**PID Manager**

**Subject = Operating Systems Design**

**Major = ITM**

**Student ID = 17102062**

**Name = Lee Yong Hoon(이용훈)**

Table

**1. How you can execute**

**2. Execution screenshots**

**Case1 : using getPID() / created thread number is under MAX\_PID-MAX\_PID+1**

**Case2 : using getPID() / created thread number is over MAX\_PID-MAX\_PID+1**

**Case3 : using getPIDWait() / created thread number is under MAX\_PID-MAX\_PID+1 (wait not happening)**

**Case4 : using getPIDWait() / created thread number is over MAX\_PID-MAX\_PID+1 (wait not happening)**

**Case5 : using getPIDWait() / created thread number is over MAX\_PID-MAX\_PID+1 (wait happening)**

**3. Code analysis**

**1. global variable**

**2. Main()**

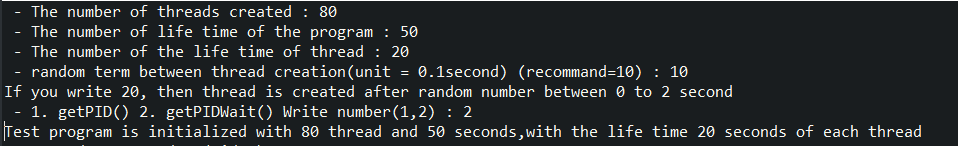
**3. run()**

**4. getPID()**

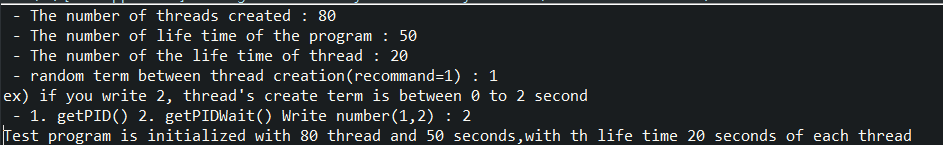
**5. getPIDWait()**

**6. releasePID()**

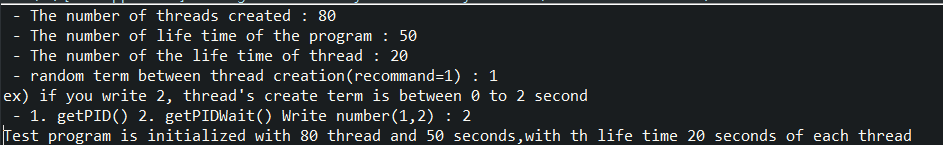
1. How you can execute



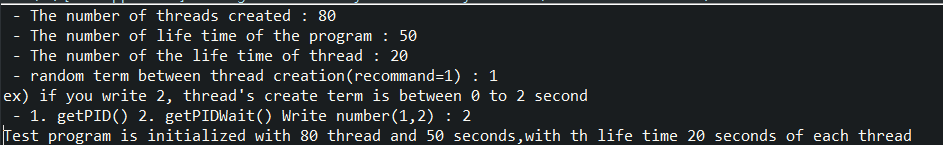
**1. you write the number of threads created**



**2. you write the number of life time of the program**



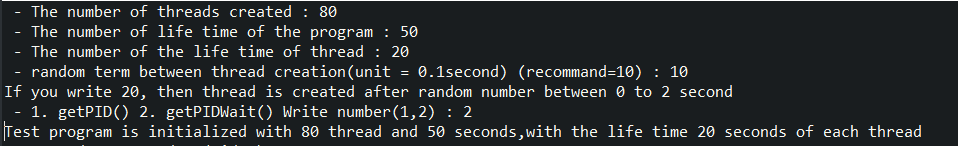
**3. you write the number of life time of thread**



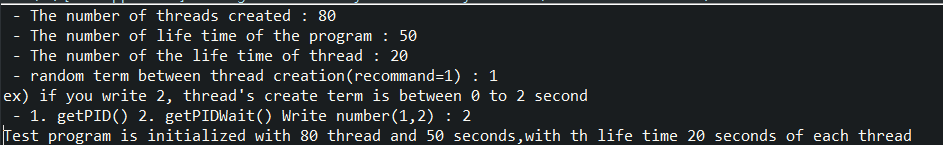
**4. you write the random term number between thread creation**

**If you write 2, threads create term Is between 0 to 2 second.**

**The number is random value between 0 to 2.**

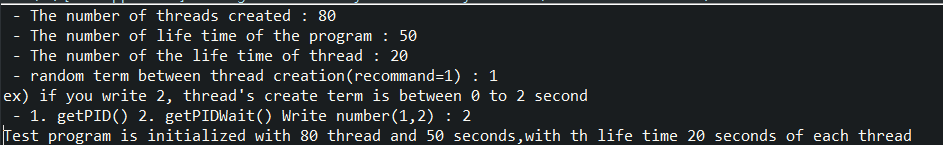


**5. you write number 1 or 2. 1 is using getPID() and 2 is using getPIDWait()**



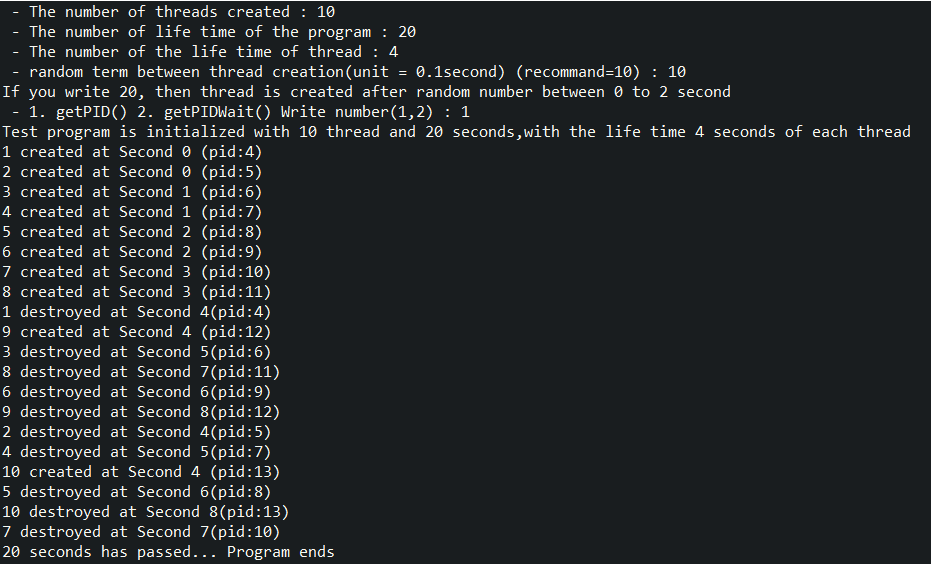
**6. you can see that “Test program is initialized with 80 thread and 50 seconds, with the life time 20 seconds of each thread”**

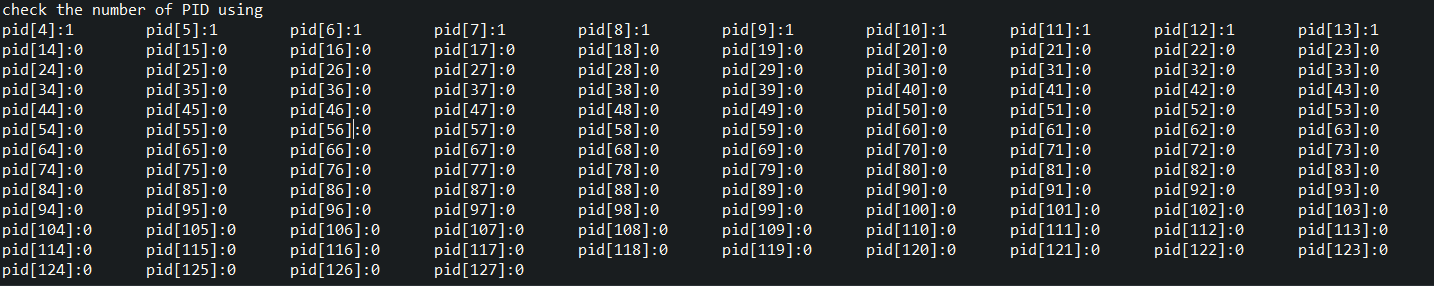
**And automatically run program and after time ends, it terminate**



Execution screenshots

**Case1 : using getPID() / created thread number is under MAX\_PID-MAX\_PID+1**

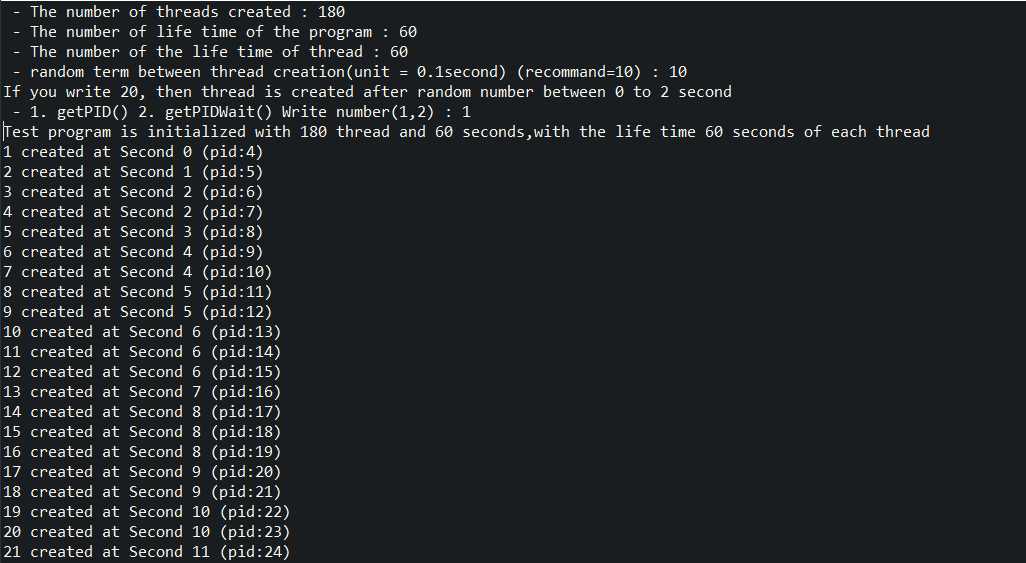




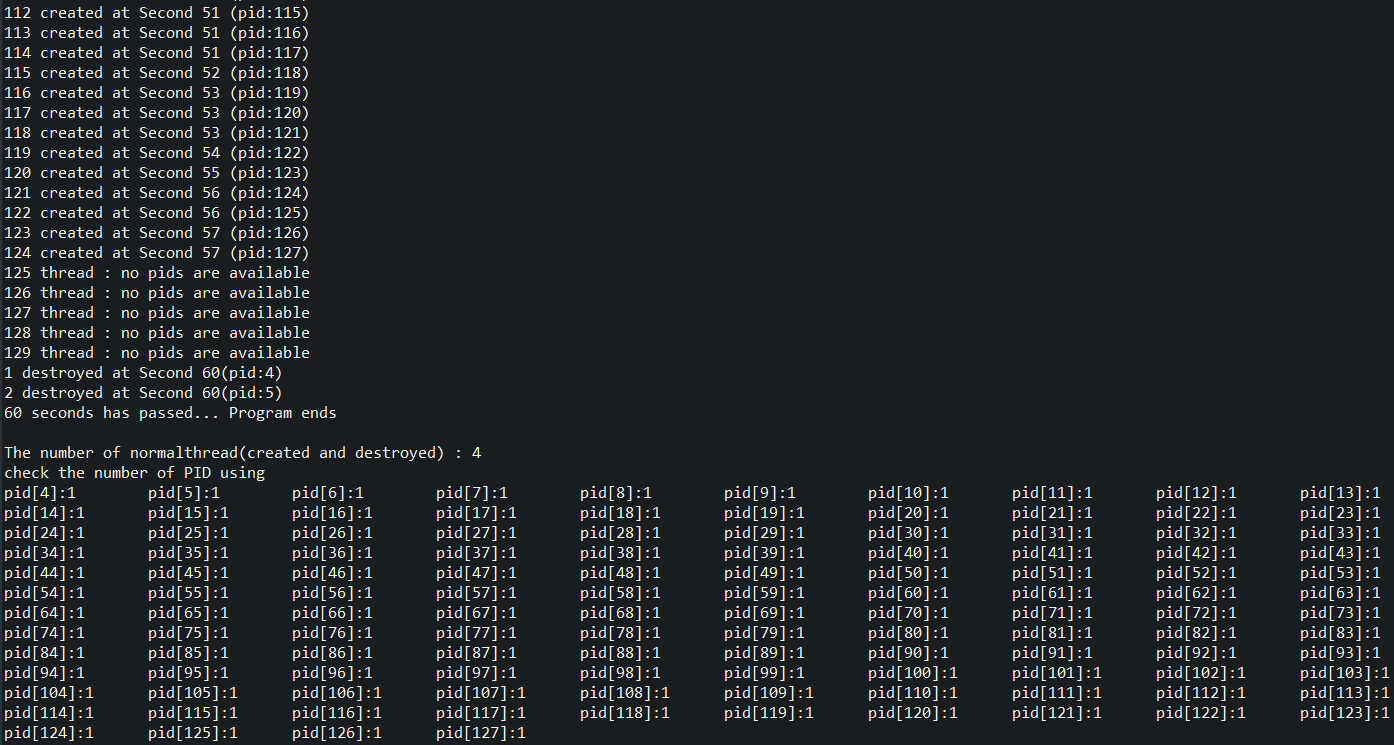
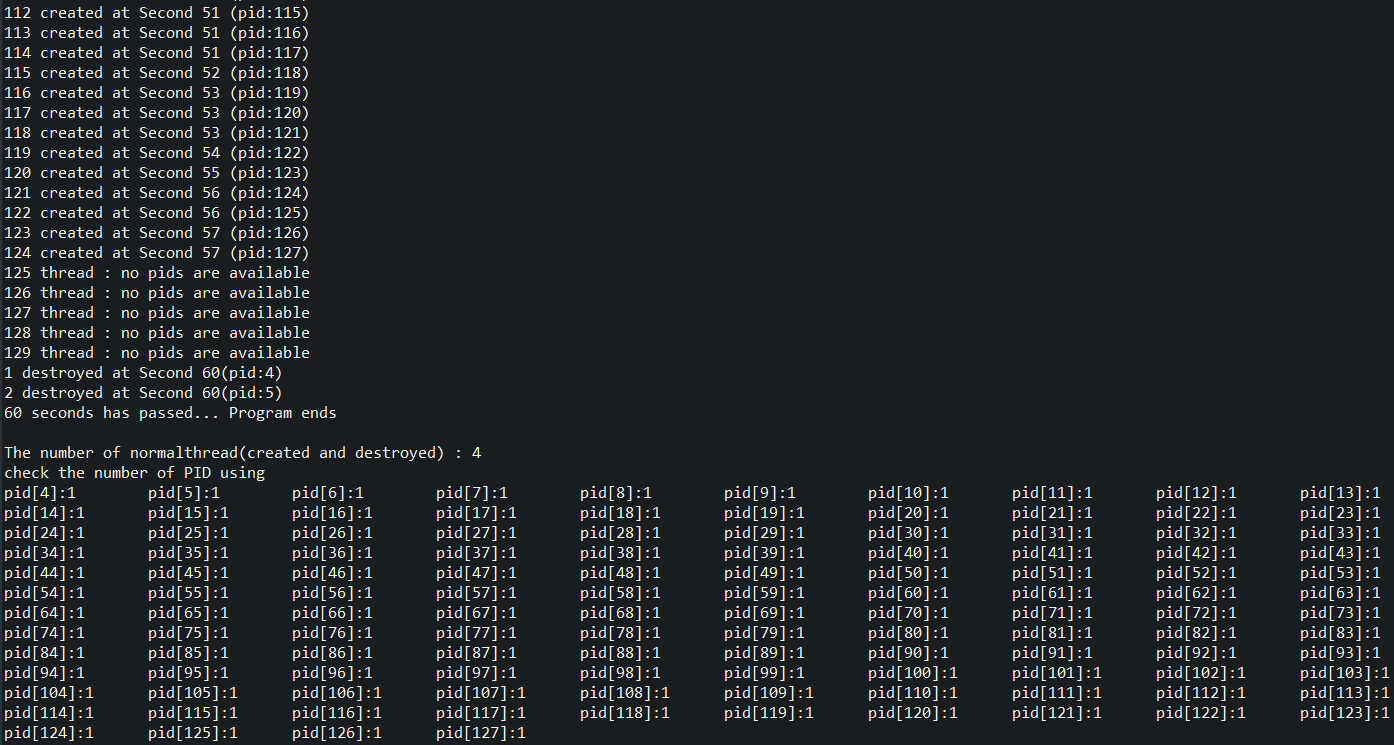
**You can see 10 threads are created and destroyed.**

**you can check the number of PID using.**

**Case2 : using getPID() / created thread number is over MAX\_PID-MAX\_PID+1**



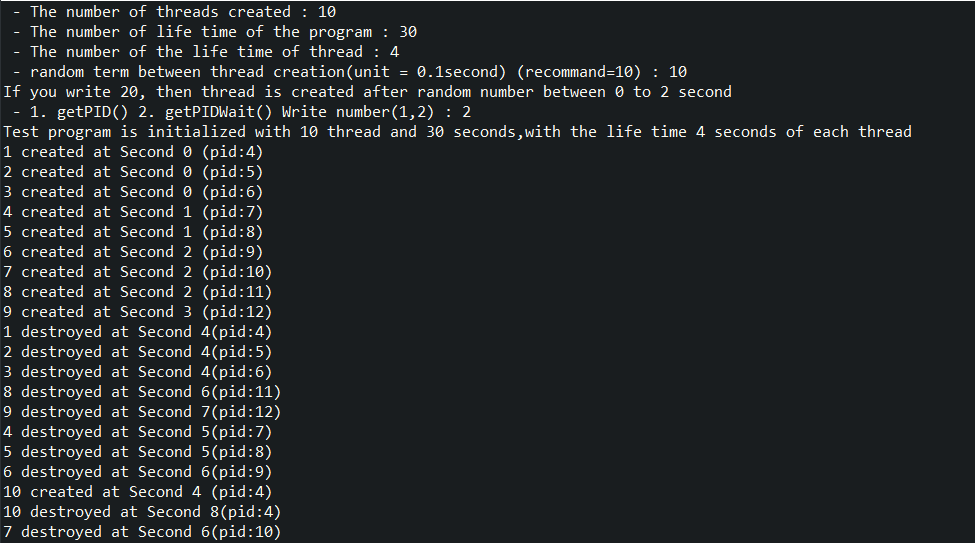
…..skip----

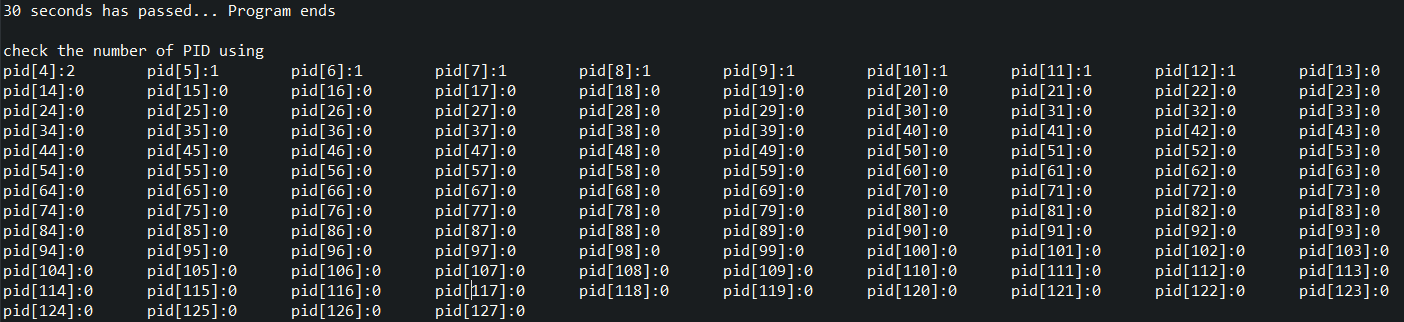


**After allocate all pid, since 125 thread show that no pids are available.**

**And you can see number of PID using that all using number is 1 per each pid**

**Case3 : using getPIDWait() / created thread number is under MAX\_PID-MAX\_PID+1 (wait not happening)**

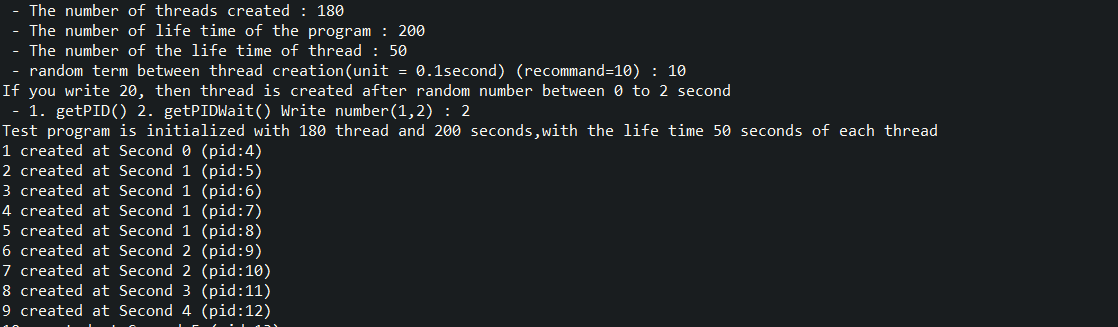




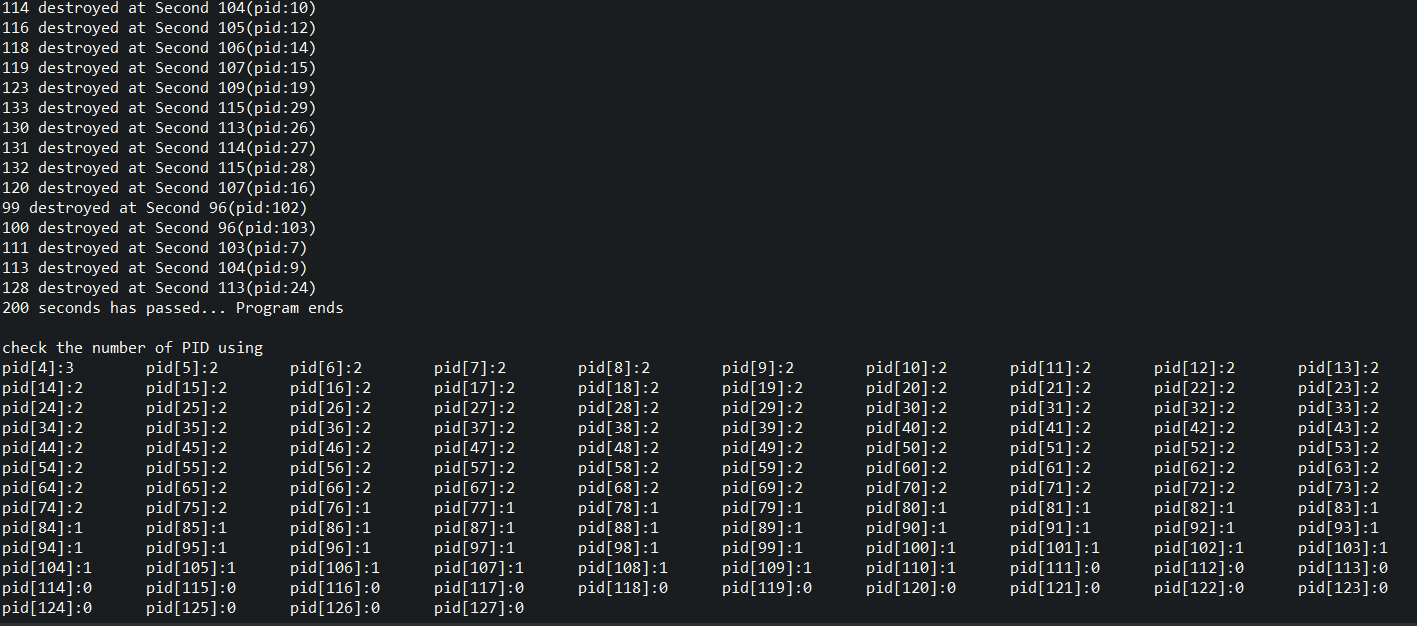
**You can see that if one thread(pid 4) terminated, next created thread is allocated pid 4.**

**You can see that in the number of PID using, pid 4 is used twice.**

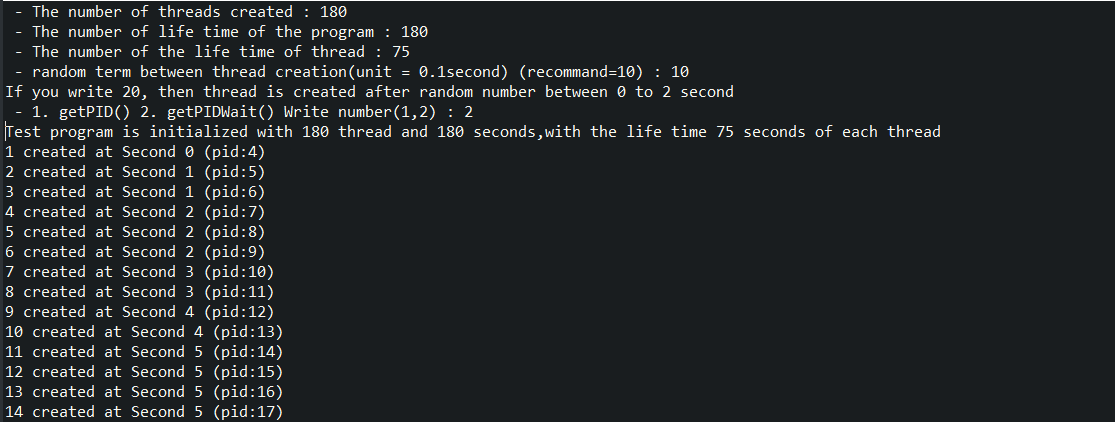
**Case4 : using getPIDWait() / created thread number is over MAX\_PID-MAX\_PID+1 (wait not happening)**



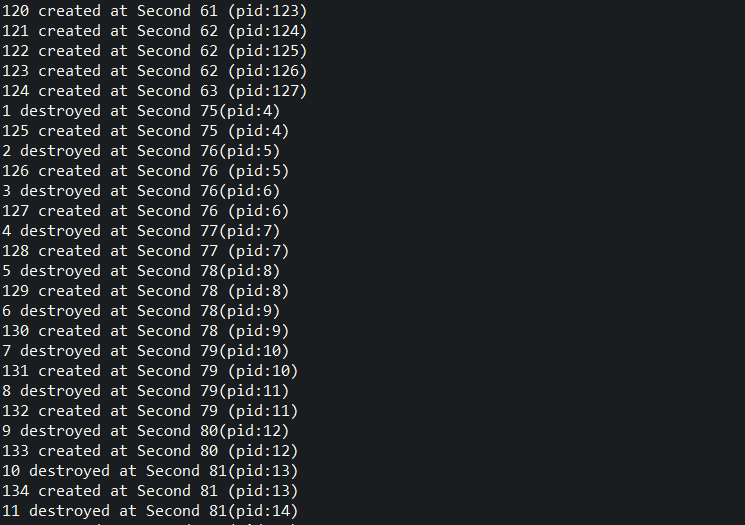
…skip…



**Case5 : using getPIDWait() / created thread number is over MAX\_PID-MAX\_PID+1 (wait happening)**



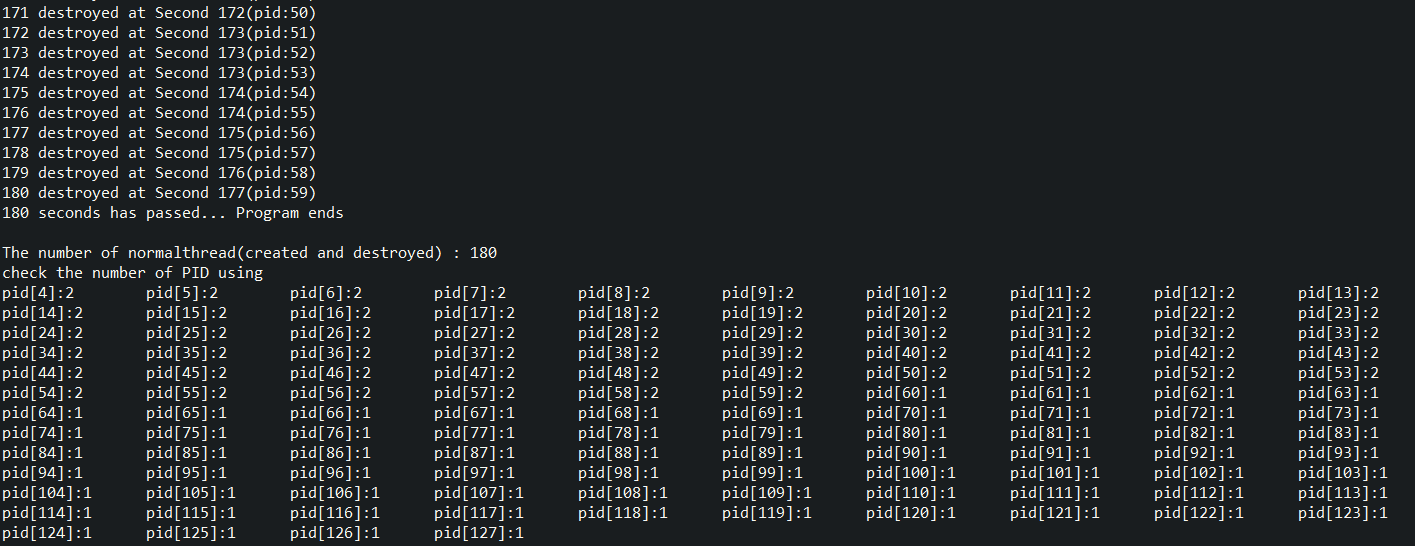
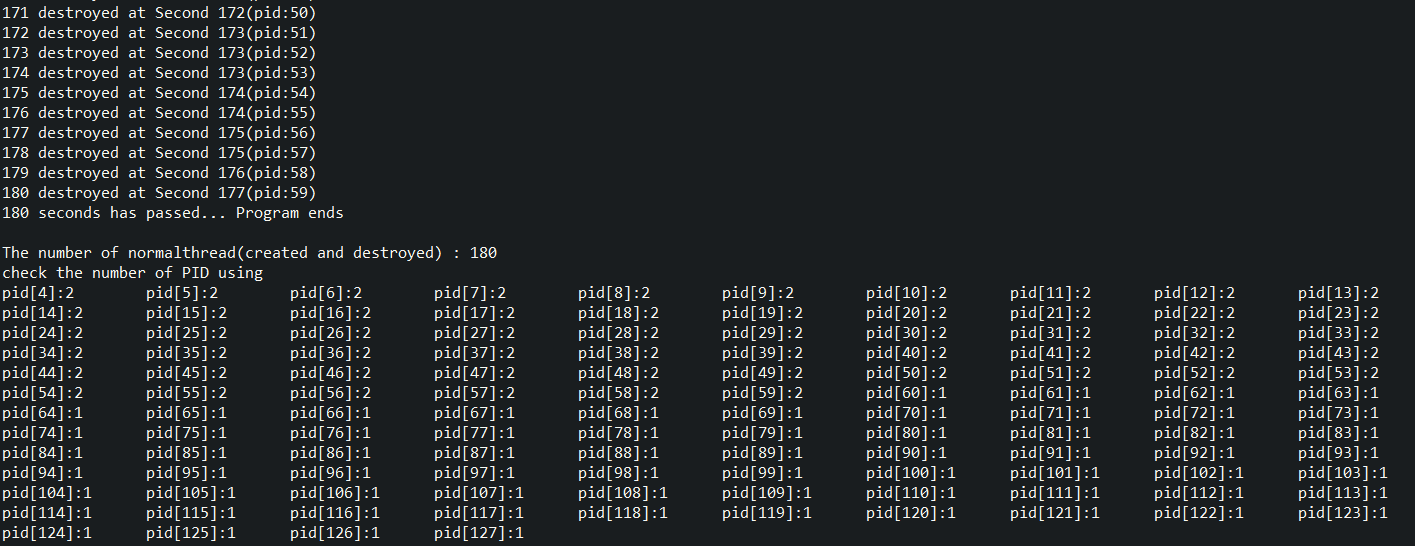
…skip…



**124 created at second 63(pid:127) and**

**125 created at second 75(pid:4).**

**In 10 second, wait happens and after 10 seconds, thread is destroyed and release pid and allocate released pid to wait thread in order( to 125, 126 …)**

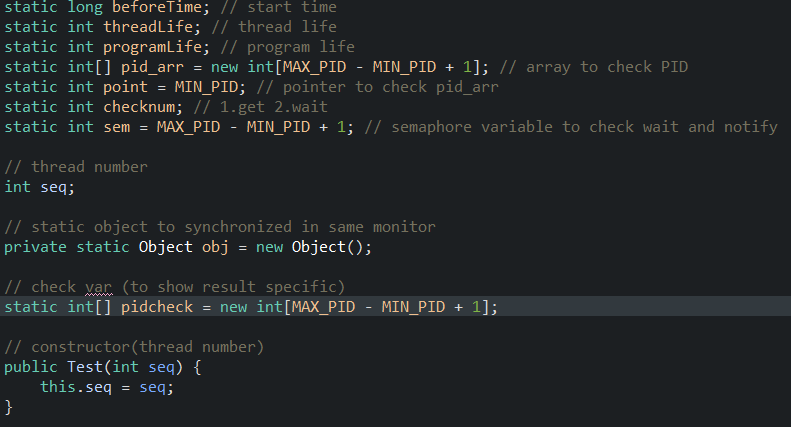


**The number of created and destroyed thread is 180**

**And you can check the number of PID using**

Code Explanation

**1. global variable**



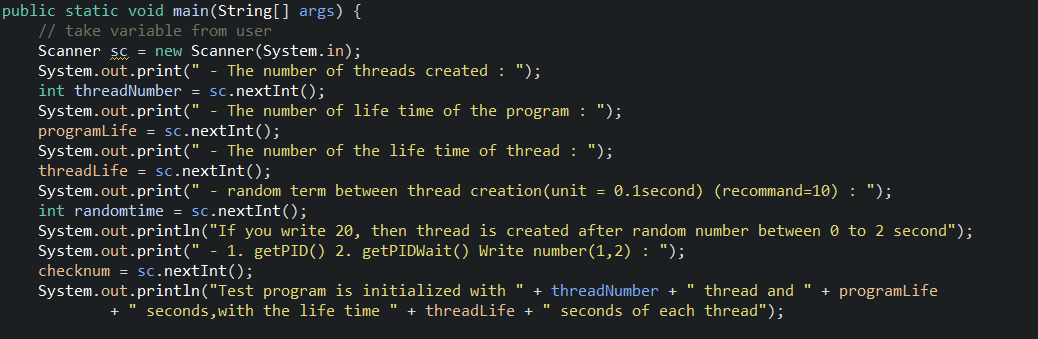
**I use private static Object obj to use wait and notify in same monitor**

**Therefore you can see synchronized(obj) in getPIDWait() and releasePID(int pid)**

**Also I set pidcheck and normalcheck to show more detail after program end**

**Pidcheck is showing how much pid used**

**2. Main()**



**You can write number**

**1. the number of threads created**

**2. the number of life time of the program**

**3. the number of the life time of thread**

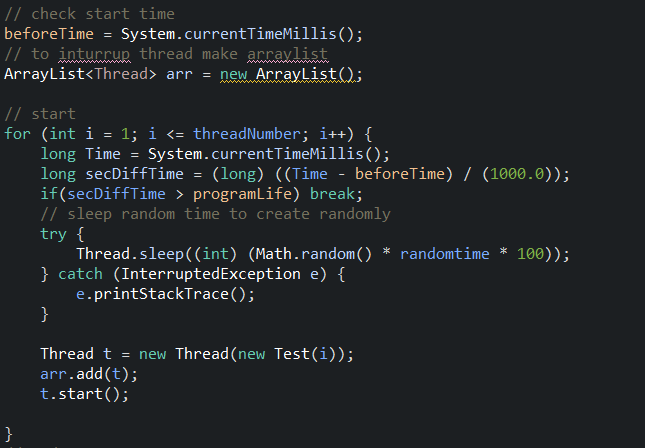
**4. the number of random term between thread creating(unit = 0.1second)(recommend=10)**

**This means after one thread created, there is some random time to sleep to create thread randomly.**

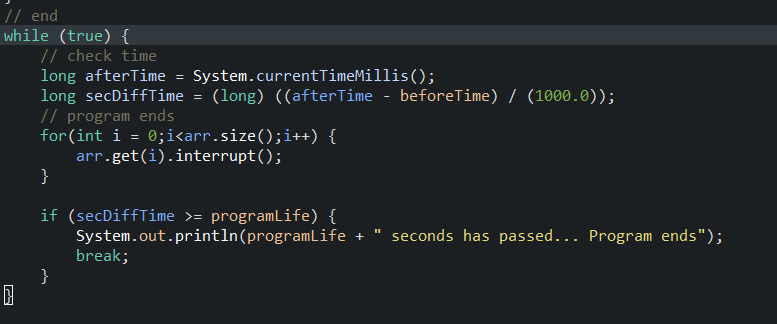
**If you write 20, then it is created after random second between 0 to 2 second.**

**To get result more fast, I recommend you to set 5~10.**

**5. you can choose between 1.getPID() and 2.getPIDWait()**



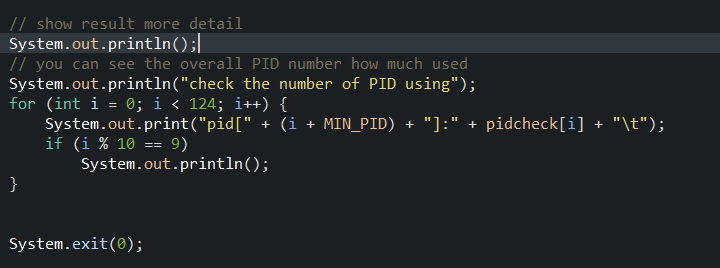
**Using for and using randomtime from your input, thread is created**



**This is end part**

**When time is over, all thread are stopped by using interrupt**

**After checking program time, print (“ㅁㅁ second has passed… program ends”)**



**To show more detail, you can see and overall status of pid using**

**After showing detail, system exit.**

**3. run()**



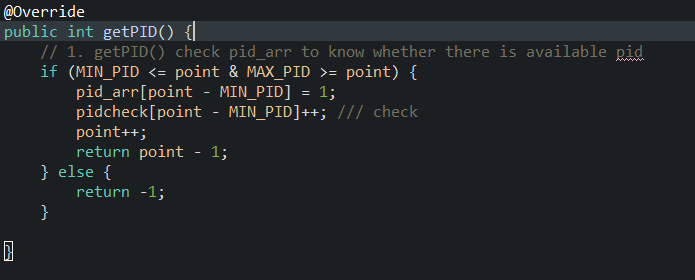
**Using check number it is classify whether it uses getPID() or getPIDWait()**

**Check time and print that thread is created**

**Check time and print that thread is destroyed**

**And releasePID.**

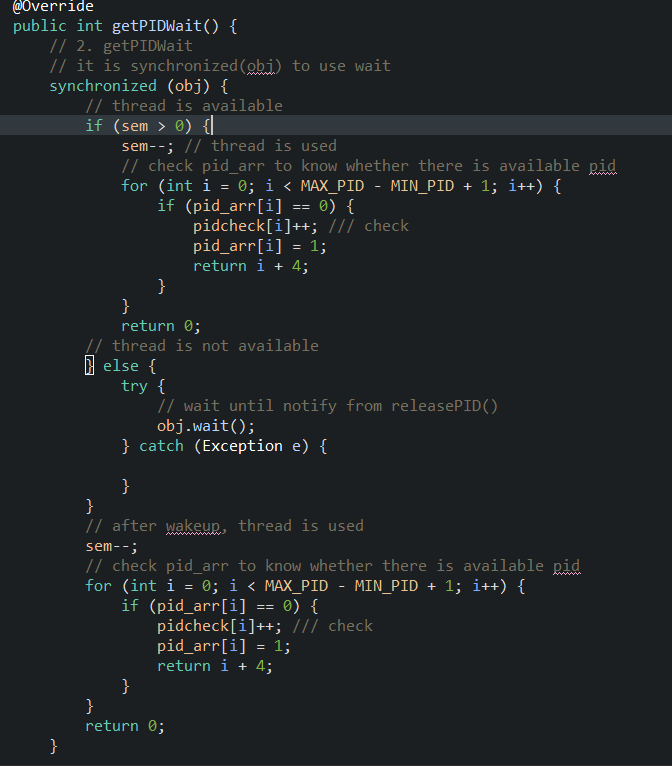
**4. getPID()**



**Check Pid\_arr to find whether there is available pid**

**If not, return -1**

**5. getPIDWait()**





**All Code is in synchronized(obj) block**

**(sem variable : how many thread can be used)**

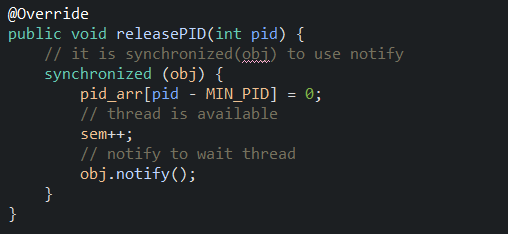
**If sem variable is bigger than 0, it means thread can be allocated pid.**

**If sem variable is smaller or equal than 0, it means thread can not be allocated pid.**

**Thread wait by using obj.wait()**

**After wakeup, sem--(allocate pid to thread) and allocate pid to thread**

**6. releasePID()**



**Code is in synchronized(obj) block.**

**After thread is destroyed, pid\_arr[pid-MIN\_PID] is changed to 0 and sem++ and obj.notify() to wake up wait thread**