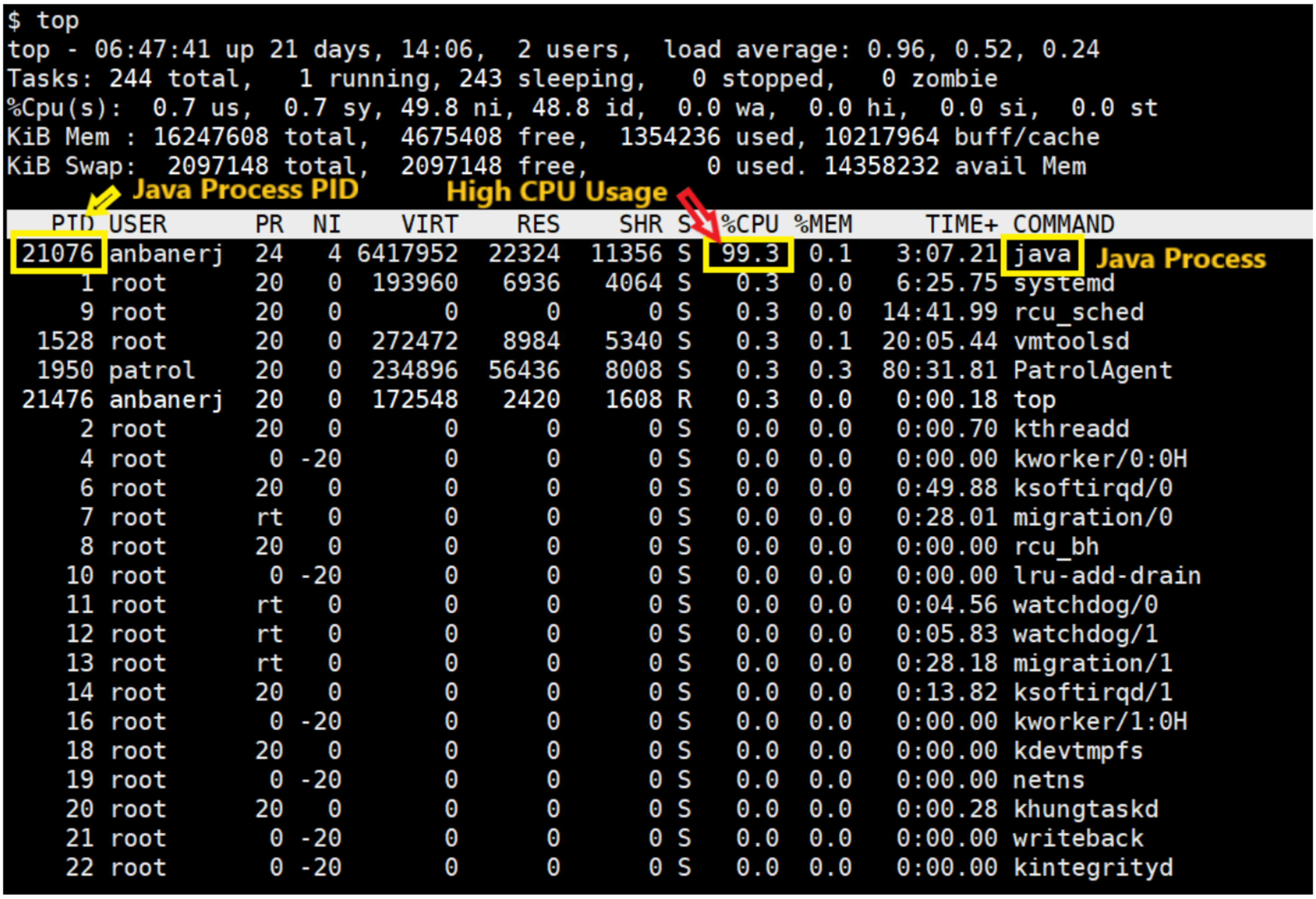
# How to Troubleshoot Java High CPU Usage Issues in Linux?

High CPU in java applications are due to huge volume of incoming requests or some intensive task performed within the code. To troubleshoot such issues, follow the below simple steps to correlate the CPU threads called LWPs (Light Weight Processes) and the Java threads.

Step1.

Execute the top command to detect the exact Java process ID (PID) consuming High CPU



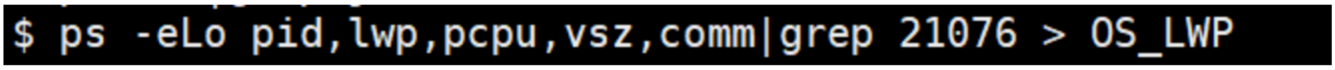
```
$ top
top - 06:47:41 up 21 days, 14:06,  2 users,  load average: 0.96, 0.52, 0.24
Tasks: 244 total,   1 running, 243 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.7 us,  0.7 sy, 49.8 ni, 48.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 16247608 total,  4675408 free,  1354236 used, 10217964 buff/cache
KiB Swap:  2097148 total,  2097148 free,        0 used. 14358232 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
21076 anbanerj  24   4 6417952  22324  11356 S  99.3  0.1   3:07.21 java
    1 root      20   0  193960   6936   4064 S   0.3  0.0   6:25.75 systemd
    9 root      20   0       0      0      0 S   0.3  0.0  14:41.99 rcu_sched
 1528 root      20   0  272472   8984   5340 S   0.3  0.1  20:05.44 vmtoolsd
 1950 patrol    20   0  234896  56436   8008 S   0.3  0.3  80:31.81 PatrolAgent
21476 anbanerj  20   0  172548   2420   1608 R   0.3  0.0   0:00.18 top
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.70 kthreadd
    4 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
    6 root      20   0       0      0      0 S   0.0  0.0   0:49.88 ksoftirqd/0
    7 root      rt   0       0      0      0 S   0.0  0.0   0:28.01 migration/0
    8 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
   10 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 lru-add-drain
   11 root      rt   0       0      0      0 S   0.0  0.0   0:04.56 watchdog/0
   12 root      rt   0       0      0      0 S   0.0  0.0   0:05.83 watchdog/1
   13 root      rt   0       0      0      0 S   0.0  0.0   0:28.18 migration/1
   14 root      20   0       0      0      0 S   0.0  0.0   0:13.82 ksoftirqd/1
   16 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:0H
   18 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kdevtmpfs
   19 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 netns
   20 root      20   0       0      0      0 S   0.0  0.0   0:00.28 khungtaskd
   21 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 writeback
   22 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kintegrityd
```

Step2.

Execute the following command to get the LWP details.

```
ps -eLo pid,lwp,pcpu,vsz,comm | grep 21076 > OS_LWP
```

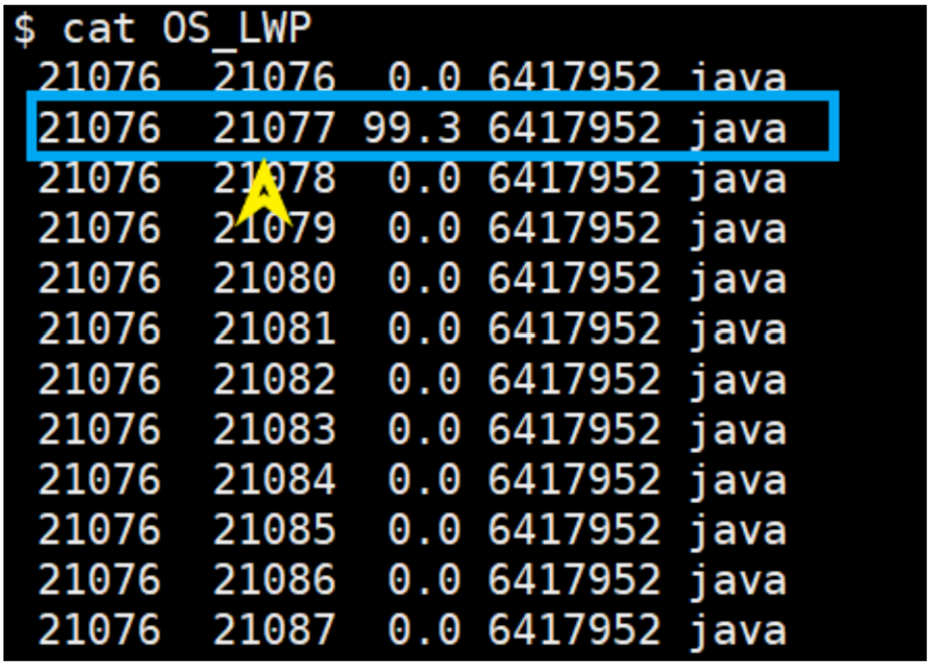


```
$ ps -eLo pid,lwp,pcpu,vsz,comm|grep 21076 > OS_LWP
```

Step3.

Output from Step2 displays all the Light Weight Process IDs (LWP) associated with the Java Process (PID = 21076)

LWPs denote the CPU threads. From below output, the problematic CPU thread ID is 21077



```
$ cat OS_LWP
 21076  21076   0.0 6417952 java
 21076  21077 99.3 6417952 java
 21076  21078   0.0 6417952 java
 21076  21079   0.0 6417952 java
 21076  21080   0.0 6417952 java
 21076  21081   0.0 6417952 java
 21076  21082   0.0 6417952 java
 21076  21083   0.0 6417952 java
 21076  21084   0.0 6417952 java
 21076  21085   0.0 6417952 java
 21076  21086   0.0 6417952 java
 21076  21087   0.0 6417952 java
```

Step4.

Get a thread dump of the Java process using any of the below commands

```
kill -3 PID > thread_dump
```

OR

```
jstack PID > jstack_thread_dump
```

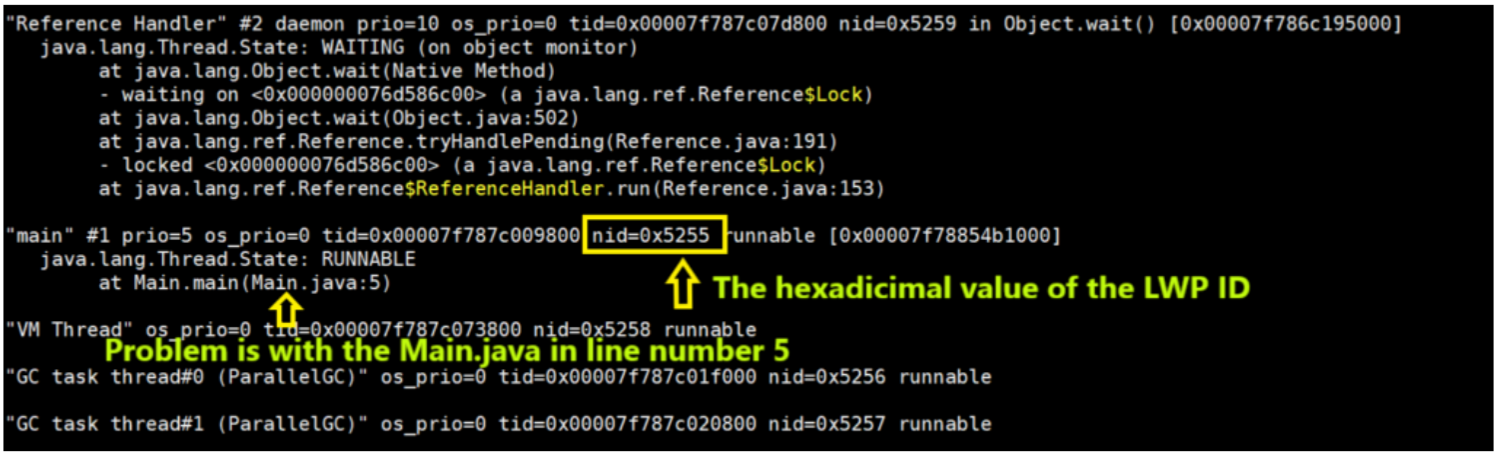
Step5.

The LWP ID is in decimal format, whereas the corresponding ID in java thread dump is in hexadecimal format. Converting LWP ID (=21077) to hexadecimal yields 5255, which is represented as 0x5255

Step6.

Search for the hexacimal ID (0x5255) in the thread dump, and it shows the exact Java thread causing the high CPU. Thread dumps log these CPU threads as nid (Native Thread ID)

From below image, we clearly see the thread with nid=0x5255, and is currently in the runnable state. It also shows the Java Class (Main.java) and the line number (=5) which causes the High CPU Usage.



```
"Reference Handler" #2 daemon prio=10 os_prio=0 tid=0x00007f787c07d800 nid=0x5259 in Object.wait() [0x00007f786c195000]
   java.lang.Thread.State: WAITING (on object monitor)
        at java.lang.Object.wait(Native Method)
        - waiting on <0x000000076d586c00> (a java.lang.ref.Reference$Lock)
        at java.lang.Object.wait(Object.java:502)
        at java.lang.ref.Reference.tryHandlePending(Reference.java:191)
        - locked <0x000000076d586c00> (a java.lang.ref.Reference$Lock)
        at java.lang.ref.Reference$ReferenceHandler.run(Reference.java:153)

"main" #1 prio=5 os_prio=0 tid=0x00007f787c009800 nid=0x5255 runnable [0x00007f78854b1000]
   java.lang.Thread.State: RUNNABLE
        at Main.main(Main.java:5)

"VM Thread" os_prio=0 tid=0x00007f787c073800 nid=0x5258 runnable

"GC task thread#0 (ParallelGC)" os_prio=0 tid=0x00007f787c01f000 nid=0x5256 runnable

"GC task thread#1 (ParallelGC)" os_prio=0 tid=0x00007f787c020800 nid=0x5257 runnable
```

The hexadicimal value of the LWP ID

Problem is with the Main.java in line number 5

So, now you know how to correlate the CPU and Java threads to find the exact cause of HIgh CPU Usage in Linux.

Code.

```
class Main{
    public static void main(String args[]){
        Double i= 2342324242424242422342.0;
        Double k= 22342342424242424242342.0;
        for(int j= 0;j<i;j++){
            Double m = Math.pow(i,k);
        }
    }
}
```