# Otimização do Programa da Aula 6

Desenvolvida por: Raul A. Gonzalez Augusto, RA.: 211023698

Objetivo: objetivo desta atividade é utilizar técnicas já aplicadas anteriormente nesta disciplina para otimizar um programa que extrai textos ocultos de uma imagem.

# Computador Usado:

Avell High Performance A52 LIV

* Processador: Intel® Core™ i5-10300H CPU @ 2.50GHz
  + Cores: 4
  + Threads: 8
  + Frequência Máxima: 4.50 GHz
  + Cache: 8 MB Intel® Smart Cache
  + Bus Speed: 8 GT/s
* Placa de Vídeo: NVIDIA GeForce GTX 1650 Ti e Intel® UHD Graphics
* RAM: 16 GB DDR4-2666 Dual Channel
* Armazenamento: 500 GB nvme m.2 SSD 2GB/s E 1 Tera HDD 5400 rpm
* Sistema Operacional: Windows 10 pro 64bits

# Programas

Compilado no gcc máquina virtual Linux.

# Operadores de bits

São operadores que funcionam de bit a bit. Tem a mesma lógica das portas logicas aplicada em circuitos eletrônicos.

|: OR (União booleana), seria equivalente a uma soma. Exemplo: 1 + 1 = 1, 1 + 0 = 1, 0 + 0 = 0.

&: AND (intersecção booleana), equivalente a uma multiplicação. Exemplo: 1 \* 1 = 1, 1 \* 0 = 0 , 0 \* 0 = 0

^: XOR(diferença simétrica booleana), é o ou exclusivo, ou seja, 0 + 0 e 1 + 1 resultam em 0 (falso) e 0 + 1 e 1 + 0 em 1 (verdadeiro).

~: NOT (complemento) inverte o resultado logico, é a negação, exemplo: ~(1 + 1) = 0

<<: deslocamento a esquerda, exemplo: 00000100 (4)<< 2 = 00010000 (16), é como uma multiplicação por inteiro. 4 \* 2 \* 2 = 16.

>>: deslocamento a direita, exemplo: 00010110 (22) >> 1 = 00001011 (11) é como uma divisão por inteiro. 22 / 2 = 11.

# Explicando operações do programa

c |= 1 << nBit;: c recebe c OR 1 deslocado nBit para a esquerda.

c &= ~(1 << nBit);: c recebe c AND a negação de 1 deslocado nBit para a esquerda.

return (p.r & 1) != 0;: retorna o resultado da expressão logica p.r AND 1 diferente de 0?

# Executando o programa original

Tempos:

0.808813

0.815336

0.807948

0.808376

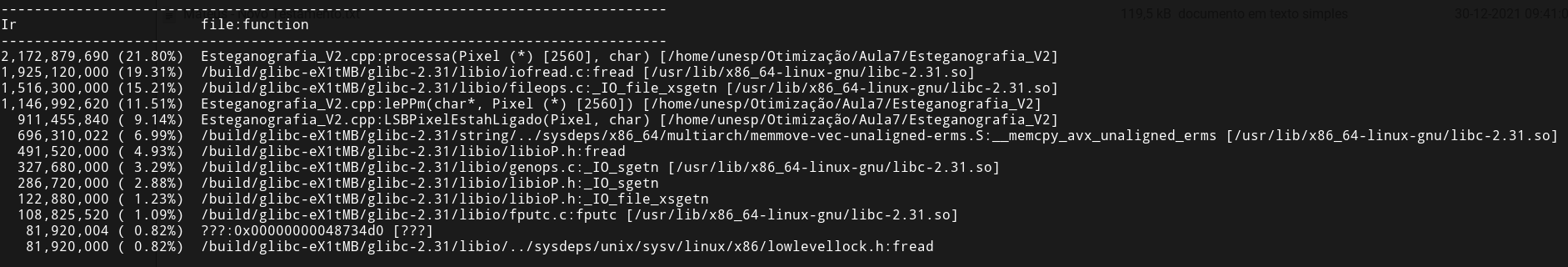
0.837254

Media: 0.8155454

Foram gerados os arquivos: Joao e Atos - Novo Testamento.txt, Marcos e Lucas - Novo Testamento.txt, Mateus - Novo Testamento.txt.

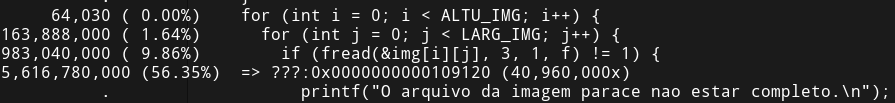
Os 3 textos estão legíveis.

file:function:



# Função das bibliotecas que está gastando muito tempo

Auto-annotated source: Esteganografia\_V2.cpp:

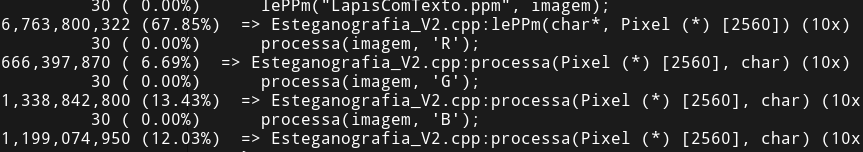


O fread está gastando muito tempo, com uma porcentagem de 56.35% do tempo no Auto\_annotated source e 19.31% no file:function.

Eu pensei e tentei diversas maneiras de otimizar o fread, mas sem sucesso, ou resultava em Segmentation fault, ou os textos não eram salvos nos arquivos e até deu um erro de compilação então optei em não mexer no fread.

# Função do programa que está gastando muito tempo

Auto-annotated source: Esteganografia\_V2.cpp:



A função lePPm está gastando muito tempo, com uma porcentagem de 67.85% do tempo no auto-annotated source e 11.51% no file:function e a função processa está gastando 6.69%, 13.43% e 12.03% em cada uma de suas chamadas no auto-annotated source e 21.80% no file:function .

A lePPm esta consumindo muito tempo de execução devido ao fread, porém como não encontrei uma maneira de otimizar o fread acabei não mexendo nela.

# Desenrolar código

Tinha fors no processaR, processaG, processaB, e no lePPm que eu poderia desenrolar, porém não o fiz pois achei que ia aumentar muito o tamanho do código.

# Separar canais RGB do processa

Separei os canais para cada um usar uma função própria pro canal.

# Otimização

Eu diminui a quantidade de ifs na função processa, além disso dividi ela em 3 funções, processaR, processaG, processaB, uma para cada canal de cor e removi a função LSBPixelEstahLigado, como era uma função simples implementei o código dentro do processa, para assim reduzir a chamadas de funções.

Tempos:

0.736957

0.710054

0.711219

0.727291

0.721246

Media: 0.7213534

Porcentagem: 11.54% de aumento de velocidade.

Foram gerados os arquivos: Joao e Atos - Novo Testamento.txt, Marcos e Lucas - Novo Testamento.txt, Mateus - Novo Testamento.txt.

Os 3 textos estão legíveis.

# Programa Otimizado:

#include <cctype>

#include <cmath>

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <ctime>

#define LARG\_IMG 2560

#define ALTU\_IMG 1600

struct Pixel {

  unsigned char r, g, b;

};

Pixel imagem[ALTU\_IMG][LARG\_IMG];

Pixel imagemSai[ALTU\_IMG][LARG\_IMG];

char \*leLinha(FILE \*f) {

  static char str[1000];

  unsigned char c;

  int j = 0;

  c = fgetc(f);

  while (c == '#') {

    do {

      c = fgetc(f);

    } while (c != '\n');

    while (isspace(c = fgetc(f)))

      ;

  }

  do {

    str[j] = c;

    j++;

  } while (!isspace(c = fgetc(f)));

  str[j] = 0;

  return str;

}

void lePPm(char nomeArq[], Pixel img[ALTU\_IMG][LARG\_IMG]) {

  int larg, altu, cores, tam;

  char \*p;

  FILE \*f = fopen(nomeArq, "rb");

  if (!f) {

    printf("O arquivo da imagem não pode ser aberto para leitura.\n");

    exit(1);

  }

  if (strcmp(leLinha(f), "P6")) {

    printf("Não é um arquivo no formato PPM (P6) de imagem.\n");

    exit(1);

  }

  p = leLinha(f);

  larg = atoi(p);

  if (LARG\_IMG != larg) {

    printf("A imagem apresenta largura %d, mas o valor desejado e' %d.\n", larg,

           LARG\_IMG);

    exit(1);

  }

  p = leLinha(f);

  altu = atoi(p);

  if (ALTU\_IMG != altu) {

    printf("A imagem apresenta altura %d, mas o valor desejado e' %d.\n", altu,

           ALTU\_IMG);

    exit(1);

  }

  p = leLinha(f);

  cores = atoi(p);

  if (cores != 255) {

    printf("A imagem tem um valor máximo de %d, mas a valor de cor é de 255.\n",

           cores);

    exit(1);

  }

  for (int i = 0; i < ALTU\_IMG; i++) {

    for (int j = 0; j < LARG\_IMG; j++) {

      if (fread(&img[i][j], 3, 1, f) != 1) {

        printf("O arquivo da imagem parace nao estar completo.\n");

        exit(1);

      }

    }

  }

  fclose(f);

}

void ligaBit(char &c, int nBit) { c |= 1 << nBit; }

void desligaBit(char &c, int nBit) { c &= ~(1 << nBit); }

void processaR(Pixel img[ALTU\_IMG][LARG\_IMG]) {

  char letra;

  int posBit;

  char nomeArq[200];

  int indLetra = 0;

  bool test;

  for (int j = 0; j < LARG\_IMG; j++) {

    posBit = j % 8;

    test = (img[0][j].r & 1) != 0;

    if (posBit == 0) {

      letra = 0;

      if (test) {

        letra |= 0x01;

      }

    } else if (posBit == 1 && test) {

        letra |= 0x02;

    } else if (posBit == 2 && test) {

        letra |= 0x04;

    } else if (posBit == 3 && test) {

        letra |= 0x08;

    } else if (posBit == 4 && test) {

        letra |= 0x10;

    } else if (posBit == 5 && test) {

        letra |= 0x20;

    } else if (posBit == 6 && test) {

        letra |= 0x40;

    } else if (posBit == 7) {

      if (test) {

        letra |= 0x80;

      }

      nomeArq[indLetra++] = letra;

      if (letra == '\0')

        break;

    }

  }

  FILE \*fp = fopen(nomeArq, "wb");

  for (int i = 1; i < ALTU\_IMG; i++) {

    for (int j = 0; j < LARG\_IMG; j++) {

      posBit = j % 8;

      test = (img[i][j].r & 1) != 0;

      if (posBit == 0) {

        letra = 0;

        if (test) {

          letra |= 0x01;

        }

      } else if (posBit == 1 && test) {

          letra |= 0x02;

      } else if (posBit == 2 && test) {

          letra |= 0x04;

      } else if (posBit == 3 && test) {

          letra |= 0x08;

      } else if (posBit == 4 && test) {

          letra |= 0x10;

      } else if (posBit == 5 && test) {

          letra |= 0x20;

      } else if (posBit == 6 && test) {

          letra |= 0x40;

      } else if (posBit == 7) {

        if (test) {

          letra |= 0x80;

        }

        if (letra == '\0') {

          i = ALTU\_IMG;

          break;

        }

        fputc(letra, fp);

      }

    }

  }

  fclose(fp);

}

void processaG(Pixel img[ALTU\_IMG][LARG\_IMG]) {

  char letra;

  int posBit;

  char nomeArq[200];

  int indLetra = 0;

  bool test;

  for (int j = 0; j < LARG\_IMG; j++) {

    posBit = j % 8;

    test = (img[0][j].g & 1) != 0;

    if (posBit == 0) {

      letra = 0;

      if (test) {

        letra |= 0x01;

      }

    } else if (posBit == 1 && test) {

        letra |= 0x02;

    } else if (posBit == 2 && test) {

        letra |= 0x04;

    } else if (posBit == 3 && test) {

        letra |= 0x08;

    } else if (posBit == 4 && test) {

        letra |= 0x10;

    } else if (posBit == 5 && test) {

        letra |= 0x20;

    } else if (posBit == 6 && test) {

        letra |= 0x40;

    } else if (posBit == 7) {

      if (test) {

        letra |= 0x80;

      }

      nomeArq[indLetra++] = letra;

      if (letra == '\0')

        break;

    }

  }

  FILE \*fp = fopen(nomeArq, "wb");

  for (int i = 1; i < ALTU\_IMG; i++) {

    for (int j = 0; j < LARG\_IMG; j++) {

      posBit = j % 8;

      test = (img[i][j].g & 1) != 0;

      if (posBit == 0) {

        letra = 0;

        if (test) {

          letra |= 0x01;

        }

      } else if (posBit == 1 && test) {

          letra |= 0x02;

      } else if (posBit == 2 && test) {

          letra |= 0x04;

      } else if (posBit == 3 && test) {

          letra |= 0x08;

      } else if (posBit == 4 && test) {

          letra |= 0x10;

      } else if (posBit == 5 && test) {

          letra |= 0x20;

      } else if (posBit == 6 && test) {

          letra |= 0x40;

      } else if (posBit == 7) {

        if (test) {

          letra |= 0x80;

        }

        if (letra == '\0') {

          i = ALTU\_IMG;

          break;

        }

        fputc(letra, fp);

      }

    }

  }

  fclose(fp);

}

void processaB(Pixel img[ALTU\_IMG][LARG\_IMG]) {

  char letra;

  int posBit;

  char nomeArq[200];

  int indLetra = 0;

  bool test;

  for (int j = 0; j < LARG\_IMG; j++) {

    posBit = j % 8;

    test = (img[0][j].b & 1) != 0;

    if (posBit == 0) {

      letra = 0;

      if (test) {

        letra |= 0x01;

      }

    } else if (posBit == 1 && test) {

        letra |= 0x02;

    } else if (posBit == 2 && test) {

        letra |= 0x04;

    } else if (posBit == 3 && test) {

        letra |= 0x08;

    } else if (posBit == 4 && test) {

        letra |= 0x10;

    } else if (posBit == 5 && test) {

        letra |= 0x20;

    } else if (posBit == 6 && test) {

        letra |= 0x40;

    } else if (posBit == 7) {

      if (test) {

        letra |= 0x80;

      }

      nomeArq[indLetra++] = letra;

      if (letra == '\0')

        break;

    }

  }

  FILE \*fp = fopen(nomeArq, "wb");

  for (int i = 1; i < ALTU\_IMG; i++) {

    for (int j = 0; j < LARG\_IMG; j++) {

      posBit = j % 8;

      test = (img[i][j].b & 1) != 0;

      if (posBit == 0) {

        letra = 0;

        if (test) {

          letra |= 0x01;

        }

      } else if (posBit == 1 && test) {

          letra |= 0x02;

      } else if (posBit == 2 && test) {

          letra |= 0x04;

      } else if (posBit == 3 && test) {

          letra |= 0x08;

      } else if (posBit == 4 && test) {

          letra |= 0x10;

      } else if (posBit == 5 && test) {

          letra |= 0x20;

      } else if (posBit == 6 && test) {

          letra |= 0x40;

      } else if (posBit == 7) {

        if (test) {

          letra |= 0x80;

        }

        if (letra == '\0') {

          i = ALTU\_IMG;

          break;

        }

        fputc(letra, fp);

      }

    }

  }

  fclose(fp);

}

int main() {

  int j;

  clock\_t inicio, final;

  double duracao;

  printf("Iniciando processamento:\n");

  inicio = clock();

  for (j = 0; j < 10; j++) {

    lePPm("LapisComTexto.ppm", imagem);

    processaR(imagem);

    processaG(imagem);

    processaB(imagem);

  }

  final = clock();

  duracao = (double)(final - inicio) / CLOCKS\_PER\_SEC;

  printf("Tempo utilizado no processamento = %f\n", duracao);

  return 0;

}