МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Тихоокеанский государственный университет»

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Лабораторная работа №3

По предмету «Операционные системы и системное ПО»

«Синхронизация потоков»

Выполнил:

студент группы ПО(аб) –71

Маслеников М.В.

Проверил :

Тормозов В.С.

Хабаровск – 2020г.

**Цели**: Изучение функций, предназначенных для синхронизации потоков в ОС Linux.

**Задачи**: Получение практических навыков использования мьютексов и условных переменных для решения задач доступа к ресурсам.

**Порядок выполнения работы:**

1. Выполните следующие действия. Для каждого изменения опишите, что происходит, и дайте объяснение – почему.

1) Уберите в первом примере функцию sleep.

Текст первого примера:

#include <pthread.h>

#include <stdio.h>

int x=1;

void\* compute\_thread(void \* argument)

{

printf("X value in thread before sleep = %d\n",x);

printf("X value in thread is incremented by 1 before sleep\n");

x++;

sleep(2);

printf("X value in thread after sleep = %d\n",x);

return;

}

main( )

{

pthread\_t tid;

pthread\_attr\_t attr;

pthread\_attr\_init(&attr);

pthread\_create(&tid, &attr, compute\_thread,(void \*)NULL);

sleep(1);

x++;

printf("Main thread increments X, after that X value is %d\n",x);

pthread\_join(tid,NULL);

exit(0);

}

Единственный способ приостановки потока — сделать это из кода самого потока. Другими словами, можно только приостановить текущий поток. Что бы это сделать, можно вызвать функцию sleep().

После создания потока в функции main() следом идет функция sleep(1). После вызова этой функции программа начинает работать в таком порядке:

printf("X value in thread before sleep = %d\n",x);

printf("X value in thread is incremented by 1 before sleep\n");

x++;

После этого вызывается функция sleep(2), и мы переходим к следующей части кода:

x++;

printf("Main thread increments X, after that X value is %d\n",x);

После приостановка потока завершается и вызывается следующий фрагмент кода:

x++;

printf("Main thread increments X, after that X value is %d\n",x);

Поэтому результат работы программы с вызовами функции sleep будет следующим:

X value in thread before sleep = 1

X value in thread is incremented by 1 before sleep

Main thread increments X, after that X value is 3

X value in thread after sleep = 3

Если функции sleep убрать, то приостановка потока совершаться не будет и результат будет следующим:

Main thread increments X, after that X value is 2

X value in thread before sleep = 2

X value in thread is incremented by 1 before sleep

X value in thread after sleep = 3

2) Уберите во втором примере функцию освобождения мьютекса из порожденного потока.

Мьютекс захватывается вызовом функции **pthread\_mutex\_lock**. Он освобождается вызовом функции **pthread\_mutex\_unlock**. **pthread\_mutex\_lock** или захватывает мьютекс, или приостанавливает выполнение вызывающего потока, пока владелец мьютекса (то есть поток, который захватил мьютекс вызовом функции **pthread\_mutex\_lock** ранее) не освободит его вызовом **pthread\_mutex\_unlock**. Так как взаимная блокировка не была освобождена, программа не может завершиться.

3) Уберите в третьем примере функцию pthread\_cond\_signal.

Функция pthread\_cond\_wait() используется, чтобы атомарно освободить мьютекс и заставить вызывающий поток блокироваться по переменной состояния. Блокированный поток пробуждается с помощью вызовов pthread\_cond\_signal(). Процедура pthread\_cond\_wait() всегда возвращает запертый мьютекс, который принадлежит вызывающему потоку, даже если возникла ошибка. Эта функция блокируется, пока не придет сообщение о нужном состоянии. Она атомарно освобождает связанный с ней закрытый мьютекс перед блокированием, и атомарно захватывает его снова, перед возвратом. Чтобы разблокировать определенный поток, используется функция pthread\_cond\_signal(). Если никакие потоки не блокированы по переменной состояния, вызов pthread\_cond\_signal () не будет иметь никакого эффекта. Таким образом, без функции pthread\_cond\_signal поток не может быть разблокирован, и программа не может завершиться.

2. Решите классическую проблему «поставщик – потребитель» с использованием описанных в лабораторной работе средств синхронизации.

Постановка задачи: Один поток производит данные, другой поток их потребляет. В промежуток времени между изготовлением и потреблением данные хранятся в буфере.

Пример использования: Конвейер команд в Unix.

Исходные данные: Данные хранятся в циклическом буфере. Циклический буфер описывается некоторой областью памяти, указателем начала данных и указателем конца данных. Поток-поставщик записывает данные в конец буфера, поток-потребитель считывает их с начала буфера. После записи или чтения соответствующим образом меняются указатели начала и конца.

Операции чтения/записи должны быть выполнены как взаимоисключающие. Если операция чтения выполняется над пустым буфером (указатель начала = указатель конца), поток-потребитель должен быть заблокирован на условной переменной до тех пор, пока поток-поставщик не запишет в буфер какие-нибудь данные. Если операция записи выполняется над полным буфером, поток-поставщик должен также быть заблокирован на условной переменной до тех пор, пока поток-потребитель не считает из буфера какие-нибудь данные.

Размер буфера – не менее 10 символов. Поток-поставщик и поток-потребитель работают в бесконечном цикле. Поток-поставщик производит по одному символу в последовательности 0,1,2...9,0,1,... и записывает его в буфер через случайный интервал времени 0,5 – 2 сек. Поток-потребитель считывает по одному символу через случайный интервал времени 0,5 – 2 сек из буфера и выводит их на экран в виде сообщений (например, Символ 0, Символ 1,...)

Каждый поток совершая операцию с буфером выводит на экран информацию о текущем состоянии буфера до и после операции, тип операции, символ, состояние условной переменной.

Текст программы:

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

int buffer[11];

int bdp=1;

int edp=1;

pthread\_mutex\_t mymtx;

pthread\_cond\_t mysv;

void\* sourser (void \*t) {

int t1;

while (1) {

pthread\_mutex\_lock (&mymtx);

if ((bdp==1 && edp==10)||(edp==bdp-1))

pthread\_cond\_wait (&mysv, &mymtx);

else {

buffer[edp]=edp;

printf ("Symbol %d in\n", buffer[edp]);

edp=edp+1;

if (edp>10)

edp=1;

pthread\_cond\_signal (&mysv);

}

pthread\_mutex\_unlock (&mymtx);

t1=rand() % 1500000 + 500000;

usleep(t1);

}

return;

}

void\* extructor (void \*d) {

int t2;

while (1) {

pthread\_mutex\_lock (&mymtx);

if (edp==bdp)

pthread\_cond\_wait (&mysv, &mymtx);

else {

buffer[bdp];

printf ("Symbol %d out\n", buffer[bdp]);

buffer[bdp]=0;

bdp=bdp+1;

if (bdp>10)

bdp=1;

pthread\_cond\_signal (&mysv);

}

pthread\_mutex\_unlock (&mymtx);

t2=rand() % 1500000 + 500000;

usleep(t2);

}

return;

}

int main() {

int res;

pthread\_t sid, eid;

res=pthread\_mutex\_init (&mymtx, NULL);

if (res != 0) {

perror("Mutex initialization failed");

exit(EXIT\_FAILURE);

}

pthread\_cond\_init (&mysv, NULL);

res=pthread\_create (&sid, NULL, &sourser, NULL);

if (res != 0) {

perror("Thread creation failed");

exit(EXIT\_FAILURE);

}

res=pthread\_create (&eid, NULL, &extructor, NULL);

if (res != 0) {

perror("Thread creation failed");

exit(EXIT\_FAILURE);

}

res=pthread\_join (sid, NULL);

if (res != 0) {

perror("Thread join failed");

exit(EXIT\_FAILURE);

}

res=pthread\_join (eid, NULL);

if (res != 0) {

perror("Thread join failed");

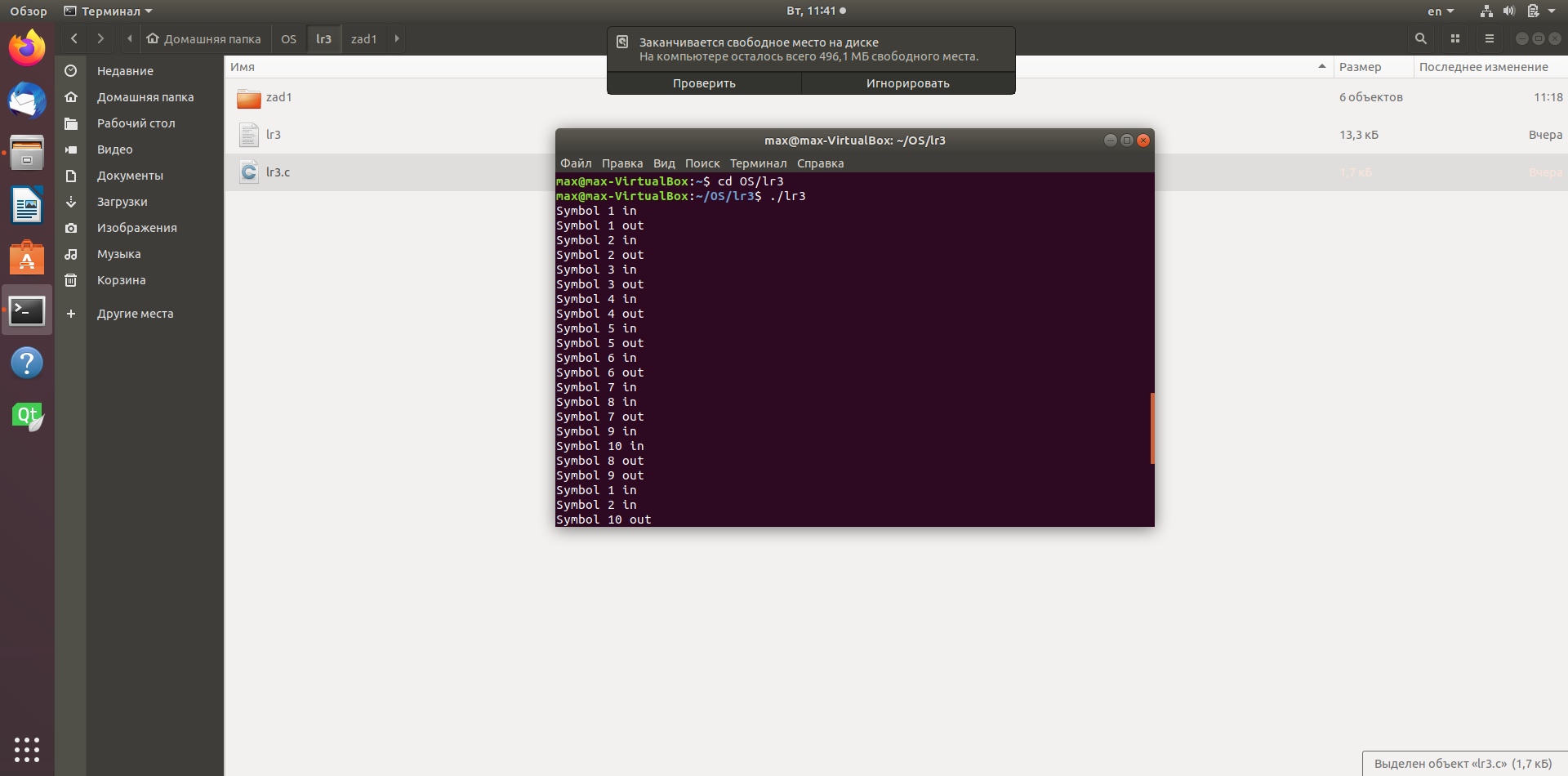
exit(EXIT\_FAILURE);

}

return;

}

Результат работы программы: в цикле записываются и считываются символы от 1 до 10.



Вывод: в ходе лабораторной работы были рассмотрены такие понятия как мьютексы и условные переменные, а также изучены функции для синхронизации потоков.