ZÁPADOČESKÁ
UNIVERZITA
V PLZNI

Semestrální práce z předmětu

KIV/PPR

Transformace obrazu

Milan Balcar (A06072)
Narozen 16. 1. 1984
mbalcar@students.zcu.cz

Obsah

1 Zadání

Realizujte transformaci digitalizovaného šedotónového obrazu zadaného jako matice (m, n) s celočíselnými prvky. Hodnota prvků je v rozmezí **<0,k>** a udává jas příslušného pixelu. Transformace spočívá ve vytvoření "negativu" původního obrazu.

Povinné parametry:

- počet vláken (není nutné u MPI, tam se zadává počet vláken při spuštění prostředí)
- velikost kusu práce přidělovaného vláknům (opět není nutné u některých úloh v MPI)
- matice (není nutné načítat opravdové obrázky (plus body))
- hodnota **k** (nemusí být zadána explicitně - může být např. v souboru s daty)

Poznámky k řešení:

- negativ se vytváří podle vzorce new(x,y) = k - old(x,y)

Zadanou úlohu individuálně vypracujte ve dvou verzích:

1. Paralelní program pro systém se sdílenou pamětí (použitelné prostředky: prog. jazyk Java nebo vlákna POSIX).
2. Paralelní program pro systém s distribuovanou pamětí (použitelné prostředky: PVM nebo MPI).

2 Analýza

Podle zadané úlohy si musíme nejprve uvědomit, jak aplikaci rozdělit a také jaké formě budeme potřebovat vstupní data. Rozhodl jsem se, že aplikace bude transformovat skutečné obrázky, tudíž vstupem i výstupem bude obrázek. Vybral jsem si formát BMP, který musí splňovat následující – bitmapa musí být truecolor (24-bit) a nekomprimovaná. Ke zpracování šeodotónových obrázkům stačila sice jen 16-bitová paleta, ale protože taková bitmapa obsahuje paletu 256 barev a zbylá data jsou jen indexy do této palety, stačilos měnit jen barvy v paletě a nezáleželo by na velikosti bitmapy. Práce by byla stejná. Na rozdíl v 24-bit paletě není žádná tabulka barev. A tedy každý pixel je reprezentován 3 bajty. Protože bitmapa je truecolor lze použít na vstupu i barevné obrázky. Jednotlivé složky barvy jsou brány jako jasy jednotlivých složek a upraveny podle stejného vzorce.

2.1 Vstupy

Vstupních parametrů je celkem pět. První je počet vláken (dělníků) v kterých má výpočet běžet, druhý je velikost práce přidělovaná vláknům (je to počet pixelů obrázku), třetí je hodnota k v intervalu <0;255> a poslední dva parametry jsou cesty k souborům. První z nich je bitmapa, kterou chceme transformovat a druhý je, kam chceme výsledek transformace uložit. První tři parametry musí být celočíselné a kromě k větší jak 0.

2.2 Výstupy

Výstupem aplikace je transformovaný obrázek uložený na zadané cestě.

3 Implementace

Bez rozdílu v čem je aplikace napsána, nejprve zkontroluje parametry, zkontroluje bitmapu (zkontroluje, zda zadaný soubor je bitmapa, zda je truecolor a zda je nekomprimovaná). Celá bitmapa je pak rozdělena v obou aplikacích stejně. Podle zadané velikosti práce (počet pixelů) je vytvořeno dvourozměrné pole, kde druhá dimenze je právě kus přidělované práce (= počet pixelů * 3[1]) a první dimenze se dopočítá podle velikosti obrazových dat.

3.1 Java

Aplikace je rozdělena do tří balíků podle funkčnosti. Použitá verze JRE byla 1.6.0_01.

3.1.1 Balík ppr

Balík ppr obsahuje hlavní třídu Negative, která načte bitmapu, vytvoří frontu s prací, kterou si rozebírají jednotliví dělníci, nastartují se všechna vlákna a počká, dokud všechna nedoběhnou. Po té uloží výsledek transformace a skončí.

3.1.2 Balík ppr.image

Tento balík obsahuje jedinou třídu Bitmap, která se stará o načtení bitmapy, o její validaci a ve finále i o její uložení.

3.1.3 Balík ppr.threading

V tomto balíku jsou dvě třídy. Třída ThreadCounter, která hlídá, zda všechny vlákna už skončila a blokuje hlavní vlákno do té doby než tomu tak bude. Třída WorkingThread běží, dokud není fronta prázdná. Pokud je fronta prázdná, zavolá ThreadCounter, že skončila a ukončí se. Pokud ve frontě je „nějaká práce“, vezme si ji a zpracuje daný kus práce a to se celé opakuje, dokud fronta není prázdná.

3.2 PVM

3.2.1 Farmer

Hlavní aplikace, která vytvoří všechny potřebné dělníky, načte bitmapu a začne přidělovat dělníkům práci. Pokud nějaký dělník vrátí výsledek, je dělníkovi okamžitě přidělena další část práce a to tak dlouho, dokud se neprojde celé pole s prací. Po skončení transformace vyšle farmer zprávu multicastem všem dělníkům, aby se ukončili, a sám uloží výsledek a ukončí se.

3.2.2 Worker

Modul worker se přihlásí do PVM a v nekonečné smyčce přijímá zprávy. Pokud přijatá zpráva je práce, načte si všechna data z přijímacího zásobníku a přijaté pole bajtů zpracuje a výsledek pošle zpátky modulu farmer. Pokud zpráva je typu MSG_QUIT přeruší se nekonečná smyčka a dělník se ukončí.

3.2.3 Překlad

Ke zdrojovým kódům je připojen soubor makefile, který pomůže s překladem. Jednoduchým zavoláním příkazu make přeložíte řídící modul resp. dělníka takto:

```
make EXE=farmer
make EXE=worker
```

Získáte soubory farmer.exe a worker.exe.

[1] Na jeden pixel připadá v truecolor bitmapě 3 bajty

4 Uživatelský manuál

4.1 Java

Vzhledem k tomu, že aplikace nevyžaduje žádné vstupy od uživatele v průběhu výpočtu, stačí aplikaci spustit se všemi parametry následujícím způsobem:

java –jar ppr.jar.

K dispozici je soubor run.bat,který výše zmíněný příkaz volá sám.

Dále samozřejmě musíme zadat všechny potřebné parametry. Pokud tak neučiníme nebo budou špatně zadány, program vypíše nápovědu.

Příkladem může být následovně spuštěný program:

run 10 150 255 obr.bmp vystup.bmp

Aplikace se spustí v 10 vláknech, každé vlákno zpracuje 150 pixelů obrázku, než dostane přidělenou další práci. Vstupem bude obr.bmp a výsledek transformace se uloží v aktuálním adresáři do souboru vystup.bmp.

4.2 PVM

U PVM se aplikace lehce liší. Je složená ze dvou spustitelných souborů – farmer.exe a worker.exe. Po přeložení si musíme pustit PVM daemona (návod viz stránky předmětu PPR) a stačí spustit farmer.exe, což je aplikace, která přiděluje dělníkům práci, a která se stará o spuštění příslušného počtu dělníků.

Dále samozřejmě musíme zadat všechny potřebné parametry. Pokud tak neučiníme nebo budou špatně zadány, program vypíše nápovědu.

Příkladem může být následovně spuštěný program:

./farmer.exe 10 150 255 obr.bmp vystup.bmp

Aplikace se spustí v 10 vláknech, každé vlákno zpracuje 150 pixelů obrázku, než dostane přidělenou další práci. Vstupem bude obr.bmp a výsledek transformace se uloží v aktuálním adresáři do souboru vystup.bmp.

5 Závěr

Obě aplikace transformují obrázky a vytvářejí jejich negativy podle zadaného parametru k. Příklad je jednoduchá transformace a vzhledem k tomu, že v Jave programuje neustále, nebylo její naprogramování nikterak obtížné. Nejtěžší bylo správně rozdělit aplikaci a určit práci pro jednotlivá vlákna. Oproti tomu aplikace napsaná pro PVM v jazyce C už nebyla tak snadná. Opět vytvoření transformace nebylo nikterak složité, ale narazil jsem na potíže s oblíbenými ukazateli. Program havaroval při pokusu uvolnit paměť zabranou pro data bitmapy. Pak jsem si všiml, že jsem špatně alokoval první dimenzi dvourozměrného pole. Použil jsem menší velikost pro funkci malloc. Sice stačilo odstranit řádku s pokusem o uvolnění paměti a nechat to tak na systému, ale nedalo mi to a hledal jsem tak dlouho, dokud sem chybu nenašel.

6 Příloha

6.1 Java

6.1.1 Třída Negative.java

```
package ppr;
import java.io.IOException;
import java.util.ArrayList;
import ppr.image.Bitmap;
import ppr.threading.ThreadCounter;
import ppr.threading.WorkingThread;

public class Negative {
 private ArrayList<Integer> queue;
 public Negative(){
  this.queue = new ArrayList<Integer>();
 }
 public void transform(int threads,int pixels,int k,String bitmap,String
negative){
  try {
   Bitmap bmp = new Bitmap(bitmap);
   if(bmp.checkBitmap() == false){
    System.err.println("Vstupni obrazek neni bitmapa nebo nema spravny
format.");
    System.err.println("Aplikace akceptuje pouze nekomprimovany truecolor
bitmapy.");
    return;
   }
   //load bitmap
   byte[][] data = bmp.loadData(pixels);
   //prepare queue
   for(int i = 0; i < data.length; i++)
    this.queue.add(i);

   //start all threads
   ThreadCounter tc = new ThreadCounter(threads);
   for(int i = 0; i < threads ; i++){
    new WorkingThread(this.queue,data,k,tc).start();
   }

   //wait till all threads complete
   tc.waitForAll();

   //save transformed bitmap
   bmp.saveBitmap(negative, data);
   bmp.close();
  } catch (IOException e) {
   e.printStackTrace();
  } catch (Exception e) {
   e.printStackTrace();
  }
 }

 private static void printInfo(){
  System.out.println();
  System.out.println("Vytvoreni negativu k puvodnimu obrazku");
  System.out.println("======================================");
  System.out.println();
  System.out.println("Semestralni prace na PPR");
  System.out.println("Milan Balcar\tA06072");
```

```
  System.out.println();
  System.out.println("Aplikaci je nunto spoustet s 5 parametry.");
  System.out.println();
  System.out.println("java ppr.Negative <pocet vlaken> <velikost prace> <k>
<vstupni soubor> <vystupni soubor>");
  System.out.println("   <pocet vlaken>      - pocet vlaken, v kterych
vypocet pobezi");
  System.out.println("   <velikost prace>    - pocet pixelu, ktera ma jedno
vlakno zpracovat");
  System.out.println("   <k>                 - jas pixelu v rozsaho
<0;255>");
  System.out.println("   <vstupni soubor>    - bitmapa s obrazkem");
  System.out.println("   <vystupni soubor>   - vysledek bude ulozen do
tohoto souboru");
 }
 public static void main(String[] args) {
  if(args.length != 5){
   System.err.println("Nespravny pocet argumentu.");
   Negative.printInfo();
   return;
  }

  try {
   int threads = Integer.parseInt(args[0]);
   int pixels = Integer.parseInt(args[1]);
   int k = Integer.parseInt(args[2]);

   if(threads <= 0 || pixels <= 0 || k < 0 || k > 255)
    throw new NumberFormatException();

   //Negative n = new Negative(threads,pixels,k,args[2],args[3]);
   Negative n = new Negative();

   long time = System.currentTimeMillis();
   n.transform(threads,pixels,k,args[3],args[4]);
   System.out.println(System.currentTimeMillis() - time + " ms");
  } catch (NumberFormatException e) {
   System.err.println("Pocet vlaken a velikost prace musi byt vetsi jak 0,
velikost k musi byt v rozsahu <0;255>.");
  }
 }
}
```

6.1.2 Třída Bitmap.java

```
package ppr.image;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

public class Bitmap {
 public static final int HEADER_LENGTH = 54;
 public static final int BYTES_PER_PIXEL = 3;

 private byte[] header;
 private FileInputStream bitmapFile;
 private long length;

 public Bitmap(String fileName)throws Exception{
  File f = new File(fileName);
  this.length = f.length() - Bitmap.HEADER_LENGTH;
  this.bitmapFile = new FileInputStream(f);
 }

 private void loadHeader()throws IOException{
  this.header = new byte[Bitmap.HEADER_LENGTH];
  this.bitmapFile.read(this.header);
 }

 public boolean checkBitmap()throws IOException{
  this.loadHeader();

  if(this.header[0] != 0x42 || this.header[1] != 0x4D)
   return false;

  if(((this.header[28] << 8) | this.header[29]) != 0x1800)
   return false;

  if(((this.header[30] << 24) | (this.header[31] << 16) | (this.header[32]
<< 8) | this.header[33]) != 0x0)
   return false;

  return true;
 }

 public byte[][] loadData(int pixels)throws IOException{
  int bytes = pixels * Bitmap.BYTES_PER_PIXEL;
  int mod = (int)(this.length%bytes);
  int size = (int)(this.length/bytes);

  if(mod != 0) size++;

  if(this.header == null) this.loadHeader();

  byte[][] data = new byte[size][bytes];

  for(int i = 0; i < data.length; i++){
   this.bitmapFile.read(data[i]);
  }

  return data;
 }
```

```
 public void saveBitmap(String file,byte[][] data){
  try {
   FileOutputStream out = new FileOutputStream(file);

   out.write(this.header);

   for(int i = 0; i < data.length-1; i++){
    out.write(data[i]);
   }

   long len = data[data.length-1].length - data.length*data[data.length-
1].length + this.length;
   out.write(data[data.length-1],0,(int)len);

   out.close();
  } catch (FileNotFoundException e) {
   e.printStackTrace();
  } catch (IOException e) {
   e.printStackTrace();
  }
 }

 public void close(){
  try {
   this.bitmapFile.close();
  } catch (IOException e) {}
 }
}
```

6.1.3 Třída ThreadCounter.java

```
package ppr.threading;

public class ThreadCounter {
	private int threadNo = 0;
	private int counter = 0;
	private Object lock = new Object();

	public ThreadCounter(int threadNo){
		this.threadNo = threadNo;
	}

	public synchronized void increment(){
		this.counter++;
		synchronized(this.lock){
			this.lock.notifyAll();
		}
	}

	public void waitForAll(){
		synchronized(lock){
			while(this.counter < this.threadNo){
				try {
					this.lock.wait();
				} catch (InterruptedException e) {}
			}
		}
	}
}
```

6.1.4 Třída ThreadWorker.java

```
package ppr.threading;

import java.util.ArrayList;

public class WorkingThread extends Thread {
	private byte[][] data;
	private ArrayList<Integer> queue;
	private ThreadCounter tc;
	private int k;

	public WorkingThread(ArrayList<Integer> queue,byte[][] data, int
k,ThreadCounter tc){
		this.queue = queue;
		this.data = data;
		this.k = k;
		this.tc = tc;
	}

	@Override
	public void run() {
		int index;
		while(true){

			synchronized(this.queue){
				if(this.queue.size() == 0){
					this.tc.increment();
					break;
				}else{
					index = this.queue.remove(0);
				}
			}
			//System.out.println(this.getName() + " altering index "
+ index );
			//process data
			int j;
			if(index < 0 && index >= data.length) continue;
			for(int i = 0; i < data[index].length; i++){
				j = (data[index][i] < 0)?
data[index][i]+256:data[index][i];
				data[index][i] = (byte)((k-j < 0)? 0:k-j);
			}
		}
	}

}
```

6.2 PVM

6.2.1 Farmer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pvm3.h>
#include "ppr.h"

int ntask;
int pixels;
int k;
long bmp_length;

/*
 * Zkontroluje zda zadany soubor je bitmapa a zda ma spravny format.
 * Musi byt nekomprimovana a v 24bitove palete.
 */
int checkHeader(byte * header){
    //kontrola zda je to bitmapa
    if(header[0] != 0x42 || header[1] != 0x4D)
        return 0;
    //kontrola na pocet barev
    if(((header[28] << 8) | header[29]) != 0x1800)
        return 0;
    //a kontrola zda neni bitmapa komprimovana
    if(((header[30] << 24) | (header[31] << 16) | (header[32] << 8) |
header[33]) != 0x0)
        return 0;

    return 1;
}

/*
 * Nacte bitmapu a vrati odkaz na matici dat,
 * rozdelenych podle parametru programu.
 * Druhy rozmer je vzdy velikost pridelene prace (pocet pixelu ke
zpracovani) * 3.
 * Je 3x vetsi, protoze truecolor bitmapa pouziva 3 byty pro jeden pixel.
 */
byte ** loadBitmap(byte * bmp, byte * header, int * size){
    int len, mod, i, bytes;
    byte ** data;
    FILE * fr;

    printf("Nacitam bitmapu %s.\n", bmp);

    fr = fopen(bmp, "rb");
    if(fr == NULL) return (byte **)0;

    //potrebuju zjistit velikost bitmapy abych mohl urcit prvni rozmer pole
    fseek(fr, 0L, SEEK_END);
    bmp_length = ftell(fr)-HEADER_LENGTH;
    fseek(fr, 0L, SEEK_SET);

    bytes = pixels*BYTES_PER_PIXEL;
    mod = bmp_length%(bytes);
    len = bmp_length/(bytes);
    *size = len;
    if(mod != 0) len++;

    fread(header, sizeof(byte), HEADER_LENGTH, fr);
```

```
    if(checkHeader(header) == 0){
        printf("Zadany soubor neni bitmapa nebo nema spravny format.\n");
        printf("Aplikace akceptuje pouze truecolor nekomprimovane
bitmapy.\n");
        return (byte **)0;
    }

    data = (byte **)malloc(sizeof(byte *)*len);
    for(i = 0; i < len-1; i++){
        data[i] = (byte *)malloc(sizeof(byte)*bytes);
        fread(data[i], sizeof(byte), bytes, fr);
    }
    //donactu blokove zbytek souboru
    data[len-1] = (byte *)malloc(sizeof(byte)*bytes);
    fread(data[len-1], sizeof(byte), bmp_length-(bytes*(len-1)), fr);
    printf("Bitmapa nactena.\n");

    fclose(fr);
    return data;
}

/*
 * Ullozi vysledek transformace zpatky jako bitmapu.
 */
int saveBitmap(byte * bmp, byte * header, byte ** data, int data_len){
    int i, len;
    FILE * fw;

    fw = fopen(bmp, "wb");
    if(fw == NULL) return 0;

    fwrite(header, sizeof(byte), HEADER_LENGTH, fw);
    fflush(fw);

    printf("Ukladam bitmapu do %s.\n", bmp);
    for(i = 0; i < data_len-1; i++){
        fwrite(data[i], sizeof(byte), pixels*BYTES_PER_PIXEL, fw);
    }

    len = bmp_length + (1-data_len)*pixels*BYTES_PER_PIXEL;
    fwrite(data[data_len-1], sizeof(byte), len, fw);

    fclose(fw);
    return 1;
}

/*
 * Zasle praci (pole bytu) delnikovi se zadanym tid.
 * Posila se i index, aby pri prijmuti vedel farmar, kam ma dane pole
vratit.
 */
void send_work(int index, int tid, byte * data){
    pvm_initsend(PvmDataDefault);
    pvm_pkint(&index, 1, 1);
    pvm_pkint(&pixels, 1, 1);
    pvm_pkint(&k, 1, 1);
    pvm_pkbyte(data, pixels*BYTES_PER_PIXEL, 1);
    printf("Posilam data ke zpracovani delnikovi %d.\n", tid);
    pvm_send(tid, MSG_TODO);
}
```

```
/*
 * Hlavni smycka programu, prideluje data dokud nejaka jsou a vsichni
delnici neskonci.
 */
void run(byte ** bmp_data, int data_len, int * tids){
    int pracujici, i, index, tid;
    pracujici = 0;

    for(i = 0; i < ntask;i++){
        send_work(i, tids[i], bmp_data[i]);
        pracujici++;
    }

    while(1){
        if(pvm_recv(-1, MSG_RESULT) > 0){
            pvm_upkint(&tid, 1, 1);
            pvm_upkint(&index, 1, 1);
            printf("Prijimam data od delnika %d.\n", tid);
            pvm_upkbyte(bmp_data[index], pixels*BYTES_PER_PIXEL, 1);

            if(i < data_len){
                send_work(i++, tid, bmp_data[i]);
            }else pracujici--;
            if(pracujici == 0) break;
        }
    }
}
/*
 * Vypise napovedu o programu.
 */
void printInfo(void){
    printf("Vytvoreni negativu k puvodnimu obrazku\n");
    printf("======================================\n\n");
    printf("Semestralni prace na PPR\n");
    printf("Milan Balcar\tA06072\n\n");
    printf("Aplikaci je nunto spoustet s 5 parametry.\n\n");
    printf("./farmer.exe <pocet vlaken> <velikost prace> <k> <vstupni
soubor> <vystupni soubor>\n");
    printf("   <pocet vlaken>      - pocet vlaken, v kterych vypocet
pobezi\n");
    printf("   <velikost prace>    - pocet pixelu, ktera ma jedno vlakno
zpracovat\n");
    printf("   <k>                 - jas pixelu v rozsaho <0;255>\n");
    printf("   <vstupni soubor>    - bitmapa s obrazkem\n");
    printf("   <vystupni soubor>   - vysledek bude ulozen do tohoto
souboru\n");
}

int main(int argc, char ** args){
    int mytid,i,size;
    byte * bmp_original;
    byte * bmp_result;
    byte bmp_header[HEADER_LENGTH];
    byte ** bmp_data;
    int * tids;

    if(argc != 6){
        printf("Spatny pocet argumentu.\n");
        printInfo();
        exit(1);
```

```
    }

    ntask = atoi(args[1]);
    pixels = atoi(args[2]);
    k = atoi(args[3]);
    bmp_original = args[4];
    bmp_result = args[5];

    if(ntask <= 0 || pixels <= 0 || k < 0 || k > 255){
        printf("Je nam lito, ale zadane argumenty jsou mimo jejich platne
rozsahy.\n");
        printf("pocet vlaken musi byt vetsi nez 0.\n");
        printf("pocet pixelu musi byt vetsi jak 0.\n");
        printf("k musi byt z rozsahu <0;255>.\n");
        return 1;
    }

    tids = (int *)malloc(sizeof(int)*ntask);

    //nacteme bitmapu
    if((bmp_data = loadBitmap(bmp_original, bmp_header, &size)) == NULL){
        printf("Chyba pri nacitani bitmapy.\n");
        exit(1);
    }

    //nastartujem pvm
    mytid = pvm_mytid();
    printf("Spoustim delniky...\n");
    i =     pvm_spawn("worker.exe", (char **)0, PvmTaskDefault, (char *)0,
ntask, tids);
    if(ntask != i){
        printf("Nebylo mozne spustit vsechny delniky, koncim.\n");
        pvm_exit();
        exit(1);
    }

    for(i = 0; i < ntask; i++){
        printf("Delnik %d pripraven.\n" , i);
        pvm_recv(-1, MSG_READY);
    }
    //zacnem pridelovat praci
    run(bmp_data, size, tids);

    printf("Ukoncuji delniky.\n");
    //ukoncime delniky
    pvm_mcast(tids, ntask, MSG_QUIT);
    //ulozime negativ
    if(saveBitmap(bmp_result, bmp_header, bmp_data, size) == 0)
        printf("Bitmapa nebyla ulozena.\n");
    //vymazeme bitmapu z pameti
    free(bmp_data);
    bmp_data = 0;

    pvm_exit();

    return 0;
}
```

6.2.2 Worker.c

```
#include <stdlib.h>
#include <pvm3.h>
```

```
#include "ppr.h"
/*
 *Projde pole a podle vzorce upravi vsechny hodnoty
 * new(x,y) = k - old(x,y)
 */
void zpracuj(byte * data, int k, int data_len){
    int i = 0;

    for(i = 0; i < data_len; i++ ){
        data[i] = (k-data[i] < 0)? 0:k-data[i];
    }
}

/*
 * Nacte prijata data z bufferu a preda je funkci na zpracovani negativu.
 * Upravene pole hodnot posle rodicovi zpatky.
 * A smaze veskerou pridelenou pamet.
 */
void processData(int mytid){
    int pixels, k, index;
    byte * data;

    pvm_upkint(&index, 1, 1);
    pvm_upkint(&pixels, 1, 1);
    pvm_upkint(&k, 1, 1);
    data = (byte *)malloc(pixels*BYTES_PER_PIXEL*sizeof(byte));
    pvm_upkbyte(data, pixels*BYTES_PER_PIXEL, 1);

    //zpracuju pole
    zpracuj(data, k, pixels*BYTES_PER_PIXEL);

    //poslu zpatky
    pvm_initsend(PvmDataDefault);
    pvm_pkint(&mytid, 1, 1);
    pvm_pkint(&index, 1, 1);
    pvm_pkbyte(data, pixels*BYTES_PER_PIXEL, 1);
    pvm_send(pvm_parent(), MSG_RESULT);

    //uvolnim alokovanou pamet
    free(data);
    data = 0;
}

int main(int argc, char ** argv){

    int mytid;

    //prihlaseni do pvm
    mytid = pvm_mytid();
    //zasleme informaci o tom ze jsme pripraveni
    pvm_initsend(PvmDataDefault);
    pvm_send(pvm_parent(), MSG_READY);

    while(1){
        //mame praci
        if(pvm_nrecv(pvm_parent(), MSG_TODO) > 0){
            processData(mytid);
        }

        //skoncime
        if(pvm_nrecv(pvm_parent(), MSG_QUIT) > 0){
```

```
            break;
        }
    }

    //odesleme potvrzeni o ukonceni
    //pvm_initsend(PvmDataDefault);
    //pvm_send(pvm_parent(),MSG_QUIT);

    pvm_exit();

    return 0;
}
```

6.2.3 Ppr.h

```
#define MSG_READY   4
#define MSG_TODO    5
#define MSG_RESULT  6
#define MSG_QUIT    7


#define HEADER_LENGTH 54
#define BYTES_PER_PIXEL 3

typedef unsigned char byte;
```