```
# -*- coding: iso-latin-1 -*-
import multiprocessing as mp
from factorize import factorize_naive
import socket
import os
import select

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

def server(port, max_queue=5):

    # Questa è la socket su cui accetto le connessioni
    main_sock = socket.socket(socket.AF_INET,
                              socket.SOCK_STREAM)
    main_sock.bind( ("", port) )
    main_sock.listen(max_queue)

    # dizionario dei risultati
    res = dict()

    # lista degli N numeri da fattorizzare, divisa poi in blocchi di
    # block_size elementi.
    N = 999
    nums = make_nums(99999999999, N)
    block_size = 43
    blocks = [nums[i:i + block_size]
              for i in range(0, len(nums), block_size)]

    # socket da cui "leggo". All'inizio c'è solo main_sock, poi si
    # aggiungono quelle aperte per ciascuna connessione.
    read_socks = [main_sock]

    while read_socks:
        # socket pronte per leggere, scrivere, ...
        rr, ww, ee = select.select(read_socks, [], [])
        for r in rr:
            if r == main_sock:
                # connessione iniziale di un client, creo una nuova
                # socket e la aggiungo alle socket da cui leggere.
                ns, addr = r.accept()
                print ("Server: new client at %s" % str(addr))
                read_socks.append(ns)

                # gli mando il primo blocco e lo cancello
                ns.send(repr(blocks[0]))
                del blocks[0]

                # se non ci sono più blocchi il server ha finito
                if not blocks:
                    print ("Server: done")
                    read_socks.remove(main_sock)
                    main_sock.close()
            else:
                # connessione di un client specifico, da cui devo
                # leggere il risultato della fattorizzazione.

                data = r.recv(5024)       # BAD!!!
                data = eval(data)
                res.update(data)
                r.close()
                read_socks.remove(r)
                print ("Server: client done")
    main_sock.close()
    return res
```

```
def client(ip, port):

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    try:
        sock.connect( (ip, port) )
    except Exception as e:
        print ("Client: server closed ... leaving")
        sys.exit(0)

    try:
        # Leggo la lista di numeri da fattorizzare, rappresentata come
        # stringa da "repr".
        data = sock.recv(1024)
        if not data:
            return
        # Passo dalla stringa all'oggetto (undo di repr)
        data = eval(data)
        print("Client %s\ngot range: %d-%d" % (
            os.getpid(), data[0], data[-1]))

        # Calcolo i fattori, li metto nel dizionario come valori dei
        # numeri, poi mando il repr del dizionario al server e poi
        # saluto e vado.
        d = dict()
        for n in data:
            ff = factorize_naive(n)
            d.update({n: ff})
        sock.send(repr(d))
    finally:
        sock.close()

def check_res(res):
    ok_bad = [0,0]
    for k,v in res.items():
        t = 1
        for n in v:
            t *= n
        c = (k == t)
        ok_bad[1-c] += 1
        s = ["ERROR", "OK"][c]
    print ("OK: %d BAD: %d" % tuple(ok_bad))

if __name__ == '__main__':

    import sys
    import random
    port = 5000
    args = sys.argv[1:]
    if not args:
        sys.exit(1)
    if args[0] == "-s":
        res = server(port)
        for k,v in res.items():
            print k, 40 * "-"
            print v

        check_res(res)
    if args[0] == "-c":
        if len(args) > 1:
            ip = args[1]
        else:
            ip = "127.0.0.1"
        client(ip, port)
```