Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа № 3**

по курсу

«Операционные системы»

|  |  |
| --- | --- |
| Студент: | Пшеницын А. А. |
| Группа: | М8О-206Б-18 |
| Вариант: | 18 |
| Преподаватель: | Соколов Андрей Алексеевич |
| Оценка: |  |
| Дата: |  |
| Подпись: |  |

Москва, 2019

**Цель работы**

Приобретение практических навыков в:

* Управление потоками в ОС
* Обеспечение синхронизации между потоками

**Задание (Вариант № 18)**

На вход подается текстовый файл. Необходимо рассчитать частотность встречаемых букв.

Количество используемой оперативной памяти должно задаваться «ключом».

**Общие сведения о программе**

В программе используются следующие функции:

* int pthread\_create(pthread\_t \*thread, const pthread\_attr\_t \*attr, void \*(\*start)(void \*), void \*arg) — создание нового потока
* int pthread\_join (pthread\_t THREAD\_ID, void \*\* DATA) — ожидание завершения переданного потока

# int open(const char \*filename, int access) — открытие файла с именем filename с модификатором доступа access

* int close(int fd) – закрытие файла
* int write(int handle, void \*buf, int count) — переписывает count байт из буфера buf в файл, соответствующий дескриптору файла handle
* int read(int fd, void \*buf, unsigned count) - переписывает count байт в буфер buf из файла, соответствующего дескриптору файла handle

**Код программы**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <pthread.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#define SIZE 30

typedef struct{

char c;

int rep;

}CharRepeat;

CharRepeat\* CR\_array\_create(){

CharRepeat\* res = (CharRepeat\*) malloc(26 \* sizeof(CharRepeat));

for(int i = 0; i < 26; i ++){

res[i].c = i + 'a';

res[i].rep = 0;

}

return res;

}

typedef struct{

char\* str;

CharRepeat\* res;

}pthrData;

void repeat(CharRepeat\* res, char\* str){

for(int i = 0; i < strlen(str); i ++){

if(str[i] >= 'a' && str[i] <= 'z'){

res[str[i] - 'a'].rep ++;

} else if(str[i] >= 'A' && str[i] <= 'Z'){

res[str[i] - 'A'].rep ++;

} else if(str[i] == '\n'){

break;

}

}

}

void\* thread\_func(void\* thread\_data){

pthrData\* data = (pthrData\*) thread\_data;

repeat(data -> res, data -> str);

return NULL;

}

int int\_read(char\* buf){

int res = 0;

for(int i = 0; i < strlen(buf); i ++){

if(buf[i] >= '0' && buf[i] <= '9'){

res = res \* 10 + (buf[i] - '0');

} else {

return -1;

}

}

return res;

}

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

if(argc != 3){

printf("Incorrect number of arguments\n");

exit(1);

}

int fr;

fr = open(argv[1], O\_RDONLY, 0);

if (fr == -1) {

printf("Cannot open file.\n");

exit(1);

}

int number\_thread = int\_read(argv[2]);

if(number\_thread < 0){

printf("Incorrect input of the second argument");

exit(1);

}

struct stat st;

if(fstat(fr, &st) < 0){

printf("fstat faild\n");

exit(1);

}

int n = 0;

int fsize = st.st\_size;

char str[fsize];

CharRepeat\* res = CR\_array\_create();

read(fr, str, fsize);

close(fr);

for(int i = 0; i < fsize; i++){

if(str[i] == EOF || str[i] == '\n'){

n++;

}

}

pthread\_t\* threads = (pthread\_t\*) malloc(n \* sizeof(pthread\_t));

pthrData\* threadData = (pthrData\*) malloc(n \* sizeof(pthrData));

int k = 0;

int i = 0;

while(k < number\_thread && k < n && i < fsize){

threadData[k].res = CR\_array\_create();

threadData[k].str = (char\*) malloc(SIZE \* sizeof(char));

int j = 0;

while(i < fsize){

if(str[i] == EOF || str[i] == '\n'){

break;

}

threadData[k].str[j] = str[i];

j ++;

i ++;

}

int tmp = pthread\_create(&(threads[k]), NULL, thread\_func, &threadData[k]);

if(tmp != 0){

printf("Maximum Thread Exceeded");

thread\_func(&threadData[1]);

k ++;

i ++;

break;

}

k ++;

i ++;

}

for(int h = 0; h < k; h++){

pthread\_join(threads[h], NULL);

}

free(threads);

if(i != fsize){

char buf[fsize - i];

int f = 0;

while(i < fsize){

if(str[i] == '\n'){

i ++;

continue;

}

buf[f] = str[i];

f ++;

i ++;

}

repeat(res, buf);

}

for(int i = 0; i < k; i ++){

for(int j = 0; j < 26; j ++){

res[j].rep += threadData[i].res[j].rep;

}

}

free(threadData);

for(int i = 0; i < 26; i ++){

if(res[i].rep != 0){

printf("%c - %d \n", res[i].c, res[i].rep);

}

}

}

**Принцип работы**

В функцию main(int argc, char\* argv[]) передаются следующие аргументы командной строки:

1. первый аргумент — имя файла, в котором будет рассчитывается частотность встречаемых букв (только английского алфавита)
2. второй аргумент — количество потоков number\_thread, где number\_thread > 0

С файла считывается одна строка с помощью read(). С помощью pthread\_create создается поток, в котором вычисляется частотность повторения букв в данной строке. Так продолжается до тех пор, пока:

* файл не закончится. Тогда просто складываем результаты выполнения каждого потока и получаем результат.
* не закончится количество максимально возможных потоков, задаваемых системой или number\_threat. Тогда в цикле вычисляем частотность букв оставшихся строк, а уже затем складываем результаты выполнения каждого потока и последнего цикла.

**Вывод**

Многопоточность в программировании является важным механизмом в наше время. Поток в Linux является отдельным процессом. Но в определенных обстоятельствах создание нового потока обладает явно выраженными преимуществами по сравнению с созданием нового процесса. Главное преимущество потоков в том, что накладные расходы при создании нового потока существенно меньше, чем при создании нового процесса.