**Лабораторная работа №4**

**Задание 1**

**Необходимые знания**

1. Функция kill

"Сигнал - программное прерывание, доставляемое процессу. Как уже отмечалось, "доставка прерывания" означает, что при возникновении некоторого события операционная система вызывает ЗАДАННЫЙ ПРОЦЕССОМ обработчик этого события.

Сигнал может быть послан процессу не непосредственно операционной системой, но также и другим процессом, что можно использовать, например, для синхронизации процессов. Посылка сигнала осуществляется системным вызовом kill()

По умолчанию, если мы не указываем какой сигнал посылать, посылается сигнал SIGTERM (от слова termination — завершение).

SIGTERM указывает процессу на то, что необходимо завершиться.

Сигнал SIGTERM может и не остановить процесс (например, при перехвате или блокировке сигнала),

SIGKILL же выполняет уничтожение процесса всегда, так как его нельзя перехватить или проигнорировать.

1. Неблокирующий wait c WNOHANG

Функция wait приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс не завершится, или до появления сигнала, который либо завершает текущий процесс,либо требует вызвать функцию-обработчик.

Если дочерний процесс к моменту вызова функции уже завершился (так называемый "зомби" ("zombie")),то функция немедленно возвращается. Системные ресурсы, связанные с дочерним процессом, освобождаются.

Функция waitpid приостанавливает выполнение текущего процесса до тех пор, пока дочерний процесс, указанный в параметре pid, не завершит выполнение, или пока не появится сигнал, который либо завершает текущий процесс либо требует вызвать функцию-обработчик.

Если указанный дочерний процесс "зомби", то функция немедленно возвращается. Системные ресурсы, связанные с дочерним процессом, освобождаются.

Значение options создается путем логического сложения нескольких следующих констант:

WNOHANG

означает немедленное возвращение управления, если ни один дочерний процесс не завершил выполнение.

1. Функция alarm, сигнал SIGALRM, функция signal.

РЕАКЦИЯ на сигнал осуществляется посредством вызова signal(). Имеет 2 параметра, первый - номер сигнала, на который требуется установить обработчик, второй - указатель на функцию-обработчик. Последняя должна принимать один параметр типа int и не должна возвращать никакого значения. В качестве второго параметра можно указать SIG\_IGN или SIG\_DFL, что означает игнорировать данный сигнал и восстановить реакцию на данный сигнал по умолчанию, соотвественно. Вызов возвращает указатель на старый обработчик или константу SIG\_ERR в случае ошибки.

SIGALRM (от англ. signal и alarm — тревога) — применяемый в POSIX-системах сигнал по истечении времени, предварительно заданного функцией alarm() SIGALRM посылается ядром через целое количество секунд после вызова alarm()

Дополнить программу parallel\_min\_max.c из ***лабораторной работы №3***, так чтобы после заданного таймаута родительский процесс посылал дочерним сигнал SIGKILL. Таймаут должен быть задан, как именной необязательный параметр командной строки (--timeout 10). Если таймаут не задан, то выполнение программы не должно меняться.

#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"

#include <signal.h>

bool kill\_flag = false;

void handle\_alarm(int sig)

{

    kill\_flag = true;

}

int main(int argc, char \*\*argv) {

  int seed = -1;

  int array\_size = -1;

  int pnum = -1;

  int timeout = -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},

                                      {"timeout", 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 positive");

                seed = -1;

            }

            break;

          case 1:

            array\_size = atoi(optarg);

            // your code here

            // error handling

            if (array\_size <= 0)

            {

                printf("arraysize must be positive");

                array\_size = -1;

            }

            break;

          case 2:

            pnum = atoi(optarg);

            if (pnum <= 0)

            {

                printf("pnum is positive number");

                pnum = -1;

            }

            break;

          case 3:

            timeout = atoi(optarg);

            if (timeout <= 0)

            {

                printf("timeout must be positive\n");

                return 1;

            }

            break;

          case 4:

            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) {

        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\" --timeout \"timeout\"\n",

            argv[0]);

        return 1;

    }

    int \*array = malloc(sizeof(int) \* array\_size);

    GenerateArray(array, array\_size, seed);

    int active\_child\_processes = 0;

    struct timeval start\_time;

    gettimeofday(&start\_time, NULL);

    int pipefd[2];

    pipe(pipefd);

    if (timeout != -1){

        signal( SIGALRM, handle\_alarm );

        alarm( timeout );

    }

    int n = array\_size / pnum;

    pid\_t currentPid[pnum];

    for (int i = 0; i < pnum; i++)

    {

        pid\_t w, child\_pid = -1;

        child\_pid = fork();

        if (child\_pid >= 0)

        {

            // successful fork

            active\_child\_processes += 1;

            if (child\_pid == 0)

            {

                // child process

                // parallel somehow

                struct MinMax min\_max;

                if (i != pnum-1)

                {

                    min\_max = GetMinMax(array, i\*n, (i+1)\*n - 1);

                }

                else

                {

                    min\_max = GetMinMax(array, i\*n, array\_size-1);

                }

                if (with\_files)

                {

                    FILE \*fp;

                    if (i == 0)

                    {

                        fp = fopen("output.txt", "w");

                    }

                    else

                    {

                        fp = fopen("output.txt", "a");

                    }

                    fprintf(fp, "%d %d\n", min\_max.min, min\_max.max);

                    fclose(fp);

                }

                else

                {

                    write(pipefd[1], &min\_max, sizeof(struct MinMax));

                }

                return 0;

            }

            if (child\_pid > 0)

            currentPid[i] = child\_pid;

        //printf("%d\*\*",currentPid[i]);

        }

            else

            {

            printf("Fork failed!\n");

            return 1;

            }

    }

    int status;

    while (active\_child\_processes > 0) {

    // your code here

    if (kill\_flag == true){

        printf("\nThe child processes is killing...");

        for (int i =0; i<pnum; i++)

            kill(currentPid[i], SIGKILL);

        return 1;

    }

    else     printf("\nThe child processes is not killing...");

    while (waitpid(-1, &status, WNOHANG) > 0)

    {

        active\_child\_processes -= 1;

    }}

  struct MinMax min\_max;

  min\_max.min = INT\_MAX;

  min\_max.max = INT\_MIN;

  for (int i = 0; i < pnum; i++) {

    int min = INT\_MAX;

    int max = INT\_MIN;

    if (with\_files) {

      struct MinMax min\_max;

      FILE \*fp;

      int j;

      fp = fopen("output.txt", "r");

      for (j = 0; j < i || (i == 0 && j <= i); j++)

      {

        fscanf(fp, "%d %d", &min\_max.min, &min\_max.max);

        if (min < min\_max.min)

          min\_max.min = min;

        if (max > min\_max.max)

          min\_max.max = max;

      }

      fclose(fp);

      min = min\_max.min;

      max = min\_max.max;

    }

    else {

      struct MinMax min\_max;

      read(pipefd[0], &min\_max, sizeof(struct MinMax));

      min = min\_max.min;

      max = min\_max.max;

    }

    if (min < min\_max.min) min\_max.min = min;

    if (max > min\_max.max) min\_max.max = max;

  }

  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;

}

[jboss@workspacehj5n2js983kq5lxl src]$ make

gcc -o utils.o -c utils.c -I.

gcc -o find\_min\_max.o -c find\_min\_max.c -I.

gcc -o parallel\_min\_max utils.o find\_min\_max.o parallel\_min\_max.c -I.

[jboss@workspacehj5n2js983kq5lxl src]$ ./parallel\_min\_max --seed 3 --array\_size 20 --pnum 3

1205554746

483147985

844158168

953350440

612121425

310914940

1210224072

1856883376

1922860801

495649264

8614858

989089924

378651393

1344681739

2029100602

1816952841

21468264

552076975

87517201

953369895

Process killedProcess killedProcess killedProcess killedProcess killedProcess killedMin: 8614858

Max: 2029100602

Elapsed time: 0.381000ms

**Задание 2**

**Необходимые знания**

1. Что такое зомби процессы, как появляются, как исчезают.

Если один из дочерних процессов всё же завершился, а его родительский процесс не смог получить об этом информацию, то такой дочерний процесс становится зомби.Зомби процессы Linux не выполняются и убить их нельзя, даже с помощью sigkill, они продолжают висеть в памяти, пока не будет завершён их родительский процесс.

Создать программу, с помощью которой можно продемонстрировать зомби процессы. Необходимо объяснить, как появляются зомби процессы, чем они опасны, и как можно от них избавиться.

#include <stdlib.h>

#include <sys/types.h>

#include <unistd.h>

/\*Процесс при завершении освобождает все свои ресурсы и становится

«зомби» — пустой записью в таблице процессов, хранящей статус завершения

предназначенный для чтения родительским процессом.

Зомби процесс существует до тех пор, пока родительский процесс не прочитает

его статус с помощью системного вызова wait(), в результате чего запись в

таблице процессов будет освобождена.\*/

int main()

{

    pid\_t child\_pid;

    /\* Создание дочернего процесса. \*/

    child\_pid = fork();

    if (child\_pid > 0)

    {

        /\* Это родительский процесс — делаем минутную паузу. \*/

        sleep(60);

    }

    else

    {

        /\* Это дочерний процесс — немедленно завершаем работу. \*/

        exit(0);

    }

    return 0;

}

[jboss@workspacehj5n2js983kq5lxl src]$ gcc -c zombie.c

[jboss@workspacehj5n2js983kq5lxl src]$ gcc zombie.o -o zombie

[jboss@workspacehj5n2js983kq5lxl src]$ ./zombie

[jboss@workspacehj5n2js983kq5lxl src]$ ./zombie

[jboss@workspacehj5n2js983kq5lxl src]$ ps aux| grep defunct

jboss 89 0.0 0.0 0 0 pts/0 Z 08:23 0:00 [zombie] <defunct>

jboss 143 0.0 0.0 0 0 pts/0 Z+ 08:25 0:00 [zombie] <defunct>

jboss 145 0.0 0.0 16376 504 pts/1 S+ 08:25 0:00 grep --color=auto defunct

**Задание 3**

**Необходимые знания**

1. Работа виртуальной памяти.

Скомпилировать process\_memory.c. Объяснить, за что отвечают переменные etext, edata, end.

1)Адреса этих символов соответствуют концам различных программных сегментов:

Etext - Первый адрес за концом сегмента текста (кода программы).

Edata- Первый адрес за концом сегмента инициализированных данных.

End - Первый адрес за концом сегмента неинициализированных данных (сегмента BSS).

[jboss@workspacehj5n2js983kq5lxl src]$ chmod u+rwx process\_memory.c

[jboss@workspacehj5n2js983kq5lxl src]$ gcc -c process\_memory.c

process\_memory.c:20:1: warning: return type defaults to ‘int’ [-Wimplicit-int]

main() {

^~~~

[jboss@workspacehj5n2js983kq5lxl src]$ gcc process\_memory.o -o process\_memory

[jboss@workspacehj5n2js983kq5lxl src]$ ./process\_memory

Address etext: 4009F5

Address edata: 601068

Address end : 6010A0

ID main is at virtual address: 400786

ID showit is at virtual address: 4008B4

ID cptr is at virtual address: 601060

ID buffer1 is at virtual address: 601080

ID i is at virtual address: 65415A7C

A demonstration

ID buffer2 is at virtual address: 65415A58

Alocated memory at 1AF46B0

This message is output by the function showit()

**Задание 4**

Создать makefile, который собирает программы из задания 1 и 3.

CC=gcc

CFLAGS=-I.

all:     parallel\_min\_max process\_memory.o utils.o find\_min\_max.o

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)

process\_memory.o:

    $(CC) -c process\_memory.c $(CFLAGS)

    $(CC) process\_memory.o -o process\_memory $(CFLAGS)

clean :

    rm utils.o find\_min\_max.o  parallel\_min\_max process\_memory

**Задание 5**

**Необходимые знания**

1. POSIX threads: как создавать, как дожидаться завершения.
2. Как линковаться на бибилотеку pthread

Системные вызовы fork и exec создают новые копии страниц памяти и списка файловых описателей.

Ядро ОС может создать новый поток на порядок быстрее, чем новый процесс.

int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr,

void \*(\*start)(void \*), void \*arg);

Первый параметр вызова pthread\_create() является адресом для хранения идентификатора создаваемого потока типа pthread\_t.

Аргумент start является указателем на потоковую void \* функцию, принимающей бестиповый указатель в качестве единственной переменной.

Аргумент arg — это бестиповый указатель, содержащий аргументы потока.

Чаще всего arg указывает на глобальную или динамическую переменную,но если вызываемая функция не требует наличия аргументов, то в качестве arg можно указать NULL.

Аргумент attr также является бестиповым указателем атрибутов потока pthread\_attr\_t.

Если этот аргумент равен NULL, то поток создается с атрибутами по умолчанию.

Чтобы подключить библиотеку Pthread к программе, нужно передать компоновщику опцию -lpthread

**Поток завершает выполнение задачи когда:**

потоковая функция выполняет return и возвращает результат произведенных вычислений;

в результате вызова завершения исполнения потока pthread\_exit();

тут при выходе старшей нити из main с помощью этой штуки, остальные нити работают в результате вызова отмены потока pthread\_cancel();

одна из нитей совершает вызов exit()

основная нить в функции main() выполняет return, и в таком случае все нити процесса резко сворачиваются.

Функция pthread\_join() ожидает завершения потока обозначенного THREAD\_ID.

Если этот поток к тому времени был уже завершен, то функция немедленно возвращает значение.

Смысл функции в том, чтобы синхронизировать потоки.

int pthread\_join (pthread\_t THREAD\_ID, void \*\* DATA);

Если указатель DATA отличается от NULL, то туда помещаются данные,

возвращаемые потоком через функцию pthread\_exit() или через инструкцию return потоковой функции.

Доработать parallel\_sum.c так, чтобы:

* Сумма массива высчитывалась параллельно.
* Массив генерировался с помощью функции GenerateArray из ***лабораторной работы №3***.
* Программа должна принимать входные аргументы: количество потоков, seed для генерирования массива, размер массива (./psum --threads\_num "num" --seed "num" --array\_size "num").
* Вместе с ответом программа должна выводить время подсчета суммы (генерация массива не должна попадать в замер времени).
* Вынести функцию, которая считает сумму в отдельную библиотеку.

#include <stdint.h>

#include <stdio.h>

#include <stdlib.h>

#include <getopt.h>

#include <stdbool.h>

#include <limits.h>

#include <sys/time.h>

#include "utils.h"

#include "Sum.h"

#include <pthread.h>

void \*ThreadSum(void \*args) {

  struct SumArgs \*sum\_args = (struct SumArgs \*)args;

  return (void \*)(size\_t)Sum(sum\_args);

}

int main(int argc, char \*\*argv) {

  /\*

   \*  TODO:

   \*  threads\_num by command line arguments

   \*  array\_size by command line arguments

   \*    seed by command line arguments

   \*/

  uint32\_t threads\_num = 0;

  uint32\_t array\_size = 0;

  uint32\_t seed = 0;

  static struct option options[] = {{"seed",required\_argument, 0, 0},

                                    {"array\_size",  required\_argument, 0, 0},

                                    {"threads\_num", required\_argument, 0, 0},

                                    {0, 0, 0, 0}};

        while (true) {

        int option\_index = 0;

        int c = getopt\_long(argc, argv, "", options, &option\_index);

        if (c == -1)

            break;

        switch (c) {

            case 0: {

                switch (option\_index) {

                    case 0: {

                        seed = atoi(optarg);

                        if (seed < 0) {

                            printf("Seed must be a positive number or 0. Now seed is %d\n", seed);

                            return -1;

                        }

                        break;

                    }

                    case 1:

                        array\_size = atoi(optarg);

                        if (array\_size < 0) {

                            printf("Array\_size must be a positive number. Now array\_size is %d\n", array\_size);

                            return -1;

                        }

                        break;

                    case 2: {

                        threads\_num = atoi(optarg);

                        if (threads\_num <= 0) {

                            printf("Threads\_num must be 1 or more. Now threads\_num is %d\n", threads\_num);

                            return -1;

                        }

                        break;

                    }

                }

                break;

            }

            case '?':

                break;

            default:

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

        }

    }

    if (threads\_num <= 0) {

        printf("Threads\_num must be 1 or more. Now threads\_num is %d\n", threads\_num);

        return -1;

    }

    printf("\nseed: %d, array\_size: %d, threads\_num: %d",seed, array\_size, threads\_num);

    pthread\_t threads[threads\_num];

  /\*

   \* TODO:

   \* your code here

   \* Generate array here

   \*/

  int \*array = malloc(sizeof(int) \* array\_size);

    GenerateArray(array, array\_size, seed);

    /\*for(int i = 0; i < array\_size; i++)

    {

       printf("%d ", array[i]);

    }

    printf("\n");

\*/

  struct SumArgs args[threads\_num];

    for(int i = 0; i < threads\_num; i++)

    {

        args[i].array = array;

        args[i].begin = i \* array\_size/threads\_num;

        if (i == threads\_num - 1 )

            args[i].end = array\_size;

        else

            args[i].end = (i + 1) \* array\_size/threads\_num;

    }

    struct timeval start\_time;

    gettimeofday(&start\_time, NULL);

  for (uint32\_t i = 0; i < threads\_num; i++) {

    if (pthread\_create(&threads[i], NULL, ThreadSum, (void \*)&args)) {

      printf("Error: pthread\_create failed!\n");

      return 1;

    }

  }

  int total\_sum = 0;

  for (uint32\_t i = 0; i < threads\_num; i++) {

    int sum = 0;

    pthread\_join(threads[i], (void \*\*)&sum);

    total\_sum += sum;

  }

    struct timeval finish\_time;

    gettimeofday(&finish\_time, NULL);

    double elapsed\_time = (finish\_time.tv\_usec - start\_time.tv\_usec) / 1000.0;

  free(array);

  printf("Total: %d\n", total\_sum);

  printf("It took %f milliseconds\n", elapsed\_time);

  return 0;

}

#include "Sum.h"

#include <stdio.h>

int Sum(const struct SumArgs \*args) {

  int sum = 0;

    for (int i = args->begin; i < args->end; i++) {

        sum += args->array[i];

    }

    return sum;

}

struct SumArgs {

  int \*array;

  int begin;

  int end;

};

int Sum(const struct SumArgs \*args);

**Задание 6**

Создать makefile для parallel\_sum.c.

C=gcc

CFLAGS=-I.

all: parallel\_sum

parallel\_sum : utils.o Sum.o utils.h Sum.h

$(CC) -pthread -o parallel\_sum utils.o Sum.o parallel\_sum.c $(CFLAGS)

utils.o : utils.h

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

Sum : Sum.h

$(CC) -o Sum.o -c Sum.c $(CFLAGS)

clean :

rm utils.o Sum.o parallel\_sum

[jboss@workspace0r5pfp8k507mwrgw src]$ make -f makefile2

clang -o utils.o -c utils.c -I.

clang -I. -c -o Sum.o Sum.c

clang -pthread -o parallel\_sum utils.o Sum.o parallel\_sum.c -I.

[jboss@workspace0r5pfp8k507mwrgw src]$ ./parallel\_sum --seed 2 --array\_size 200000 --threads\_num 5

seed: 2, array\_size: 200000, threads\_num: 5Total: 1656684000

It took 0.859000 milliseconds