**Отчет лабораторной работы №3**

**Тема «** **Мютексы и потоковые семафоры»**

**Султанов Шакир КА**

Студенты должны знать:

1. мютекс (критическая секция);
2. Потоковый семафор;
3. Назначение фунций: pthread\_mutex\_lock(), pthread\_mutex\_unlock(), sem\_wait(), sem\_pos().

**Задание**

Главный поток приложения запускает k одинаковых потоков для обработки некоторых запросов. Затем, главный поток читает запросы из текстового файла fin.txt и помещает их в очередь запросов и ждет нажатия клавиши для завершения работы приложения.

Параллельный поток выполняет в бесконечном цикле следующие действия: извлекает запрос из очереди запросов; обрабатывает запрос; выводит результат обработки запроса на консоль.

**КОД**

#include <malloc.h>

#include <pthread.h>

#include <semaphore.h>

//#include <math.h>

struct job {

struct job\* next;

int n;

};

struct job\* job\_queue;

/\* Исключающий семафор, защищающий очередь. \*/

pthread\_mutex\_t job\_queue\_mutex = PTHREAD\_MUTEX\_INITIALIZER;

/\*Потоковый семафор, подсчитывающий число заданий

в очереди. \*/

sem\_t job\_queue\_count;

/\* Начальная инициализация очереди. \*/

void initialize\_job\_queue () {

/\* Вначале очередь пуста. \*/

job\_queue = NULL;

/\* Устанавливаем начальное значение счетчика

семафора равным 0. \*/

sem\_init (&job\_queue\_count, 0, 0);

}

void process\_job(struct job\* next\_job) {

int n = next\_job->n;

int i;

for (i = n-1; i > 1; i--) {

double s = i / 2;

int j;

int flag = 0;

for (j = 2; j <= s; j++) {

if (i % j == 0) {

flag = 1;

break;

}

}

if (flag == 0) break;

}

printf("Поток #%d, полученное число:%d, результат:%d\n", (int)pthread\_self(), n, i);

}

/\* Обработка заданий до тех пор,

пока очередь не опустеет. \*/

void\* thread\_function (void\* arg) {

struct job\* next\_job;

while (1) {

sem\_wait(&job\_queue\_count);

/\* Захват исключающего семафора,

защищающего очередь. \*/

pthread\_mutex\_lock(&job\_queue\_mutex);

next\_job = job\_queue;

/\* Удаляем задание из списка. \*/

job\_queue = job\_queue->next;

/\* Освобождаем исключающий семафор,

так как работа с очередью окончена. \*/

pthread\_mutex\_unlock (&job\_queue\_mutex);

/\* Выполняем задание. \*/

process\_job(next\_job);

/\* Очистка. \*/

free(next\_job);

}

return NULL;

}

/\* Добавление нового задания в начало очереди. \*/

void enqueue\_job(int n) {

struct job\* new\_job;

/\* Выделение памяти для нового объекта задания. \*/

new\_job = (struct job\*) malloc (sizeof (struct job));

/\* Заполнение остальных полей структуры JOB... \*/

new\_job->n=n;

/\* Захватываем исключающий семафор, прежде чем

обратиться к очереди. \*/

pthread\_mutex\_lock (&job\_queue\_mutex);

/\* Помещаем новое задание в начало очереди. \*/

new\_job->next = job\_queue; job\_queue = new\_job;

/\* Устанавливаем семафор, сообщая о том, что в

очереди появилось новое задание.

Если есть потоки, заблокированные в ожидании

семафора, один из них будет разблокирован и

обработает задание. \*/

sem\_post(&job\_queue\_count);

/\* Освобождаем исключающий семафор. \*/

pthread\_mutex\_unlock(&job\_queue\_mutex);

}

int main() {

initialize\_job\_queue();

pthread\_t t;

int i;

for (i = 0; i < 3; i++) {

pthread\_create(&t, NULL, thread\_function, NULL);

}

FILE \*pfile;

pfile = fopen("flab3in.txt", "r");

int n;

while(1) {

if (fscanf(pfile, "%d", &n) == EOF) {

break;

}

enqueue\_job(n);

}

printf("Press any key to finish the job\n");

char c;

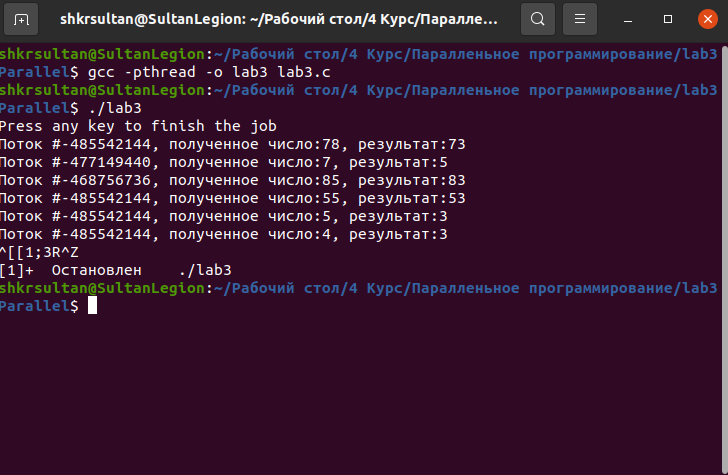
scanf("%c", &c);

fclose(pfile);

return 0;

}

**Результат**

****