Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №4 по курсу «Операционные системы»

Студент: Э. Л. Носов
Преподаватель: Е. С. Миронов
Группа: М8О-307Б
Дата:
Оценка:
Подпись:

Москва, 2021

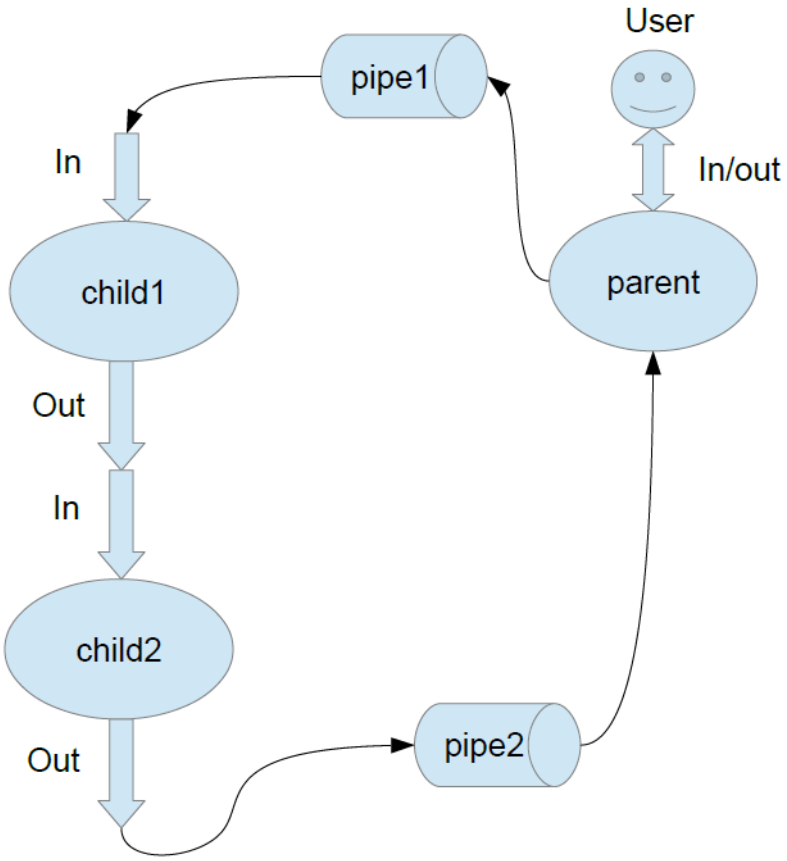
Лабораторная работа №4

Цель работы: Приобретение практических навыков в:

- Освоение принципов работы с файловыми системами
- Обеспечение обмена данных между процессами посредством технологии «File mapping»

Задание: 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 через системные сигналы/события и/или через отображаемые файлы (memory-mapped files). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Вариант №12: Child1 переводит строки в верхний регистр. Child2 убирает все задвоенные пробелы.



1 Описание

Родительский процесс создает два дочерних процесса. Перенаправление стандартных потоков ввода-вывода показано на картинке выше. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и записывает их в отображенный файл. Процесс child1 и child2 производят работу над строками. Child2 выводит полученный результат в стандартный поток вывода.

2 Исходный код

В main-файле при помощи функции shm_open() выделяется общий фрагмент памяти, затем функция ftruncate обрезает объем выделенной памяти до размера страницы, получаемого с помощью sysconf, после чего при помощи функции mmap() создается отображение на этот фрагмент памяти. При помощи семафоров из semaphore.h, которые хранятся все в той же выделенной памяти, контролируется доступ к общей памяти. При помощи функции fork() создается новый процесс - двойник старого процесса. Функция execl заменяет код данного процесса на другой код, выполняющий заданную функцию.

lab4.c

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


int get_line(char* string, int size){
    char c;
    int n=0;
    while(read(STDIN_FILENO, &c, 1)>0){
        if((c!='\n')&&(n<size)){
        string[n]=c;
        n++;
        }
        else {
            string[n]='\0';
            break;
        }
    }
    return n;
}

int main(){
    const int MAX_CHAR=256;
    char* mem_map_file = "memmap";
    int fd = shm_open(mem_map_file, O_CREAT | O_RDWR, S_IRWXU);
    if (fd < 0) {
        printf("can't open shared memmory segmet\n");
        return 0;
    }
    int page_size = sysconf(_SC_PAGE_SIZE);
    if(ftruncate(fd, page_size) == -1){
        perror("ftruncate fail\n");
        return 0;
    }
    const char *child1_name = "child1", *child2_name="child2";
    void* sh_file = mmap(0, page_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    close(fd);
    sem_t* parent = sh_file;
    sem_t* child1 = sh_file + sizeof(sem_t);
    sem_t* child2 = sh_file + 2 * sizeof(sem_t);
    sem_init(parent, 1, 1);
    sem_init(child1, 1, 0);
    sem_init(child2, 1, 0);
```

```
    char* string_shared = sh_file + sizeof(sem_t) * 3;
    string_shared[0] = 0;


    int id = fork();
    if(id == -1){
        printf("child1 fork error");
        return -1;
    }else if(id == 0 ){//child1
    if (execl(child1_name, child1_name, mem_map_file, NULL) == -1) {
      printf("Failed to exec\n");
      return -1;
    }
    }
    else{
        int di=fork();
        if(di == -1){
            printf("child2 fork error");
            return -1;
        }
        else if(di == 0){//child2
        if (execl(child2_name, child2_name, mem_map_file, NULL) == -1) {
          printf("Failed to exec\n");
          return -1;
        }
        }
        else{//parent
            char string[MAX_CHAR];
            int i=0;
            while(get_line(string, MAX_CHAR)>0){
                sem_wait(parent);
                i=0;
                while(string[i]!=0){
                    string_shared[i]=string[i];
                    i++;
                }
                string_shared[i]=string[i];
                sem_post(child1);
            }
            string_shared[0]=-1;
            sem_post(child1);
            sem_post(child2);
            sem_close(parent);
            sem_close(child1);
            sem_close(child2);
            shm_unlink(mem_map_file);
        }
    }
  return 0;
}
```

child1.c

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

```

```
11
12  int touppercase(char *string){
13      int i=0;
14      while(string[i]!=0){
15          if((string[i]>='a')&&(string[i]<='z')) string[i]+='A'-'a';
16          i++;
17      }
18  }
19
20  int main(int argc, char*argv[]){
21      char* mem_map_file = argv[1];
22      int fd = shm_open(mem_map_file, O_RDWR, S_IRWXU);
23      if (errno == EACCES) {
24          printf("EACCES\n");
25      }
26      int page_size = sysconf(_SC_PAGE_SIZE);
27      if (ftruncate(fd, page_size) == -1) {
28          perror("ftruncate failure\n");
29          return 0;
30      }
31      void* sh_file = mmap(0, page_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
32      sem_t* parent = sh_file;
33      sem_t* child1 = sh_file + sizeof(sem_t);
34      sem_t* child2 = sh_file + 2 * sizeof(sem_t);
35      char* string_shared = sh_file + sizeof(sem_t) * 3;
36      string_shared[0] = 0;
37
38
39      const int MAX_CHAR= 256;
40      char string[MAX_CHAR];
41      while(1){
42          sem_wait(child1);
43          if(string_shared[0] == -1){
44              break;
45          }
46          else if(string_shared[0]>0){
47              touppercase(string_shared);
48              printf("%s\n",string_shared);
49          }
50          sem_post(child2);
51      }
52      sem_close(parent);
53      sem_close(child2);
54      sem_close(child1);
55      return 0;
56  }
```

child2.c

```
1   #include <unistd.h>
2   #include <stdio.h>
3   #include <semaphore.h>
4   #include <pthread.h>
5   #include <string.h>
6   #include <sys/mman.h>
7   #include <sys/stat.h>
8   #include <fcntl.h>
9   #include <errno.h>
10  #include <stdlib.h>
11
12
13  void doublespaces(char *string, int length){
14      char *str;
```

```
    str=(char*)malloc(sizeof(char)*length);
    int j=1, i=1;
    str[0]=string[0];
    while(string[i]!=0){
        if((str[j-1]!=' ')&&(string[i]==' ')){
            str[j]=string[i];
            j++;
        }
        else if(string[i]!=' '){
            str[j]=string[i];
            j++;
        }
        i++;
    }
    for(int i=0; i<=j; i++){
        string[i]=str[i];
    }
    free(str);
}

int main(int argc, char*argv[]){
    char* mem_map_file = argv[1];
    int fd = shm_open(mem_map_file, O_RDWR, S_IRWXU);
    if (errno == EACCES) {
        printf("EACCES\n");
    }
    int page_size = sysconf(_SC_PAGE_SIZE);
    if (ftruncate(fd, page_size) == -1) {
        perror("ftruncate failure\n");
        return 0;
    }
    void* sh_file = mmap(0, page_size, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
    sem_t* parent = sh_file;
    sem_t* child1 = sh_file + sizeof(sem_t);
    sem_t* child2 = sh_file + 2 * sizeof(sem_t);
    char* string_shared = sh_file + sizeof(sem_t) * 3;
    string_shared[0] = 0;


    const int MAX_CHAR = 256;
    char string[MAX_CHAR];
    while(1){
        sem_wait(child2);
        if(string_shared[0] == -1){
            break;
        }
        else if(string_shared[0]>0){
            doublespaces(string_shared, MAX_CHAR);
            printf("%s\n",string_shared);
        }
        sem_post(parent);
    }
    sem_close(parent);
    sem_close(child1);
    sem_close(child2);
    return 0;
}
```

3 Консоль

```
maloletniydebil@LAPTOP-LNCHG0M3:/mnt/d/X-Files/MAI/3 sem/OS/labs/lab4$ ./lab4
text   text text
TEXT TEXT TEXT
text    tex text te   x t
TEXT TEX TEXT TE X T
text      text text text text                                                text
TEXT TEXT TEXT TEXT TEXT TEXT
```

4 Выводы

Выполнив лабораторную работу №4 по курсу «Операционные системы», я освоил взаимодействие процессов при помощи технологии "File-mapping"в операционной системе семейства Unix. В ходе работы я применил на практике системные вызовы shm_open и shm_unlink для управления общей памятью, а также mmap для реализации отображения на эту память, набор функций для инициализации семафоров и управления ими.