# Investigating Threads

## Learning Outcomes

At the end of this lab you should be able to:

LO1. Write a source code that creates threads

LO2. Use the simulator to compile and run the code

LO3. Display the list of processes/threads and the tree of processes showing the parent/child process relationship

LO4. Demonstrate that threads share their parent's data areas

LO5. Modify the source code to create a version not using threads to be able to compare the two versions

LO6. Simulate a multi-threaded server program

## What are threads?

Threads are similar to processes and they too get scheduled by the OS. However, they do not normally exist in isolation. All threads have parent processes or other parent threads. Threads are usually referred to as "light-weight processes" (lwp). This is because thread creation and management is not as demanding and time-consuming as processes. Also, threads normally share their parents' (and grandparents') global data spaces and other resources.

## Tutorial Exercises

In this tutorial, you'll investigate the threads (or light-weight processes). To do this we'll use the CPU/OS simulator. You also need to use the "teaching" compiler that is part of the simulator. Please note that the method of creating threads, as shown below, is specific to the "teaching" compiler and the simulator; other languages such as Java and C++ use different methods.

### LO1. Write a source code that creates threads.

Start the CPU/OS simulator. In the compiler window enter the following source code:

```
program ThreadTest1
     sub thread1 as thread
          writeln("In thread1")
          while true
          wend
     end sub

     sub thread2 as thread
          call thread1
```

```
        writeln("In thread2")
        while true
        wend
    end sub

    call thread2
    writeln("In main")

    do
    loop
end
```

**Notes**:

1. The "**as thread**" construct marks the subroutine executable as a thread.
2. You may wish to save the above code in a file or paste it in Notebook.

Briefly explain what the above code is doing:

The above code is 2 subroutines that creaties 2 threads where one prints "thread1" and the other "thread2". In thread1 we are printing "in thread1" while true and in thread2 we are calling thread1 and printing "in thread2" There is also a main that calls thread2 and prints out "in main"

List the order in which you expect the text to be displayed by the "writeln" statements:

"in thread1"
"in thread2"
"in main"

**LO2. Use the simulator to compile and run the code.**

**LO3. Display the list of processes/threads and the tree of processes showing the parent/child process relationship.**

a) Compile the above source and load the generated code in memory.

b) Make the console window visible by clicking on the **INPUT/OUTPUT…** button. Also make sure the console window stays on top by checking the **Stay on top** check box.

c) Now, go to the OS simulator window (use the **OS…** button in the CPU simulator window) and create a single process of program *ThreadTest1* in the program list view. For this use the **CREATE NEW PROCESS** button.

d) Make sure the scheduling policy selected is **Round Robin** and that the simulation speed is set at maximum.

e) Hit the **START** button and at the same time observe the displays on the console window.

Briefly explain your observations:

The output was, “in main in thread2in thread1” The first statement was in main and the second was in thread2 and then thread1

How many processes are created?

1 process

Identify, by name, which is a process and which is a thread:

Process: threadtest1 Thread: P1T0, P1T0T1

f) Now, click on the **Views** tab and click on the **VIEW PROCESS LIST…** button. Observe the contents of the window now displaying.

Briefly explain your observations:

This gives you a list of all the process and threads thats currently in the system and the status and memory they’re using.

What do you think the **PPID** field signifies?

It signifies the parent PID of any given process or thread.

Is there equivalent information in MS Windows?

The task manager does the same in MS Windows

g) In the Process List window hit the **PROCESS TREE…** button. Observe the contents of the window now displaying.

Briefly explain your observations:

There’s a root of a tree and then theres a child / “leaves” which are just subtrees of the root.

How are the parent/child process relationships represented?

The parent process is at the top while the child processes are under with dotted lines going under. The child processes are branched under the parent

Identify the parent and the children processes:

Parent: ThreadTest which has a child: P1T0, which has a child P1T0T1

Is there equivalent information in MS Windows?

We don’t have a tree like structure in out window

h) Stop the running processes by repeatedly using the **KILL** button in the OS simulator window.

## LO4. Demonstrate that threads share their parent’s data areas.

We now need to modify the above program statements for the next set of exercises. You can do this in two ways: 1) Modify the existing source, 2) Copy the code and paste it into a new editor window (you can use the **NEW…** button for this in the compiler window). The

required modifications are in bold and underlined. Also make sure that this modified program has a different program name as shown below.

```
program ThreadTest2
     var s1 string(6)
     var s2 string(6)

     sub thread1 as thread
          s1 = "hello1"
          writeln("In thread1")
          while true
          wend
     end sub

     sub thread2 as thread
          call thread1
          s2 = "hello2"
          writeln("In thread2")
          while true
          wend
     end sub

     call thread2
     writeln("In main")
     wait

     writeln(s1)
     writeln(s2)
end
```

**Notes**:

1. The "**wait**" statement allows the parent process to wait for its children to terminate before it continues.

Briefly explain what effect the modifications will have:

The modifications above will have the parent process wait until the children process terminates before it starts executing the code in itself.

a) Compile the above source and load the generated code in memory.

b) Click on the **SYMBOL TABLE…** button in the compiler window. In the displayed window, observe the information on variables **s1** and **s2**.

Make a note below of the data memory addresses for variables **s1** and **s2**:

s1 = 0006

s2 = 0014

c) Make the console window visible by clicking on the **INPUT/OUTPUT…** button. Also make sure the console window stays on top by checking the **Stay on top** check box.

d) Now, go to the OS simulator window and create a single process of program *ThreadTest2* in the program list view. For this use the **CREATE NEW PROCESS** button.

e) Make sure the scheduling policy selected is **Round Robin** and that the simulation speed is set at maximum.

f) Hit the **START** button and after all the displays on the console window are done use the **SUSPEND** button to temporarily suspend the processes. Identify and select the grandparent process.

g) Now, click on the **SHOW MEMORY…** button. This action will display the data area of the selected process. Observe the contents of the memory areas of variables **s1** and **s2** (use the address values you noted down in (b) above).

h) Display the memory areas of the child processes and again observe the contents of the memory areas of variables **s1** and **s2** (note that each string variable's data starts with the first byte set to 03).

Briefly explain what your general observations are and suggest what the significance of your observations is:

They both show the same data memory. It doesn't change because they share the same data space as their parents.

i) Click on the **RESUME** button and stop the running processes by repeatedly using the **KILL** button in the OS simulator window.

**LO5. Modify the source code to create a version not using threads to be able to compare the two versions.**

a) Modify the source code for *ThreadTes1* by removing the two instances of "**as thread**" from the two subroutine declarations. You may wish to call this modified source code *ThreadTest3.*

b) Compile the code and load in memory.

c) Create a process of it and run in the OS simulator.

d) Observe the display in the console window.

Briefly describe what you observe and explain how and why this differs from the results of the previous processes:

There is no longer a thread because we removed that part so now its just like any other function. There is also an infinite loop so it will be stuck in the first function.

e) Stop the running process by using the **KILL** button in the OS simulator window.

**LO6. Simulate a multi-threaded server program**

In the compiler window enter the following source code:

```
program ServerTest
     var p integer

     sub ServiceThread as thread
          writeln("Started service")
          for i = 1 to 30
          next
          writeln("Finished service")
     end sub

     sub ServerThread as thread
          while true
               read(nowait, p)
               select p
                    case '1'
                         call ServiceThread
                    case 'q'
                         break
                    case else
```

```
            end select
        wend
    end sub

    call ServerThread
    wait
end
```

a. Compile the above source, load the generated code in memory and run it.

b. Make a note below of what you observe on the OS simulator window:

I notice that on the OS simulator window we have a thread for our main process and a child process

c. On the console window type **1** four times one after the other and observe the displays on the console.

d. When all the threads finish then make a note of the displays on the console.

When we type it four times it says “started service” after each 1 and then finally displays “finished service” at the end for each 1 entered.

e. Type **q** on the console to stop the server program.

f. Now, modify the above code by removing the “**as thread**” construct from the subroutine *ServiceThread*.

g. Compile the above source, load the generated code in memory and run it.

h. On the console window type **1** four times one after the other and observe the displays on the console.

i. When all the threads finish then make a note of the displays on the console.

The console will display the input 1 with “started service” and then finish with “finished service” before starting service of the next one.

j. Type **q** on the console to stop the server program.

k. Compare the displays in (l) with those in (q) above. Do they differ? If they do then explain below why they differ?

Both displays in i and q differ in when a program is started and finished. While in i the programs are started and then finished, q doesn't start another until the one prior is finished. They differ because since it's not a thread the process can't run simultaneously.