Threads

What is unique to each Thread?

- Threads share
  - Code
  - Data
  - File descriptors
  - Signals and signal handlers
  - Environment
  - User and group ids
- Each thread has unique
  - Thread id
  - Registers, stack area, local variables
  - Signal mask
  - Priority
  - Return value
  - errno

The Linux Implementation of Threads

- A thread in Linux kernel is nothing but a process that shares certain resources, which are
  - Address space
  - File system resources
  - File descriptors
  - Signal handlers

Credits: Linux Kernel Development 3rd Edition, Robert Love

POSIX Threads(pthreads)

- POSIX Threads is an API defined by the standard POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995).
- Most Unix-Like system provide implementation for *pthreads*
  - Linux: (Native POSIX Thread Library)
  - FreeBSD
  - macOS
  - Android
- The header file for C Language is *pthread.h*
- We need to link our programs with *pthread* library
  - gcc -pthread main.c
  - g++ -pthread main.cpp

Source: https://en.wikipedia.org/wiki/Pthreads

pthreads: thread creation

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

- `pthread_t *thread:` Buffer that stores newly created thread id
- `const pthread_attr_t *attr:` Attributes for newly created thread; `NULL` is default
- `void *(*start_routine) (void *):` The function that the newly created thread will execute
- `void *arg:` Arguments to the function
- Returns:
  - On success, 0
  - Otherwise, an error number

pthreads: thread creation, e.g.

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

...
int pthread_retcode;

int arg0 = 15;
pthread_t id;
if((pthread_retcode = pthread_create(&id, NULL, print_hello_world, &arg0)) != 0) {
    fprintf(stderr, "Error while creating a thread: %s\n", strerror(pthread_retcode));
    exit(EXIT_FAILURE);
}
...
```

pthreads: stack size

- To check newly created stack size, execute the following bash builtin command
  - *ulimit -s*
- *If RLIMIT_STACK* is “unlimited”, then the default stack size is *2MB* for *x86_64* architecture

pthreads: returning a value

- We can use following methods to return a value from the routine that the thread executes:
  - Calling *pthread_exit* function
  - Or simply by using `return` statement
- E.g.

```
#include <pthread.h>

void* print_hello_world(void* arg) {
        ...
    int arg0 = *((int*)arg);
    int* retval = (int*) malloc(sizeof(int));
    *retval = arg0 * arg0;
    return (void*) retval; // pthread_exit((void*) retval);
}
```

pthreads: joining and handling returned value

- To handle returning value from a running thread or to wait it to finish its execution, we have to join it within the current thread by calling *pthread_join* function:
  - `int pthread_join(pthread_t thread, void **retval);`
- E.g.

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <string.h>
...
int pthread_retcode;
int* thread_retval;
if((pthread_retcode = pthread_join(id, (void**)&thread_retval)) != 0) {
    fprintf(stderr, "Error while joining the thread with id %lu: %s\n", id, strerror(pthread_retcode));
    exit(EXIT_FAILURE);
}
printf("Return val: %d\n", *thread_retval);
free(thread_retval);
...
```

pthreads: e.g.

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

void* print_hello_world(void* arg) {
    printf("Hello world!\n");
    printf("My thread id: %lu\n", pthread_self());

    int arg0 = *((int*)arg);
    int* retval = (int*) malloc(sizeof(int));
    *retval = arg0 * arg0;
    return (void*) retval; // pthread_exit((void*) retval);
}
```

```c
int main() {
    int pthread_retcode;

    int arg0 = 15;
    pthread_t id;
    if((pthread_retcode = pthread_create(&id, NULL,
print_hello_world, &arg0)) != 0) {
        fprintf(stderr, "Error while creating a thread:
%s\n", strerror(pthread_retcode));
        exit(EXIT_FAILURE);
    }

    int* thread_retval;
    if((pthread_retcode = pthread_join(id,
(void**)&thread_retval)) != 0) {
        fprintf(stderr, "Error while joining the thread
with id %lu: %s\n", id, strerror(pthread_retcode));
        exit(EXIT_FAILURE);
    }
    printf("Return val: %d\n", *thread_retval);
    free(thread_retval);

    return 0;
}
```