Lab 2

1. Starting Code: Thread creation based on the textbook code, pthread_tb.c

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

int sum;

void *runner(void *param){
   int i, upper = atoi(param);
   sum = 0;

   for(i=0; i<=upper; i++)
          sum += i;
   pthread_exit(0);
}

int main(int argc, char *argv[]){
   pthread_t tid;
   pthread_attr_t attr;

   if (argc != 2){
          fprintf(stderr, "usage: a.out <integer value>\n");
          return -1;
   }

   if (atoi(argv[1]) < 0){
          fprintf(stderr, "%d must be >= 0\n", atoi(argv[1]));
          return -1;
   }

   pthread_attr_init(&attr);
   pthread_create(&tid, &attr, runner, argv[1]);
   pthread_join(tid, NULL);

   printf("sum = %d\n", sum);

   return 0;
}
```

Compilation:

```
gcc  pthread_tb.c -o pthread.out -lpthread
```

Output:

```
jurn@ZBookG4:~/work/cs332/thread$ ./pthread.out 3
sum = 6
jurn@ZBookG4:~/work/cs332/thread$ ./pthread.out 4
sum = 10
```

2. Lab Practices

- How to read the Linux thread information?

  ps -eLf

  ```
  UID      PID  PPID   LWP  C NLWP STIME TTY        TIME CMD
  root     1       0     1  0    1 Jan15 ?      00:00:04 /sbin/init splash
  root     2       0     2  0    1 Jan15 ?      00:00:00 [kthreadd]
  ```

  ($ man ps)

  ```
  lwp      LWP     light weight process (thread) ID of the dispatchable entity
                   (alias spid, tid).
  nlwp     NLWP          number of lwps (threads) in the process.  (alias thcount).
  ```

  (e.g.,)

  ```
  jurn     23488 31947 23488  0   5 12:51 pts/22   00:00:00 ./pthread.out 4
  jurn     23488 31947 23489  0   5 12:51 pts/22   00:00:00 ./pthread.out 4
  jurn     23488 31947 23490  0   5 12:51 pts/22   00:00:00 ./pthread.out 4
  jurn     23488 31947 23491  0   5 12:51 pts/22   00:00:00 ./pthread.out 4
  jurn     23488 31947 23492  0   5 12:51 pts/22   00:00:00 ./pthread.out 4
  ```

- How to check the process state?

  ```
  jurn@ZBookG4:~/work/cs332$ ps -au
  USER   PID %CPU %MEM     VSZ   RSS TTY         STAT START   TIME COMMAND
  ```

  For the STAT, after $man ps, and then read the description ‘PROCESS STATE CODES’ part

  ```
  PROCESS STATE CODES
         Here are the different values that the s, stat and state output specifiers (header
         "STAT" or "S") will display to describe the state of a process:

         D      uninterruptible sleep (usually IO)
         R      running or runnable (on run queue)
         S      interruptible sleep (waiting for an event to complete)
         T      stopped by job control signal
         t      stopped by debugger during the tracing
  ```

W	paging (not valid since the 2.6.xx kernel)
X	dead (should never be seen)
Z	defunct ("zombie") process, terminated but not reaped by its parent

For BSD formats and when the stat keyword is used, additional characters may be displayed:

<	high-priority (not nice to other users)
N	low-priority (nice to other users)
L	has pages locked into memory (for real-time and custom IO)
s	is a session leader
l	is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
+	is in the foreground process group

POSIX APIs

- **int pthread_create(pthread_t ****thread***, const pthread_attr_t ****attr***, void *(****start_routine***) (void *), void ****arg***);**
- **void pthread_exit(void **retval*);**
- **int pthread_join(pthread_t *thread*, void ***retval*);**

3. Lab Assignment: Submission date on the moodle (7/2, Friday 11 am)
and Online Grading (7/2, 1 pm during lecture session or using the submitted files)

- Check the incorrect input cases like the **Output** examples.

- Create K POSIX threads passing the thread argument (*i.e.,* **void ****arg*), assuming K <= 10. (e.g., if the thread number = 3 (argv[1]), the *thread_arg* will be 0,1,2)

- In each thread routine,
  - print the *thread_arg* information
  - after printing the message of 'and sleeping for 60 secs', sleep for 60 seconds.
  - after sleeping 60 seconds, print 'thread_arg # was completed'.

- The main thread waits until the K threads are finished, and prints the join message ('thread_arg # is joined well') for each thread. And, it finally prints the message of 'the main thread exits'.

- And test your program using the **Output**.

Compilation:

```
jurn@ZBookG4:~/work/cs332/thread$ gcc lab2.c -o lab2.out -lpthread
```

Output : (some results depend on the machine; for example, *thread ID* or the order of *thread_arg* etc).

```
jurn@ZBookG4:~/work/cs332/thread$ ./lab2.out
usage: lab2.out <integer value>

jurn@ZBookG4:~/work/cs332/thread$ ./lab2.out -1
-1 must be >= 0

jurn@ZBookG4:~/work/cs332/thread$ ./lab2.out 3 &
[1] 3727
jurn@ZBookG4:~/work/cs332/thread$ thread number = 3
thread_arg = 0 was created    and sleeping for 60 secs
thread_arg = 1 was created    and sleeping for 60 secs
thread_arg = 2 was created    and sleeping for 60 secs
```

(For 60 seconds: NOT printed)

Please check your threads information during the 60 seconds.

```
jurn@ZBookG4:~/work/cs332/thread$ ps -eLf | grep lab2
jurn      3655  2947  3655  0      1 15:06 pts/19   00:00:00 vim lab2.c
jurn      3727  3414  3727  0      4 15:12 pts/2    00:00:00 ./lab2.out 3
jurn      3727  3414  3728  0      4 15:12 pts/2    00:00:00 ./lab2.out 3
jurn      3727  3414  3729  0      4 15:12 pts/2    00:00:00 ./lab2.out 3
jurn      3727  3414  3730  0      4 15:12 pts/2    00:00:00 ./lab2.out 3
```

(After 60 seconds: the below will be printed)

```
thread_arg 0 was completed
thread_arg 2 was completed
thread_arg 1 was completed
thread_arg 0  is joined well
thread_arg 1  is joined well
thread_arg 2  is joined well
the main thread exits

[1]+  Done                    ./lab2.out 3

jurn@ZBookG4:~/work/cs332/thread$ ./lab2.out 10 &
[1] 3772
jurn@ZBookG4:~/work/cs332/thread$ thread number = 10
thread_arg = 0 was created    and sleeping for 60 secs
thread_arg = 1 was created    and sleeping for 60 secs
thread_arg = 2 was created    and sleeping for 60 secs
thread_arg = 3 was created    and sleeping for 60 secs
thread_arg = 4 was created    and sleeping for 60 secs
thread_arg = 5 was created    and sleeping for 60 secs
thread_arg = 7 was created    and sleeping for 60 secs
thread_arg = 8 was created    and sleeping for 60 secs
thread_arg = 9 was created    and sleeping for 60 secs
thread_arg = 6 was created    and sleeping for 60 secs
```

(After 60 seconds: NOT printed)

```
thread_arg 1 was completed
thread_arg 4 was completed
thread_arg 3 was completed
thread_arg 0 was completed
thread_arg 5 was completed
thread_arg 6 was completed
thread_arg 7 was completed
thread_arg 9 was completed
thread_arg 8 was completed
thread_arg 2 was completed
thread_arg 0  is joined well
thread_arg 1  is joined well
thread_arg 2  is joined well
thread_arg 3  is joined well
thread_arg 4  is joined well
thread_arg 5  is joined well
thread_arg 6  is joined well
thread_arg 7  is joined well
thread_arg 8  is joined well
thread_arg 9  is joined well
the main thread exits

[1]+  Done                    ./lab2.out 10
```