МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский ядерный университет «МИФИ»

(НИЯУ МИФИ)

Институт интеллектуальных кибернетических систем

Кафедра Кибернетики

Лабораторная работа №3 по курсу «Разработка ПО ОС UNIX»

Выполнил студент группы Б15-501: Огнянович Павел

Проверил: Ктитров С.В.

Москва, 2018

Задание

Разработать программу для Solaris, реализующую центральную доску объявлений. С помощью первой программы в разделяемой памяти публикуется объявление, а программы-клиенты отображают его, причем при изменении клиенты обновляют текст. Программа должна собираться из нескольких файлов с использованием make.

Код программы

Lab-3-server.c

```
#include <stdlib.h>

#include <stdio.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/types.h>

#include <sys/mman.h>

#include <pthread.h>

#include <string.h>


#define SH_MEM_FILE_NAME "lab-3-server.c"

#define SH_MEM_SIZE 1120

#define PROJ_ID 12

#define BUFF_SIZE 1024


int main(int argc, char *argv[]) {

        int smesch = sizeof(pthread_mutex_t)

                        +sizeof(pthread_cond_t)

                        +sizeof(int)

                        +sizeof(int);

        pthread_mutex_t mutex;

                pthread_mutex_init(&mutex, NULL);

        pthread_cond_t cond;

                pthread_cond_init(&cond, NULL);
```

```
	int has_new = 0;				//flag for new messages
	int message_size = 0;			//current size of message
	int message_number = -1;		//number of message
	char buffer[BUFF_SIZE];				//buffer for message
	char *sh_mem;				//pointer to shared memory
	int shmid;
	key_t key;
//	generating key
	//printf(">>>key = %d\n", key);
	if((key = ftok(SH_MEM_FILE_NAME, 12)) < 0) {
		//printf(">>>key = %d\n",key);
		printf(">>>FTOK_ERROR!\n");
		return 1;
	}
//	creating shared memory
	if((shmid = shmget(key, SH_MEM_SIZE*sizeof(char), 0666 | IPC_CREAT)) < 0) {
		//printf(">>>shmid = %d\n", shmid);
		printf(">>>SHMGET_ERROR!\n");
		return 1;
	}
//	getting pointer to shared memory
	if((sh_mem = (int*)shmat(shmid, NULL, 0)) == (int*)(-1)) {
		//printf(">>>sh_mem = %d\n", sh_mem);
		printf(">>>SHMAT_ERROR!\n");
		return 1;
	}
//TEST_OF_SH_MEM #1
//			int test = 100500;
//			memcpy(sh_mem, &test, sizeof(int));
//			int res = 0;
//			memcpy(&res, sh_mem, sizeof(int));
//			printf(">>>test = %d\n>>>res = %d\n", test, res);
```

```
//	copying mutex to shared memory
	printf(">>>COPYING_<MUTEX>_TO: sh_mem[0]\n");
	memcpy(sh_mem, &mutex, sizeof(pthread_mutex_t));
//	copying cond to shared memory
	printf(">>>COPYING_<COND>_TO: sh_mem[%ld]\n", sizeof(pthread_mutex_t));
	memcpy(sh_mem+sizeof(pthread_mutex_t), &cond, sizeof(pthread_cond_t));
//	copying message_size to shared memory
	printf(">>>COPYING_<MESSAGE_SIZE>_TO: sh_mem[%ld]\n",
sizeof(pthread_mutex_t)+sizeof(pthread_cond_t));
	memcpy(sh_mem+sizeof(pthread_mutex_t)+sizeof(pthread_cond_t), &message_size,
sizeof(int));
//	copying message_number to shared memory
	printf(">>>COPYING_<MESSAGE_NUMBER>_TO: sh_mem[%ld]\n", sizeof(pthread_mutex_t));
	memcpy(sh_mem+sizeof(pthread_mutex_t)+sizeof(pthread_cond_t)+sizeof(int),
&message_number, sizeof(pthread_cond_t));


//TEST_OF_SH_MEM #2
//			char *messag = "hello world!";
//			memcpy(sh_mem+smesch, messag, 12*sizeof(char));
//			char *res;
//			memcpy(res, sh_mem+smesch, 12*sizeof(char));
//			printf(">>>RESULT:\n>>>");
//			for(int i=0;i<12;i++)
//				printf("%c",res[i]);
//			printf("\n");


	for(;;) {
		printf(">>>1 - new message;\n>>>2 - exit\n");
		char symb;
		switch(symb = getchar()) {
			case '1':
```

```
                        getchar();
                        pthread_mutex_lock((pthread_mutex_t*)sh_mem);
                        //-------------CREATE_MESSAGE-------
                        message_size = 0;
                        message_number++;
                        printf(">>>MESSAGE_NUMBER: %d\n", message_number);
                        printf(">>>ENTER_MESSAGE:\n");
//                      memset(buffer, 0, BUFF_SIZE*sizeof(char));
//                      for(int i=0;i<BUFF_SIZE;i++) {
//                          message_size++;
//                          if((buffer[i]=getchar())=='\n')
//                              break;
//                      }
                        create_message(buffer, (int)BUFF_SIZE, &message_size);
                        printf(">>>MESSAGE_SIZE: %d\n", message_size);
                        //---------------------------------
                        //copying message
                        memcpy(sh_mem+smesch,
                            buffer,
                            BUFF_SIZE*sizeof(char));
                        //copying message_size
                        memcpy(sh_mem+sizeof(pthread_mutex_t)+sizeof(pthread_cond_t),
                            &message_size,
                            sizeof(int));
                        //copying message_number

            memcpy(sh_mem+sizeof(pthread_mutex_t)+sizeof(pthread_cond_t)+sizeof(int),
                            &message_number,
                            sizeof(int));

            pthread_cond_broadcast((pthread_cond_t*)(sh_mem+sizeof(pthread_mutex_t)));
                        pthread_mutex_unlock((pthread_mutex_t*)sh_mem);
                        break;
                case '2':
                        pthread_mutex_lock((pthread_mutex_t*)sh_mem);
                        message_number = -1;
                        //setting buffer in sh_mem to 0
                        memset(sh_mem+smesch, 0, BUFF_SIZE*sizeof(char));
                        //setting message_size to 0
                        memset(sh_mem+sizeof(pthread_mutex_t)+sizeof(pthread_cond_t), 0,
sizeof(int));

        memcpy(sh_mem+sizeof(pthread_mutex_t)+sizeof(pthread_cond_t)+sizeof(int),
                            &message_number,
                            sizeof(int));

        pthread_cond_broadcast((pthread_cond_t*)(sh_mem+sizeof(pthread_mutex_t)));
                        pthread_mutex_unlock((pthread_mutex_t*)sh_mem);
                        if(shmdt(sh_mem)<0) {
                            printf(">>>SHMDT_ERROR!\n");
                            return 1;
                        }
                        return 0;
                        break;
                default:
                        printf(">>>UNKNOWN_COMMAND!\n");
                        if(shmdt(sh_mem)<0)
                            printf(">>>SHMDT_ERROR!\n");
                        return 1;
        }
        }
        if(shmdt(sh_mem)<0) {
        printf(">>>SHMDT_ERROR!\n");
        return 1;
        }
        return 0;
```

```
}
```

Lab-3-client.c

```
#include <stdlib.h>
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <pthread.h>
#include <string.h>

#define SH_MEM_FILE_NAME "lab-3-server.c"
#define SH_MEM_SIZE 1120
#define PROJ_ID 12
#define BUFF_SIZE 1024

int main(int argc, char *argv[]) {
	int has_new = 0;
	int message_size = 0;
	int message_number = -1;
	int message_number_old = -1;
	char *sh_mem;
	int shmid;
	key_t key;
//	generating key
//	printf(">>>key = %d\n", key);
	if((key = ftok(SH_MEM_FILE_NAME, 12)) < 0) {
//		printf(">>>key = %d\n",key);
		printf(">>>FTOK_ERROR!\n");
		return 1;
	}
//	creating shared memory
	if((shmid = shmget(key, SH_MEM_SIZE*sizeof(char), 0666)) < 0) {
//		printf(">>>shmid = %d\n", shmid);
		printf(">>>SHMGET_ERROR!\n");
		return 1;
	}
//	getting pointer to shared memory
	if((sh_mem = (int*)shmat(shmid, NULL, 0)) == (int*)(-1)) {
//		printf(">>>sh_mem = %d\n", sh_mem);
		printf(">>>SHMAT_ERROR!\n");
		return 1;
	}
	for(;;) {
		pthread_mutex_lock((pthread_mutex_t*)sh_mem);
		memcpy(&message_number, sh_mem+92, sizeof(int));
		while(message_number == sh_mem[92])
			pthread_cond_wait((pthread_cond_t*)(sh_mem+sizeof(pthread_mutex_t)),
(pthread_mutex_t*)sh_mem);
		memcpy(&message_number,
			sh_mem+sizeof(pthread_mutex_t)+sizeof(pthread_cond_t)+sizeof(int),
			sizeof(int));
		if(message_number == -1) {
			printf(">>>SERVER_DISCONNECTED!\n");
			printf(">>>MESSAGE_NUMBER: -1\n");

		} else {
			printf(">>>GETTING_MESSAGE:\n");
			printf(">>>MESSAGE_NUMBER: %d\n", message_number);
			memcpy(&message_size,
				sh_mem+sizeof(pthread_mutex_t)+sizeof(pthread_cond_t),
				sizeof(int));
			printf(">>>MESSAGE_SIZE = %d\n", message_size);
```

```
                        for(int i=0;i<message_size;i++) {
                                printf("%c", (sh_mem+96)[i]);
                        }
                }
                pthread_mutex_unlock((pthread_mutex_t*)sh_mem);
        }
        return 0;
}
```

function.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void create_message(char *buff, int buffer_size, int *msg_size) {
        memset(buff, 0, buffer_size*sizeof(char));
        //printf(">%d\n", buffer_size);
        for(int i=0;i<buffer_size;i++) {
                msg_size[0]++;
                //printf("CR_M>>>msg_size=%d\n", msg_size[0]);
                if((buff[i]=getchar())=='\n')
                        break;
        }
}
```

makefile

```
all: lab-3-server.exe lab-3-client.exe
lab-3-server.exe: lab-3-server.o function.o
        gcc lab-3-server.o function.o -o lab-3-server.exe
functoin.o: function.c
```

```
	gcc -c finction.c
lab-3-server.o: lab-3-server.c
	gcc -c lab-3-server.c
lab-3-client.exe:
	gcc lab-3-client.o -o lab-3-client.exe
lab-3-client.o: lab-3-client.c
	gcc -c lab-3-client.c
clean:
	rm -rf *.o
```