Московский Авиационный Институт

(Национальный Исследовательский Университет)

Кафедра 806 «Вычислительная информатика и программирование»

Факультет: «Информационные технологии и прикладная математика»

Лабораторная работа

Дисциплина: «Операционные системы»

I I семестр

Задание 2

|  |  |
| --- | --- |
| Группа: | М8О-208Б-18, №9 |
| Студент: | Игитова Александра Андреевна |
| Преподаватель: | Миронов Евгений Сергеевич |
| Оценка: |  |
| Дата: | 01.11.2019 |

Москва, 2019

1. **Задание**

(*вариант № 17*): 17. Дочерний процесс представляет собой сервер по работе с очередями и принимает команды со стороны родительского процесса.

1. **Код программы на С++**

Lab1.c

#include <stdbool.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include "queue.h"

void writeCommands(const int ostream);

int checkCommand(const char \*a);

bool findSubString(const char \*where, const char \*what);

void itoa(int n, char \*s);

void reverse(char \*s);

char \*slice(const char \*a, const int begin, const int end);

int main(){

const int READ = 0;

const int WRITE = 1;

int toChild[2];

int toParent[2];

int error1 = pipe(toChild);

int error2 = pipe(toParent);

if(error1 == -1 || error2 == -1){

printf("Can not create pipe(s). Exiting with status \n");

exit(-1);

}

pid\_t sp = fork();

char arr[101];

if(sp < 0){

printf("Не может создать новый процесс. \n");

exit(-1);

}else if(sp > 0){

//parent

writeCommands(STDOUT\_FILENO);

while(true){

write(STDOUT\_FILENO, "> ", 2);

int len = read(STDIN\_FILENO, arr, 100);

arr[len - 1] = '\0';

write(toChild[WRITE], arr, strlen(arr) + 1);

len = read(toParent[READ], arr, 100);

if(len > 1) {

printf("%s\n", arr);

} else {

break;

}

}

}else{

//child

queue \*q = create();

while(true){

int len = read(toChild[READ], arr, 100);

int x = checkCommand(arr);

if(x == -1){

write(toParent[WRITE], "Ошибка: некорректная команда.\n", 30);

}else{

switch(x){

case 0:

write(toParent[WRITE], "q", 1);

break;

case 1:

show(q, toParent[WRITE]);

break;

case 2:

if(length(q) > 0){

write(toParent[WRITE], first(q), strlen(first(q)));

write(toParent[WRITE], "\n\0", 2);

}else{

write(toParent[WRITE], "\0\0", 2);

}

break;

case 3:

if(length(q) > 0){

write(toParent[WRITE], last(q), strlen(last(q)));

write(toParent[WRITE], "\n\0", 2);

}else{

write(toParent[WRITE], "\0\0", 2);

}

break;

case 4:

push(q, slice(arr, 5, strlen(arr)-1));

write(toParent[WRITE], "\0\0", 2);

break;

case 5:

pop(q);

write(toParent[WRITE], "\0\0", 2);

break;

case 6:{

char s[100];

itoa(length(q), s);

write(toParent[WRITE], s, strlen(s));

write(toParent[WRITE], "\n\0", 2);

break;

}

case 7:

if(isEmpty(q)){

write(toParent[WRITE], "true\n", 5);

}else{

write(toParent[WRITE], "false\n", 6);

}

break;

}

}

}

}

close(toChild[READ]);

close(toChild[WRITE]);

close(toParent[READ]);

close(toParent[WRITE]);

}

void writeCommands(const int ostream){

write(ostream, "Welcome to the UI!\n", 19);

write(ostream, "List of the commands:\n", 22);

write(ostream, "0. exit - exit from program\n", 28);

write(ostream, "1. show - show the queue\n", 25);

write(ostream, "2. first - top of the queue\n", 28);

write(ostream, "3. last - back of the queue\n", 28);

write(ostream, "4. push VALUE - push the VALUE to the queue\n", 44);

write(ostream, "5. pop - pop the first element from the queue\n", 46);

write(ostream, "6. len - length of the queue\n", 29);

write(ostream, "7. empty - is the queue empty?\n", 31);

write(ostream,"\n-----\nAttention! The limit of input string is 100 symbols!\n-----\n", 66);

}

bool findSubString(const char \*where, const char \*what){

const int whereSize = strlen(where);

const int whatSize = strlen(what);

if(whereSize < whatSize){

return false;

}else if(whereSize == whatSize){

for(int i = 0; i < whereSize; i++){

if(what[i] != where[i]){

return false;

}

}

return true;

}else{

for(int i = 0; i <= whatSize; i++){

if(what[i] == '\0' && (where[i] == ' ' || where[i] == '\t')){

return true;

}

if(what[i] != where[i]){

return false;

}

}

}

}

int checkCommand(const char \*a){

char \*commands[] = {"exit", "show", "first", "last", "push", "pop", "len", "empty"};

for(int i = 0; i <= 7; i++){

if(findSubString(a, commands[i]) == true){

return i;

}

}

return -1;

}

void itoa(int n, char \*s){

int i, sign;

if((sign = n) < 0){

n = -n;

}

i = 0;

do{

s[i++] = n % 10 + '0';

}while ((n /= 10) > 0);

if(sign < 0){

s[i++] = '-';

}

s[i] = '\0';

reverse(s);

}

void reverse(char \*s){

int i, j;

char c;

for(i = 0, j = strlen(s)-1; i < j; i++, j--){

c = s[i];

s[i] = s[j];

s[j] = c;

}

}

char \*slice(const char \*a, const int begin, const int end){ //[begin, end]

if(strlen(a) <= begin){

return NULL;

}

char \*result = malloc( sizeof(char) \* (end-begin+2) );

int i;

for(i = begin; i <= end; i++){

result[i-begin] = a[i];

}

result[i] = '\0';

return result;

}

Queue.c

#include "queue.h"

queue \*create(){

queue \*q = malloc(sizeof(queue));

q->size = 0;

q->first = q->last = NULL;

return q;

}

bool isEmpty(const queue \*q){

return q->size == 0;

}

char \*pop(queue \*q){

if(isEmpty(q)){

return 0;

}

char \*ans = malloc( sizeof(char) \* (strlen(q->first->value)+1) );

strcpy(ans, q->first->value);

if(q->size == 1){

free(q->first);

q->first = NULL;

}else{

q->first = q->first->next;

q->first->prev->next = NULL;

free(q->first->prev);

q->first->prev = NULL;

}

q->size--;

return ans;

}

void push(queue \*q, char \*data){

if(data == NULL){

return;

}

if(isEmpty(q)){

q->first = malloc(sizeof(list));

q->first->next = NULL;

q->first->prev = NULL;

q->last = q->first;

q->first->value = data;

}else{

q->last->next = malloc(sizeof(list));

q->last->next->prev = q->last;

q->last->next->next = NULL;

q->last->next->value = data;

q->last = q->last->next;

}

q->size++;

}

int length(const queue \*q){

return q->size;

}

void show(const queue \*q, const int ostream){

list \*temp = q->first;

write(ostream, "queue: ", 7);

while(temp != NULL){

write(ostream, temp->value, strlen(temp->value));

write(ostream, " | ", 3);

temp = temp->next;

}

write(ostream, "\n\0", 2);

}

char \*first(const queue \*q){

if(isEmpty(q)){

return "\0";

}

return q->first->value;

}

char \*last(const queue \*q){

if(isEmpty(q)){

return "\0";

}

return q->last->value;

}

Queue.h

#ifndef QUEUE\_H

#define QUEUE\_H

#include <stdbool.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

typedef struct queue queue;

typedef struct list list;

struct queue{

list \*first;

list \*last;

int size;

};

struct list{

list \*next;

list \*prev;

char \*value;

};

queue \*create();

char \*first(const queue \*q);

char \*last(const queue \*q);

void push(queue \*q, char \*data);

char \*pop(queue \*q);

bool isEmpty(const queue \*q);

int length(const queue \*q);

void show(const queue \*q, const int ostream);

#endif

1. **Вывод**

Я изучила некоторые основы Операционных систем. Разобрала подробнее команды fork и pipe. Узнала подробности внутреннего устройства файлов, их идентификаторов и т.п. Научилась лучше работать с очередями. Работа выполнена благодаря моим умным друзьям, которые через пот и слезы пытались вбить в мою голову всю теорию. Спасибо им, я стала немного умнее.