CS232L Operating Systems Lab
Lab 10: Introduction to POSIX Threads

CS Program
Habib University

Fall 2019

1 Introduction

A thread is a unit of execution. Each process has at least one thread. Sometimes multiple threads may be associated with a process.

In this lab we will learn:

1. a brief introduction to Posix Threads
2. create multiple threads
3. pass arguments to a thread
4. receive return values from a thread
5. wait for the termination of created threads

2 What is a thread

Threads are a mechanism that enable a process to perform multiple tasks concurrently. A single process can have multiple threads as shown in Figure 1 on page 2.
All of these threads are independently executing the same program, and they all share the same global memory, including the initialized data, uninitialized data, and heap segments. (A traditional UNIX process is simply a special case of a multithreaded processes; it is a process that contains just one thread.)

Threads in a process can execute concurrently. On a multi-processor system, the threads can execute in parallel[1].

In Linux the variable errno is maintained separately for each thread.

3 Thread creation

[1]While the same multiple processes can achieve the same effects, they have the following disadvantages:

- processes are expensive to create.
- it is more difficult to share data among processes.

This is why threads are sometimes refered to as light-weight processes.

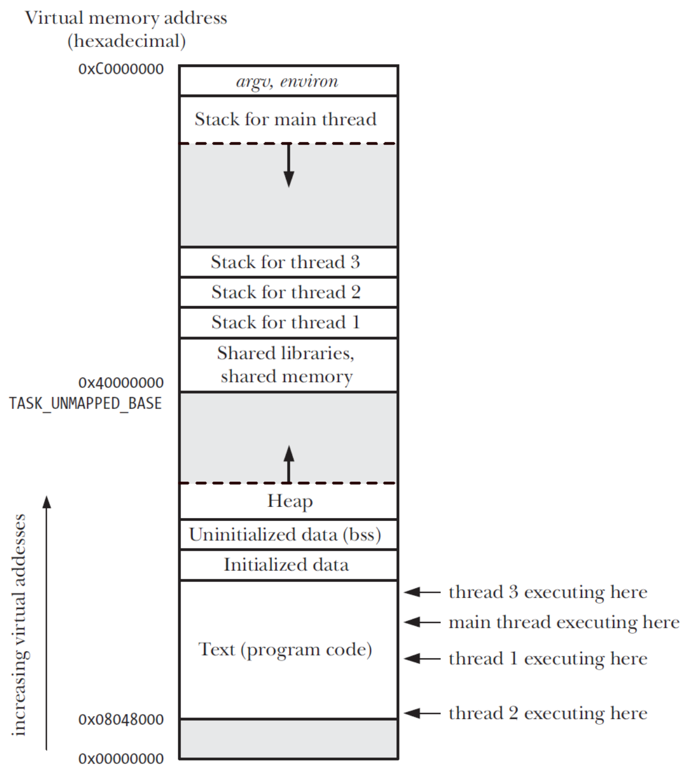

Figure 1: A Linux process running three threads

```c
/* hello_pthread.c*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <pthread.h>


void * hello_fun(void * args){

  printf("Hello World!\n");

  return NULL;
}

int main(int argc, char * argv[]){

  pthread_t thread;  //thread identifier

  //create a new thread have it run the function hello_fun
  pthread_create(&thread, NULL, hello_fun, NULL);

  //wait until the thread completes
  pthread_join(thread, NULL);

  return 0;
}
```

Listing 1: Creating a thread

The function pthread_create() [2] creates a thread of the function passed to it as argument. The listing 1 shows the use of pthread_create to execute the function hello_fun() in a separate thread.

[2]see manual page for the function

Do not forget to link with the pthread library when compiling a program using POSIX threads.

3.1 Todo:

- Create multiple threads in your program and make them perform different functions.
- Create multiple threads in your program and make them display their thread identfiers (TIDs) [3] and process identifiers (PIDs).

4 Passing arguments to threads

```c
/* hello_args_pthread.c*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <pthread.h>


void * hello_arg(void * args){

  char * str = (char *) args;

  printf("%s\n", str);

  return NULL;
}

void * hello_arg_int(void * args){

  int * str = (int *) args;

  printf("%d\n", *str);

  return NULL;
}

int main(int argc, char * argv[]){

  char hello[] = "Hello World!";
  int x = 33;

  pthread_t t1, t2;  //thread identifier

  //create a new thread that runs hello_arg with argument hello
  pthread_create(&t1, NULL, hello_arg, hello);
  pthread_create(&t2, NULL, hello_arg_int, &x);

  //wait until the thread completes
  pthread_join(t1, NULL);
  pthread_join(t2, NULL);

  return 0;
}
```

Listing 2: Passing arguments to threads

The listing 2 shows how we can pass arguments to a thread when creating it.

4.1 Todo:

Make multiple threads and pass them different strings which they will each display along with their TIDs and PIDs.

[3] Checkout the function pthread_self().

5 Waiting for threads

The pthread_join() function lets you wait for a thread to terminate. It is a blocking call which would block the calling thread until the thread specified as the join's argument has finished. Usually the model is that there is a main thread which creates all the other threads and then waits for all the threads to finish before exiting itself. However, nothing stops other threads from waiting for each other by calling pthread_join().

```c
/* hello_pthread_bad.c*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <pthread.h>


void * hello_fun(){

  printf("Hello World!\n");

  return NULL;
}

int main(int argc, char * argv[]){

  pthread_t thread;

  pthread_create(&thread, NULL, hello_fun, NULL);

  return 0;
}
```

Listing 3: Failing to wait

If the main thread exits, all the threads created by it are joined even if they have not yet finished their execution. This is illustrated by the listing 3.
Observe the behaviour of this program by compiling and running it.

5.1 Todo:

- Correct the program in listing 3 so that it displays the output.
- Create multiple threads in a program and see the effect of calling pthread_exit() and exit() in those threads.

6 Returning values from threads

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

#include <pthread.h>

static void *
threadFunc(void *arg)
{
  char *s = (char *) arg;
  printf("%s", s);
  return (void *) strlen(s);
}

int
```

```c
main(int argc, char *argv[])
{
  pthread_t t1;
  void *res;
  int s;

  s = pthread_create(&t1, NULL, threadFunc, "Hello world\n");
  if (s != 0){
    perror("error: pthread_create");
    exit(EXIT_FAILURE);
  }

  printf("Message from main()\n");

  s = pthread_join(t1, &res);
  if (s != 0) {
    perror("error: pthread_join");
    exit(EXIT_FAILURE);
  }

  printf("Thread returned %ld\n", (long) res);
  exit(EXIT_SUCCESS);
}
```

Listing 4: Returning values from threads

Listing 4 how the second argument of pthread_join() can be used to receive the return value from a thread.

6.1 Todo:

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


int
main(int argc, char *argv[])
{
  pthread_t t1;
  void *res;
  int s;

  srand(time(NULL));

  //int a[1000000000];
  int *a = malloc(100000000*sizeof(int));
  if (a == NULL) {
    perror("error: memory failure");
    exit(EXIT_FAILURE);
  }


  for(int i=0; i<1000000000; i++)
    a[i%100000000] = rand()%1000000000;


  free (a);
  exit(EXIT_SUCCESS);
}
```

Listing 5: Initializing a large array (dummy.c)

- Listing 5 initializes a large array by writing 10 times a random value in each of its indices. Compile and run the program to see how long it takes. See if you could make it run faster via threads.

- Create a global variable counter. Write a function that increments this variable 10 million times. Launch this function in 4 threads and wait in main() for these threads to finish and then display the value of counter in main(). Explain the observed value.

7 Other functions

Check out the documentation for funcitons and try to use them in your programs:

- pthread_exit()
- pthread_self()
- pthread_equal()
- pthread_detach()