Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа №4
по курсу «Операционные системы»

Студент:	Марков А.Н.
Группа:	М80-208Б-18
Преподаватель:	Миронов Е.С.
Оценка:	
Дата:	

Москва
2019

Содержание

1. Постановка задачи.

Составить и отладить программу на языке Си, осуществляющую работу с процессами и взаимодействие между ними в одной из двух операционных систем. В результате работы программа (основной процесс) должен создать для решения задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляются через системных сигналы/события и/или через отображаемые файлы. Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант задания №25. Родительский процесс отвечает за ввод и вывод. Дочерний процесс осуществляет поиск образца в строке.

2. Общие сведения о программе.

Исходный код хранится в файле main.c. В данном файле используются заголовочные файлы unistd.h, stdlib.h, string.h, ctype.h, stdio.h, limits.h, fcntl.h, semaphore.h, sys/stat.h, sys/mman.h, sys/types.h. В программе используются следующие вызовы:

1. sem_open — для создания нового именованного семафора или открытия уже существующего.
2. sem_unlink — для удаления именованного семафора.
3. fork — для создания дочернего процесса.
4. sem_post — для увеличения (разблокировки) семафора.
5. sem_wait — для уменьшения (блокировки) семафора.
6. mmap — для отображения файла в адресное пространство процесса.

3. Общий метод и алгоритм решения.

Для реализации поставленной задачи необходимо:

1. Считать данные.
2. Прозвести маппинг файла.
3. Создать два семафора для синхронизации работы с файлом, отображенным в память.
4. Манипулировать семафорами так, чтобы:
   - Дочерний процесс доходит до sem_wait(s2) и встает в ожидание.
   - Родительский процесс выполняет запись в файл, доходит до sem_post(s1).
   - Затем дочерний процесс начинает работу. А родительский процесс следующим действием доходит до sem_wait(s2) и встает в ожидание.
   - Дочерний процесс выполняет чтение из файла, поиск образца в строке, записывает результат в файл. Оповещает родительский процесс sem_post(s2).
   - Родительский процесс выводит результат поиска.

4. Основные файлы программы.

main.c

```
#include <unistd.h>
#include <stdlib.h> // for exit, atoi
#include <string.h>
#include <ctype.h>
#include <stdio.h>
```

```c
#include <limits.h>
#include <fcntl.h>
#include <semaphore.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/types.h>

struct string_type {
char *s;
int capacity;
int size;
};

typedef struct string_type string_type;

void string_init(string_type *str) {
str->s = (char *) malloc(sizeof(char) * 1);
str->capacity = 0;
str->size = 0;
}

void string_scan(string_type *str) {
char ch;
scanf("%c", &ch);
for (int i = 0; ch != '\n'; i++) {
if (i == str->size) {
if (str->size == 0) {
str->size = 1;
char *temp = realloc(str->s, str->size);
if (temp == NULL) {
printf("realloc error.\n");
exit(1);
}
} else {
char *temp = realloc(str->s, str->size * 2);
if (temp == NULL) {
printf("realloc error.\n");
exit(1);
}
str->size *= 2;
}
}
str->s[i] = ch;
str->capacity++;
scanf("%c", &ch);
}
}

void string_print(string_type *str) {
for (int i = 0; i < str->capacity; i++) {
printf("%c", str->s[i]);
}
}

void write_to_memory(char a, char *memory, int position) {
char *n = (char *) (memory + sizeof(char) * position);
*n = a;
}

char get_from_memory(char *memory, int position) {
char *res = (char *) (memory + sizeof(char) * position);
return *res;
```

```
}

char get_from_memory_for_string(char *memory, int position, string_type *str) {
char ch = *(memory + sizeof(char) * position);
if (str->capacity == str->size) {
if (str->size == 0) {
str->size = 1;
char *temp = realloc(str->s, str->size);
if (temp == NULL) {
printf("realloc error.\n");
exit(1);
}
} else {
char *temp = realloc(str->s, str->size * 2);
if (temp == NULL) {
printf("realloc error.\n");
exit(1);
}
str->size *= 2;
}
}
str->s[str->capacity++] = ch;
}

int get_int_from_memory(char *memory, int position) {
string_type temp;
string_init(&temp);
for (int i = 0; isdigit(*(i + memory + sizeof(char) * position)); i++) {
if (i == temp.size) {
if (temp.size == 0) {
temp.size = 1;
char *t = realloc(temp.s, temp.size);
if (t == NULL) {
printf("realloc error.\n");
exit(1);
}
} else {
char *t = realloc(temp.s, temp.size * 2);
if (t == NULL) {
printf("realloc error.\n");
exit(1);
}
temp.size *= 2;
}
}
temp.s[i] = *(i + memory + sizeof(char) * position);
}
return atoi(temp.s);
}

int number_of_digits(int n) {
int cnt = 1;
while (n / 10 != 0) {
n /= 10;
cnt++;
}
return cnt;
}

void reverse(char *a) {
int temp;
int size = strlen(a);
for (int i = 0; i < size / 2; i++) {
```

```
temp = a[i];
a[i] = a[size - 1 - i];
a[size - 1 - i] = temp;
}
}

void my_itoa(int n, char *a) {
int i;
for (i = 0; n / 10 != 0; i++) {
a[i] = (char) (n % 10 + 48);
n /= 10;
}
a[i] = (char) (n % 10 + 48);
a[i + 1] = '\0';
reverse(a);
}

int bmh(char *str, int str_len, char *pattern, int pat_len) {
int table[CHAR_MAX + 1];

if (str_len < pat_len || pat_len <= 0 || !str || !pattern) {
return -1;
}

for (register int i = 0; i < CHAR_MAX + 1; ++i) {
table[i] = pat_len;
}

for (register int i = 1; i < pat_len; ++i) {
if (table[(int) pattern[pat_len - i - 1]] != pat_len) {
continue;
}
else {
table[(int) pattern[pat_len - i - 1]] = i;
}
}
for (register int i = 0; i < str_len; ++i) {
int match = 0;
for (register int j = pat_len - 1; j >= 0; --j) {
if (str[i + j] != pattern[j] && !match) {
i += table[(int) str[i + j]] - 1;
break;
}
else if(str[i + j] != pattern[j] && match) {
i += table[(int) pattern[pat_len - 1]] - 1;
match = 0;
break;
}
else {
match = 1;
}
}
if (match) {
while (i != 0 && str[i - 1] != ' ') {
i--;
}
return i;
}
}
return -1;
}
```

```
int main() {
/* Указатель на область с отраженными данными. */
char *ptr;
/* Количество байт, которые отражаются в ОЗУ с помощью mmap. */
int length;
/* Сколько char нужно выделить под результат поиска. */
int num_of_digits;
/* Семафор. */
sem_t *semaphore1 = sem_open("/sem1", O_CREAT, 0666, 0);
sem_t *semaphore2 = sem_open("/sem2", O_CREAT, 0666, 0);
if (semaphore1 == SEM_FAILED || semaphore2 == SEM_FAILED) {
printf("Semaphores doesn't create\n");
exit(1);
}
sem_unlink("s1");
sem_unlink("s2");
/* Образец. */
string_type pattern;
/* Строка. */
string_type string;

/* Инициализация образца. */
string_init(&pattern);
/* Инициализация строки. */
string_init(&string);

/* Считывание образца. */
string_scan(&pattern);
/* Считывание строки. */
string_scan(&string);

/* Открывается файл, в который будут записываться образец и строка и
откуда будет они будут считываться. Также в этот файл занесется
результат поиска. */
int fd = open("mapped", O_RDWR | O_CREAT, 0666);
if (fd == -1) {
printf("File didn't open\n");
exit(1);
}

num_of_digits = number_of_digits((pattern.capacity + string.capacity)*
sizeof(char));

length = (pattern.capacity + string.capacity) * sizeof(char) +
num_of_digits;

/* Устанавливаем длину файла в length байт. */
if (ftruncate(fd, length) == -1) {
printf("ftruncate error\n");
exit(1);
}

/* Отражение файла в ОЗУ. */
ptr = (char *) mmap(NULL, length, PROT_WRITE | PROT_READ, MAP_SHARED,
fd, 0);
if (ptr == MAP_FAILED) {
printf("Memory mapping failed\n");
exit(1);
}

/* Создание дочернего процесса, который будет осуществлять поиск
образца в строке. */
```

```
int proc = fork();

if (proc == -1) {
printf("Can\'t create child process\n");
exit(1);
} else if (proc > 0) {
int i;
/* Запись в файл образца. */
for (i = 0; i < pattern.capacity; i++) {
write_to_memory(pattern.s[i], ptr, i);
}
/* Запись в файл строки. */
int old_i = i;
for (; i < string.capacity + old_i ; i++) {
write_to_memory(string.s[i - old_i], ptr, i);
}

// printf("parent ptr: %s; length ptr: %ld\n", ptr, strlen(ptr));

/* Ожидаем завершение дочернего процесса. */
sem_post(semaphore1);
sem_wait(semaphore2);
/* Вывод результата поиска. */
if (get_from_memory(ptr, (pattern.capacity + string.capacity) * sizeof(char)) == 'e') {
printf("No match found.\n");
} else {
printf("%d\n", get_int_from_memory(ptr,
(pattern.capacity + string.capacity) * sizeof(char)));
}
sem_close(semaphore1);
sem_close(semaphore2);
/* Удаление отражения из данной области. */
munmap(ptr, length);
/* Закрытие файла. */
close(fd);
} else if (proc == 0) {
string_type child_pattern, child_string;
string_init(&child_pattern);
string_init(&child_string);
/* Ожидаем передачи доступа */
sem_wait(semaphore1);

int i;
/* Чтение из файла образца. */
for (i = 0; i < pattern.capacity; i++) {
get_from_memory_for_string(ptr, i, &child_pattern);
}
/* Чтение из файла строки. */
int old_i = i;
for (; i < string.capacity + old_i ; i++) {
get_from_memory_for_string(ptr, i, &child_string);
}

/* Результат поиска образца в строке. */
int result = bmh(child_string.s, child_string.capacity,
child_pattern.s, child_pattern.capacity);
if (result != -1) {
/* Массив для представления результата поиска в виде строки. */
char res_char[num_of_digits + 1];
/* Перевод из числа в строку. */
my_itoa(result, res_char);
/* Запись результата в память. */
```

```
for (int h = 0; h < num_of_digits; h++) {
write_to_memory(res_char[h], ptr, (pattern.capacity + string.capacity) *
sizeof(char)+ h);
}
} else {
write_to_memory('e', ptr, (pattern.capacity + string.capacity) * sizeof(char));
}

sem_post(semaphore2);

sem_close(semaphore1);
sem_close(semaphore2);
close(fd);
}

return 0;
}
```

5. Демонстрация работы программы.

```
oem@Alex-PC:~/Documents/OS/lab4$ ./main
kol
lo kool kolokol
8
oem@Alex-PC:~/Documents/OS/lab4$ ./main
7 days
7 days mini kr
0
oem@Alex-PC:~/Documents/OS/lab4$ ./main


No match found.
oem@Alex-PC:~/Documents/OS/lab4$ ./main
f
gggggg
No match found.
```

6. Вывод.

Отображение файла в память позволяет всему файлу или некоторой его части поставить в соответствие определенный участок памяти. Чтение данных из этого участка памяти фактически приводит к чтению данных из отображаемого файла, а запись данных приводит к записи этих данных в файл.

Достоинством такого способа работы с файлами является меньшая по сравнению с чтением/записью нагрузка на операционную систему, поскольку при использовании отображений ОС не загружает в память сразу весь файл, а делает это по мере необходимости, блоками размером со страницу памяти (4 кб). Таким образом, даже имея небольшое количество физической памяти, можно легко отобразить файл большего размера.

В этой лабораторной я получил опыт работы с семафорами, которые хорошо подходят для синхронизации процессов, в основе которых лежит счетчик.