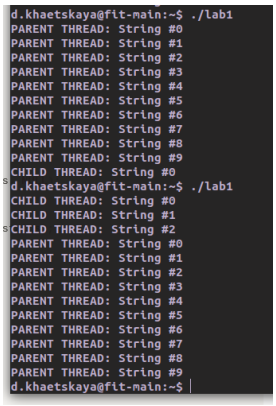
OS2 Lab1

checkbox	☑
due date	@September 3, 2021
class	OS
Status	approved

Task

```
Напишите программу, которая создает нить. Используйте атрибуты по умолчанию. Родительская и вновь созданная нити должны распечатать десять
```

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

void* printStrings(void *args) {
    for (int i = 0; i < 10; i++) {
        printf("CHILD THREAD: String #%d\n", i);
    }
    return((void *)1);
}

int main() {
    pthread_t thread;
    int status = pthread_create(&thread, NULL, printStrings, NULL);

    if (status != 0) {
        perror("failed to create thread.");
        exit(-1);
    }

    for (int i = 0; i < 10; i++) {
        printf("PARENT THREAD: String #%d\n", i);
    }

    return 0;
}
```



Основной поток выполнился быстрее и вызвал функцию `exit()` что привело к завершению процесса и соответственно, завершению всех потоков процесса.

Notes

Pthreads

```
A thread is an independent source of execution within a process. Every
      process is created with a single thread, which calls the main function.
      process may have multiple threads, all of which are scheduled
      independently by the system and may run concurrently. Threads within a
      process all use the same address space and as a result can access all
      data in the process; however, each thread is created with its own
      attributes and its own stack. When a thread is created, it inherits the
      signal mask of the thread which created it, but it has no pending
      signals.
```

```
  All threads of execution have their own, independent life time, th
Ais ultimately bounded by the life time of the process. If t
      terminates for any reason, whether due to a call to exit(3C
      of a fatal signal, or some other reason, then all threads w
      process are terminated. Threads may themselves exit and sta
      information of them may be obtained, for more information,
      pthread_detach(3C), pthread_join(3C), and pthread_exit(3C)
      their equivalents as described in the tables later on in th
```

Как создать поток?

Традиционная модель процессов в UNIX поддерживает только один поток управления на процесс. Концептуально это то же, что и модель, основанная на потоках, в случае, когда каждый процесс состоит из одного потока. При наличии поддержки `pthread` программа также запускается как процесс, состоящий из одного потока управления. Поведение такой программы ничем не отличается от поведения традиционного процесса, пока она не создаст дополнительные потоки управления. Создание дополнительных потоков производится с помощью функции `pthread_create`.

```
cc -mt [ flag... ] file... -lpthread [ library... ]
#include <pthread.h>

int pthread_create(pthread_t *restrict thread,
    const pthread_attr_t *restrict attr,
    void *(*start_rtn)(void*), void *restrict arg);
```

```
The pthread_create() function is used to create a new thread, with
 attributes specified by attr, within a process. If attr is NULL, the
 default attributes are used. (See pthread_attr_init(3C)).  If the
 attributes specified by attr are modified later, the thread's attributes
 are not affected. Upon successful completion, pthread_create() stores the
 ID of the created thread in the location referenced by thread.


 The thread is created executing start_routine with arg as its sole
 argument. If the start_routine returns, the effect is as if there was an
 implicit call to pthread_exit() using the return value of start_routine
 as the exit status. Note that the thread in which main() was originally
 invoked differs from this. When it returns from main(), the effect is as
 if there was an implicit call to exit() using the return value of main()
 as the exit status.
```

Вновь созданный поток начинает выполнение с функции `start_rtn`. Эта функция принимает единственный аргумент `arg` — нетипизированный указатель. Если функции `start_rtn` потребуется передать значительный объем информации, ее следует сохранить в виде структуры и передать в `arg` указатель на структуру.
При создании нового потока нельзя заранее предположить, кто первым получит управление — вновь созданный поток или поток, вызвавший функцию `pthread_create`. Новый поток имеет доступ к адресному пространству процесса и наследует от вызывающего потока среду окружения арифметического сопроцессора и маску сигналов, однако набор сигналов, ожидающих обработки, для нового потока

Квант процессорного времени

Квантом процессорного времени называется время, по истечении которого выполнение нити с алгоритмом планирования SCHED_RR будет прервано, а управление будет передано другой нити с тем же приоритетом. С помощью опции -t команды **schedtune** можно увеличить число тактов процессора в кванте времени. Другими словами, квант времени можно увеличивать за счет прерываний по 10 миллисекунд (см. раздел Изменение кванта времени планировщика с помощью команды schedtune).

Примечание: Квант времени не является гарантированным квантом времени процессора. Он задает максимальное время, в течение которого нити может принадлежать управление до ее замены другой нитью. Существует множество причин, по которым нить может потерять управление CPU до истечения кванта времени.

```
393  /*
394   * POSIX definitions are same as defined in thread.h and synch.h.
395   * Any changes made to here should be reflected in corresponding
396   * files as described in comments.
397   */
398  typedef uint_t  pthread_t;      /* = thread_t in thread.h */
399  typedef uint_t  pthread_key_t;  /* = thread_key_t in thread.h */
```

```
50  #ifndef _ASM
51  typedef unsigned int thread_t;
52  typedef unsigned int thread_key_t;
53
```

SCHEDULING

POSIX Threads

illumos supports the following three POSIX scheduling policies:

SCHED_OTHER
Traditional Timesharing scheduling policy. It is based on the timesharing (TS) scheduling class.

SCHED_FIFO
First-In-First-Out scheduling policy. Threads scheduled to this policy, if not preempted by a higher priority, will proceed until completion. Such threads are in real-time (RT) scheduling class. The calling process must have a effective user **ID** of **0**.

SCHED_RR
Round-Robin scheduling policy. Threads scheduled to this policy, if not preempted by a higher priority, will execute for a time period determined by the system. Such threads are in real-time (RT) scheduling class and the calling process must have a effective user **ID** of **0**.

USAGE

-mt compiler option
The **-mt** compiler option compiles and links for multithreaded code. It compiles source files with **-D_REENTRANT** and augments the set of support libraries properly.

Users of other compilers such as gcc and clang should manually set **-D_REENTRANT** on the compilation line. There are no other libraries or flags necessary.

Завершение процесса системным вызовом `exit(2)` или возвратом из функции `main` приводит к завершению всех нитей процесса. Это поведение описано в стандарте POSIX, поэтому ОС, которые ведут себя иначе (например, старые версии Linux), не соответствуют стандарту. Если вы хотите, чтобы нити вашей программы продолжали исполнение после завершения `main`, следует завершать `main` при помощи вызова `pthread_exit(3C)`.