011	1100

Passos:

Durante a Escrita:

- 1 – Cálculo do bit de teste **C**: **Paridade** dos bits de dados **M** respeitando a tabela 2;
- 2 – Aloca os bits de dados **M** e de teste **C** na mesma palavra.

Durante a Leitura:

- 1 – Lê-se a palavra completa, e extrair os bits de Dados **M** e Teste **C**;
- 2 – Cálculo do novo bit de teste **C'**;
- 3 – Compara-se o bit de teste **C** com novo bit de teste **C'** (com **ou-exclusivo**);
- 4a – Palavra síndrome diferente de 0 e menor do que M:
Dados **M** incorreto -> Identificar e recuperar;
Teste **C** incorreto -> Assume-se que os bits **M** estejam corretos.
- 4b – Palavra síndrome maior do que M
Dados **M** incorreto e impossível de recuperar (Rejeição Automática)
- 4b – Palavra síndrome igual a 0: Dados **M** correto.

Hamming SEC-DED – Single Error Correction – Double Error Detection:

Adiciona-se um bit extra **G** que representa a paridade geral da palavra, evitando piorar a situação caso a palavra contenha mais do que um único erro.

Tabela 4: Posicionamento do bit **G**, dos bits de teste **C** e dos bits de Dados **M**.

12	11	10	9	8	7	6	5	4	3	2	1	0
M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1	G
1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001	1101

Gravação:

- O bit **G** é calculado após o cálculo da palavra **C**, em conjunto com **M**.
Aplicação de ou-exclusivo (bit de paridade!)

Leitura:

- Após o processo de recuperação SEC, calcula-se um novo bit **G'**, utilizando **M** corrigida e **C'**
Compara-se **G** com **G'**
Se forem iguais, a palavra **M** está correta
Se forem diferentes, 2 erros aconteceram e **SEC** piorou a situação
Palavra **M** deve ser Rejeitada!

Hamming SEC-DED – Exemplos

Para armazenamento com integridade Hamming:

Considere a palavra de um byte **0x61**, que representa o caractere **a** em ASCII.

0x61 → **0b0110 0001**

Tamanho da palavra de teste:

$$2^k - 1 \geq m + k \rightarrow 2^4 - 1 \geq 8 + 4 \rightarrow 15 \geq 12 \rightarrow k = 4$$

Dados **M**:

M8	M7	M6	M5	M4	M3	M2	M1
0	1	1	0	0	0	0	1

Teste **C**:

C8	C4	C2	C1

Tabela inicial SEC-DED:

M8	M7	M6	M5	M4	M3	M2	M1
0	1	1	0	0	0	0	1

Cálculo dos bits de teste **C** com base na tabela 2:

M8	M7	M6	M5	M4	M3	M2	M1		
0	1	1	0	0	0	0	1		xor
	1		0	0		0	1	C1	0
	1	1		0	0		1	C2	1
0				0	0	0		C4	0
0	1	1	1					C8	0

Cálculo do bit G:

M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1	xor
0	1	1	0	0	0	0	0	0	1	1	0	0

Resultado para armazenamento da palavra M+C+G:

12	11	10	9	8	7	6	5	4	3	2	1	0
M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1	G
0	1	1	0	0	0	0	0	0	1	1	0	0

Palavra final com **Padding** de 13 bits para 16 bits:

0000 1100 0000 1100 → 0x0C0C

Para leitura com integridade Hamming SEM FALHA:

Palavra armazenada originalmente: 0x0C0C

Palavra lida: 0x0C0C

Extrair M, C e G:

12	11	10	9	8	7	6	5	4	3	2	1	0
M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1	G
0	1	1	0	0	0	0	0	0	1	1	0	0

Cálculo dos novos bits de teste **C'** com base na tabela 2:

M8	M7	M6	M5	M4	M3	M2	M1		
0	1	1	0	0	0	0	1		xor
	1		0	0		0	1	C1'	0
	1	1		0	0		1	C2'	1
0				0	0	0		C4'	0
0	1	1	0					C8'	0

Cálculo da palavra síndrome:

0010 ← **C'**

xor 0010 ← **C**

0000 → palavra síndrome é zero → **ERRO não detectado!**

Cálculo de novo **G'**:

M8	M7	M6	M5	C8'	M4	M3	M2	C4'	M1	C2'	C1'	xor
0	1	1	0	0	0	0	0	0	1	1	0	0

0 ← **G'**

xor 0 ← **G**

0 → paridade geral é igual → **PALAVRA ACEITA**

Para leitura com integridade Hamming COM 1 FALHA em M:

Palavra armazenada originalmente: 0x0C0C

Palavra lida: 0x0C4C

Extrair M, C e G:

12	11	10	9	8	7	6	5	4	3	2	1	0
M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1	G
0	1	1	0	0	0	1	0	0	1	1	0	0

Cálculo dos novos bits de teste **C'** com base na tabela 2:

M8	M7	M6	M5	M4	M3	M2	M1		
0	1	1	0	0	1	0	1		xor
	1		0	0		0	1	C1'	0
	1	1		0	1		1	C2'	0
0				0	1	0		C4'	1
0	1	1	0					C8'	0

Cálculo da palavra síndrome:

0100 ← **C'**

xor 0010 ← **C**

0110 → palavra síndrome é 6 → **ERRO detectado no índice 6!**

Correção:

12	11	10	9	8	7	6	5	4	3	2	1	0
M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1	G
0	1	1	0	0	0	1	0	0	1	1	0	0
0	1	1	0	0	0	0	0	0	1	1	0	0

Cálculo de novo **G'** após correção:

M8	M7	M6	M5	C8'	M4	M3	M2	C4'	M1	C2'	C1'	xor
0	1	1	0	0	0	0	0	1	1	0	0	0

0 ← **G'**

xor 0 ← **G**

0 → paridade geral é igual → **PALAVRA ACEITA!**

Para leitura com integridade Hamming COM 1 FALHA em C:

Palavra armazenada originalmente: 0x0C0C

Palavra lida: 0x0D0C

Extrair M, C e G:

12	11	10	9	8	7	6	5	4	3	2	1	0
M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1	G
0	1	1	0	1	0	0	0	0	1	1	0	0

Cálculo dos novos bits de teste **C'** com base na tabela 2:

M8	M7	M6	M5	M4	M3	M2	M1		
0	1	1	0	0	0	0	1		xor
	1		0	0		0	1	C1'	0
	1	1		0	0		1	C2'	1
0				0	0	0		C4'	0
0	1	1	0					C8'	0

Cálculo da palavra síndrome:

0010 ← **C'**

xor 1010 ← **C**

1000 → palavra síndrome é 8 → **ERRO detectado no índice 8!**

Correção (opcional, pois C não é usado para cálculo de G', mas sim C'):

12	11	10	9	8	7	6	5	4	3	2	1	0
M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1	G
0	1	1	0	1	0	0	0	0	1	1	0	0
0	1	1	0	0	0	0	0	0	1	1	0	0

Cálculo de novo **G'** após correção:

M8	M7	M6	M5	C8'	M4	M3	M2	C4'	M1	C2'	C1'	xor
0	1	1	0	0	0	0	0	0	1	1	0	0

0 ← **G'**

xor 0 ← **G**

0 → paridade geral é igual → **PALAVRA ACEITA!**

Para leitura com integridade Hamming COM 2 FALHAS – índice maior do que m+k:

Palavra armazenada originalmente: 0x0C0C

Palavra lida: 0x0D4C

Extrair M, C e G:

12	11	10	9	8	7	6	5	4	3	2	1	0
M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1	G
0	1	1	0	1	0	1	0	0	1	1	0	0

Cálculo dos novos bits de teste C' com base na tabela 2:

M8	M7	M6	M5	M4	M3	M2	M1		
0	1	1	0	0	1	0	1		xor
	1		0	0		0	1	C1'	0
	1	1		0	1		1	C2'	0
0				0	1	0		C4'	1
0	1	1	0					C8'	0

Cálculo da palavra síndrome:

0100 ← C'

xor 1010 ← C

1110 → palavra síndrome é 0xE → ERRO detectado no índice 14(!?)

Correção:

Não aplicável, já que índice síndrome é maior do que limite da palavra m+k (13)

Cálculo de novo G' após correção:

M8	M7	M6	M5	C8'	M4	M3	M2	C4'	M1	C2'	C1'	xor
0	1	1	0	0	0	1	0	1	1	0	0	1

1 ← G'

xor 0 ← G

1 → paridade geral é diferente → PALAVRA REJEITADA!

Para leitura com integridade Hamming COM 2 FALHAS – índice menor do que m+k:

Palavra armazenada originalmente: 0x0C0C

Palavra lida: 0x0C14

Extrair M, C e G:

12	11	10	9	8	7	6	5	4	3	2	1	0
M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1	G
0	1	1	0	0	0	0	0	1	0	1	0	0

Cálculo dos novos bits de teste **C'** com base na tabela 2:

M8	M7	M6	M5	M4	M3	M2	M1		
0	1	1	0	0	0	0	0		xor
	1		0	0		0	0	C1'	1
	1	1		0	0		0	C2'	0
0				0	0	0		C4'	0
0	1	1	0					C8'	0

Cálculo da palavra síndrome:

0001 ← **C'**

xor 0110 ← **C**

0111 → palavra síndrome é 7 → **ERRO detectado no índice 7!**

Correção:

Aplicável, já que índice síndrome é menor do que limite da palavra m+k (13)

12	11	10	9	8	7	6	5	4	3	2	1	0
M8	M7	M6	M5	C8	M4	M3	M2	C4	M1	C2	C1	G
0	1	1	0	0	0	0	0	1	1	1	0	0
0	1	1	0	0	1	0	0	1	1	1	0	0

Cálculo de novo **G'** após correção:

M8	M7	M6	M5	C8'	M4	M3	M2	C4'	M1	C2'	C1'	xor
0	1	1	0	0	1	1	0	0	0	0	1	1

1 ← **G'**

xor 0 ← **G**

1 → paridade geral é diferente → **PALAVRA REJEITADA!**