Proiect JAVA

Image resizing (Zooming +/-) – keeping aspect ratio. Pixel replication method

1. Introducere:

a. Prezentarea Proiectului:

i. În cadrul proiectului propus, am dezvoltat o aplicație de prelucrare a imaginilor în format BMP, respectând criterii stricte de implementare. Am pus accent pe utilizarea exclusivă a algoritmilor și secvențelor de cod low-level, excluzând orice altă metodă de procesare.

ii. Implementarea include în întregime conceptele de programare orientată pe obiect (POO), precum incapsulare, moștenire, polimorfism și abstractizare. Codul sursă respectă în totalitate standardele de codificare și este bine comentat pentru o înțelegere clară a funcționalităților.

b. Obiectivele proiectului:

i. Am integrat operațiuni de lucru cu fișiere, permițând utilizatorului să interacționeze atât prin tastatură, cât și prin parametri liniei de comandă pentru gestionarea fișierelor de intrare și setările de execuție.

ii. Arhitectura aplicației este mult modulară, cu cel puțin trei niveluri de moștenire între clase, evidențiind organizarea logică și eficientă a componentelor.

iii. De asemenea, am inclus varargs, constructori, blocuri de inițializare și elemente de interfață în codul sursă pentru a adăuga flexibilitate și extensibilitate.

iv. Implementarea cuprinde și gestionarea excepțiilor pentru a asigura stabilitatea și siguranța aplicației în fața unor situații neprevăzute.

v. Pentru a îmbunătăți eficiența, am integrat un model Producer-Consumer cu comunicare multithread, unde fiecare thread își îndeplinește rolul în procesarea imaginii.

vi. De asemenea, aplicația utilizează comunicarea prin pipes, cu transmiterea și recepția segmentelor de informație, contribuind astfel la o structură bine organizată și modulară.

2. Algoritmi Low-Level:

a. Clasa RGB:

i. Metoda "*RGBRead*" inițializează un obiect de tip „*SharedBuffer*" și un array bidimensional numit „*pixelArray*", având rolul de a stoca valorile RGB ale pixelilor. Se efectuează și calculul numărului de coloane per thread pentru a eficientiza procesul de procesare a imaginii. Apoi, într-o buclă for, se inițializează și se lansează obiecte de tip „*Producer*" și „*Consumer*", creând astfel thread-uri pentru a procesa imaginea. Așteaptă apoi ca toate thread-urile „*Consumer*" să se termine de executat folosind metoda „*join*". Măsoară timpul total de citire și afișează rezultatul în milisecunde. În final, imaginea este procesată ulterior cu ajutorul metodei *„imageProcess"*.

ii. Metoda „imageProcess" are rolul de a redimensiona imaginea la noile dimensiuni specificate. În primul rând, calculează raportul de aspect pentru a păstra proporțiile inițiale ale imaginii. Apoi, creează o imagine redimensionată și calculează factorii de scalare pe axele X și Y. Într-un dublu for loop, parcurge imaginea redimensionată și ajustează valorile RGB ale fiecărui pixel conform unui factor de multiplicare (în acest caz, 1.5). Se asigură că valorile rezultate rămân în intervalul valid **0-255**.

iii. Metodele „*getRGBValues*" și „*setRGBValues*" sunt folosite în procesul de extragere și setare a valorilor RGB ale unui pixel specificat din imagine, contribuind astfel la manipularea și ajustarea acestor valori în cadrul operațiilor de prelucrare a imaginii.

3. Partea Teoretică:

a. Fundamentul teoretic al proiectului:

i. **Programarea Orientată pe Obiect** (POO) este o paradigmă de programare care se bazează pe conceptul de "obiecte". Un obiect este o instanță a unei clase și reprezintă o entitate care încorporează date și comportamente asociate cu acele date.

ii. **Modelul Producer-Consumer** este un model de cooperare între două tipuri de thread-uri: unul care produce date și le pune într-un buffer comun, și altul care consumă acele date din acel buffer.

iii. **Comunicarea prin Pipes** se referă la schimbul de date între două procese sau două thread-uri, utilizând structuri denumite "pipes". Principalele operații în comunicarea prin pipes includ scrierea și citirea de date. Procesele sau thread-urile pot utiliza pipes pentru a trimite și a primi date în mod eficient, facilitând colaborarea între componente separate ale unui sistem.

b. Concepte și tehnologii cheie:
   i. Rolul **producătorului** constă în a produce și adăuga date în buffer. Are capacitatea de a genera date cu o viteză mai mare decât consumatorul poate prelua.
   ii. **Consumatorul** este responsabil pentru preluarea și procesarea datelor din buffer. Poate procesa date într-un ritm mai rapid decât producătorul le furnizează.
   iii. **Buffer (sau Coada)** reprezintă o zonă comună unde producătorul introduce date și de unde consumatorul le extrage. Necesită o gestionare atentă pentru a evita conflictele și pentru a asigura sincronizarea între producător și consumator.
   iv. **Sincronizarea** este esențială pentru a preveni condițiile de cursă și pentru a asigura accesul securizat la buffer între thread-urile producător și consumator.
   v. **!!** Modelul poate implica blocarea producătorului sau consumatorului atunci când buffer-ul este gol sau plin. Această abordare asigură o cooperare eficientă între cele două entități și previne utilizarea inutilă a resurselor.

4. Descrierea Implementării:

***Pachetul** PackWork - **Clasa** RGB (sursa principală a implementării)

a. Atribuții statice:
   - *numThreads*: Numărul de thread-uri utilizate pentru procesarea imaginii.
   - *w, h*: Dimensiunile imaginii procesate.
   - *endTime1, endTime2*: Timpii de încheiere a citirii și procesării imaginii.

b. Bloc de inițializare:
   - Inițializează numThreads cu 4 în cazul în care utilizatorul nu selectează alt număr.

c. Metode:
   - *getEndTime2()*: Returnează endTime2; *setThreads(int threads)*: Setează numărul de thread-uri; *setDimension(int w, int h, long endTime1)*: Setează dimensiunile imaginii și timpul de începere al procesului.

d. Metoda *RGBRead(BufferedImage image)*:
   - Inițializează un obiect *SharedBuffer*.
   - Calculează numărul de coloane per thread.
   - Inițializează array-ul bidimensional *pixelArray*.
   - Creează și pornește thread-urile *Producer* și *Consumer* pentru fiecare segment de imagine.
   - Așteaptă finalizarea tuturor thread-urilor Consumer.

- Măsoară timpul total de citire și afișează rezultatul.
- Procesează imaginea cu metoda *imageProcess*.
- Returnează imaginea redimensionată.

e. Metoda *imageProcess(int[][] pixelArray, BufferedImage image, int newWidth, int newHeight)**:***
   - Redimensionează imaginea la noile dimensiuni specificate.
   - Ajustează valorile RGB conform unui factor de multiplicare și le menține în intervalul 0-255.

f. Metode ajutătoare:
   - *getRGBValues(BufferedImage image, int x, int y*): Extrage valorile RGB ale unui pixel specificat din imagine.
   - *setRGBValues(BufferedImage resizedImage, int x, int y, int[] rgbValues):* Setează noile valori RGB pentru un pixel specificat într-o imagine.

***Pachetul** PackWork **- Clasa** FileManipulation **(extinde clasa** RGB**):**

g. Atribute:
   - **file, OutputPath**: Calea fișierului de intrare și de ieșire.
   - **resizedImage**: Imaginea procesată.

h. Metode:
   - *FileRead():* Realizează citirea fișierului și apelul metodei RGBRead pentru procesare.
   - *FileWrite():* Realizează scrierea fișierului și utilizarea modelului Producer-Consumer pentru transmiterea datelor între thread-uri prin pipe-uri.
   - *Traverse(int[][] pixelArray, BufferedImage finalImage):* Plasează valorile pixelilor în imaginea finală.
   - *SaveImage(BufferedImage finalImage, String OutputPath):* Salvează imaginea într-un fișier.
   - *DisplayImage(BufferedImage image):* Afișează imaginea într-o fereastră (opțional).

***Pachetul** PackWork - **Interfața** și **Clasa** ImageResultWriter:

i. Interfața WriterResult:
   - Definește metodele *setPixelValue* și *getPixelValue* pentru a manipula valorile pixelilor.

j. Clasa ImageResultWriter (implementează WriterResult):
   - Oferă o implementare a interfeței, gestionând un array bidimensional de pixeli și un array corespunzător de obiecte de tip **Lock** pentru sincronizare.
   - Folosește un mecanism de *locking fin-grained* pentru a asigura accesul sigur la valorile pixelilor.

***Pachetul** PackTest - **Clasa** MyMain:

k. Metoda main(String[] args):
   - Colectează input-ul utilizatorului pentru selecția imaginii, numărul de fire de execuție, dimensiunile și calea de ieșire.
   - Inițializează un obiect FileManipulation în funcție de opțiunea aleasă și setările utilizatorului.
   - Apelurile **FileRead** și **FileWrite** sunt făcute pentru prelucrarea și salvarea imaginii.

5. Evaluare Performanțe:

Pentru a evalua performanțele am ales ca exemplu:

a. Imaginea *Tesla.bmp* (640x427 pixeli) cu citire pe 4 threaduri , redimensionată la 400x420 pixeli cu scrierea în *Out.bmp*

```
Image to choose:
 1-Tesla
 2-Corgi
 3-Something
Select:
1
Number of proceses for execution:
4
Resize width and height:
400 427
Name an output:
Out
Elapsed Time Identification: 9614413300 milliseconds
Trying to open the file...
File open Tesla.bmp
Image width: 640
Image height: 427
Image type: 5
Elapsed Time Reading: 741388100 milliseconds
Elapsed Time Reading: 952329000 milliseconds
Elapsed Time Processing: 50195100 milliseconds
Image saved successfully to: Out.bmp
Elapsed Time Writing: 435201300 milliseconds
```

Figură 1 Test 1

b. Imaginea Corgi.bmp (640x960 pixeli) cu citire pe 5 threaduri , redimensionată la 500x480 pixeli cu scrierea în *Out1.bmp*

```
Image to choose:
 1-Tesla
 2-Corgi
 3-Something
Select:
2
Number of proceses for execution:
5
Resize width and height:
500 480
Name an output:
Out1
Elapsed Time Identification: 19345078700 milliseconds
Trying to open the file...
File open Corgi.bmp
Image width: 640
Image height: 960
Image type: 5
Elapsed Time Reading: 942011300 milliseconds
Elapsed Time Reading: 1271115500 milliseconds
Elapsed Time Processing: 54321000 milliseconds
Image saved successfully to: Out1.bmp
Elapsed Time Writing: 504271700 milliseconds
```

c. Imaginea Something.bmp (640x800 pixeli) cu citire pe 3 threaduri , redimensionată la 300x220 pixeli cu scrierea în *Out2.bmp*

```
Image to choose:
 1-Tesla
 2-Corgi
 3-Something
Select:
3
Number of proceses for execution:
3
Resize width and height:
300 220
Name an output:
Out2
Elapsed Time Identification: 15108120800 milliseconds
Trying to open the file...
File open Something.bmp
Image width: 640
Image height: 800
Image type: 5
Elapsed Time Reading: 834948200 milliseconds
Elapsed Time Reading: 1075419900 milliseconds
Elapsed Time Processing: 42137200 milliseconds
Image saved successfully to: Out2.bmp
Elapsed Time Writing: 239838700 milliseconds
```

6. Concluzie:

În concluzie, proiectul implementat reușește să îmbine eficientitatea procesării imaginilor cu conceptele avansate de programare orientată pe obiect. Utilizarea modelului Producer-Consumer și comunicarea prin Pipes contribuie la o gestionare eficientă a datelor, iar implementarea modulară și respectarea standardelor de cod conferă proiectului o structură clară și ușor de înțeles. Performanța sistemului în ceea ce privește procesarea imaginilor este optimizată prin utilizarea mai multor thread-uri, iar rezultatele obținute sunt vizualizate și salvate în mod corespunzător. Proiectul evidențiază aplicarea concretă a principiilor de programare orientată pe obiect într-un context practic și reprezintă o soluție robustă pentru manipularea eficientă a imaginilor folosind algoritmi de procesare low-level.

De menționat că datorită unei erori rezultate din concurența proceselor la nivel de scriere , imaginea rezultată conține pixeli amestecați. Am încercat remedierea acestui impediment, reușind doar 70% din imagine corectă.