Programming Assignment 2

CS 240 Bolden

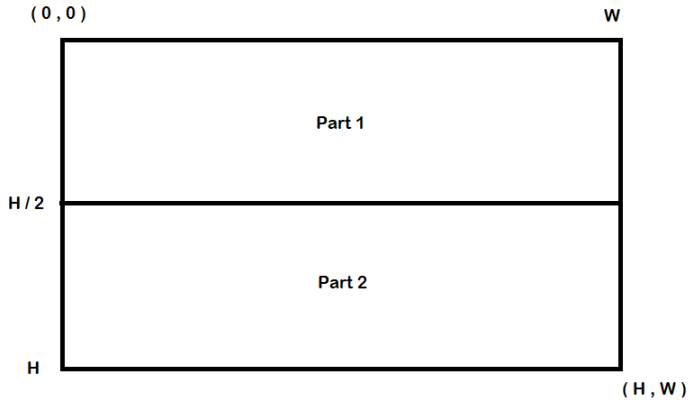
May 4th, 2021

David C Bush

1 Program Design

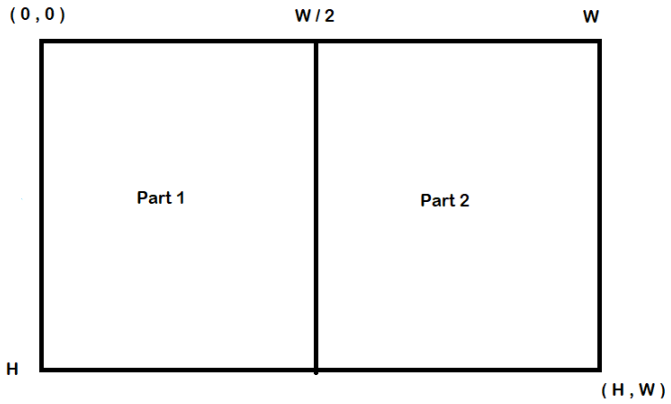
GOAL: Break image into separate parts and use fork() to create child processes that will work simultaneously, each extracting a different part of the image. Then combine the pieces back together and save the resulting image.

Horizontal:



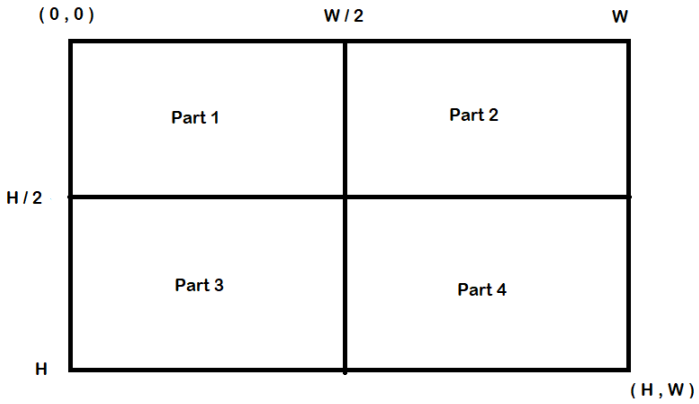
	heightStart	heightEnd	widthStart	widthEnd
Part 1	0	h/2	0	w
Part 2	h/2	h	0	w

Vertical:



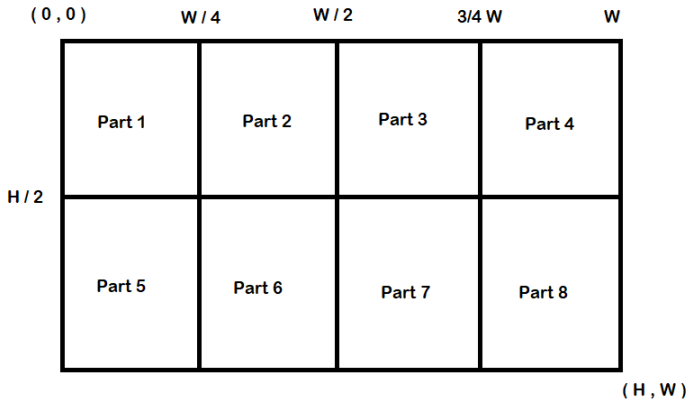
	heightStart	heightEnd	widthStart	widthEnd
Part 1	0	h	0	w/2
Part 2	0	h	w/2	w

4 Parts:



	heightStart	heightEnd	widthStart	widthEnd
Part 1	0	h/2	0	w/2
Part 2	0	h/2	w/2	w
Part 3	h/2	h	0	w/2
Part 4	h/2	h	w/2	w

8 Parts:



	heightStart	heightEnd	widthStart	widthEnd
Part 1	0	h/2	0	w/4
Part 2	0	h/2	w/4	w/2
Part 3	0	h/2	w/2	3/4w
Part 4	0	h/2	3/4w	w
Part 5	h/2	h	0	w/4
Part 6	h/2	h	w/4	w/2
Part 7	h/2	h	w/2	3/4w
Part 8	h/2	h	3/4w	w

2 Program Log

5/4/21:

- I started by creating a LaTeX document to contain the assignment, and then looked over the assignment pdf.
- I then began Part 1 of the assignment by modifying the provided source code in order to convert the image to red, blue, or green. I tested it and made sure it compiled and worked.

5/5/21:

- I worked on the Program Design by drawing diagrams with Paint and creating tables with LaTeX. This helped me get a better understanding of how I needed to break apart the image and assign a process to each section.
- I rewrote ExtractRGB to include a start height and width so I could-divide the image into different parts. I also made it so it would return a Pixel**.
- Next I began working on creating just one fork and I ran into a pretty big problem once I realized that the two processes would be working on separate image objects (processes don't share information like threads). In order to solve this issue I would need to use some form of IPC (Inner Process Communication), and this is when I started to do research on pipes.

5/6/21:

- My solution was simple but the implementing it into code could be challenging. I wanted to use pipes to write the child image part back to the parent process and then have the parent combine the images together and save the combined image to the output. So both processes would do their work (ExtractRGB) and then one process (parent) would save the result to the output file.
- As I predicted this ended up taking more time then I originally had planned for and I spent many hours debugging before I was able to get the Read/Write between the processes and the image merger to work correctly.
- After the one fork program was working I started the two fork program. This would divide the program into four separate processes and each one would work on a different part of the image. Keeping track of each part that each process worked on was difficult.

5/7/21:

- I was running into a lot of problems using pipes with transferring the buffer array and couldn't get multiple processes to work with the larger images (also using pipes slowed down the overall speed a lot). After attending class I decided to look into mmap() and take a second look at the assignment.

- I actually found using the mmap() system call to be much easier at transferring the data between processes. What I ended up doing was taking my early program and removing the read/write pipe calls and adding mmap() instead.

- This turned out to be a good decision and I was able to fix the previous bugs I was having and get both 2 and 4 processes to work with all the images.

5/8/2021:

- Today I worked on writing the three fork program which would divide the image into 8 parts, each having a separate process to work on it. The code was very similar to the two fork program and I was fortunate not to run into that many problems. The hardest part was setting the dimensions for each part (odd vs even images are treated different).

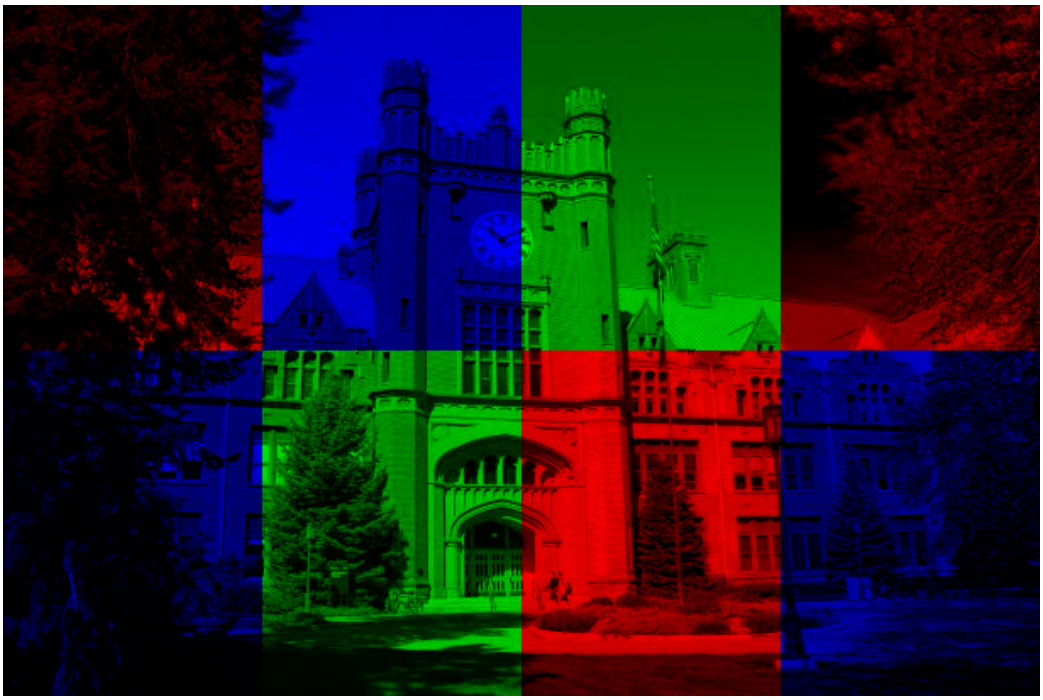
- Recorded tests of all fork programs for my output log and finished writing the conclusion.

3 Output

```
Script started on 2021-05-08 16:16:32-0700
user: time ./noFork uiAdmin.jpg test.jpg 2

real    0m0.110s
user    0m0.078s
sys     0m0.031s

user: time ./oneFork uiAdmin.jpg test.jpg 2

real    0m0.149s
user    0m0.063s
sys     0m0.078s

user: time ./twoFork uiAdmin.jpg test.jpg 2

real    0m0.142s
user    0m0.094s
sys     0m0.031s

user: time ./threeFork uiAdmin.jpg test.jpg 2

real    0m0.160s
user    0m0.109s
sys     0m0.047s

user: time ./noFork sugarBear.jpg test.jpg 3

real    0m0.028s
user    0m0.000s
sys     0m0.016s

user: time ./oneFork sugarBear.jpg test.jpg 3

real    0m0.040s
user    0m0.000s
```

```
sys     0m0.016s

user: time ./twoFork sugarBear.jpg test.jpg 3

real    0m0.039s
user    0m0.000s
sys     0m0.031s

user: time ./threeFork sugarBear.jpg test.jpg 3

real    0m0.047s
user    0m0.000s
sys     0m0.047s

exit

Script done on 2021-05-08 16:18:28-0700
```

Example Image (set the child processes to purposely do different colors to show contrast between parts):



4 Conclusion

During this assignment I was able to learn more about processes, inter process communication, converting different array types, and dividing an image into smaller sections. I started out by drawing diagrams and tables to illustrate how to divide the image into smaller parts. This was very useful later on when I began programming but I still ran into an issue since even and odd images would be divided differently(different size parts).

I figured out eventually that I could take the height, h, and width, w, and use the mod operator to make it so when there were odd values it would add an extra pixel to certain parts of the image, [Example Code: (h/2)+(h%2), (w/2)+(w%2)]. Next I used the fork() function to create child processes, and saved the return value to the variable pid. This allows us to distinguish between the parent process(pid>0) and child process (pid==0). By calling fork() multiple times I was able to create more and more processes and I kept track of them by checking each pid value, [Example Code: if(pid1==0 pid2==0 pid3==0)], this would check for the child/child/child process if there were 3 forks.

Now that the child processes were each working on a different part, there needs to be way for them to communicate. This is when I started doing research on IPC. I first started to use pipes but that ended up being the wrong choice as I was running into countless errors and pipes were a lot

slower(I/O is bad). Then during class, mmap() was suggested and I decided to give it a try. This turned out to work really well as I could have the processes share the parts between each other, but they would still need to be converted between a 1D char array and 2D Pixel array.

I designed it so that one process would be in charge of reading all the child parts and then combine them back together into the final image. This worked fine but I think this is what led to the speeds being slower since there was more work going on converting the images between arrays.

Finally I began to experiment with the different programs. There were 4 programs I used; one with no forks(1 process), one fork(2 processes), two forks(4), 3 forks(8). I first made an example image where I changed the 8 parts to alternating colors to show that I had properly divided the image. Then I began running time tests on each program and I was a little surprised by the results. The noFork program was actually the fastest in all the tests I did by a decent margin. Then the next fastest was usually the twoFork program, and after that the oneFork. The threeFork was the slowest program which probably is due to extra work going on by the parent process since its in charge of converting each child image and combining the pieces. There probably are better ways to do what I was trying to accomplish that would have better time results.

I found the assignment to be challenging, interesting, and a good learning experience.

5 Source Code

5.1 HighlightsofPrograms.txt

NOTE: I removed the repetitive parts to use less space and highlight the important parts. The full code is attached in the email.

```
/**********************************
*          imageUtils.c
*        Utility Functions
**********************************/
Pixel ** ExtractPartRGB( Pixel **image, int heightStart, int heightEnd, int widthStart, int widthEn
{
  int h = heightEnd-heightStart;
  int w = widthEnd-widthStart;
  Pixel **newImage = (Pixel **)malloc(sizeof(Pixel *) * h);
  newImage[0] = (Pixel *)malloc(sizeof(Pixel) * (h * w));
  for( int i = 1 ; i < h ; i++ ) {
    newImage[i] = (*newImage + (w * i));
  }
  for( int i = heightStart; i < heightEnd; i++ ) {
    for( int j = widthStart; j < widthEnd; j++ ) {
      if( m == RED )
      {
        //image[i][j].red;      // unchanged
        image[i][j].green = 0;
        image[i][j].blue  = 0;
      }
      else if( m == GREEN )
```

```
        {
            image[i][j].red = 0;
            //image[i][j].green = 0; // unchanged
            image[i][j].blue  = 0;
        }
        else if( m == BLUE )
        {
            image[i][j].red = 0;
            image[i][j].green = 0;
            //image[i][j].blue  = 0;  // unchanged
        }
        newImage[i-heightStart][j-widthStart] = image[i][j];
        //printf("i: %d, j: %d\t", i, j);
    }
  }
  return newImage;
}

void ConvertPixelto1D(Pixel **image, unsigned char** arr1D, int height, int width){
  int x = 0;
  for( int i = 0 ; i < height ; i++ ) {
    for( int j = 0 ; j < width ; j++ ) {
      //printf("i: %d, j: %d\n", i, j);
      (*arr1D)[x+0] = image[i][j].red;
      (*arr1D)[x+1] = image[i][j].green;
      (*arr1D)[x+2] = image[i][j].blue;
      x += 3;
```

```
    }
  }
}

Pixel ** Convert1DtoPixel(unsigned char *arr1D, int height, int width){
  Pixel **newImage = (Pixel **)malloc(sizeof(Pixel *) * height);
  newImage[0] = (Pixel *)malloc(sizeof(Pixel) * (height * width));
  for( int i = 1 ; i < height ; i++ ) {
    newImage[i] = (*newImage + (width * i));
  }
  int z = 0;
  for(int i=0; i<height; i++){
    for(int j=0; j<width; j++){
      newImage[i][j].red = arr1D[z+0];
      newImage[i][j].green = arr1D[z+1];
      newImage[i][j].blue = arr1D[z+2];
      z += 3;
    }
  }
  return newImage;
}

void CombineImagePieces(Pixel **mainImage, Pixel **partImage, int startHeight, int endHeight, int s
  for(int i=startHeight; i<endHeight; i++){
    for(int j=startWidth; j<endWidth; j++){
      //printf("i: %d, j: %d \n", i, j);
      mainImage[i][j] = partImage[i-startHeight][j-startWidth];
```

```
    }
  }
  return;
}


/**********************************
* noFork_Image.c Base Program
**********************************/
#include <stdlib.h>
#include <stdio.h>
#include "colorUtils.h"
#include "imageUtils.h"
int main(int argc, char **argv) {

    char *inputFile  = NULL;
    char *outputFile = NULL;
    int mode = 1;
    if( argc == 3 || argc == 4 ) {
        inputFile  = argv[1];
        outputFile = argv[2];
        if( argc == 4 ) {
            mode = atoi(argv[3]);
        }
    }
    else {
        fprintf(stderr, "Usage:" );
```

```
    fprintf(stderr, " %s inputFileName outputFileName [mode]\n", argv[0]);
    fprintf(stderr, "   [mode: 1 = Gray, 2 = Red,\n" );
    fprintf(stderr, "          3 = Blue,          4 = Green]\n" );
    exit(1);
}
int h, w;
Pixel **image = LoadImage(inputFile, &h, &w);
Pixel **newImage;
if( mode == 1 ) {
    ImageToGrayScale( image, h, w, AVERAGE);
}
else if( mode == 2 ) {
    ExtractRGB(image, h, w, RED);
}
else if (mode == 3){
    ExtractRGB(image, h, w, BLUE);
}
else if (mode == 4){
    ExtractRGB(image, h, w, GREEN);
}
else {
    fprintf(stderr, "Invalid mode\n");
}
// show numerical values DumpImage( image, h, w );
// save resulting image
SaveImage(outputFile, image, h, w);
return 0;
```

```
}


/**********************************
*           oneFork_Image.c
*        Program with 2 Processes
**********************************/
Pixel **startImage = LoadImage(inputFile, &h, &w);
    unsigned char *part1 = mmap(NULL, (h/2)*w*3, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANON, 0,
    if(part1 == MAP_FAILED){
        printf("Mapping Failed\n");
        return 1;
    }
    Pixel **childImage;
    Pixel **parentImage;
    if( mode == 1 ) {
        ImageToGrayScale( startImage, h, w, AVERAGE);
    }
    else if( mode == 2 ) {
        // fork here
        pid = fork();
        if(pid == 0){ // its the child
            Pixel ** otherImage = ExtractPartRGB(startImage, 0, h/2, 0, w, RED);
            ConvertPixelto1D(otherImage, &part1, h/2, w);
            exit(0);
        }
        else if(pid > 0){ //parent
```

```
        parentImage = ExtractPartRGB( startImage, h/2, h, 0, w, RED );
        waitpid(pid, NULL, 0);
        childImage = Convert1DtoPixel(part1, h/2, w);
    }
}
else if (mode == 3){
    // BLUE
    }
}
else if (mode == 4){
    // GREEN
    }
}
else {
    fprintf(stderr, "Invalid mode\n");
}
// show numerical values DumpImage( startImage, h, w );
// Parent process should combine images and save output
if(pid>0){
    Pixel **finalImage = (Pixel **)malloc(sizeof(Pixel *) * h);
    finalImage[0] = (Pixel *)malloc(sizeof(Pixel) * (h * w));
    for( int i = 1 ; i < h ; i++ ) {
        finalImage[i] = (*finalImage + (w * i));
    }
    // combine parts
    CombineImagePieces(finalImage, childImage, 0, h/2, 0, w);
    CombineImagePieces(finalImage, parentImage, h/2, h, 0, w);
```

```
        // save resulting image
        SaveImage(outputFile, finalImage, h, w);
    }
    return 0;


/**********************************
*           twoFork_Image.c
*       Program with 4 Processes
**********************************/

    Pixel **startImage = LoadImage(inputFile, &h, &w);
    // define mapped parts (going to store 1D version of image in)
    unsigned char *part1 = mmap(NULL, (h/2)*(w/2)*3, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANON
    unsigned char *part2 = mmap(NULL, (h/2)*(w/2)*3, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANON
    unsigned char *part3 = mmap(NULL, (h/2)*(w/2)*3, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANON
    Pixel **childImage1;
    Pixel **childImage2;
    Pixel **childImage3;
    Pixel **parentImage;
    if( mode == 1 ) {
        ImageToGrayScale( startImage, h, w, AVERAGE);
    }
    else if( mode == 2 ) {
        // fork here (making 4 processes)
        pid1 = fork();
        pid2 = fork();
```

```
    if(pid1 == 0 && pid2 == 0){ //child/child *part1
        Pixel ** otherImage = ExtractPartRGB(startImage, 0, h/2, 0, w/2, RED);
        ConvertPixelto1D(otherImage, &part1, h/2, w/2);
        exit(0);
    }
    else if(pid1 == 0 && pid2 > 0) { //child/parent *part2
        Pixel ** otherImage = ExtractPartRGB(startImage, 0, h/2, w/2, w, RED);
        ConvertPixelto1D(otherImage, &part2, h/2, (w/2)+(w%2));
        exit(0);
    }
    else if(pid1 > 0 && pid2 == 0) { //parent/child *part3
        Pixel ** otherImage = ExtractPartRGB(startImage, h/2, h, 0, w/2, RED);
        ConvertPixelto1D(otherImage, &part3, (h/2)+(h%2), w/2);
        exit(0);
    }
    else if(pid1 > 0 && pid2 > 0){ // parent/parent
        parentImage = ExtractPartRGB( startImage, h/2, h, w/2, w, RED );
        waitpid(pid2, NULL, 0);
        waitpid(pid1, NULL, 0);
        childImage1 = Convert1DtoPixel(part1, h/2, w/2);
        childImage2 = Convert1DtoPixel(part2, h/2, (w/2)+(w%2));
        childImage3 = Convert1DtoPixel(part3, (h/2)+(h%2), w/2);
    }
}
else if (mode == 3){
    // BLUE
}
```

```
    else if (mode == 4){
        // GREEN
    }
    else {
        fprintf(stderr, "Invalid mode\n");
    }
    // show numerical values
    DumpImage( startImage, h, w );
    // Parent process should combine images and save output
    if(pid1>0 && pid2 >0){
        Pixel **finalImage = (Pixel **)malloc(sizeof(Pixel *) * h);
        finalImage[0] = (Pixel *)malloc(sizeof(Pixel) * (h * w));
        for( int i = 1 ; i < h ; i++ ) {
            finalImage[i] = (*finalImage + (w * i));
        }
        // combine parts
        CombineImagePieces(finalImage, childImage1, 0, h/2, 0, w/2);
        CombineImagePieces(finalImage, childImage2, 0, h/2, w/2, w);
        CombineImagePieces(finalImage, childImage3, h/2, h, 0, w/2);
        CombineImagePieces(finalImage, parentImage, h/2, h, w/2, w);
        // save resulting image
        SaveImage(outputFile, finalImage, h, w);
    }
    return 0;
}

/*********************************
```

```
*           threeFork_Image.c
*       Program with 8 Processes
************************************/
    Pixel **startImage = LoadImage(inputFile, &h, &w);\
    // define mapped parts (going to store 1D version of image in)
    unsigned char *part1 = mmap(NULL, (h/2)*(w/4)*3, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANON
    unsigned char *part2 = mmap(NULL, (h/2)*(w/4)*3, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANON
    unsigned char *part3 = mmap(NULL, (h/2)*(w/4)*3, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANON
    unsigned char *part4 = mmap(NULL, (h/2)*(w/4)*3, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANON
    unsigned char *part5 = mmap(NULL, (h/2)*(w/4)*3, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANON
    unsigned char *part6 = mmap(NULL, (h/2)*(w/4)*3, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANON
    unsigned char *part7 = mmap(NULL, (h/2)*(w/4)*3, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANON
    Pixel **childImage1, **childImage2, **childImage3, **childImage4, **childImage5, **childImage6
    Pixel **parentImage;
    if( mode == 1 ) {
        ImageToGrayScale( startImage, h, w, AVERAGE);
    }
    else if( mode == 2 ) {
        // fork here (making 8 processes)
        pid1 = fork();
        pid2 = fork();
        pid3 = fork();
        if(pid1 == 0 && pid2 == 0 && pid3 == 0){ //child/child/child *part1
            Pixel ** otherImage = ExtractPartRGB(startImage, 0, h/2, 0, w/4, RED);
            ConvertPixelto1D(otherImage, &part1, h/2, w/4);
            exit(0);
        }
```

```
else if(pid1 == 0 && pid2 == 0 && pid3 > 0) { //child/child/parent *part2
    Pixel ** otherImage = ExtractPartRGB(startImage, 0, h/2, w/4, w/2, RED);
    ConvertPixelto1D(otherImage, &part2, h/2, (w/4)+(w%4));
    exit(0);
}
else if(pid1 == 0 && pid2 > 0 & pid3 == 0) { //child/parent/child *part3
    Pixel ** otherImage = ExtractPartRGB(startImage, 0, h/2, w/2, (0.75)*w, RED);
    ConvertPixelto1D(otherImage, &part3, h/2, (w/4)+(w%4));
    exit(0);
}
else if(pid1 == 0 && pid2 > 0 & pid3 > 0) { //child/parent/parent *part4
    Pixel ** otherImage = ExtractPartRGB(startImage, 0, h/2, 0.75*w, w, RED);
    ConvertPixelto1D(otherImage, &part4, h/2, (w/4)+(w%4));
    exit(0);
}
else if(pid1 > 0 && pid2 == 0 & pid3 == 0) { //parent/child/child *part5
    Pixel ** otherImage = ExtractPartRGB(startImage, h/2, h, 0, w/4, RED);
    ConvertPixelto1D(otherImage, &part5, (h/2)+(h%2), (w/4)+(w%4));
    exit(0);
}
else if(pid1 > 0 && pid2 == 0 & pid3 > 0) { //parent/child/parent *part6
    Pixel ** otherImage = ExtractPartRGB(startImage, h/2, h, w/4, w/2, RED);
    ConvertPixelto1D(otherImage, &part6, (h/2)+(h%2), (w/4)+(w%4));
    exit(0);
}
else if(pid1 > 0 && pid2 > 0 & pid3 == 0) { //parent/parent/child *part7
    Pixel ** otherImage = ExtractPartRGB(startImage, h/2, h, w/2, 0.75*w, RED);
```

```
        ConvertPixelto1D(otherImage, &part7, (h/2)+(h%2), (w/4)+(w%4));
        exit(0);
    }
    else if(pid1 > 0 && pid2 > 0 && pid3 > 0){ // parent/parent/parent
        parentImage = ExtractPartRGB( startImage, h/2, h, 0.75*w, w, RED );
        waitpid(pid3, NULL, 0);
        waitpid(pid2, NULL, 0);
        waitpid(pid1, NULL, 0);
        childImage1 = Convert1DtoPixel(part1, h/2, w/4);
        childImage2 = Convert1DtoPixel(part2, h/2, (w/4)+(w%4));
        childImage3 = Convert1DtoPixel(part3, h/2, (w/4)+(w%4));
        childImage4 = Convert1DtoPixel(part4, h/2, (w/4)+(w%4));
        childImage5 = Convert1DtoPixel(part5, (h/2)+(h%2), (w/4)+(w%4));
        childImage6 = Convert1DtoPixel(part6, (h/2)+(h%2), (w/4)+(w%4));
        childImage7 = Convert1DtoPixel(part7, (h/2)+(h%2), (w/4)+(w%4));
    }
}
else if (mode == 3){
    // BLUE
}
else if (mode == 4){
    // GREEN
}
else {
    fprintf(stderr, "Invalid mode\n");
}
// show numerical valuesDumpImage( startImage, h, w );
```

```
    // Parent process should combine images and save output
    if(pid1>0 && pid2 >0 && pid3 > 0){
        Pixel **finalImage = (Pixel **)malloc(sizeof(Pixel *) * h);
        finalImage[0] = (Pixel *)malloc(sizeof(Pixel) * (h * w));
        for( int i = 1 ; i < h ; i++ ) {
            finalImage[i] = (*finalImage + (w * i));
        }
        // combine parts
        CombineImagePieces(finalImage, childImage1, 0, h/2, 0, w/4); //part1
        CombineImagePieces(finalImage, childImage2, 0, h/2, w/4, w/2); //part2
        CombineImagePieces(finalImage, childImage3, 0, h/2, w/2, 0.75*w); //part3
        CombineImagePieces(finalImage, childImage4, 0, h/2, 0.75*w, w); //part4
        CombineImagePieces(finalImage, childImage5, h/2, h, 0, w/4); //part5
        CombineImagePieces(finalImage, childImage6, h/2, h, w/4, w/2); //part6
        CombineImagePieces(finalImage, childImage7, h/2, h, w/2, 0.75*w); //part7
        CombineImagePieces(finalImage, parentImage, h/2, h, 0.75*w, w); //part8
        // save resulting image
        SaveImage(outputFile, finalImage, h, w);
    }
    return 0;
}
```