**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №7

по дисциплине

«Операционные системы»

Вариант 9

Студент

Группа АИ-20-1 \_\_\_\_\_\_\_\_\_\_ Глубоков Г.В.

Подпись, дата

Руководитель

к.т.н.,доц. Батищев Р. В.

Подпись, дата

Липецк 2022г.

# Задание кафедры:

Реализовать программу-сервер и группу программ-клиентов, взаимодействующих с сервером с помощью механизма разделяемой памяти. Каждая программа-клиент обращается к серверу для реализации запроса на выполнение функции, заполняя поля структуры запроса в блоке разделяемой памяти. Сервер обрабатывает запрос, выдает результат, помещая его в поле результата структуры запроса в разделяемой памяти. Если сервер занят выполнением запроса и очередь клиентов на обслуживание не заполнена, клиент ставится в конец очереди. Клиент «уходит» (заканчивает работу), если обнаруживает, что очередь заполнена (функцию и максимальное количество клиентов в очереди N следует взять из табл. 6 прил.).

|  |  |  |
| --- | --- | --- |
| 9 | Поиск последнего вхождения подстроки s в строку s1 | 5 |

# Теоретические сведения:

# Для работы с разделяемлй памятью предусмотрены следующие функции:

Области совместно используемой памяти (System V) прикрепляются с помощью вызова shmat() и открепляются вызовом shmdt(); Для явного запроса областей совместно используемой памяти с другими процессами процессы могут задействовать системные вызовы shmget() и mmap().

Ход работы:Код программы на C++:

#include <iostream>

#include <pthread.h>

#include <string>

#include <sys/types.h>

#include <sys/wait.h>

#include <sys/mman.h>

#include <unistd.h>

#include <semaphore.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <unistd.h>

#define QUEUE\_SIZE 5

#define COUNT\_CLIENTS 7

#define FILE\_NAME "shared\_memory"

#define LINE\_SIZE 12

using namespace std;

sem\_t sQueue;

sem\_t sHandleQuery;

sem\_t sReadResponse;

pthread\_mutex\_t mAccess = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_mutex\_t mComeInToQueue = PTHREAD\_MUTEX\_INITIALIZER;

pthread\_mutex\_t mStdoutAccess = PTHREAD\_MUTEX\_INITIALIZER;

struct Query{

char line[LINE\_SIZE]={0};

char subline[LINE\_SIZE]={0};

int response=0;

Query(char line[LINE\_SIZE],char subline[LINE\_SIZE],int response){

for(int i=0;i < LINE\_SIZE ; i++){

this->line[i] = line[i];

this->subline[i] = subline[i];

}

this->response = response;

}

};

int

getSharedBlock(std::string fileName){

key\_t key;

key = ftok(fileName.c\_str(),0);

return shmget(key,1\*sizeof(Query),IPC\_CREAT | 0666);

}

void \*

attachMemoryBlock(int blockId){

if(blockId==-1){

return NULL;

}

void\* result;

result=(void\*)shmat(blockId,NULL,0);

if(result == (void\*)-1){

return NULL;

}

return result;

}

void

detachMemoryBlock(void\* block){

shmdt(block);

}

void

destroyMemoryBlock(int blockId){

shmctl(blockId,IPC\_RMID,NULL);

}

int

performQueryToServer(std::string line, std::string subline){

return 1;

}

int

indexLastSubline(std::string line,std::string subline){

int lastIndex=-1;

int isSub=true;

for(int i=0; i < line.length() ; i++){

for(int j=0; j < subline.length(); j++){

if(i+j >= line.length()){

isSub=false;

break;

}

if(line[i+j]!=subline[j]){

isSub=false;

break;

}

}

if(isSub){

lastIndex=i;

}

isSub=true;

}

return lastIndex;

}

static void \*

serverThread(void\* something){

int i=0;

while(i<5){

sem\_wait(&sHandleQuery);

system("sleep 0");

int blockId;

Query\* shm;

blockId = getSharedBlock(FILE\_NAME);

shm = (Query\*) attachMemoryBlock(blockId);

shm->response = indexLastSubline(shm->line, shm->subline);

detachMemoryBlock(shm);

sem\_post(&sReadResponse);

i++;

}

pthread\_exit(NULL);

}

static void \*

clientThread(void\* arg){

int countFreePlaces;

Query\* passedQuery=(Query\*) arg;

sem\_getvalue(&sQueue,&countFreePlaces);

pthread\_mutex\_lock(&mComeInToQueue);

if(!countFreePlaces){

pthread\_mutex\_lock(&mStdoutAccess);

std::cout <<"CLIENT{"<< std::endl;

std::cout <<" line: " << passedQuery->line << std::endl;

std::cout <<" subline: " << passedQuery->subline << std::endl;

std::cout <<" response: error: the queue is full" << std::endl;

std::cout <<"}"<< std::endl << std::endl;

pthread\_mutex\_unlock(&mStdoutAccess);

pthread\_mutex\_unlock(&mComeInToQueue);

pthread\_exit(NULL);

} else{

sem\_wait(&sQueue);

pthread\_mutex\_unlock(&mComeInToQueue);

}

pthread\_mutex\_lock(&mAccess);

int blockId;

Query\* shm;

blockId = getSharedBlock(FILE\_NAME);

shm = (Query\*) attachMemoryBlock(blockId);

for(int i = 0 ; i < LINE\_SIZE ; i++){

shm->line[i]=passedQuery->line[i];

shm->subline[i]=passedQuery->subline[i];

}

sem\_post(&sHandleQuery);

sem\_wait(&sReadResponse);

pthread\_mutex\_lock(&mStdoutAccess);

std::cout <<"CLIENT{";

std::cout <<" line: " << shm->line;

std::cout <<" subline: " << shm->subline;

std::cout <<" response: " << shm->response;

std::cout <<"}"<< std::endl;

pthread\_mutex\_unlock(&mStdoutAccess);

detachMemoryBlock(shm);

destroyMemoryBlock(blockId);

pthread\_mutex\_unlock(&mAccess);

pthread\_exit(NULL);

}

int

main() {

pthread\_t serverPid;

pthread\_t clientPids[COUNT\_CLIENTS];

Query queryes[7]={

{"i love c++","love",0},

{"fbgdebd","b",0},

{"","gdhb",0},

{"ttty jjf ssss","s",0},

{"aa cc aacc","cc",0},

{"v","v",0},

{"ttttt","tt",0} };

sem\_init(&sQueue,0,QUEUE\_SIZE);

sem\_init(&sHandleQuery,0,0);

sem\_init(&sReadResponse,0,0);

pthread\_create(&serverPid,NULL,&serverThread,NULL);

for(int i=0 ; i < COUNT\_CLIENTS ; i++){

pthread\_create(&clientPids[i],NULL,&clientThread,(void\*)&queryes[i]);

}

for(int i =0 ; i < COUNT\_CLIENTS ; i++){

pthread\_join(clientPids[i],NULL);

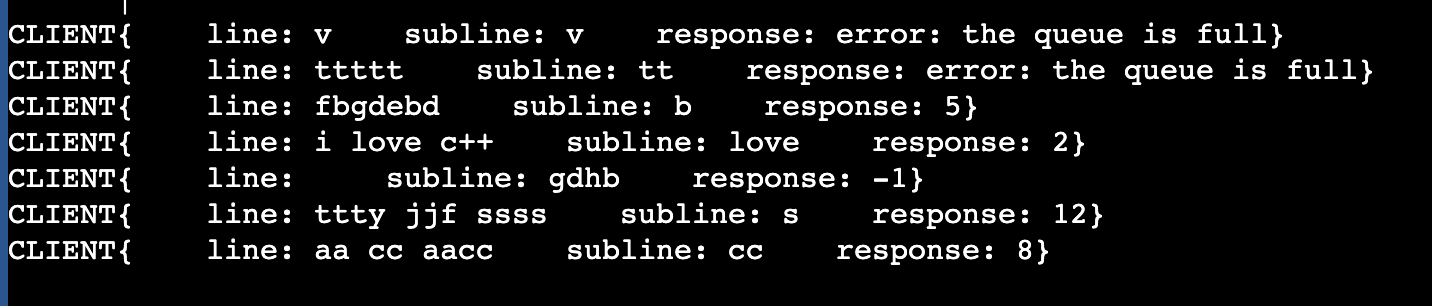
}

pthread\_join(serverPid,NULL);

sem\_destroy(&sQueue);

sem\_destroy(&sHandleQuery);

sem\_destroy(&sReadResponse);

}

В ходе выполнения лабораторной работы были получены основные навыки работы с разделяемой памятью Linux API