```python
#!/usr/bin/env python
"""
 Author: Michael Lange
 pass the data to the ros monitoring system
 This scripts lists all running rosnodes and displays all values the
 psutil package is providing for each node.
 Supports a filter function in order to only retrieve information of specific
 nodes with specific values.

# TODO: node name, PID into own Datatype (1) DONE
# TODO: comments in Dpxygen style (2) DONE
# TODO: Log error as ros error (3) DONE
# TODO: get all psutil process info (6) DONE
# TODO: rename file (4) DONE
# TODO: Integrate Whitelist Param-stuff (5) DONE
# TODO: Integrate default if Whitelist Param-stuff n/a  DONE
# TODO: Integrate into Monitoring (7) DONE
# TODO: Set return value of get_process_info(pid) to named tuple containing all values
        see documentation at https://docs.python.org/3/library/collections.html#collections.namedtuple
        DONE
# TODO: Add second yaml file for general config, (frequency, black/whitelist etc) DONE
# TODO: Clean up get_process_info() nonsense DONE
# TODO: create additional launch files
# TODO: integrate blacklist filter feature DONE
# TODO: Add units to monitor output
"""



import rospy
import psutil
import rosnode
import xmlrpclib

from std_msgs.msg import String
from monitoring_core.monitor import Monitor
from collections import namedtuple

ID = "NODEINFO"
filter_type = None

class node:
        def __init__(self, name, pid):
            self.pid = pid
            self.name = name


def init():
    """
    Init rospy node
    check for frequency parameter (default = 1Hz)
    check for filter_type (0 = default (list all), 1 = whitelist, 2= black list)
    """
    rospy.init_node('node_ressource_monitor', anonymous=True)
    if rospy.has_param('node_ressource_monitor/frequency'):
        frequency = rospy.get_param('node_ressource_monitor/frequency')
    else:
        frequency = 1
    if rospy.has_param('node_ressource_monitor/filter_type'):
        filter_type = rospy.get_param('node_ressource_monitor/filter_type')
    else:
        filter_type = 0
    return frequency, filter_type

def get_node_list():
    """
    List all ROS Nodes
    Return: List containing ROS Nodes(name, pid)
    """
    rospy.loginfo("GET_NODE_LIST:")
    node_array_temp = rosnode.get_node_names()
    node_list = []
    j = 0
    for node_name in node_array_temp:
        node_api = rosnode.get_api_uri(rospy.get_master(), node_name)
        code, msg, pid = xmlrpclib.ServerProxy(node_api[2]).getPid(ID)
        node_list.append(node(node_name, pid))
        rospy.loginfo("Node_name: " + node_list[j].name + " Node_PID: " + str(node_list[j].pid))
        j=j+1
```

```
    rospy.loginfo("============================")
    return node_list

def get_process_info(pid):
    """
    gather all information provided by psutil.Process(pid)
    """
    node_process_info = psutil.Process(pid)
    return node_process_info

def print_to_console_and_monitor(name, pid):
    """
    print information to console
    pass the data to the ros monitoring system
    """
    #get all values belonging to pid
    try:
        node_process_info = get_process_info(pid)
    except psutil._exceptions.NoSuchProcess:
        pass

    monitor = Monitor("node_ressource_monitor")
    no_param_available = False
    #get list of nodes set as filter values
    try:
        node_filter = rospy.get_param('/node_ressource_monitor/nodes')
    except KeyError:
        no_param_available = True
        rospy.logerr("No parameters set - setting default")

    if filter_type == 2:
        for element in node_filter:
            if name in rospy.get_param('/node_ressource_monitor/nodes/' + element).get("name"):
                rospy.logerr(name + " is blacklisted!")
                return

    #set default elements in node_filter if no parameters are set, filter_type is default
    #or filter_type is blacklist
    if no_param_available or filter_type == 0 or filter_type == 2:
        node_filter = {'default_node':{'name': name, 'values': ['children', 'cmdline', \
            'connections', 'cpu_affinity', 'cpu_percent', 'cpu_times', 'create_time', \
            'cwd', 'exe', 'gids', 'io_counters', 'ionice', 'is_running', 'memory_info', \
            'memory_info_ex', 'memory_maps', 'memory_percent', 'name', \
            'nice', 'num_ctx_switches', 'num_fds', 'num_threads', 'open_files', \
            'parent', 'ppid', 'status', 'terminal', 'threads', 'uids', 'username']}}

    for element in node_filter:

        #look for whitelisted values, use default if no parameters are set,
        #filter_type is default or blacklisting is activated
        if no_param_available or filter_type == 0 or filter_type == 2:
            value_filter = node_filter['default_node']
        else:
            value_filter = rospy.get_param('/node_ressource_monitor/nodes/' + element)

        #lookup name of given node in whitelisted_nodes/values
        if name in value_filter.get("name"):

            rospy.logout("Node-name: " + name)
            #check whitelisted_values dictionary for values to print
            for element in value_filter.get("values"):
                if element == 'children':
                    rospy.loginfo("children: " + str(node_process_info.children()))
                    monitor.addValue("children", str(node_process_info.children()), "", 0.0)
                if element == 'cmdline':
                    rospy.loginfo("cmdline: " + str(node_process_info.cmdline()))
                    monitor.addValue("cmdline", str(node_process_info.cmdline()), "", 0.0)
                if element == 'connections':
                    rospy.loginfo("connections: " + str(node_process_info.connections()))
                    monitor.addValue("connections", str(node_process_info.connections()), "", 0.0)
                if element == 'cpu_affinity':
                    rospy.loginfo("cpu_affinity: " + str(node_process_info.cpu_affinity()))
                    monitor.addValue("cpu_affinity", str(node_process_info.cpu_affinity()), "", 0.0)
                if element == 'cpu_percent':
                    rospy.loginfo("cpu_percent: " + str(node_process_info.cpu_percent()))
                    monitor.addValue("cpu_percent", str(node_process_info.cpu_percent()), "", 0.0)
                if element == 'cpu_times':
                    rospy.loginfo("cpu_times: " + str(node_process_info.cpu_times()))
```

```
            monitor.addValue("cpu_times", str(node_process_info.cpu_times()), "", 0.0)
        if element == 'create_time':
            rospy.loginfo("create_time: " + str(node_process_info.create_time()))
            monitor.addValue("create_time", str(node_process_info.create_time()), "", 0.0)
        if element == 'cwd':
            rospy.loginfo("cwd: " + str(node_process_info.cwd()))
            monitor.addValue("cwd", str(node_process_info.cwd()), "", 0.0)
        if element == 'exe':
            rospy.loginfo("exe-path: " + str(node_process_info.exe()))
            monitor.addValue("exe-path", str(node_process_info.exe()), "", 0.0)
        if element == 'gids':
            rospy.loginfo("gids: " + str(node_process_info.gids()))
            monitor.addValue("gids", str(node_process_info.gids()), "", 0.0)
        if element == 'io_counters':
            rospy.loginfo("io_counters: " + str(node_process_info.io_counters()))
            monitor.addValue("io_counters", str(node_process_info.io_counters()), "", 0.0)
        if element == 'ionice':
            rospy.loginfo("ionice: " + str(node_process_info.ionice()))
            monitor.addValue("ionice", str(node_process_info.ionice()), "", 0.0)
        if element == 'is_running':
            rospy.loginfo("is_running: " + str(node_process_info.is_running()))
            monitor.addValue("is_running", str(node_process_info.is_running()), "", 0.0)
        if element == 'memory_info':
            rospy.loginfo("memory_info: " + str(node_process_info.memory_info()))
            monitor.addValue("memory_info", str(node_process_info.memory_info()), "", 0.0)
        if element == 'memory_info_ex':
            rospy.loginfo("memory_info_ex: " + str(node_process_info.memory_info_ex()))
            monitor.addValue("memory_info_ex", str(node_process_info.memory_info_ex()), "", 0.0)
        if element == 'memory_maps':
            rospy.loginfo("memory_maps: " + str(node_process_info.memory_maps()))
            monitor.addValue("memory_maps", str(node_process_info.memory_maps()), "", 0.0)
        if element == 'memory_percent':
            rospy.loginfo("memory_percent: " + str(node_process_info.memory_percent()))
            monitor.addValue("memory_percent", str(node_process_info.memory_percent()), "", 0.0)
        if element == 'name':
            rospy.loginfo("name: " + str(node_process_info.name()))
            monitor.addValue("name", str(node_process_info.name()), "", 0.0)
        if element == 'nice':
            rospy.loginfo("nice: " + str(node_process_info.nice()))
            monitor.addValue("nice", str(node_process_info.nice()), "", 0.0)
        if element == 'num_ctx_switches':
            rospy.loginfo("num_ctx_switches: " + str(node_process_info.num_ctx_switches()))
            monitor.addValue("num_ctx_switches", str(node_process_info.num_ctx_switches()), "",
            0.0)
        if element == 'num_fds':
            rospy.loginfo("num_fds: " + str(node_process_info.num_fds()))
            monitor.addValue("num_fds", str(node_process_info.num_fds()), "", 0.0)
        if element == 'num_handles':
            rospy.loginfo("num_handles: " + str(node_process_info.num_handles()))
            monitor.addValue("num_handles", str(node_process_info.num_handles()), "", 0.0)
        if element == 'num_threads':
            rospy.loginfo("num_threads: " + str(node_process_info.num_threads()))
            monitor.addValue("num_threads", str(node_process_info.num_threads()), "", 0.0)
        if element == 'open_files':
            rospy.loginfo("open_files: " + str(node_process_info.open_files()))
            monitor.addValue("open_files", str(node_process_info.open_files()), "", 0.0)
        if element == 'parent':
            rospy.loginfo("parent: " + str(node_process_info.parent()))
            monitor.addValue("parent", str(node_process_info.parent()), "", 0.0)
        if element == 'pid':
            rospy.loginfo("pid: " + str(node_process_info.pid()))
            monitor.addValue("pid", str(node_process_info.pid()), "", 0.0)
        if element == 'ppid':
            rospy.loginfo("ppid: " + str(node_process_info.ppid()))
            monitor.addValue("ppid", str(node_process_info.ppid()), "", 0.0)
        if element == 'rlimit':
            rospy.loginfo("rlimit: " + str(node_process_info.rlimit()))
            monitor.addValue("rlimit", str(node_process_info.rlimit()), "", 0.0)
        if element == 'status':
            rospy.loginfo("status: " + str(node_process_info.status()))
            monitor.addValue("status", str(node_process_info.status()), "", 0.0)
        if element == 'terminal':
            rospy.loginfo("terminal: " + str(node_process_info.terminal()))
            monitor.addValue("terminal", str(node_process_info.terminal()), "", 0.0)
        if element == 'threads':
            rospy.loginfo("threads: " + str(node_process_info.threads()))
            monitor.addValue("threads", str(node_process_info.threads()), "", 0.0)
        if element == 'uids':
```

```
                    rospy.loginfo("uids: " + str(node_process_info.uids()))
                    monitor.addValue("uids", str(node_process_info.uids()), "", 0.0)
                if element == 'username':
                    rospy.loginfo("username: " + str(node_process_info.username()))
                    monitor.addValue("username", str(node_process_info.username()), "", 0.0)

            monitor.publish()
        else:
            rospy.logerr(name + " not in whitelist")

def gather_info():
    """
    obtains list of all running nodes by calling get_node_list()
    calls get_process_info() for each retrieved node
    calls print_info_to_console() for each retrieved node
    """
    node_list = get_node_list()
    for i in node_list:
        try:
            get_process_info(i.pid)
            print_to_console_and_monitor(i.name, i.pid)
            rospy.loginfo("------------------------------")
        except psutil._exceptions.NoSuchProcess:
            rospy.logerr("----------NO_SUCH_PROCESS_ERROR---------------")
            pass

    rospy.loginfo("==============================")

if __name__ == '__main__':
    frequency, filter_type = init()
    rate = rospy.Rate(frequency)
    rospy.loginfo(frequency)
    while not rospy.is_shutdown():
        try:
            gather_info()
            rate.sleep()
        except rospy.ROSInterruptException:
            rospy.loginfo("ERROR")
            pass
```