```
# -*- coding: iso-latin-1 -*-

import time
import Queue
import threading
import argparse
from factorize import factorize_naive

def factorizer_worker(job_q, res_q):
    while not job_q.empty():
        try:
            job = job_q.get()
            out_dict = {n: factorize_naive(n) for n in job}
            res_q.put(out_dict)
        except Queue.Empty:
            return

def make_nums(base, count):
    return [base + i * 2 for i in range(count)]

def parse_args(args):

    parser = argparse.ArgumentParser(
        description="Multiprocess factorization.")

    add = parser.add_argument

    add('-b', '--base-number', type=int, default=999999,
        help="The smallest number to factorize.")

    add('-n', '--numbers-count', type=int, default=999,
        help="The number of numbers to factorize.")

    add('-t', '--threads-count', type=int, default=5,
        help="The number of threads to spawn.")

    return parser.parse_args(args)

def main(args):

    options = parse_args(args)

    job_q = Queue.Queue()
    res_q = Queue.Queue()

    print("Factorizing %d odd numbers starting from %d using %d threads." % (
        options.numbers_count, options.base_number, options.threads_count))

    nums = make_nums(options.base_number, options.numbers_count)
    chunksize = 43
    for i in range(0, len(nums), chunksize):
        job_q.put(nums[i:i + chunksize])

    tt = list()
    for i in range(options.threads_count):
        t = threading.Thread(
                target=factorizer_worker,
                args=(job_q, res_q))
        tt.append(t)

    print("Starting threads.")
    for t in tt: t.start()
    for t in tt: t.join()

    print("Collecting results.")
    d = dict()
    while not res_q.empty():
        d.update(res_q.get())
```

```
    print("Printing results.")
    for k in sorted(d):
        print k, d[k]

if __name__ == '__main__':

    import sys
    sys.exit(main(sys.argv[1:]))
```