Computer Communication 352/552/652 (Index No. 04522/10094/302954)

CURTIN UNIVERSITY OF TECHNOLOGY
Department of Computing

Worksheet 6

1. Copy all the C program files from the UNIT DIRECTORY under `worksheets/practical7`, into your own home directory.

2. You have to be careful when you want to pass argument to a thread. Remember that threads share all memory in a process. To see the passing argument problem, do the followings.
   a) In program `pass_arg1.c`, the main thread prints its ID, and passes the ID to a new thread. The new thread then prints the passing argument (parent thread ID).
      - Compile and run program `pass_arg1.c`. We expect to see the same ID numbers printed.

      Note, you must use **`-lpthread`** to compile your program (`cc pass_arg1.c -lpthread`).
   b) Modify the program to include the `sleep (2)` in the `thread_function()` so that the new thread gets the passing argument after the function `create_thread ()` returns.
      - Compile and run the modified program. Do you still get correct output (the printed Ids are the same)?
   c) To correct the problem, you have to pass argument as a pointer (i.e., use `malloc()`
      - Modify the program, compile and run the program. Do you get correct results?

3. The argument that is passed to the `thread_function()` is a single pointer. If more arguments are required, you have to set up a structure and have the argument point to it. Modify `pass_arg1.c` to do the followings:
   - In the `create_thread ()`let the parent thread send a message "Hello from your creator", and the creator's thread ID.
   - In the `thread_function (),` let the new thread print the message and the ID.

4. The completion status is a convenient way of returning the results of a thread's execution to another thread.
   - Modify your program so that the parent thread may wait for the termination of its child thread (`using pthread_join()`), and print the returned status from the child thread.
     - For this example, let the parent thread pass two integers to the child thread, and let the child return the sum of the integers to the parent thread.

5. `strclithread.c` is the `str_cli` function that uses threads. Using `makeclient1`, create your TCP echo client program that includes `strclithread.c`. Run your TCP

echo server from your previous practicals, and then run your new created client. Does it work?

6. `tcpserv01.c` is a concurrent TCP echo server using threads (for each client connection, the server creates one thread). However, in the program we pass a value of the `connfd` through `pthread_create` (ANSI C that does guarantee that this works). Create the executable for the server using `makeserver1`, and run it. Use the echo client program to connect to the server. Does it work?

7. `tcpserv02.c` is another version of TCP echo server using threads. In this version, we create a copy of the `connfd` for each thread, and pass it through `pthread_create`. This is a better solution compared to `tcpserv01.c`. Repeat #6 using `tcpserv02.c`. Does it work? Make sure you understand the program!

8. `readlinethread.c` is a thread-safe `readline` function. Create the executable for `tcpserv01.c` with `readlinethread.c` and run it. Use the echo client program to connect to the server. Does it work? Make sure you understand the program!

9. In the `example01.c`, we create three threads, each of which will increment a global variable `counter` 1000 times. Each time the `counter` is incremented, each thread will print its TID, and the current counter value. Compile the example01.c. Run it several times. Does the program always give 3000 (the correct output) as the results?

10. In the `example02.c`, we use a `mutex` to protect the shared variable `counter`. Compile the example02.c. Run it several times. Does the program always give 3000 (correct output) as the results?

11. In the `example03.c`, we simulate producer-consumer bounded queue problem. The producer will write the squares of integers from 1 to 100, while the consumer thread removes the values and sum them. However, the producer and consumer synchronization is not correct. Please note that the BUFSIZE is set to 8 (less than 100).
    - Compile the program, and run it. Does it give correct result?
    - Revise the program so that the producer runs first. Does it give same result as before? Why?
    - Revise this example by using condition variable to synchronize the producer and consumer. For the revised version, does it matter if the producer or the consumer runs first?

12. The web03.c is the web client for simultaneous connections using threads and condition variables. Compile the program using makeclient5. Does it run? Read the source code to understand the program.

13. REMEMBER TO KILL YOUR RUNNING PROCESSES AFTER YOU FINISH YOUR PRACTICAL.