|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| voenmeh | МИНОБРНАУКИ РОССИИ  федеральное государственное бюджетное образовательное учреждение  высшего образования  **«Балтийский государственный технический университет «ВОЕНМЕХ» им. Д.Ф. Устинова»**  **(БГТУ «ВОЕНМЕХ» им. Д.Ф. Устинова»)** | | | | | |
| БГТУ.СМК-Ф-4.2-К5-01 | | | | | |
| Факультет | |  | И |  | Информационные и управляющие системы |  |
|  | |  |  |  |  |  |
| Кафедра | |  | И9 |  | Систем управления и компьютерных технологий |  |
|  | |  |  |  |  |  |
| Дисциплина | |  | Программное обеспечение систем реального времени | | |  |

Лабораторная работа №2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнила студентка группы | | | |  | | И992 |
| Шнабель М.А. | | | | | | |
|  | | | | | | |
| **ПРЕПОДАВАТЕЛЬ** | | | | | | |
| Карасев А.А. | |  |  | | | |
|  | | | | | | |
| Оценка |  | | | |  | |
| «\_\_\_\_\_» |  | | | | 2023 г. | |

САНКТ-ПЕТЕРБУРГ

2023 г.

# Постановка задачи

Первая задача: написать программу, которая создаст два процесса с помощью fork(). В этой программе проверить, в каком процессе сейчас находится пользователь - родительском или потомке. Родитель должен выводить имя, потомок - фамилию и группу. Убедиться, что два процесса действительно создались.

Выполнить запуск программы в фоновом режиме и фильтрацию по имени исполняемого файла с помощью утилит pidin и grep.

Вторая задача: переписать код, представленный на рисунке 1, изменив его следующим образом: функции, которые исполняют потоки, должны быть в динамической библиотеке. (Необходимо написать Makefile, который собирает библиотеку и подключает её к программе). Заменить sleep в main на функцию, которая бы ожидала завершения самого долгого потока.

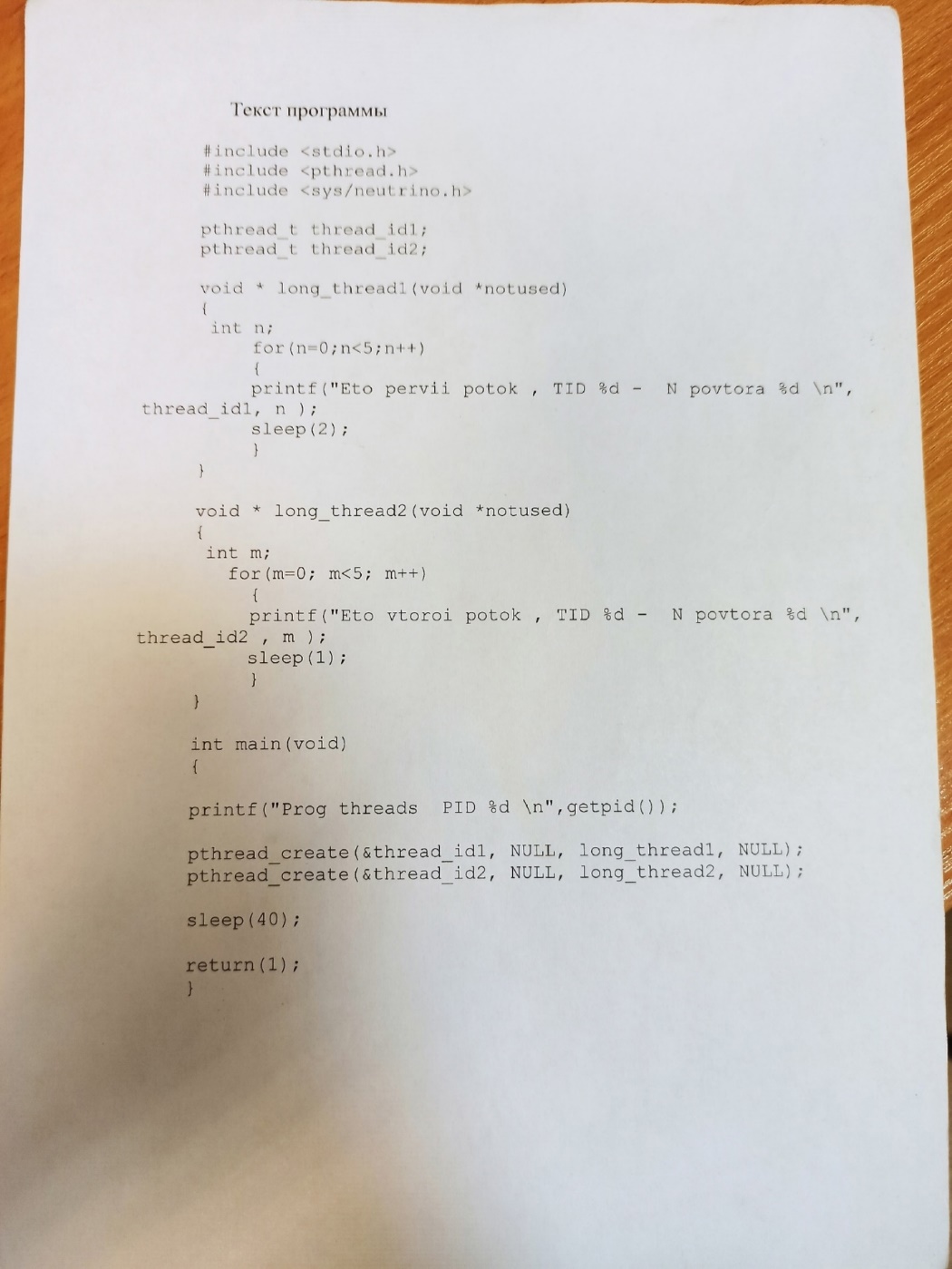


Рисунок 1 – Код второй задачи

# Понятия процесс и поток, утилиты для отображения информации о процессах

Grep - название представляет собой акроним английской фразы «search globally for lines matching the regular expression, and print them» — «искать везде строки, соответствующие регулярному выражению, и выводить их». Ищет и возвращает элементы массива, которые соответствуют шаблону, заданному в "pattern\_string".

Pidin – утилита, отображающая информацию о процессах, запущенных в QNX Neutrino system.

Процесс – программа на стадии выполнения; "объект", которому выделено процессорное время; асинхронная работа. Для описания состояний процессов используется несколько моделей. Самая простая модель - это модель трех состояний. Модель состоит из: состояния выполнения, состояния ожидания, состояния готовности.

С помощью процессов можно организовать параллельное выполнение программ. Для этого процессы клонируются вызовами fork() или exec(), а затем между ними организуется взаимодействие средствами IPC. Это довольно дорогостоящий в отношении ресурсов процесс.

С другой стороны, для организации параллельного выполнения и взаимодействия процессов можно использовать механизм многопоточности. Основной единицей здесь является поток.

Поток представляет собой облегченную версию процесса. При выполнении нескольких потоков они будут неизменно взаимодействовать друг с другом, чтобы синхронизироваться. Существует несколько средств синхронизации потоков. Это: блоки взаимного исключения (Mutex); переменные состояния; семафоры.

Мьютексы используются для синхронизации доступа к грубо говоря памяти (например, к объекту)). Формально - это механизм синхронизации, который используется для защиты общих ресурсов от конкурентного доступа несколькими потоками. Мьютекс позволяет обеспечить взаимное исключение между потоками, то есть гарантировать, что только один поток будет иметь доступ к общим ресурсам в определенный момент времени.

В отличие от многопоточности, где несколько потоков могут одновременно работать над разными задачами в рамках одного процесса, мьютекс может использоваться для синхронизации доступа к общим ресурсам в рамках одного потока.

# Решение поставленных задач

**Задача первая**

В создании нового процесса помогает fork(). Потом проверяется значение, которое возвращает fork(), чтобы определить, в каком процессе пользователь находится - родительском или потомке. Если значение fork() меньше 0, то произошла ошибка и выводится сообщение об этом. Если значение равно 0, то пользователь находится в потомке, тогда выводится информация о фамилии и группе. Если значение больше 0, то текущее местоположение - в родительском процессе, и выводится информацию об имени.

После вызова fork() теперь есть два процесса, которые выполняют одинаковый код, поэтому оба процесса будут выводить сообщения о том, что они являются родительским и потомком-процессами. Но в зависимости от значения, возвращаемого fork(), можно определить, какой процесс какую информацию выводит.

Для проверки нужно произвести запуск программы в фоновом режиме и выполнить фильтрацию по имени исполняемого файла. Для этого в командной оболочке запускается собранная программа (lab2) в фоновом режиме и добавляется амперсанд - & в конец команды. А рядом записываются утилиты pidin | grep, чтобы найти процесс lab2 в списке запущенных процессов.

Результат сборки программы и поиска процесса в списке запущенных процессов представлен на рисунке 2.

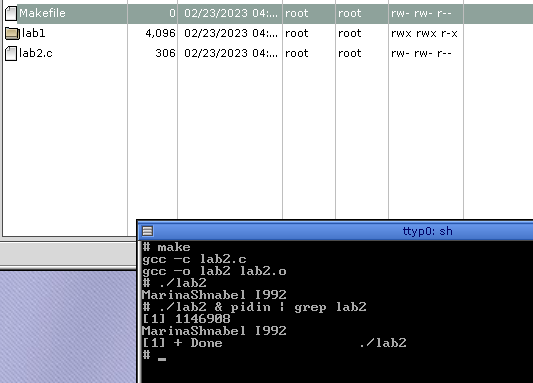


Рисунок 2 – Результат сборки программы и поиска процесса в списке запущенных процессов

**Листинг lab2.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main(){

pid\_t pid = fork();

if(pid<0){

fprintf(stderr, “Fork failed :( \n”);

exit(1);

} else if (pid==0){

printf(“Shnabel I992 \n”);

} else {

printf(“Marina\n”);

}

return 0;

}

**Листинг Makefile**

all: lab2

lab2: lab2.o

gcc -o lab2 lab2.o

lab2.o: lab2.c

gcc -c lab2.c

clean:

rm \*.out

**Вторая задача**

Конфигурируются и создаются два потока с разными приоритетами. Системный вызов sleep() переводит поток в состояние блокировки NANOSLEEP. В стандартной библиотеке С нет функции, которая бы ожидала завершения самого долгого потока. В таком случае можно использовать механизмы синхронизации для ожидания завершения потоков.

Один из способов решения данной задачи - использование функции pthread\_join(), которая позволяет ожидать завершения выполнения указанного потока.

Поток с id1 увеличивает переменную counter, pthread\_mutex\_lock() и pthread\_mutex\_unlock() используется для защиты доступа к переменной counter. Функция pthread\_mutex\_init() используется для инициализации мьютекса перед его использованием, а функция pthread\_mutex\_destroy() - для освобождения ресурсов, занятых мьютексом. Мьютекс в данном случае используется только для защиты общей переменной в рамках одного потока.

Сначала я сделала все необходимые изменения и сборку без динамической библиотеки. Листинг программы try.c и мейкфайл, а также результат работы программы и сборки без динамической библиотеки представлен на рисунке 3.

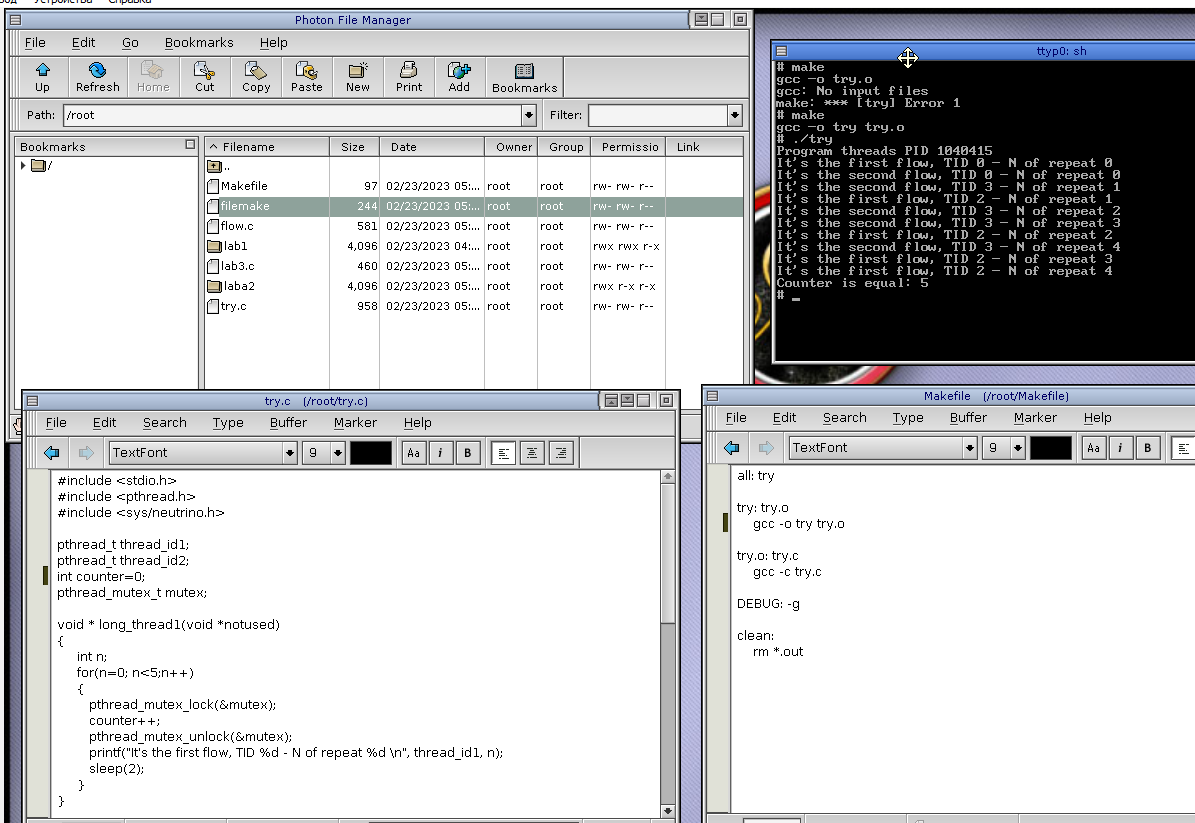


Рисунок 3 - Результат работы программы и сборки без динамической библиотеки

Затем я вынесла функции работы с потоками и мьютексом в отдельные файлы и сделала сборку с динамической библиотекой. Полученную библиотеку libflow.so нужно переместить в /lib директорию из /root.

Результат работы программы и сборки с динамической библиотекой представлен на рисунке 4.

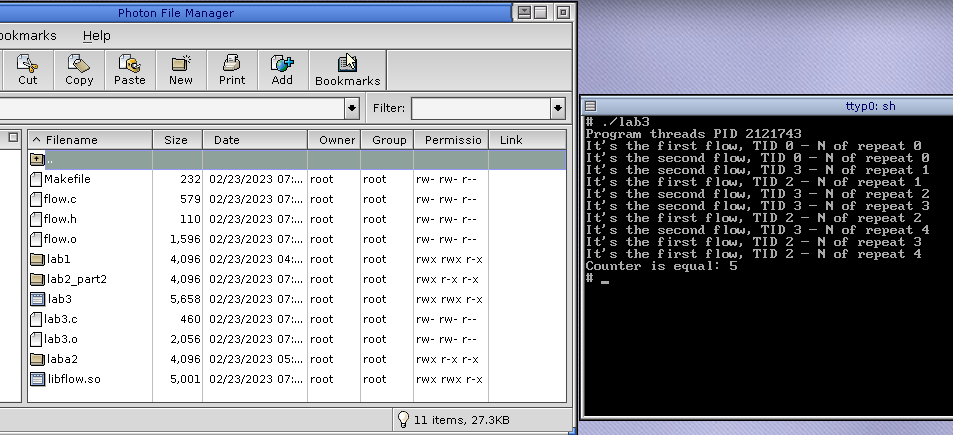


Рисунок 4 - Результат работы программы и сборки с динамической библиотекой

**Листинг lab3.c**

#include <stdio.h>

#include <pthread.h>

#include <sys/neutrino.h=>

#include “flow.c”

int main (void)

{

printf(“Program threads PID %d \n”, getpid());

pthread\_mutex\_init (&mutex, NULL);

pthread\_create(&thread\_idl, NULL, long\_thread1, NULL);

pthread\_create(&thread\_id2, NULL, long\_thread2, NULL);

pthread\_join(thread\_id1, NULL);

pthread\_join(thread\_id2, NULL);

pthread\_mutex\_destroy(&mutex);

printf(“Counter is equal: %d \n”, counter);

return 0;

}

**Листинг flow.h**

#ifndef FLOW\_H

#define FLOW\_H

void \*long\_thread1(void \*notused);

void \*long\_thread2(void \*notused);

#endif

**Листинг flow.c**

#include <stdio.h>

#include <pthread.h>

#include <sys/neutrino.h>

#include "flow.h"

pthread\_t thread\_idl;

pthread\_t thread\_id2;

int counter=0;

pthread\_mutex\_t mutex;

void \*long\_thread (void \*notused)

{

int n;

for(n=0; n<5; n++)

{

pthread\_mutex\_lock(&mutex);

counter++;

pthread\_mutex\_unlock(&mutex);

printf(“It’s the first flow, TID %d - N of repeat %d \n”, thread\_id1, n);

sleep(2);

}

}

void \*long\_thread2(void \*notused)

{

int m;

for(m=0; m<5; m++)

{

printf(“It’s the second flow, TID %d - N of repeat %d \n”, thread\_id2, m);

sleep(1);

}

}

# Листинг Makefile

all: lab3

lab3: lab3.o libflow.so

gcc -o lab3 lab3.o -L. lflow

lab3.o: lab3.c

gcc -c lab3.c

libflow.so: flow.o

gcc -shared -o libflow.so flow.o

flow.o: flow.c

gcc -c -fPIC flow.c

DEBUG: -g

clean:

rm -f\*.o \*.so lab3

В данном Makefile all собирает программу, lab3 собирает программу из файла lab3.o, используя динамическую библиотеку, lab3.o компилирует файл lab3.c, libflow.so собирает динамическую библиотеку из файла flow.o, flow.o компилирует файл flow.c, clean удаляет файлы, созданные в процессе сборки. Флаг -fPIC используется при компиляции flow.c, чтобы создать позиционно-независимый код для динамической библиотеки.