CS-342

Operating Systems

Project #1

Kerem Ayöz
21501569 / Section:02

PART A

phistogram.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <sys/syscall.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/time.h>

// Child process function, creates the intermediate files by reading the data in input files.
void create_intermediate(char *file_name, double min_val, double max_val, int bin_count, int turn) {

  // Find w and create the array to store statistics.
  double w = ceil((max_val-min_val) / (bin_count*1.0));
  int bins[bin_count];

  for (int i = 0; i < bin_count; i++)
    bins[i] = 0;

  // Open file to read.
  FILE *fp;
  char buf[1000];
  fp =fopen(file_name,"r");

  // Read the file and increase the count of the bin it should be placed.
  while (1) {
      int ret = fscanf(fp, "%s", buf);
        if(ret == 1)
      bins[(int)((atof(buf) - min_val) / w)]++;
        else if(ret == EOF)
            break;
  }
  fclose(fp);

  // Create intermediate files.
  char inter_file[50];
  sprintf(inter_file, "intermediate_%d.txt", turn);

  FILE *f = fopen(inter_file, "w");
  if (f == NULL) {
      printf("Error opening file!\n");
      exit(1);
  }

  for (int i = 0; i < bin_count; ++i)
    fprintf(f, "%d:%d\n",i+1,bins[i]);

  fclose(f);
}

int main(int argc, char const *argv[]) {

  struct timeval t0;
  struct timeval t1;
    gettimeofday(&t0, 0);

  // Min and max values.
  double min_val = atoi(argv[1]);
  double max_val = atoi(argv[2]);

  // Bin count.
  int bin_count = atoi(argv[3]);

  // Size.
  int n = atoi(argv[4]);

  // File names.
  char files[n][50];

  for (int i = 0; i < n; ++i)
    strcpy(files[i],argv[5+i]);

  // Process id's of the child processes.
  pid_t ids[n];

  // Fork n times with parent.
  for(int i=0;i<n;i++)  {
    ids[i] = fork();

    // Where child processes work.
    if (ids[i] == 0) {
      create_intermediate(files[i], min_val, max_val, bin_count, i);
      exit(0);
    }
  }

  // Wait until all child processes finish.
  for(int i=0;i<n;i++)
    wait(NULL);

  // Parent's turn to combine the intermediate files.
  int combined_bins[bin_count];
  for (int i = 0; i < bin_count; i++)
    combined_bins[i] = 0;

  // Read each intermediate file.
  for(int i = 0; i < n; ++i) {
    FILE *fp;
    char buf[1000];

    char inter_file[50];
    sprintf(inter_file, "intermediate_%d.txt", i);
    fp =fopen(inter_file,"r");

    // Add the counts in every bin in each file to the last result which is combined_bins.
    while (1) {
        int ret = fscanf(fp, "%s", buf);
          if(ret == 1) {
            char *bin;
            bin = strtok(buf, ":");
            char *count;
            count = strtok(NULL, ":");
            combined_bins[atoi(bin)-1] += atoi(count);
          }
          else if(ret == EOF)
              break;
    }

    // Remove intermediate files.
    remove(inter_file);
    fclose(fp);
  }

  // Output file name.
  char outfile[50];
  strcpy(outfile, argv[n+5]);

  FILE *f = fopen(outfile, "w");
  if (f == NULL) {
      printf("Error opening file!\n");
      exit(1);
  }

  // Write the result to output file.
  for (int i = 0; i < bin_count; ++i)
    fprintf(f, "%d:%d\n",i+1,combined_bins[i]);

  fclose(f);

  // Calculate the running time
  gettimeofday(&t1, 0);
  //double elapsed = (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
  //printf("Time elapsed for multiprocess: %f seconds\n", elapsed/100000.0);
  return 0;
}
```

PART B

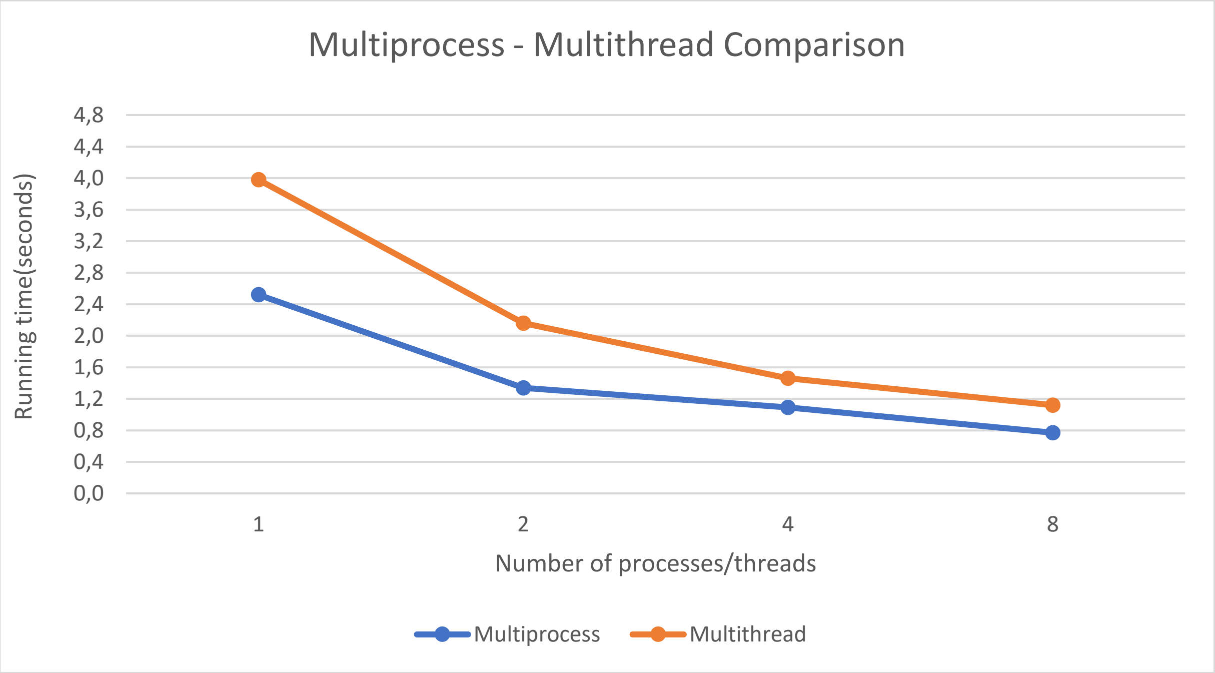
thistogram.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#include <math.h>
#include <sys/syscall.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/time.h>

//Global data structure
int **data;

//Min and max
double min_val;
double max_val;

// Bin count
int bin_count;

// Size
int n;

// File names
char **files;

// Bin width
double w;

// Thread function.
void *worker(void *arg) {

  // Get the index of thread in thread workers array created in main.
  int t_id = *((int *)arg);

  // Find the counts of numbers in each bin.
  FILE *fp;
  char buf[1000];
  fp =fopen(files[t_id],"r");

  while (1) {
      int ret = fscanf(fp, "%s", buf);
        if(ret == 1)
      data[t_id][(int)((atof(buf) - min_val) / w)] += 1;
        else if(ret == EOF)
            break;
  }

  fclose(fp);
  pthread_exit(0);
}

int main(int argc, char const *argv[]) {

  struct timeval t0;
  struct timeval t1;
```

```c
    gettimeofday(&t0, 0);

  // Initialize the parameters.
  min_val = atoi(argv[1]);
  max_val = atoi(argv[2]);
  bin_count = atoi(argv[3]);
  n = atoi(argv[4]);
  w = ceil((max_val-min_val) / (bin_count*1.0));
      files = (char **)malloc(n * sizeof(char *));
      for (int i=0; i<n; i++)
          files[i] = (char *)malloc(50 * sizeof(char));
  for (int i = 0; i < n; i++) {
    strcpy(files[i],argv[5+i]);
  }

  // Thread array to store the threads.
  pthread_t workers[n];

  // Data array; n rows for n threads, bin_count columns for each thread's bins
  data = (int **)malloc(n * sizeof(int *));
  for (int i=0; i<n; i++)
    data[i] = (int *)malloc(bin_count * sizeof(int));

  // Used that array for passing index of the thread to the worker function
  int indexes[n];

  // Create the threads
  for(int i=0;i<n;i++)  {
    indexes[i] = i;
    (void) pthread_create(&workers[i], NULL, worker, &indexes[i]);
  }

  // Wait for threads to be completed
  for (int i = 0; i < n; i++)
    (void) pthread_join(workers[i], NULL);

  // Parent's turn to generate output file from the global data array
  char outfile[50];
  strcpy(outfile, argv[n+5]);

  FILE *f = fopen(outfile, "w");
  if (f == NULL) {
      printf("Error opening file!\n");
      exit(1);
  }

  for (int i = 0; i < bin_count; ++i) {
    int total = 0;
    for (int j = 0; j < n; ++j)
      total += data[j][i];
    fprintf(f, "%d:%d\n",i+1,total);
  }
  fclose(f);

  // Calculate the running time
  gettimeofday(&t1, 0);
  //double elapsed = (t1.tv_sec-t0.tv_sec)*1000000 + t1.tv_usec-t0.tv_usec;
  //printf("Time elapsed for multiprocess: %f seconds\n", elapsed/100000.0);
  return 0;
}
```

PART C

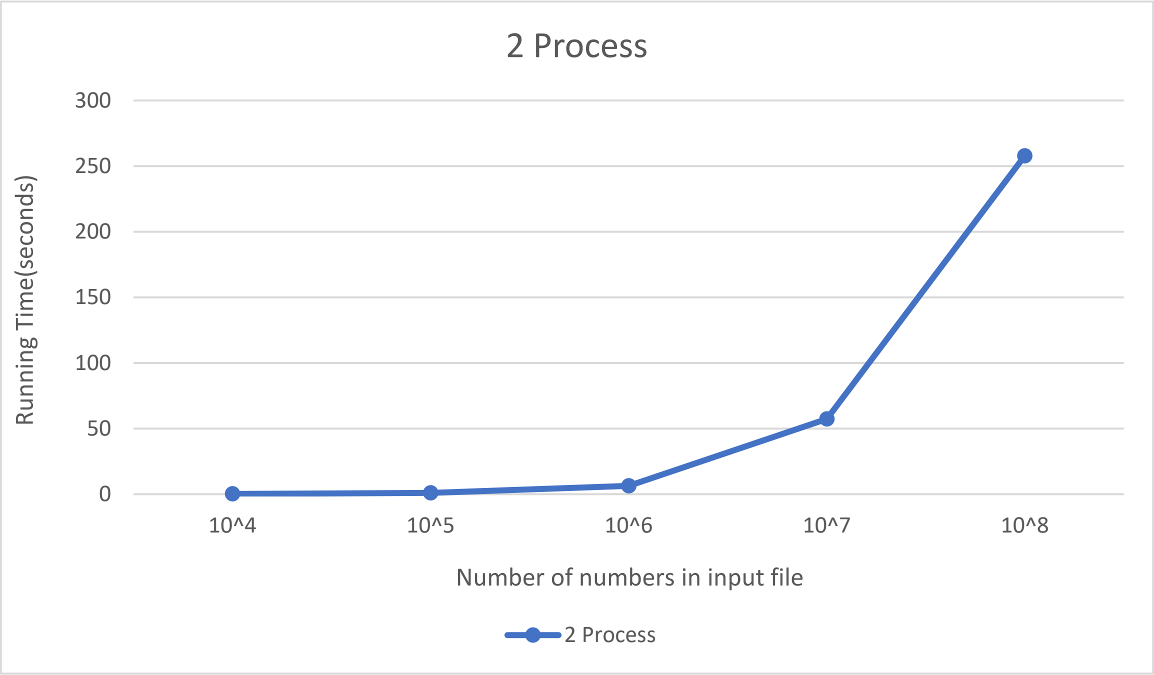
a-) To make the experiments on the same input, I generated 2^20 numbers between 0 – 100000. Each of these numbers are floating point numbers with 5 decimal digits. To use the same input with the different number of processes and threads, I divide that input to the number of threads/processes in each experiment. Total input size is fixed and contains 2^20 numbers. For instance, in the 4-process experiment, I divide the input to 4 so each input file contains 2^18 numbers. Running times of the phistogram and thistogram is in below. Mean and standard deviation is calculated from the running times of programs with different input sizes.



- **Multi-process**
  - Mean: 1.42692 seconds
  - Standard Deviation: 0.7602 seconds

- **Multi-thread**
  - Mean: 2.18318 seconds
  - Standard Deviation: 1.27747 seconds

b-) For each experiment, I generated files that contain 10^4, 10^5, 10^6, 10^7 and 10^8 floating point numbers with 5 decimal digits. Increase in input size caused running times to increase exponentially. Mean and standard deviation is calculated from the running times of the program with different input sizes.



- Mean: 64.518 seconds
- Standard Deviation: 110.679 seconds