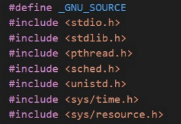
**Operating Systems MileStone 1 Report**

Team: 20

Names:

* Zeina Mohamed Fadel 55-1095
* Habiba Hesham 55-0966
* Ehab Medhat 55-4571
* Abdelrahman ElAby 55-0990
* Abdelrahman ElSamalouty 55-0842
* Hazem Mansour 55-1036

Libraries that we imported

*stdio:* for inputs/outputs

*stdlib:* standard library for C code

*pthread:* for threads

*sched:* for scheduling

*unistd:* for accessing processes’ attributes

*sys/time and sys/resource:* for getting our KPIs

Global Attributes

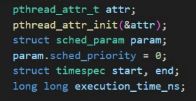


cpu\_set is of type cpu\_set\_t, used as a global variable and will be assigned later with the specific processor that we want all our threads to run on



In the main method, we first empty our set and then adds processor 0 to the set in order to run all the threads on that single processor

The Main Method’s Attributes

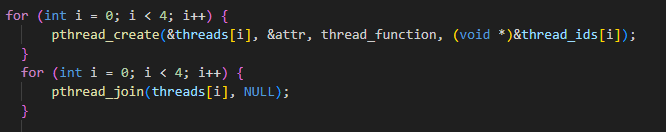
*attr:* carries all details of the thread’s behavior

*param:* structure that contains all the attributes details including the priority that is set to 0 if we’re using the ‘Other’ scheduling policy and set to a number from 1-99 if we’re using FIFO or Round Robin

*timespec / execution\_time\_ns:* used for getting our KPIs

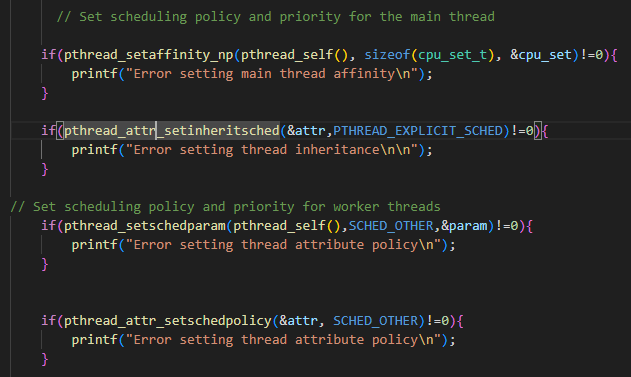
Creating the Threads:





* First, we created an array of type pthread\_t that carries our 4 threads
* We then loop to create all 4 threads in order (1,2,3,4) with the scheduling policy and parameter attr where these threads calls the “thread\_function” (explained later below) method passing to it the threads’ ID
* After creating all 4 threads, we join them with the main method to ensure that all the threads will finish executing before the main method is finished

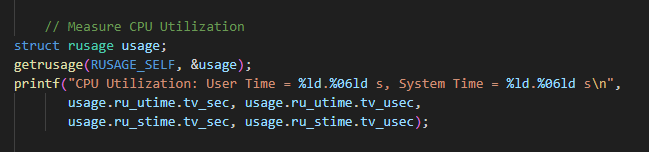
Setting The Scheduling Policies:

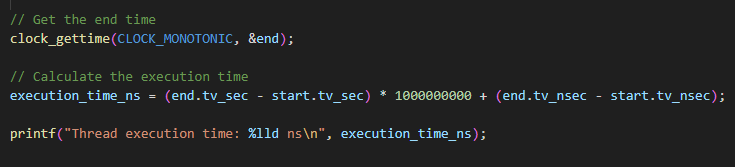


* First, we set the affinity of the main method to the processor assigned in the cpu\_set and print an error message if it doesn’t succeed
* Second, allows the attribute to work with its own attributes and not inherit from the main method by setting the inheritance mode to “EXPLICIT” and print an error message if it doesn’t succeed
* Third, Setting the scheduling policy and param of the main method to the specified scheduling policy (SCHED\_OTHER, SCHED\_FIFO, SCHED\_RR) and print an error message if it doesn’t succeed
* Fourth, Setting the scheduling policy of the attr to the specified scheduling policy (SCHED\_OTHER, SCHED\_FIFO, SCHED\_RR) and print an error message if it doesn’t succeed

KPIs:

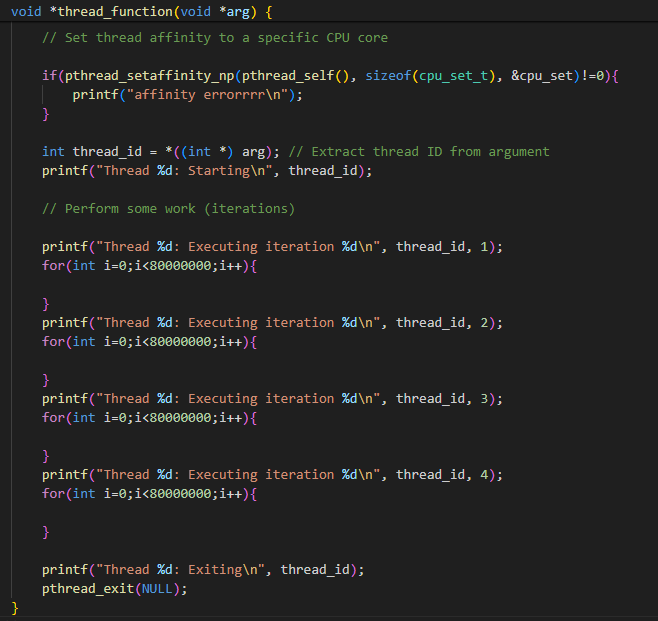






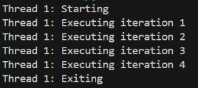
* Start time records the beginning of the main’s execution, the end time records the ending of the main’s execution and the difference records the total execution time
* We also get CPU usage time to determine which policy is more efficient

The function that our threads will execute



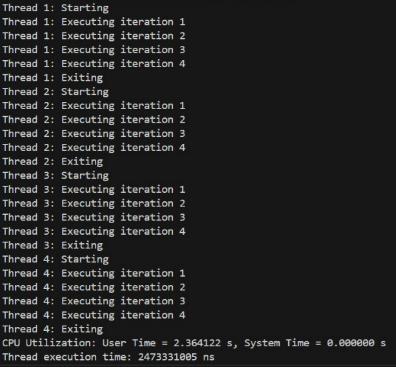
* First, we set affinity of our threads to a specific CPU core that is stored in cpu\_set and print an error message in case it does not succeed
* Next, we extract the thread ID from the function’s argument
* Then, the first print statement is executed printing “starting” when the thread starts execution
* Then, 4 print statements are written
* Between those 4 print statements, we loop for 80000000 iterations in order to increase the execution time of the function so that it would be greater than the quantum
* Lastly, we print “exiting” when the thread is done executing

The output of this function for thread 1 for example should look like this:

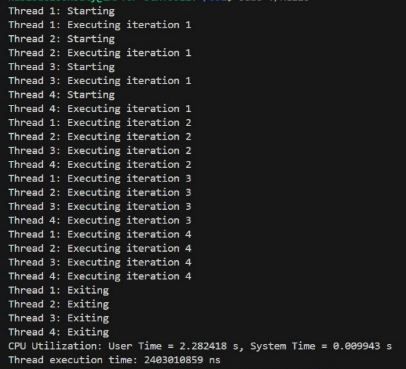


Scheduling Policies Outputs:

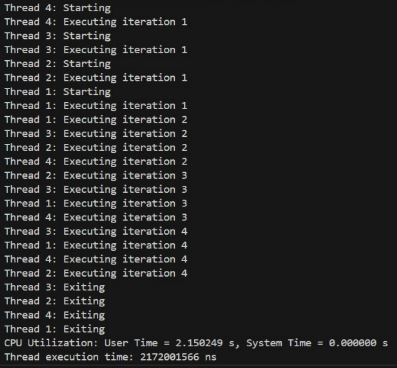
1. FIFO:



1. Round Robin:



1. Other:



Comparing between different scheduling policies:

All scheduling policies have approximately the same time for execution with minor differences where the ‘Other’ is the fastest and most efficient then comes the ‘Round Robin’ and lastly ‘FIFO’