```
#!/usr/bin/env python

# author Philippe Raipin
# author Eric Mourgaya
# licence: apache v2
# Alexis KOALLA: pymongo update for replicaset

# need package python-dev: sudo apt-get install python-dev
# need module psutil: download module tar.gz, unzip, python
setup.py install
# need pip to install pymongo: http://www.pip-
installer.org/en/latest/installing.html
# need pymongo module: download module,


from pymongo import MongoClient
from pymongo import MongoReplicaSetClient
from pymongo.read_preferences import ReadPreference

import time
import datetime

import re

# psutil to perform system command
import psutil
import pkg_resources
psutil_version = pkg_resources.get_distribution
("psutil").version.split(".")[0]

# for ceph command call
import subprocess

import os
import sys
import traceback
import getopt
import socket
from daemon import Daemon

import json
from StringIO import StringIO

from bson.dbref import DBRef

from threading import Thread, Event

import signal


# from bson.objectid import ObjectId
# db.col.find({"_id": ObjectId(obj_id_to_find)})
```

```
configfile = "/opt/inkscope/etc/inkscope.conf"
runfile = "/var/run/sysprobe/sysprobe.pid"
logfile = "/var/log/inkscope/sysprobe.log"


# load the conf (from json into file)
def load_conf():
    datasource = open(configfile, "r")
    data = json.load(datasource)

    datasource.close()
    return data


# get a field value from global conf according to the
specifpsutil_versionied ceph conf
def ceph_conf_global(cephConfPath, field):
    p = subprocess.Popen(
        args=[
            'ceph-conf',
            '-c',
            cephConfPath,
            '--show-config-value',
            field
            ],
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE)
    outdata, errdata = p.communicate()
    if len(errdata):
        raise RuntimeError('unable to get conf option %s: %s' %
(field, errdata))
    return outdata.rstrip()


# mem
def pick_mem(hostname, db):
    print str(datetime.datetime.now()), "-- Pick Mem"
    sys.stdout.flush()
    res = psutil.virtual_memory()
    # convert to base
    mem4db = {
        "timestamp": int(round(time.time() * 1000)),
        "total": res.total,
        "used": res.used,
        "free": res.free,
        "buffers": res.buffers,
        "cached": res.cached,
        "host": DBRef("hosts", hostname)
        }
    mem_id = db.memstat.insert(mem4db)
    db.hosts.update({"_id": hostname}, {"$set": {"mem": DBRef
```

```
("memstat", mem_id)}})


# swap
def pick_swap(hostname, db):
    print str(datetime.datetime.now()), "-- Pick Swap"
    sys.stdout.flush()
    res = psutil.swap_memory()
    # convert to base
    swap4db = {"timestamp": int(round(time.time() * 1000)),
               "total": res.total,
               "used": res.used,
               "free": res.free,
               "host": DBRef("hosts", hostname)
               }
    swap_id = db.swapstat.insert(swap4db)
    db.hosts.update({"_id": hostname}, {"$set": {"swap": DBRef
("swapstat", swap_id)}})


# /var/lib/ceph/osd/$cluster-$id
# partitions
def get_partitions(hostname):
    _partitions = psutil.disk_partitions()
    parts = []
    for p in _partitions:
        res = {"_id": hostname+":"+p.device,
               "dev": p.device,
               "mountpoint": p.mountpoint,
               "fs": p.fstype,
               "stat": None
           }
        parts.append(res)
    return parts



def pick_partitions_stat(hostname, db):
    print str(datetime.datetime.now()), "-- Pick Partition Stats"
    sys.stdout.flush()
    _partitions = psutil.disk_partitions()
    for p in _partitions:
        # disk usage
        part_stat = psutil.disk_usage(p.mountpoint)

        res = {
               "timestamp": int(round(time.time() * 1000)),
               "total": part_stat.total,
               "used": part_stat.used,
               "free": part_stat.free,
               "partition": DBRef("partitions",
hostname+":"+p.device)
```

```
                }
        partstat_id = db.partitionstat.insert(res)
        db.partitions.update({"_id": hostname+":"+p.device},
{"$set": {"stat": DBRef("partitionstat", partstat_id)}})


# osd_dirs = [f for f in os.listdir(ceph_osd_root_path) if
re.match(clusterName+ r'-.*', f)
# and os.path.isdir(ceph_osd_root_path+'/'+f)]


# filter the hardware list according to the class
def filter_hw(hw, cl):
    res = []
    if "children" in hw:
        for child in hw["children"]:
            if child["class"] == cl:
                res.append(child)
                continue
            else:
                res.extend(filter_hw(child, cl))
    return res


def get_hw():
    output = subprocess.Popen(['lshw', '-json'],
stdout=subprocess.PIPE).communicate()[0]
    hw_io = StringIO(output)
    hw = json.load(hw_io)
    return hw


def get_hw_disk(hostname, hw):
    local_disks = []
    # the disks
    disks = filter_hw(hw, "disk")
    for disk in disks:
        if disk['id'].startswith('disk'):
            logname = "NA"
            if "logicalname" in disk:
                if isinstance(disk['logicalname'], list):
                    logname = disk['logicalname'][0]
                else:
                    logname = disk['logicalname']
            else :
                continue
            description = "NA"
            if "description" in disk:
                description = disk["description"]
            product = "NA"
            if "product" in disk:
                product = disk["product"]
```

```
            vendor = "NA"
            if "vendor" in disk:
                vendor = disk["vendor"]
            physid = "NA"
            if "physid" in disk:
                physid = disk["physid"]
            diskversion = "NA"
            if "version" in disk:
                diskversion = disk["version"]
            serial = "NA"
            if "serial" in disk:
                serial = disk["serial"]
            disksize = 0
            if "size" in disk:
                if disk["units"] == "bytes":
                    disksize = disk["size"]
            # other units ?
            d = {"_id": hostname+":"+logname,
                 "description": description,
                 "product": product,
                 "manufacturer": vendor,
                 "physical_id": physid,
                 "logical_name": logname,
                 "version": diskversion,
                 "serial_number": serial,
                 "size": disksize,
                 "stat": None
                 }
            local_disks.append(d)
    return local_disks


def get_network_info(interface):
    output = subprocess.Popen(['ip', 'addr', 'show', 'dev',
interface], stdout=subprocess.PIPE).communicate()[0].rsplit('\n')
    mtu = re.findall('mtu ([0-9]+) ', output[0])[0]
    link_ha = re.findall('link/(\w+) ([0-9a-fA-F:]+) ', output
[1])[0]
    inet = None
    inet6 = None
    for line in output[2:]:
        if not inet:
            tinet = re.findall('inet ([0-9\.]+)/([0-9]+) ', line)
            if tinet:
                inet = {"addr": tinet[0][0], "mask": tinet[0][1]}
                continue
        if not inet6:
            tinet6 = re.findall('inet6 ([0-9a-fA-F:]+)/([0-9]+)
', line)
            if (tinet6):
                inet6 = {"addr": tinet6[0][0], "mask": tinet6[0]
[1]}
```

```
        if inet and inet6:
            break
    return {"mtu":int(mtu), "link": link_ha[0], "HWaddr": link_ha
[1], "inet": inet, "inet6": inet6}



def get_hw_net(hostname, hw):
    local_nets = []
    # the disks
    nets = filter_hw(hw, "network")
    for net in nets:
        if net['id'].startswith('network'):
            logname = "NA"
            if "logicalname" in net:
              if isinstance(net['logicalname'], list):
                  logname = net['logicalname'][0]
              else:
                  logname = net['logicalname']
            else :
              continue
            description = "NA"
            if "description" in net:
                description = net["description"]
            product = "NA"
            if "product" in net:
                product = net["product"]
            vendor = "NA"
            if "vendor" in net:
                vendor = net["vendor"]
            physid = "NA"
            if "physid" in net:
                physid = net["physid"]
            netversion = "NA"
            if "version" in net:
                netversion = net["version"]
            serial = "NA"
            if "serial" in net:
                serial = net["serial"]
            netsize = 0
            if "size" in net:
                if net["units"] == "bit/s":
                    netsize = net["size"]
            netcapacity = 0
            if "capacity" in net:
                if net["units"] == "bit/s":
                    netcapacity = net["capacity"]
                # other units ?
            d = {"_id": hostname+":"+logname,
                 "description": description,
                 "product": product,
                 "manufacturer": vendor,
```

```
                    "physical_id": physid,
                    "logical_name": logname,
                    "version": netversion,
                    "serial_number": serial,
                    "size": netsize,
                    "capacity": netcapacity,
                    "stat": None
                    }
            d_info = get_network_info(logname)
            d.update(d_info)
            local_nets.append(d)
    return local_nets


def get_hw_cpu(hostname, hw):
    local_cpus = []
    # the cpus
    cpus = filter_hw(hw, "processor")
    for cpu in cpus:
        if cpu['id'].startswith('cpu'):
            description = "NA"
            if "description" in cpu:
                description = cpu["description"]
            product = "NA"
            if "product" in cpu:
                product = cpu["product"]
            vendor = "NA"
            if "vendor" in cpu:
                vendor = cpu["vendor"]
            physid = "NA"
            if "physid" in cpu:
                physid = cpu["physid"]
            cpuversion = "NA"
            if "version" in cpu:
                cpuversion = cpu["version"]
            frequency = 0
            if "size" in cpu:
                if cpu["units"] == "Hz":
                    frequency = cpu["size"]
            capacity = 0
            if "capacity" in cpu:
                if cpu["units"] == "Hz":
                    capacity = cpu["capacity"]
            cpuwidth = 0
            if "width" in cpu:
                cpuwidth = cpu["width"]
            cores = 0
            enabledcores = 0
            threads = 0
            if "configuration" in cpu:
                config = cpu["configuration"]
                if "cores" in config:
```

```
                    cores = int(config["cores"])
                if "enabledcores" in config:
                    enabledcores = int(config["enabledcores"])
                if "threads" in config:
                    threads = int(config["threads"])

            c = {"_id": hostname+":"+physid,
                 "description": description,
                 "product": product,
                 "manufacturer": vendor,
                 "physical_id": physid,
                 "version": cpuversion,
                 "capacity": capacity,
                 "frequency": frequency,
                 "width": cpuwidth,
                 "cores": cores,
                 "enabledcores": enabledcores,
                 "threads": threads,
                 "stat": None
                 }
            local_cpus.append(c)
    return local_cpus


def init_host(hostname, db):
    hw = get_hw()

    hw_disks = get_hw_disk(hostname, hw)
    for d in hw_disks:
        db.disks.update({'_id': d['_id']}, d, upsert=True)

    partitions = get_partitions(hostname)
    for p in partitions:
        db.partitions.update({'_id': p['_id']}, p, upsert=True)

    hw_nets = get_hw_net(hostname, hw)
    for n in hw_nets:
        db.net.update({'_id': n['_id']}, n, upsert=True)

    hw_cpus = get_hw_cpu(hostname, hw)
    for c in hw_cpus:
        db.cpus.update({'_id': c['_id']}, c, upsert=True)

    host__ = {'_id': hostname, #fqdn
              "hostip": socket.gethostbyname(hostname),
              "timestamp": int(round(time.time() * 1000)),
              "mem": None,
              "swap": None,
              "disks": [DBRef("disks",  d["_id"]) for d in
hw_disks],
              "partitions": [DBRef("partitions",  p["_id"]) for p
in partitions],
```

```
                "cpus": [DBRef("cpus",  c["_id"]) for c in
hw_cpus],
                "cpus_stat": None,
                "network_interfaces": [DBRef("net",  n["_id"]) for
n in hw_nets]
                }
    db.hosts.update({'_id': hostname}, host__, upsert=True)
    return hw_disks, partitions, hw_nets, hw_cpus



disk_stat_hdr = ["rrqm_s", "wrqm_s", "r_s" , "w_s", "rkB_s",
"wkB_s"]


# disk stat
def pick_disk_stat(db, hw_disks):
    print str(datetime.datetime.now()), "-- Pick Disk Stats"
    sys.stdout.flush()
    p = subprocess.Popen(args=['iostat', '-dx']+[d
["logical_name"] for d in hw_disks], stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
    outdata, errdata = p.communicate()
    if len(errdata):
        raise RuntimeError('unable to run iostat: %s' % errdata)
    lines = outdata.rstrip().splitlines()
    for d in hw_disks:
        dev = re.findall('.*/([a-zA-Z0-9]+)', d['logical_name'])
[0]
        iostat = [line for line in lines if re.match(dev+'.*',
line)]
        if iostat:
            lineio = iostat[0].split()[1:]
            diskstat = dict([(k, float(v.replace(',', '.'))) for
k, v in zip(disk_stat_hdr, lineio)])
            diskstat["disk"] = DBRef("disks", d["_id"])
            diskstat["timestamp"] = int(round(time.time() *
1000))
            disk_stat_id = db.diskstat.insert(diskstat)
            db.disks.update({"_id": d["_id"]}, {"$set": {"stat":
DBRef("diskstat", disk_stat_id)}})


# net stat
def pick_net_stat(db, hw_nets):
    print str(datetime.datetime.now()), "-- Pick Net Stats"
    sys.stdout.flush()
    netio = psutil.net_io_counters(pernic=True)
    for n in hw_nets:
        net_interface = n["logical_name"]
        netstat = netio[net_interface]
        if netstat:
```

```
            network_interface_stat = {"network_interface": DBRef
("net", n['_id']),
                                      "timestamp": int(round
(time.time() * 1000)),
                                      "rx": {"packets":
netstat.packets_recv,
                                             "errors":
netstat.errin,
                                             "dropped":
netstat.dropin,
                                             "bytes":
netstat.bytes_recv
                                             },
                                      "tx": {"packets":
netstat.packets_sent,
                                             "errors":
netstat.errout,
                                             "dropped":
netstat.dropout,
                                             "bytes":
netstat.bytes_sent
                                            }
                                     }
            network_interface_stat_id = db.netstat.insert
(network_interface_stat)
            db.net.update({"_id": n["_id"]}, {"$set": {"stat":
DBRef("netstat", network_interface_stat_id)}})

# transform a set of object attributes to dict
def objToDict(obj, attrs):
    dict_res = {}
    for attr in attrs :
        if hasattr(obj, attr) :
            dict_res[attr] = getattr(obj, attr)
    return dict_res

# cpu stat
def pick_cpu_stat(hostname, db):
    print str(datetime.datetime.now()), "-- Pick CPU Stats"
    sys.stdout.flush()
    cputimes = psutil.cpu_times()
    cpus_stat = objToDict(cputimes, ["user", "system", "idle",
"iowait", "irq", "softirq", "steal", "guest", "guest_nice"])
    cpus_stat["timestamp"] = int(round(time.time() * 1000))
    cpus_stat["host"] = DBRef("hosts",  hostname)

    cpus_stat_hostx_id = db.cpustat.insert(cpus_stat)
    db.hosts.update({"_id": hostname}, {"$set": {"stat": DBRef
("cpus_stat", cpus_stat_hostx_id)}})


def pick_ceph_processes_v2(hostname, db):
```

```
    # Compatibility with psutil V2.x
    print str(datetime.datetime.now()), "-- Pick Ceph Processes
V2"
    sys.stdout.flush()
    iterP = psutil.process_iter()
    ceph_procs = [p for p in iterP if p.name().startswith
('ceph-')]

    for ceph_proc in ceph_procs:
        # print ceph_proc, " ", ceph_proc.cmdline()[1:]
#        options, remainder = getopt.getopt(ceph_proc.cmdline()
[1:], 'ic:f', ['cluster=', 'id', 'config', 'pid-file'])
        options, remainder = getopt.getopt(ceph_proc.cmdline()
[1:], 'ic:fd', ['cluster=', 'id', 'config', 'pid-file'])
        clust = None
        id = None

        for opt, arg in options:
            if opt in ("-i", "--id"):
                id = arg
            elif opt in '--cluster':
                clust = arg

        if db.name == clust:
            p_db = {"timestamp": int(round(time.time() * 1000)),
                    "host": DBRef("hosts",  hostname),
                    "pid": ceph_proc.pid,
                    "mem_rss": ceph_proc.memory_info_ex().rss,
                    "mem_vms": ceph_proc.memory_info_ex().vms,
                    "mem_shared": ceph_proc.memory_info_ex
().shared,
                    "num_threads": ceph_proc.num_threads(),
                    "cpu_times_user": ceph_proc.cpu_times().user,
                    "cpu_times_system": ceph_proc.cpu_times
().system,
                    }

            if ceph_proc.name() == 'ceph-osd':
                # osd
                p_db["osd"] = DBRef("osd", id)
                procid = db.processstat.insert(p_db)
                db.osd.update({'_id': id}, {"$set": {"process":
DBRef("process", procid)}})
            elif ceph_proc.name() == 'ceph-mon':
                # mon
                p_db["mon"] = DBRef("mon", id)
                procid = db.processstat.insert(p_db)
                db.mon.update({'_id': id}, {"$set": {"process":
DBRef("process", procid)}})


def pick_ceph_processes_v1(hostname, db):
```

```
    # Compatibility with psutil V1.x
    print str(datetime.datetime.now()), "-- Pick Ceph Processes
V1"
    sys.stdout.flush()
    iter_p = psutil.process_iter()
    ceph_procs = [p for p in iter_p if p.name.startswith
('ceph-')]

    for ceph_proc in ceph_procs:
        options, remainder = getopt.getopt(ceph_proc.cmdline[1:],
'i:f', ['cluster=', 'pid-file'])

        clust = None
        id = None

        for opt, arg in options:
            if opt == '-i':
                id = arg
            elif opt in  '--cluster':
                clust = arg

        if db.name == clust:
            p_db = {"timestamp": int(round(time.time() * 1000)),
                    "host": DBRef("hosts",  hostname),
                    "pid": ceph_proc.pid,
                    "mem_rss": ceph_proc.get_ext_memory_info
().rss,
                    "mem_vms": ceph_proc.get_ext_memory_info
().vms,
                    "mem_shared": ceph_proc.get_ext_memory_info
().shared,
                    "num_threads": ceph_proc.get_num_threads(),
                    "cpu_times_user": ceph_proc.get_cpu_times
().user,
                    "cpu_times_system": ceph_proc.get_cpu_times
().system,
                    }

            if ceph_proc.name == 'ceph-osd':
                # osd
                p_db["osd"] = DBRef("osd", id)
                procid = db.processstat.insert(p_db)
                db.osd.update({'_id': id}, {"$set": {"process":
DBRef("process", procid)}})
            elif ceph_proc.name == 'ceph-mon':
                # mon
                p_db["mon"] = DBRef("mon", id)
                procid = db.processstat.insert(p_db)
                db.mon.update({'_id': id}, {"$set": {"process":
DBRef("process", procid)}})
```

```
# delete the oldest stats
# window in second
def drop_stat(db, collection, window):
    before = int((time.time() - window) * 1000)
    print str(datetime.datetime.now()), "-- drop Stats:",
collection, "before", before
    db[collection].remove({"timestamp": {"$lt": before}})


def heart_beat(hostname, db):
    beat = {"timestamp": int(round(time.time() * 1000)),}
    db.sysprobe.update({'_id': hostname}, {"$set": beat},
upsert=True)


class Repeater(Thread):
    #period in second
    def __init__(self, event, function, args=[], period=5.0):
        Thread.__init__(self)
        self.stopped = event
        self.period = period
        self.function = function
        self.args = args

    def run(self):
        while not self.stopped.wait(self.period):
            try:
                # call a function
                self.function(*self.args)
            except Exception, e:
                # try later
                print str(datetime.datetime.now()), "-- WARNING :
"+self.function.__name__ + " did not worked : ", e
                exc_type, exc_value, exc_traceback = sys.exc_info
()
                traceback.print_exception(exc_type, exc_value,
exc_traceback)
                pass


def ensure_dir(f):
    d = os.path.dirname(f)
    if not os.path.exists(d):
        os.makedirs(d)


# ceph probe
# cephClient = httplib.HTTPConnection("localhost", port)

# gethostname -> hn
# if hn is mon of rank 0 -> update db
```

```
class Usage(Exception):
    def __init__(self, msg):
        self.msg = msg


evt = Event()


def handler(signum, frame):
    print 'Signal handler called with signal', signum
    evt.set()


class SysProbeDaemon(Daemon):
    def __init__(self, pidfile):
        Daemon.__init__(self, pidfile, stdout=logfile,
stderr=logfile)

    def run(self):
        print str(datetime.datetime.now())
        print "SysProbe loading"
        # load conf
        data = load_conf()
        cluster_name = data.get("cluster", "ceph")
        print "cluster = ", cluster_name

        ceph_conf_file = data.get("ceph_conf",
"/etc/ceph/ceph.conf")
        print "cephConf = ", ceph_conf_file

        fsid = ceph_conf_global(ceph_conf_file, 'fsid')
        print "fsid = ", fsid

        hb_refresh = data.get("hb_refresh", 5)
        print "hb_refresh = ", hb_refresh

        mem_refresh = data.get("mem_refresh", 60)
        print "mem_refresh = ", mem_refresh

        mem_window = data.get("mem_window", 1200)
        print "mem_window = ", mem_window

        swap_refresh = data.get("swap_refresh", 600)
        print "swap_refresh = ", swap_refresh

        swap_window = data.get("swap_window", 3600)
        print "swap_window = ", swap_window

        disk_refresh = data.get("disk_refresh", 60)
        print "disk_refresh = ", disk_refresh
```

```
        disk_window = data.get("disk_window", 1200)
        print "disk_window = ", disk_window

        partition_refresh = data.get("partition_refresh", 60)
        print "partition_refresh = ", partition_refresh

        partition_window = data.get("partition_window", 1200)
        print "partition_window = ", partition_window

        cpu_refresh = data.get("cpu_refresh", 60)
        print "cpu_refresh = ", cpu_refresh

        cpu_window = data.get("cpu_window", 1200)
        print "cpu_window = ", cpu_window

        net_refresh = data.get("net_refresh", 30)
        print "net_refresh = ", net_refresh

        net_window = data.get("net_window", 1200)
        print "net_window = ", net_window

        process_refresh = data.get("process_refresh", 60)
        print "process_refresh = ", process_refresh

        process_window = data.get("process_window", 1200)
        print "process_window = ", process_window

        mongodb_host = data.get("mongodb_host", None)
        print "mongodb_host = ", mongodb_host

        mongodb_port = data.get("mongodb_port", None)
        print "mongodb_port = ", mongodb_port

        is_mongo_replicat = data.get("is_mongo_replicat", 0)
        print "is_mongo_replicat = ", is_mongo_replicat

        mongodb_set = "'"+data.get("mongodb_set", None)+"'"
        print "mongodb_set = ", mongodb_set

        mongodb_replica_set =data.get("mongodb_replicaSet", None)
        print "mongodb_replicaSet = ", mongodb_replica_set

        mongodb_read_preference = data.get
("mongodb_read_preference", None)
        print "mongodb_read_preference = ",
mongodb_read_preference

        is_mongo_authenticate = data.get("is_mongo_authenticate",
0)
        print "is_mongo_authenticate", is_mongo_authenticate

        mongodb_user = data.get("mongodb_user", "cephdefault")
```

```
        print "mongodb_user = ", mongodb_user

        mongodb_passwd = data.get("mongodb_passwd", None)
        print "mongodb_passwd = ", mongodb_passwd

        # end conf extraction

        print "version psutil = ", psutil_version, " (", psutil.
__version__, ")"
        if (psutil.__version__ < "1.2.1") :
            print "ERROR : update your psutil to a earlier
version (> 1.2.1)"
            sys.exit(2)

        sys.stdout.flush()

        #hostname = socket.gethostname()
        hostname = socket.getfqdn()
        print "hostname = ", hostname

        if is_mongo_replicat == 1:
            print "replicat set connexion"
            client = MongoReplicaSetClient(eval(mongodb_set),
replicaSet=mongodb_replica_set, read_preference=eval
(mongodb_read_preference))
        else:
            print "no replicat set"
            client = MongoClient(mongodb_host, mongodb_port)

        db = client[fsid]

        if is_mongo_authenticate == 1:
            print "authentication  to database"
            db.authenticate(mongodb_user, mongodb_passwd)
        else:
            print "no authentication"



        HWdisks, partitions, HWnets, HWcpus = init_host(hostname,
db)

        data["_id"] = hostname
        db.sysprobe.remove({'_id': hostname})
        db.sysprobe.insert(data)

        hb_thread = None
        if hb_refresh > 0:
            hb_thread = Repeater(evt, heart_beat, [hostname, db],
hb_refresh)
            hb_thread.start()
```

```
        cpu_thread = None
        if cpu_refresh > 0:
            cpu_thread = Repeater(evt, pick_cpu_stat, [hostname,
db], cpu_refresh)
            cpu_thread.start()

        net_thread = None
        if net_refresh > 0:
            net_thread = Repeater(evt, pick_net_stat, [db,
HWnets], net_refresh)
            net_thread.start()

        mem_thread = None
        if mem_refresh > 0:
            mem_thread = Repeater(evt, pick_mem, [hostname, db],
mem_refresh)
            mem_thread.start()

        swap_thread = None
        if swap_refresh > 0:
            swap_thread = Repeater(evt, pick_swap, [hostname,
db], swap_refresh)
            swap_thread.start()

        disk_thread = None
        if disk_refresh > 0:
            disk_thread = Repeater(evt, pick_disk_stat, [db,
HWdisks], disk_refresh)
            disk_thread.start()

        part_thread = None
        if partition_refresh > 0:
            part_thread = Repeater(evt, pick_partitions_stat,
[hostname, db], partition_refresh)
            part_thread.start()

        process_thread = None
        if process_refresh > 0:
            if psutil_version == "1":
                process_thread = Repeater(evt,
pick_ceph_processes_v1, [hostname, db], process_refresh)
            else:
                process_thread = Repeater(evt,
pick_ceph_processes_v2, [hostname, db], process_refresh)
            process_thread.start()

        # drop thread
        cpu_db_drop_thread = None
        if cpu_window > 0:
            cpu_db_drop_thread = Repeater(evt, drop_stat, [db,
"cpustat", cpu_window], cpu_window)
            cpu_db_drop_thread.start()
```

```
        net_db_drop_thread = None
        if net_window > 0:
            net_db_drop_thread = Repeater(evt, drop_stat, [db,
"netstat", net_window], net_window)
            net_db_drop_thread.start()

        mem_db_drop_thread = None
        if mem_window > 0:
            mem_db_drop_thread = Repeater(evt, drop_stat, [db,
"memstat", mem_window], mem_window)
            mem_db_drop_thread.start()

        swap_db_drop_thread = None
        if swap_window > 0:
            swap_db_drop_thread = Repeater(evt, drop_stat, [db,
"swapstat", swap_window], swap_window)
            swap_db_drop_thread.start()

        disk_db_drop_thread = None
        if disk_window > 0:
            disk_db_drop_thread = Repeater(evt, drop_stat, [db,
"diskstat", disk_window], disk_window)
            disk_db_drop_thread.start()

        part_db_drop_thread = None
        if partition_window > 0:
            part_db_drop_thread = Repeater(evt, drop_stat, [db,
"partitionstat", partition_window], partition_window)
            part_db_drop_thread.start()

        process_db_drop_thread = None
        if process_window > 0:
            process_db_drop_thread = Repeater(evt, drop_stat,
[db, "processstat", process_window], process_window)
            process_db_drop_thread.start()

        signal.signal(signal.SIGTERM, handler)

        while not evt.isSet():
            evt.wait(600)

        print str(datetime.datetime.now())
        print "SysProbe stopped"


if __name__ == "__main__":
    ensure_dir(logfile)
    ensure_dir(runfile)
    daemon = SysProbeDaemon(runfile)
    if len(sys.argv) == 2:
        if 'start' == sys.argv[1]:
```

```
        daemon.start()
    elif 'stop' == sys.argv[1]:
        daemon.stop()
    elif 'status' == sys.argv[1]:
        daemon.status()
    elif 'restart' == sys.argv[1]:
        daemon.restart()
    else:
        print "Unknown command"
        sys.exit(2)
    sys.exit(0)
else:
    print "usage: %s start|stop|restart|status" % sys.argv[0]
    sys.exit(2)
```