SUPSI

Lab: Introduction

Operating Systems

Amos Brocco, Lecturer & Researcher

12 settembre 2022

Objectives

- Understand how to create and manage threads with Pthread in C

Browsing
- Get a rapid overview.

Reading
- Read it and try to understand the concepts.

Studying
- Read in depth, understand the concepts as well as the principles behind the concepts.

You are also encouraged to try out (compile and run) code examples!

How do threads work in C (with pthread) ?

Creating a thread (with pthread)

Compile and link with `-pthread`

```
#include <pthread.h>

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

- Each thread is associated with a **`pthread_t`** structure
- The "body" of the thread is defined by the **`start_routine`** procedure, which can receive parameters using the **`arg`** pointer
- The new thread is started immediately

Terminating a thread

- A thread terminates when `start_routine` returns
- A thread can explicitly terminate its execution and return a value:

```
#include <pthread.h>

void pthread_exit(void *retval);
```

- A thread can also ask another thread to terminate:

```
#include <pthread.h>

int pthread_cancel(pthread_t thread);
```

  - Exit happens as soon as possible (when a *cancellation point* is reached)

Cancellation point

```
#include <pthread.h>

void pthread_testcancel(void);
```

- With this procedure we can define a cancellation point where the thread will respond to pending cancellation requests:
  - It is possible to ignore the request with `pthread_setcancelstate`
  - Many functions provide pre-defined cancellation points (see `man pthreads`)

Who gets the exit value?

- A thread can wait for another thread to terminate and obtain its exit value:

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **retval);
```

- The return value can be obtained from **`retval`**
- Only threads which are **`JOINABLE`** *(default) can be waited for, **`DETACHED`** ones can't be waited for:
  - A joinable thread waits until the join before being freed
  - **`pthread_join`** returns 0 if the thread terminates correctly, a negative value in case of errors

* see `man pthread_attr_init`

Example

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

void *mythread (void *name)
{
    printf("Hello, I'm a new thread %s\n", (char*) name);
    sleep(3);
    return (void*) 42;
}

int main()
{
    pthread_t thread;
    int i;

    pthread_create(&thread, NULL, &mythread, "Alfred");
    sleep(2);
    pthread_join(thread, (void**) &i);
    printf("Return value is %d\n", i);
    return 0;
}
```

Example (detached thread)

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

void *mythread (void *name)
{
    printf("Hello, I'm a new thread %s\n", (char*) name);
    sleep(3);
    return (void*) 42;
}

int main()
{
    pthread_t thread;
    pthread_attr_t attr;
    int i;

    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);

    pthread_create(&thread, &attr, &mythread, "Alfred");
    sleep(2);
    pthread_join(thread, (void**) &i); // Error, cannot join detached thread
    printf("Return value is %d\n", i); // Return value is bogus
    return 0;
}
```

Example (alternate stack)

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define STACK_SIZE 2<<15

void *mythread (void *name)
{
    printf("Hello, I'm a new thread %s\n", (char*) name);
    sleep(3);
    return (void*) 42;
}

int main()
{
    pthread_t thread;
    pthread_attr_t attr;
    int i;
    void *sp;
    pthread_attr_init(&attr);
    sp = malloc(STACK_SIZE);
    pthread_attr_setstack(&attr, sp, STACK_SIZE);
    pthread_create(&thread, &attr, &mythread, "Alfred");
    sleep(2);
    pthread_join(thread, (void**) &i); // Error, cannot join detached thread
    free(sp);
    printf("Return value is %d\n", i); // Return value is bogus
    return 0;
}
```

Example (pthread_exit)

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

void *mythread (void *name)
{
    printf("Hello, I'm a new thread %s\n", (char*) name);
    sleep(3);
    pthread_exit((void*) 13);
    return (void*) 42;
}

int main()
{
    pthread_t thread;
    int i;

    pthread_create(&thread, NULL, &mythread, "Alfred");
    sleep(2);
    pthread_join(thread, (void**) &i);
    printf("Return value is %d\n", i);
    return 0;
}
```

Example (pthread_testcancel)

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

void *mythread (void *arg)
{
    printf("Thread start\n");
    while (1) {
        pthread_testcancel(); /* Cancellation point */
    }
    printf("Exiting!\n"); /* This code is never executed */
    return (void*) 42; /* This code is never executed */
}
int main()
{
    pthread_t thread;
    int i;
    pthread_create(&thread, NULL, &mythread, NULL);
    sleep(3);
    pthread_cancel(thread);
    sleep(4);
    pthread_join(thread, (void**) &i);  /* The return value 'i' is PTHREAD_CANCELED (-1)  */
    printf("Return value %d\n", i);
    return 0;
}
```

Use case for kernel threads: I/O in a separate thread

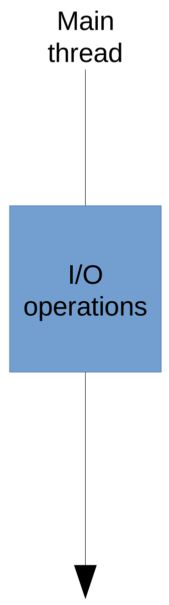
- I/O operations normally block the execution (until data is read/written)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define NBYTES 10000

void main(void)
{
        char* buffer[NBYTES];
        unsigned int bytes;

        FILE* file = fopen("/var/log/syslog", "r");
        bytes = read(fileno(file), buffer, NBYTES);
        printf("Synchronous read, got %d bytes.\n", bytes);
        close(fileno(file));
}
```



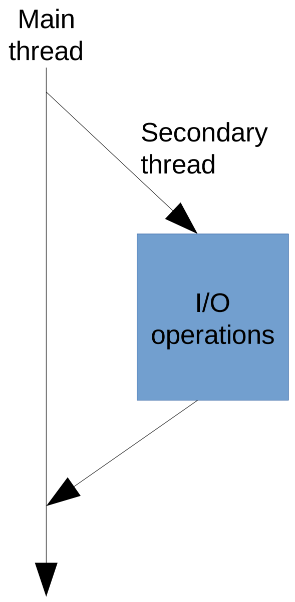
Use case for kernel threads: I/O in a separate thread

- I/O operations can be moved to a separate thread

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#define NBYTES 10000
FILE* file;
char* buffer[NBYTES];
unsigned int bytes;

void* reader() {
       bytes = read(fileno(file), buffer, NBYTES);
}

void main(void) {
       pthread_t thread_reader;
       file = fopen("/var/log/syslog", "r");
       pthread_create(&thread_reader, NULL, &reader, NULL);
       printf("... reader thread is doing its work...\n");
       sleep(5);
       pthread_join(thread_reader, NULL);
       printf("Read finished, got %d bytes.\n", bytes);
       close(fileno(file));
}
```



Read until end of Section 3.1.3