Московский авиационный институт
Национальный исследовательский университет

Операционные системы
Кафедра 806

Лабораторная работа №4

Студент: Минибаев Айдар
Группа: М8О-301Б-18
Преподаватель: Миронов Е.С.
Дата: .
Подпись: .

Москва 2020

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 решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляется с помощью отображения файла на память.

Написание собственного простого целочисленного калькулятора с операциями "+","-". В дочернем процессе должны происходить вычисления выражений. В родительском процессе ввод/вывод.

2. Метод решения

В программе изначально создается временный файл с помощью системного вызова mkstemp,который заполняется до определенного размера, после временный файл отображается в оперативную память, затем создается дочерний процесс. Родительский процесс принимает данные со стандартного потока ввода. Затем, если данные введены корректны, родительский процесс передает данные дочернему процессу через общедоступный им участок памяти, который мы получили, отобразив временный файл в оперативную память. Дочерний процесс производит вычисление и передает результат обратно в родительский процесс, после чего родительский процесс печатает результат на стандартный поток вывода и снова ждет ввода пользователя. Для синхронизации процессов используется примитив синхронизации семафор. Один семафор нужен, для того-чтобы родительский процесс оповещал дочерний, когда данные уже готовы для вычисления, а второй семафор нужен, для того-чтобы дочерний процесс оповещал родительский, когда вычисления уже произведены и доступны для родительского процесса.

3. Программная реализация

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <pthread.h>
#include <sys/mman.h>
#include <unistd.h>
#include <fcntl.h>
#include <semaphore.h>
#include <errno.h>
#include <signal.h>

typedef unsigned char uc_t;

typedef struct {
   int lhs;
   int rhs;
   char op;
} expr_t;

int expr_read(expr_t* expr) {
```

```
    return scanf("%d %c %d", &expr->lhs, &expr->op, &expr->rhs) == 3;
}

void expr_write(expr_t* expr) {
    printf("%d %c %d\n", expr->lhs, expr->op, expr->rhs);
}

int expr_sum(expr_t* expr) {
    return expr->lhs + expr->rhs;
}

int expr_sub(expr_t* expr) {
    return expr->lhs - expr->rhs;
}

void expr_share(expr_t* expr, void* memory, int start) {
    int* lhs = (int*)((uc_t*)(memory) + start);
    int* rhs = (int*)((uc_t*)(memory) + start + 5);
    char* op = (char*)((uc_t*)(memory) + start + 4);
    *lhs = expr->lhs;
    *rhs = expr->rhs;
    *op = expr->op;
}

void expr_get_from_mem(expr_t* expr, void* memory, int pos) {
    int* lhs = (int*)((uc_t *)(memory) + pos);
    int* rhs = (int*)((uc_t *)(memory) + pos + 5);
    char* op = (char*)((uc_t *)(memory) + pos + 4);
    expr->lhs = *lhs;
    expr->rhs = *rhs;
    expr->op = *op;
}

void expr_write_res(void* memory, int pos, int res) {
    int *place = (int*)((uc_t*)(memory) + pos);
    *place = res;
}

int expr_get_res(void* memory, int pos) {
    int* res = (int*)((uc_t*)(memory) + pos);
    return *res;
}

int get_temp_file(void) {
    char *tempFileName = strdup("/tmp/tmpFile.XXXXXX");
    int fd = mkstemp(tempFileName);
    unlink(tempFileName);
    free(tempFileName);
    write(fd, "\0\0\0\0\0\0\0\0\0\0\0\0\0", 13);
    return fd;
}

void* get_map_memory(int fd, int start, int size) {
    uc_t* mem = (uc_t*)mmap(NULL, size, PROT_WRITE | PROT_READ, MAP_SHARED, fd, start);
    if(mem == MAP_FAILED) {
        perror("Memmory mapping failed");
```

```
        exit(1);
    }
    return mem;
}

int main() {
    int fd = get_temp_file();
    uc_t* mem = get_map_memory(fd, 0, 13);
    sem_t* sem_calc = sem_open("/lab4", O_CREAT, 777, 0);
    sem_t* sem_output = sem_open("/output", O_CREAT, 777, 0);
    if(sem_calc == SEM_FAILED || sem_output == SEM_FAILED) {
        perror("Semaphores doesn't create");
        exit(1);
    }
    sem_unlink("/lab4");
    sem_unlink("/output");
    int err = fork();
    if(err == -1) {
        perror("Can't fork child");
        exit(1);
    } else if(err > 0) {
        expr_t expr;
        int res;
        int read_flag;
        while((read_flag = expr_read(&expr))) {
            if(expr.op != '+' && expr.op != '-') {
                printf("This operation doesn't support\n> ");
                continue ;
            }
            expr_share(&expr, mem, 0);
            sem_post(sem_calc);
            sem_wait(sem_output);
            res = expr_get_res(mem, 9);
            printf("%d\n", res);
        }
        mem[12] = 1;
        sem_post(sem_calc);
        sem_close(sem_calc);
        sem_close(sem_output);
        close(fd);
    } else {
        expr_t expr;
        int res;
        while(1) {
            sem_wait(sem_calc);
            if(mem[12] == 1)
                break;
            expr_get_from_mem(&expr, mem, 0);
            switch(expr.op) {
                case '+':
                    res = expr_sum(&expr);
                    break;
                case '-':
                    res = expr_sub(&expr);
                    break;
            }
```

```
            expr_write_res(mem, 9, res);
            sem_post(sem_output);
        }
        sem_close(sem_calc);
        sem_close(sem_output);
        munmap(mem, 13);
        close(fd);
    }
    return 0;
}
```

4. Тестирование

```
$ make
gcc lab4.c -o lab4.out -pthread -Wall -Wextra -Werror -pedantic
$ ./lab4.out
-1 + 1
0
1234 - 234
1000
10000000 +
 2345678
12345678
1024 + 1024
2048
```

5. Выводы

Отображение файлов дает удобство при работе с файлами, так как позволяет работать с областью файла как с обычным участком памяти. Другими словами, мы имеем доступ к каждому байту области памяти, которую мы отобразили и для этого не надо использовать lseek, также количество системных вызовов по чтению и записи сводится к нулю, так как мы работаем с оперативной памятью. Но также отображение файлов дает нам возможность в межпроцессорном взаимодействии. При отображении файла на участок памяти, этой память могут разделять несколько процессов, но в отличии от pipe, теперь синхронизация остается на разработчике.