Министерство науки и образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
"Московский институт электронной техники"
(МИЭТ)

Отчет по лабораторной работе № 3

Операционные системы

Выполнил: студент ПМ - 31

Мартынова Мария Олеговна

2023 г.

Задание 1

Написать функцию GetMinMax в find_max_min.c, которая ищет минимальный и максимальный элементы массива, на заданном промежутке. Разобраться, что делает программа в sequiential_min_max.c, скомпилировать, проверить, что написанный вами GetMinMax работает правильно.

lab3/src/find_min_max.c

```c
#include <limits.h>

struct MinMax GetMinMax(int *array, unsigned int begin, unsigned int end) {
  struct MinMax min_max;
  min_max.min = INT_MAX;
  min_max.max = INT_MIN;

  // your code here
int i;
  for(i = begin; i < end; i++)
  {
    if(array[i] < min_max.min)
      min_max.min = array[i];

    if(array[i] > min_max.max)
      min_max.max = array[i];
  }

  return min_max;
}
```

```
~/oslab2019$ cd lab3/src
~/.../lab3/src$ gcc -Wall find_min_max.c sequential_min_max.c utils.c
-o my_min_max
~/.../lab3/src$ ./my_min_max
Usage: ./my_min_max seed arraysize
~/.../lab3/src$ ./my_min_max 3 25
min: 8614858
max: 2029100602
~/.../lab3/src$
```

Задание 2-3

Завершить программу parallel_min_max.c, так, чтобы задача нахождения минимума и максимума в массиве решалась параллельно. Если выставлен

аргумент by_files для синхронизации процессов использовать файлы (задание 2), в противном случае использовать pipe (задание 3)

Для создания процессов используется системный вызов: **fork().** Вызов fork() создает новое адресное пространство, которое полностью идентично адресному пространству основного процесса. Другими словами, вызов fork() создает новый процесс. После выполнения этого системного вызова вы получаете два абсолютно одинаковых процесса — основной и порожденный. Функция fork() возвращает 0 в порожденном процессе и PID (Process ID — идентификатор порожденного процесса) — в основном. PID — это целое число.

pipe() создаёт однонаправленный канал данных, который можно использовать для взаимодействия между процессами. Массив *pipefd* используется для возврата двух файловых описателей, указывающих на концы канала. *pipefd[0]* указывает на конец канала для чтения. *pipefd[1]* указывает на конец канала для записи. Данные, записанные в конец канала, буферизируются ядром до тех пор, пока не будут прочитаны из конца канала для чтения.

fopen modes

The allowed modes for fopen are as follows:

```
r  - open for reading
w  - open for writing (file need not exist)
a  - open for appending (file need not exist)
r+ - open for reading and writing, start at beginning
w+ - open for reading and writing (overwrite file)
a+ - open for reading and writing (append if file exists)
```

```
#include <ctype.h>
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#include <sys/time.h>
#include <sys/types.h>
#include <sys/wait.h>

#include <getopt.h>

#include "find_min_max.h"
#include "utils.h"

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

```
int seed = -1;
int array_size = -1;
int pnum = -1;
bool with_files = false;

while (true) {
  int current_optind = optind ? optind : 1;

  static struct option options[] = {{"seed", required_argument, 0, 0},
                                    {"array_size", required_argument, 0, 0},
                                    {"pnum", required_argument, 0, 0},
                                    {"by_files", no_argument, 0, 'f'},
                                    {0, 0, 0, 0}};

  int option_index = 0;
  int c = getopt_long(argc, argv, "f", options, &option_index);

  if (c == -1) break;

  switch (c) {
    case 0:
      switch (option_index) {
        case 0:
          seed = atoi(optarg);
          // your code here
          // error handling
          if (seed<=0)
          {
            printf("Seed must be a positive number");
              return 1;
          }
          break;
        case 1:
          array_size = atoi(optarg);
          // your code here
          // error handling
          if (array_size<=0)
          {
            printf("Array size must be a positive number");
              return 1;
          }
          break;
        case 2:
          pnum = atoi(optarg);
          // your code here
          // error handling
          if (pnum<1)
          {
            printf("Process number must be > 1");
              return 1;
          }
          break;
        case 3:
          with_files = true;
          break;

        defalut:
```

```
            printf("Index %d is out of options\n", option_index);
        }
        break;
      case 'f':
        with_files = true;
        break;

      case '?':
        break;

      default:
        printf("getopt returned character code 0%o?\n", c);
    }
  }

  if (optind < argc) { // optind - количесвто комментов, которые обработал getopt_long
    printf("Has at least one no option argument\n");
    return 1;
  }

  if (seed == -1 || array_size == -1 || pnum == -1) {
    printf("Usage: %s --seed \"num\" --array_size \"num\" --pnum \"num\" \n",
           argv[0]);
    return 1;
  }

  int *array = malloc(sizeof(int) * array_size);
  GenerateArray(array, array_size, seed);
  int active_child_processes = 0;

  int fd[2];
  pipe(fd);

  FILE *f1;
  if(with_files)
  {
    f1 = fopen("task_2-3.txt", "w");
          if (!f1)
          {
            printf("Error txt");
            return 1;
          }
          fclose(f1);
  }
  struct timeval start_time;
  gettimeofday(&start_time, NULL);

  for (int i = 0; i < pnum; i++) {
    pid_t child_pid = fork();
    if (child_pid >= 0) {
      // successful fork
      active_child_processes += 1;
      if (child_pid == 0) {
        // child process
        // parallel somehow
        int begin = i*array_size/pnum;
        int end;
```

```
      if (i == pnum - 1)
        end = array_size;
      else
        end = (i + 1)*array_size/pnum;
      struct MinMax min_max = GetMinMax(array, begin, end);


      if (with_files) {
        // use files here
        FILE *f = fopen("task_2-3.txt", "a");
        if (!f)
        {
          printf("Error txt");
          return 1;
        }

        fprintf(f, "%d %d ", min_max.min, min_max.max);
        fclose(f);
      } else {
        // use pipe here
        write(fd[1], &min_max.min, sizeof(int));
        write(fd[1], &min_max.max, sizeof(int));
        close(fd[0]); // закрытие дискриптора на чтение
        close(fd[1]); // закрытие дискриптора на запись
      }
      return 0;
    }

  } else {
    printf("Fork failed!\n");
    return 1;
  }
}

close(fd[1]);
while (active_child_processes > 0) {
  // your code here
  wait(NULL);
  active_child_processes -= 1;
}

struct MinMax min_max;
min_max.min = INT_MAX;
min_max.max = INT_MIN;

FILE *f;
if(with_files)
{
   f = fopen("task_2-3.txt", "r");
        if (!f)
        {
          printf("Error txt");
          return 1;
        }
}

for (int i = 0; i < pnum; i++) {
```

```
    int min = INT_MAX;
    int max = INT_MIN;


    if (with_files) {
      // read from files
      fscanf(f, "%d %d ", &min, &max);
    } else {
      // read from pipes
      read(fd[0], &min, sizeof(int));
      read(fd[0], &max, sizeof(int));
    }

    if (min < min_max.min) min_max.min = min;
    if (max > min_max.max) min_max.max = max;
  }
  if(with_files)
    fclose(f);
  close(fd[0]);

  struct timeval finish_time;
  gettimeofday(&finish_time, NULL);

  double elapsed_time = (finish_time.tv_sec - start_time.tv_sec) * 1000.0;
  elapsed_time += (finish_time.tv_usec - start_time.tv_usec) / 1000.0;

  free(array);

  printf("Min: %d\n", min_max.min);
  printf("Max: %d\n", min_max.max);
  printf("Elapsed time: %fms\n", elapsed_time);
  fflush(NULL);
  return 0;
}
```

```
~/.../lab3/src$ gcc -Wall parallel_min_max.c utils.c find_min_max.c -o parallel_min_max
parallel_min_max.c: In function ‘main’:
parallel_min_max.c:25:9: warning: unused variable ‘current_optind’ [-Wunused-variable]
     int current_optind = optind ? optind : 1;
         ^~~~~~~~~~~~~~
parallel_min_max.c:75:11: warning: label ‘defalut’ defined but not used [-Wunused-label]
           defalut:
           ^~~~~~~
~/.../lab3/src$ ./parallel_min_max -f --seed=3 --array_size=300 --pnum=30
Min: 8370923
Max: 2129430561
Elapsed time: 5.496000ms
~/.../lab3/src$ ./parallel_min_max --seed=3 --array_size=300 --pnum=30
Min: 8370923
Max: 2129430561
Elapsed time: 3.017000ms
```

Задание 4

Изучить все targets в makefile, будьте готовы объяснить, за что они отвечают. Используя makefile, собрать получившиеся решения. Добавьте target all, отвечающий за сборку всех программ.

Для работы с утилитой make, вам понадобится так называемый **make-файл (makefile)**, который будет содержать описание требуемых действий. Как правило, make-файл описывает, каким образом нужно компилировать и компоновать программу.

Makefile — это файл, который хранится вместе с кодом в репозитории. Его обычно помещают в корень проекта. Он выступает и как документация, и как исполняемый код. Мейкфайл скрывает за собой детали реализации и раскладывает “по полочкам” команды, а утилита make запускает их из того мейкфайла, который находится в текущей директории.

lab3/src/makefile

```
CC=gcc
CFLAGS=-I.

sequential_min_max : utils.o find_min_max.o utils.h find_min_max.h
  $(CC) -o sequential_min_max find_min_max.o utils.o sequential_min_max.c $(CFLAGS)

parallel_min_max : utils.o find_min_max.o utils.h find_min_max.h
  $(CC) -o parallel_min_max utils.o find_min_max.o parallel_min_max.c $(CFLAGS)

utils.o : utils.h
  $(CC) -o utils.o -c utils.c $(CFLAGS)

find_min_max.o : utils.h find_min_max.h
  $(CC) -o find_min_max.o -c find_min_max.c $(CFLAGS)

clean :
  rm utils.o find_min_max.o sequential_min_max parallel_min_max

all : utils.o find_min_max.o utils.h find_min_max.h
  $(CC) -o sequential_min_max find_min_max.o utils.o sequential_min_max.c $(CFLAGS)
  $(CC) -o parallel_min_max utils.o find_min_max.o parallel_min_max.c $(CFLAGS)
```

```
~/.../lab3/src$ make all
gcc -o sequential_min_max find_min_max.o utils.o sequential_min_max.c -I.
gcc -o parallel_min_max utils.o find_min_max.o parallel_min_max.c -I.
```

Задание 5

Написать программу, которая запускает в отдельном процессе ваше приложение sequiential_min_max. Добавить его сборку в ваш makefile.

exec запускает исполняемый файл в контексте уже существующего процесса, заменяя предыдущий исполняемый файл. Программа меняется, а процесс остаётся.

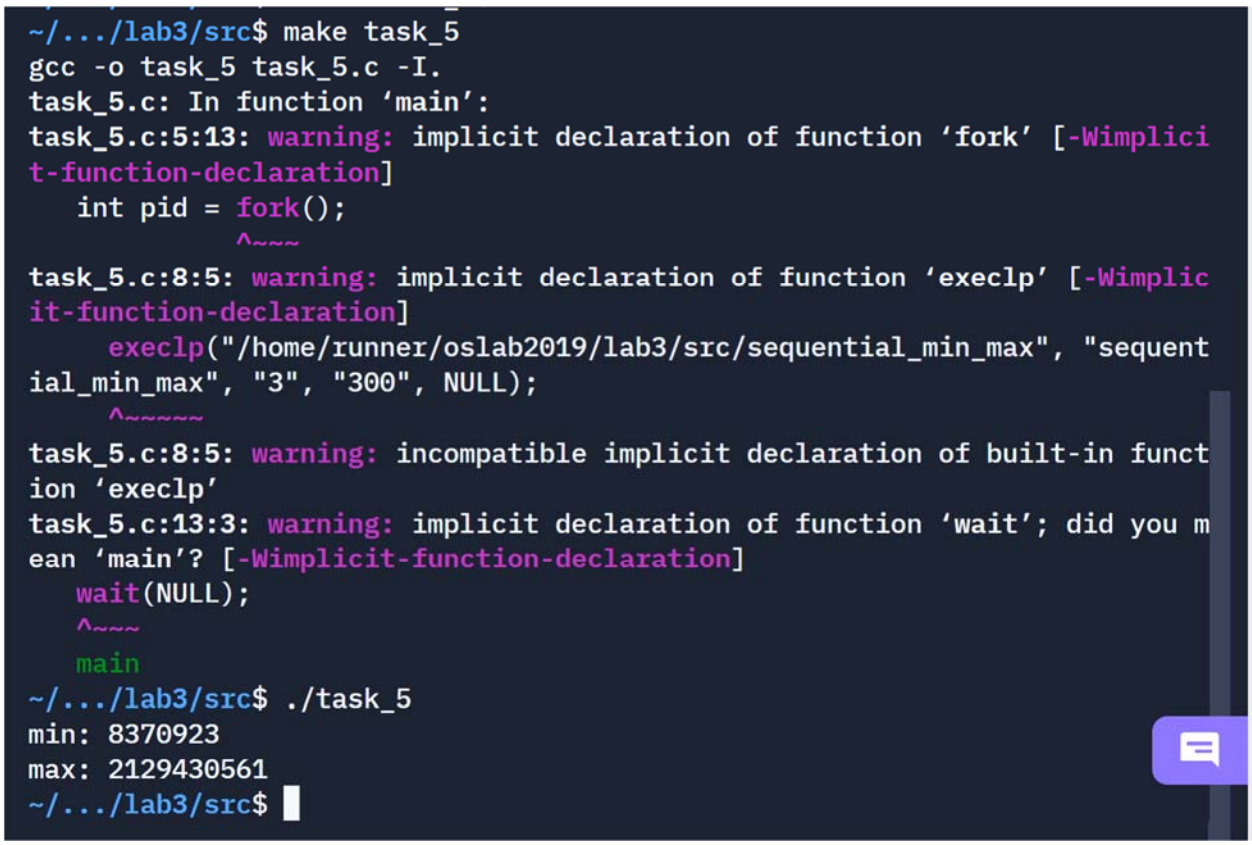
- Версия с l принимает аргументы через varargs, версия с v принимает массив строк.
- Версия с e позволяет дополнительно передать переменные окружения.
- Версия с p ищет исполняемый файл в PATH, без p требует полный путь.

lab3/src/task_5.c ×

```c
#include <stdio.h>

int main()
{
  int pid = fork();
  if (pid == 0)
  {
    execlp("/home/runner/oslab2019/lab3/src/sequential_min_max", "sequential_min_max", "3", "300", NULL);
    printf("Error");
    return 1;
  }

  wait(NULL);
  return 0;
}
```

lab3/src/makefile

```
CC=gcc
CFLAGS=-I.

sequential_min_max : utils.o find_min_max.o utils.h find_min_max.h
  $(CC) -o sequential_min_max find_min_max.o utils.o sequential_min_max.c $(CFLAGS)

parallel_min_max : utils.o find_min_max.o utils.h find_min_max.h
  $(CC) -o parallel_min_max utils.o find_min_max.o parallel_min_max.c $(CFLAGS)

utils.o : utils.h
  $(CC) -o utils.o -c utils.c $(CFLAGS)

find_min_max.o : utils.h find_min_max.h
  $(CC) -o find_min_max.o -c find_min_max.c $(CFLAGS)

clean :
  rm utils.o find_min_max.o sequential_min_max parallel_min_max

all : utils.o find_min_max.o utils.h find_min_max.h
  $(CC) -o sequential_min_max find_min_max.o utils.o sequential_min_max.c $(CFLAGS)
  $(CC) -o parallel_min_max utils.o find_min_max.o parallel_min_max.c $(CFLAGS)

task_5 :
  $(CC) -o task_5 task_5.c $(CFLAGS)
```


~/.../lab3/src$ make task_5
gcc -o task_5 task_5.c -I.
task_5.c: In function ‘main’:
task_5.c:5:13: warning: implicit declaration of function ‘fork’ [-Wimplicit-function-declaration]
int pid = fork();
^~~~
task_5.c:8:5: warning: implicit declaration of function ‘execlp’ [-Wimplicit-function-declaration]
execlp("/home/runner/oslab2019/lab3/src/sequential_min_max", "sequential_min_max", "3", "300", NULL);
^~~~~~
task_5.c:8:5: warning: incompatible implicit declaration of built-in function ‘execlp’
task_5.c:13:3: warning: implicit declaration of function ‘wait’; did you mean ‘main’? [-Wimplicit-function-declaration]
wait(NULL);
^~~~
main
~/.../lab3/src$ ./task_5
min: 8370923
max: 2129430561
~/.../lab3/src$