Московский авиационный институт (национальный исследовательский университет)

# Факультет информационных технологий и прикладной математики

**Кафедра вычислительной математики и программирования**

## Лабораторная работа №4 по курсу «Операционные системы»

**Управление процессами в операционной системе «UNIX». Осуществление обмена информацией между процессами посредством общих сегментов памяти.**

Студент: Селивёрстов Д. С. Преподаватель: Миронов Е. С.

Группа: М8О-201Б-21

Вариант 8

Дата: Оценка: Подпись:

**Москва, 2023**

# Условие

Составить и отладить программу на языке Си, осуществляющую работу с процесса- ми и взаимодействие между ними в операционной системе UNIX/LINUX. В резуль- тате работы программа (основной процесс) должен создать для решение задачи один или несколько дочерних процессов. Взаимодействие между процессами осуществляет- ся через системные сигналы/события и/или каналы (*pipe*). Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

# Задание

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 *pipe*1 или в *pipe*2 в зависимости от правила фильтрации. Процесс *child*1 и *child*2 производят работу над строками. Процессы пишут результаты своей работы в стандартный вывод.

Правило фильтрации: с вероятностью 80% строки отправляются в pipe1, иначе в pipe2. Дочерние процессы удаляют все гласные из строк.

# Код программы

## utils.h

**#ifndef** UTILS\_LAB4

**#define** UTILS\_LAB4

**#include** <s t d i o . h> **#include** <s t d l i b . h> **#include** <s t r i n g . h>

**char** ∗ Read String ( FILE∗ stream ) ;

**int** ChoosePipe ( **char** ∗ s t r ) ;

## #endif parent.h

**#ifndef** PARENT\_H

**#define** PARENT\_H

**#include** <s t d i o . h> **#include** <s t d l i b . h> **#include** <s t r i n g . h> **#include** <unistd . h> **#include** <f c n t l . h> **#include** <errno . h> **#include** <sys / wait . h> **#include** <sys / s t a t . h> **#include** <sys / types . h> **#include** <sys /mman. h> **#include** <semaphore . h> **#include** <s td b o o l . h>

**void** Parent Routine ( FILE∗ input ) ;

## #endif

**0.0.1 utils.c**

**#include** " u t i l s . h"

**char** ∗ Read String ( FILE∗ stream )

{

**i f** ( f e o f ( stream )) {

**return** NULL;

}

**const int** chunk Size = 256 ;

**char** ∗ b u f f e r = ( **char** ∗) malloc ( chunk Size ) ;

**int** b u f f e r S i z e = chunk Size ;

**i f** ( b u f f e r == NULL)

{

p r i n t f ( " Couldn ’ t ␣ a l l o c a t e ␣ b u f f e r " ) ; e x i t (EXIT\_FAILURE) ;

}

**int** readChar ;

**int** idx = 0 ;

**while** (( readChar = getc ( stream )) != EOF)

{

b u f f e r [ idx++] = readChar ;

**i f** ( idx == b u f f e r S i z e )

{

b u f f e r = r e a l l o c ( b u f f e r , b u f f e r S i z e + chunk Size ) ; b u f f e r S i z e += chunk Size ;

}

**i f** ( readChar == ’ \n ’ ) {

## break ;

}

}

b u f f e r [ idx ] = ’ \0 ’ ;

**return** b u f f e r ;

}

**int** ChoosePipe ( **char** ∗ s t r )

{

**char** ∗ vowels = {"AEIOUYaeiouy" };

**int** vowelsCnt = 0 ;

**char** ∗ consonants = { "BCDFGHJKLMNPQRSTVWXYZbcdfghjklmnpqrstvwxyz"

};

**int** consonants Cnt = 0 ;

**for** ( **int** i = 0 ; i < ( **int** ) s t r l e n ( s t r ) ; ++i )

{

**int** is Vowel = 0 ;

**for** ( **int** j = 0 ; j < ( **int** ) s t r l e n ( vowels ) ; ++j ){

**i f** ( s t r [ i ] == vowels [ j ] )

{

++vowelsCnt ; is Vowel = 1 ; **break** ;

}

}

**i f** ( is Vowel ){

## continue ;

}

**for** ( **int** j = 0 ; j < ( **int** ) s t r l e n ( consonants ) ; ++j ){

**i f** ( s t r [ i ] == consonants [ j ] )

{

++c onsonants Cnt ;

## break ;

}

}

}

**return** vowelsCnt > consonants Cnt ;

}

## parent.c

**#include** " parent . h" **#include** " u t i l s . h" **#include** <semaphore . h>

**void** Parent Routine ( FILE∗ f i n )

{

**char** ∗ f i le Name 1 = Read String ( f i n ) ;

**char** ∗ f i le Name 2 = Read String ( f i n ) ;

f i le Name 1 [ s t r l e n ( f i le Name 1 ) − 1 ] = ’ \0 ’ ; f i le Name 2 [ s t r l e n ( f i le Name 2 ) − 1 ] = ’ \0 ’ ;

unlink ( f i le Name 1 ) ; unlink ( f i le Name 2 ) ;

pid\_t output File 1 , output File 2 ;

**i f** (( output File 1 = open ( f i le Name 1 , O\_WRONLY | O\_CREAT, S\_IRWXU) ) < 0 )

{

p e r r o r ( " opening ␣output ␣ f i l e ␣1 ␣ e r r o r ␣ ) " ) ; e x i t (EXIT\_FAILURE) ;

}

**i f** (( output File 2 = open ( f i le Name 2 , O\_WRONLY | O\_CREAT, S\_IRWXU) ) < 0 )

{

p e r r o r ( " opening ␣output ␣ f i l e ␣2 ␣ e r r o r ␣ ) " ) ; e x i t (EXIT\_FAILURE) ;

}

f r e e ( f i le Name 1 ) ; f r e e ( f i le Name 2 ) ;

**const int** mapSize = 128 ;

**void** ∗ map1 = mmap( 0 , mapSize , PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_AN

**void** ∗ map2 = mmap( 0 , mapSize , PROT\_READ | PROT\_WRITE, MAP\_SHARED | MAP\_AN

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| sem\_t∗ sem1 = sem\_open ( " semaphore1 " , | O\_CREAT, | S\_IRUSR | | | S\_IWUSR, | 0 ) ; |
| sem\_t∗ sem2 = sem\_open ( " semaphore2 " , | O\_CREAT, | S\_IRUSR | | | S\_IWUSR, | 0 ) ; |

**const char** ∗ vowels = {"AEIOUYaeiouy" };

pid\_t pid 1 = f o r k ( ) ; pid\_t pid 2 = 1 ;

**i f** ( pid 1 > 0 ){ pid 2 = f o r k ( ) ;

}

**i f** ( pid 1 < 0 | | pid 2 < 0 )

{

p e r r o r ( " Creating ␣ p r o c e s s ␣ e r r o r ␣ ) " ) ; e x i t (EXIT\_FAILURE) ;

}

**i f** ( pid 1 == 0 )

{

**i f** ( dup2 ( output File 1 , 1 ) < 0 )

{

p e r r o r ( "dup2 () ␣ e r r o r ␣ ) " ) ; e x i t (EXIT\_FAILURE) ;

}

**char** ∗ s t r = ( **char** ∗) malloc ( mapSize ) ;

sem\_wait ( sem1 ) ;

msync ( map1 , mapSize , MS\_SYNC) ; memcpy( s t r , map1 , mapSize ) ;

**for** ( **int** i = 0 ; i < ( **int** ) s t r l e n ( s t r ) ; ++i )

{

**int** is Vowel = 0 ;

**for** ( **int** j = 0 ; j < ( **int** ) s t r l e n ( vowels ) ; ++j ){

**i f** ( s t r [ i ] == vowels [ j ] )

{

is Vowel = 1 ;

## break ;

}

}

**i f** ( is Vowel == 0 ) {

p r i n t f ( "%c " , s t r [ i ] ) ;

}

}

}

**else i f** ( pid 2 == 0 )

{

**i f** ( dup2 ( output File 2 , 1 ) < 0 )

{

p e r r o r ( "dup2 () ␣ e r r o r ␣ ) " ) ; e x i t (EXIT\_FAILURE) ;

}

**char** ∗ s t r = ( **char** ∗) malloc ( mapSize ) ;

sem\_wait ( sem2 ) ;

msync ( map1 , mapSize , MS\_SYNC | MS\_INVALIDATE) ; memcpy( s t r , map2 , mapSize ) ;

**for** ( **int** i = 0 ; i < ( **int** ) s t r l e n ( s t r ) ; ++i )

{

**int** is Vowel = 0 ;

**for** ( **int** j = 0 ; j < ( **int** ) s t r l e n ( vowels ) ; ++j ){

**i f** ( s t r [ i ] == vowels [ j ] )

{

is Vowel = 1 ;

## break ;

}

}

**i f** ( is Vowel == 0 ) {

p r i n t f ( "%c " , s t r [ i ] ) ;

}

}

}

## else

{

**char** ∗ s t r In p u t = NULL;

**const int** chunk Size = 1 6 ;

**int** b u f f e r S i z e 1 = chunk Size ;

**int** f r e e S p a c e 1 = b u f f e r S i z e 1 ;

**char** ∗ b u f f e r 1 = ( **char** ∗) malloc ( chunk Size ) ;

**int** b u f f e r S i z e 2 = chunk Size ;

**int** f r e e S p a c e 2 = b u f f e r S i z e 2 ;

**char** ∗ b u f f e r 2 = ( **char** ∗) malloc ( chunk Size ) ;

**while** (( s t r In p u t = Read String ( f i n )) != NULL)

{

**i f** ( ChoosePipe ( s t r In p u t ))

{

**while** ( f r e e S p a c e 1 < ( **int** ) s t r l e n ( s t r In p u t ))

{

b u f f e r 1 = ( **char** ∗) r e a l l o c ( buff er 1 , b u f f e r S i z e 1 + chunk Size

b u f f e r S i z e 1 += chunk Size ; f r e e S p a c e 1 += chunk Size ;

}

}

## else

{

s t r c a t ( buff er 1 , s t r In p u t ) ;

f r e e S p a c e 1 −= s t r l e n ( s t r In p u t ) ;

f r e e ( s t r In p u t ) ;

**while** ( f r e e S p a c e 2 < ( **int** ) s t r l e n ( s t r In p u t ))

{

b u f f e r 2 = ( **char** ∗) r e a l l o c ( buff er 2 , b u f f e r S i z e 2 + chunk Size

b u f f e r S i z e 2 += chunk Size ; f r e e S p a c e 2 += chunk Size ;

}

s t r c a t ( buff er 2 , s t r In p u t ) ;

f r e e S p a c e 2 −= s t r l e n ( s t r In p u t ) ;

f r e e ( s t r In p u t ) ;

}

}

memcpy( map1 , buf f e r 1 , s t r l e n ( b u f f e r 1 ) ) ;

msync ( map1 , mapSize , MS\_SYNC | MS\_INVALIDATE) ; sem\_post ( sem1 ) ;

memcpy( map2 , buf f e r 2 , s t r l e n ( b u f f e r 2 ) ) ;

msync ( map2 , mapSize , MS\_SYNC | MS\_INVALIDATE) ; sem\_post ( sem2 ) ;

wait(& pid 1 ) ; wait(& pid 2 ) ;

}

munmap( map1 , mapSize ) ; munmap( map2 , mapSize ) ;

sem\_close ( sem1 ) ; sem\_close ( sem2 ) ;

sem\_unlink ( " semaphore1 " ) ; sem\_unlink ( " semaphore2 " ) ;

}

# Выводы

В процессе выполнения лабораторной работы были приобретены навыки реализации обмена информацией с помощью файлов, находящихся в общей памяти процессов.