**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Операционные системы»

**Лабораторная работа № 4**

Тема: Файловые системы

Студент: Тимофеев А.В.

Группа: 80-207

Преподаватель: Миронов Е. С.

Дата:

Оценка:

**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 через системные сигналы/события и/или через отображаемые файлы (memory-mapped files).

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

ВАРИАНТ 21.

Родительский процесс создает два дочерних процесса. Первой строкой пользователь в консоль родительского процесса вводит имя файла, которое будет использовано для открытия File с таким именем на запись для child1. Аналогично для второй строки и процесса child2. Родительский и дочерний процесс должны быть представлены разными программами.

Родительский процесс принимает от пользователя строки произвольной длины и пересылает их в pipe1 или в pipe2 в зависимости от правила фильтрации. Процесс child1 и child2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Вариант 21) Правило фильтрации: нечетные строки отправляются в pipe1, четные в pipe2. Дочерние процессы инвертируют строки.

**Алгоритм решения**

Обмениваться данными между процессами будет происходить через отображаемые файлы (memory-mapped files). В родительском процессе создаем два файла, и устанавливаем их длину равной размеру страницы. Также родительский процесс считывает имена

двух файлов для записи результата. В дочернем процессе переопределяем его вывод в файл, название которого было получено в родительском процессе, вызываем exec и передаем в программу имя отображаемого файла. В это время родительский процесс отображает оба файла в память. Файлам был задан размер, равный размеру страницы, так как для получившегося массива выделяется память кратная этому размеру. Затем процесс начинает считывать строки. Передача строк в дочерние процессы происходит по описанному во 2 лабораторной алгоритму. Однако теперь запись происходит в соответствующий массив. Для синхронизации процессов используются сигналы. После того как вся строка записана, родительский процесс посылает в соответствующий дочерний сигнал SIGUSR1 и ждет в ответ такой же сигнал, чтобы продолжить чтение следующей строки. Если в массиве закончилось место, родительский процесс также отправляет и ждет сигнал, а затем продолжает запись строки с начала массива.Дочерний процесс открывает файл для отображения и отображает его в память. Затем ждет сигнал, считывает строку из массива и посылает сигнал родительскому процессу. Номер родительского процесса также передается в дочерний как аргумент. Об окончании строки сигнализирует символ перевода строки, об окончании строк вообще сигнализирует нулевой символ.

**Листинг программы**

**main.c**

#include "unistd.h"

#include "stdio.h"

#include "stdlib.h"

#include "sys/mman.h"

#include "string.h"

#include "signal.h"

#include "fcntl.h"

#include <sys/types.h>

#define check(VALUE, MSG, BADVAL) if (VALUE == BADVAL) { perror(MSG); exit(1); }

int main() {

size\_t pagesize = sysconf(\_SC\_PAGESIZE);

int pid = getpid();

char fn1[256];

char fn2[256];

if (scanf("%s", fn1) <= 0) {

perror("scanf error");

return -1;

}

if (scanf("%s", fn2) <= 0) {

perror("scanf error");

return -1;

}

getchar();

FILE\* file1 = fopen(fn1, "wt");

check(file1, "fopen 1 error", NULL)

FILE\* file2 = fopen(fn2, "wt");

check(file2, "fopen 2 error", NULL)

char mmfilename1[] = "mmfile1";

char mmfilename2[] = "mmfile2";

int mmf1 = open(mmfilename1, O\_RDWR | O\_CREAT, S\_IRWXU);

check(mmf1, "open 1 error", 0)

int mmf2 = open(mmfilename2, O\_RDWR | O\_CREAT, S\_IRWXU);

check(mmf2, "open 2 error", 0)

size\_t i1 = 0, i2 = 0;

ftruncate(mmf1, pagesize);

ftruncate(mmf2, pagesize);

int id1 = fork();// первый дочерний процесс

if (id1 == -1) {

perror("fork 1 error");

return -1;

}else if (id1 > 0){// родительский процесс

int id2 = fork();// второй дочерний процесс

if (id2 == -1) {

perror("fork 2 error");

return -1;

}else if (id2 > 0){// родительский процесс

fclose(file1);

fclose(file2);

char \*fmap1 = (char \*)mmap(NULL, pagesize, PROT\_WRITE | PROT\_READ, MAP\_SHARED, mmf1, 0);

check(fmap1, "mmap 1 error", MAP\_FAILED)

char \*fmap2 = (char \*)mmap(NULL, pagesize, PROT\_WRITE | PROT\_READ, MAP\_SHARED, mmf2, 0);

check(fmap2, "mmap 2 error", MAP\_FAILED)

sigset\_t set;

check(sigemptyset(&set), "sigemptyset error", -1)

check(sigaddset(&set, SIGUSR1), "sigaddset error", -1)

check(sigprocmask(SIG\_BLOCK, &set, NULL), "sigprocmask error", -1)

int sig;

char c;

int n = 1;

while (scanf("%c", &c) > 0){// заполняем pipes

if (c != '\n'){

if (n % 2 != 0){

fmap1[i1] = c;

if (++i1 == pagesize) {

i1 = 0;

check(kill(id1, SIGUSR1), "send signal to child.out 2 error", -1)

check(sigwait(&set, &sig), "sigwait error", -1)

}

}else{

fmap2[i2] = c;

if (++i2 == pagesize) {

i2 = 0;

check(kill(id2, SIGUSR1), "send signal to child.out 2 error", -1)

check(sigwait(&set, &sig), "sigwait error", -1)

}

}

}else{

if (n % 2 != 0){

fmap1[i1] = c;

i1 = 0;

check(kill(id1, SIGUSR1), "send signal to child.out 1 error", -1)

check(sigwait(&set, &sig), "sigwait error", -1)

}else{

fmap2[i2] = c;

i2 = 0;

check(kill(id2, SIGUSR1), "send signal to child.out 2 error", -1)

check(sigwait(&set, &sig), "sigwait error", -1)

}

n++;

}

}

c = '\0';

fmap1[0] = c;

fmap2[0] = c;

check(kill(id1, SIGUSR1), "send signal to child.out 1 error", -1)

check(kill(id2, SIGUSR1), "send signal to child.out 2 error", -1)

check(munmap(fmap1, pagesize), "munmap error", -1);

check(munmap(fmap2, pagesize), "munmap error", -1);

close(mmf1);

close(mmf2);

}else{// второй дочерний процесс

fclose(file1);

close(mmf1);

check(dup2(fileno(file2), fileno(stdout)), "dup2 error", -1)

fclose(file2);

char spid[10];

snprintf(spid, 10, "%d", pid);

char\* const args[] = {"child.out", mmfilename2, spid, (char \*)NULL};

check(execv("child.out", args), "execv child.out 2 error", -1)

}

}else{// первый дочерний процесс

fclose(file2);

close(mmf2);

check(dup2(fileno(file1), fileno(stdout)), "dup2 error", -1)

fclose(file1);

char spid[10];

snprintf(spid, 10, "%d", pid);

char\* const args[] = {"child.out", mmfilename1, spid, (char \*)NULL};

check(execv("child.out", args), "execv child.out 1 error", -1)

}

return 0;

}

**child.c**

#include "unistd.h"

#include "stdio.h"

#include "stdlib.h"

#include "sys/mman.h"

#include "string.h"

#include "signal.h"

#include "fcntl.h"

#define check(VALUE, MSG, BADVAL) if (VALUE == BADVAL) { perror(MSG); exit(-1); }

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

size\_t pagesize = sysconf(\_SC\_PAGESIZE);

char c = '\0';

int n = 0;

int cap = 256;

char\* str = (char \*)malloc(sizeof(char) \* pagesize);

check(str, "malloc error", NULL)

int mfile = open(argv[1], O\_RDWR);

check(mfile, "open error", -1)

int pid = atoi(argv[2]);

char\* fmap = (char \*)mmap(NULL, pagesize, PROT\_READ, MAP\_SHARED, mfile, 0);

check(fmap, "mmap error", MAP\_FAILED)

size\_t i = 0;

sigset\_t set;

check(sigemptyset(&set), "sigemptyset error", -1)

check(sigaddset(&set, SIGUSR1), "sigaddset error", -1)

check(sigprocmask(SIG\_BLOCK, &set, NULL), "sigprocmask error", -1)

int sig;

for(;;) {

check(sigwait(&set, &sig), "sigwait error", -1);

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

c = fmap[i];

if (c != '\n' && c != '\0') {

str[i] = c;

++n;

} else if (c == '\0') {

break;

} else {

for (int i = n-1; i >= 0; i--) {

printf("%c", str[i]);

}

printf("%c", c);

n = 0;

break;

}

}

if (c == '\0') {

break;

} else {

check(kill(pid, SIGUSR1), "send signal to parent error", -1)

}

}

free(str);

check(munmap(fmap, pagesize), "munmap error", -1)

close(mfile);

}

**Тесты и протокол исполнения**

dude@dude:~/Рабочий стол/oc/OC/mylaba4$ ./a.out

file

file1

file

file1

qwerty

jjjjjggggjjgjgjggggg

yiweuibwef44444

ujnguneanbndnfb''8558

74168411111111

122

nbjdnfbklpdfmblk

mn b

7411 dfg

mmbg 774

dude@dude:~/Рабочий стол/oc/OC/mylaba4$ cat file

elif

ytrewq

44444fewbiuewiy

11111111486147

klbmfdplkbfndjbn

gfd 1147

dude@dude:~/Рабочий стол/oc/OC/mylaba4$ cat file1

1elif

gggggjgjgjjggggjjjjj

8558''bfndnbnaenugnju

221

b nm

477 gbmm

**Вывод**:

В ходе лабораторной работы я познакомился с отображаемыми в память файлами, как со способом взаимодействия процессов и с управлением системными сигналами. Отображение файлов память позволяет удобнее работать с информацией в файле или модифицировать один файл из нескольких процессов. Также отображение в память используется для загрузки динамических библиотек. Для целей межпроцессорного взаимодействия можно использовать как обычные файлы, так и объекты разделяемой памяти. Я решил использовать обычные файлы фиксированного размера. Синхронизация процессов при помощи сигналов показалась мне удобнее, чем, например, при помощи мьютекса в разделяемой памяти. Однако такой способ менее безопасен, так как сигнал процессу может послать кто угодно.

**Список литературы:**

1. Таненбаум Э., Бос Х. Современные операционные системы. – 4-е изд. – СПб.: Издательский дом «Питер», 2018
2. Файлы, отображаемые в память

URL: https://habr.com/ru/post/55716/