·····

·····cgroup·····swap·····

swap ·····
·····

····

vmstat · swpd ·····si·so ·····majflt/s ·····
· swap··· RSS·VmSwap ·····cgroup · memory.current·memory.swap.current · memory.pressure ·····
·····

·· | ····· | ····

swpd | ··swap·· | ·····

si / so | ····· | ·····PID

RSS / VmSwap | ····· | ·····worker

memory.pressure | cgroup····· | ···cgroup·····

·····

····· smaps_rollup ····· cgroup v2 ····· PID····· shell···· JSONL ····
····· CG ····· PID ·····

```
from pathlib import Path
import json, math, sys, time

pid, cg_arg, count_arg, interval_arg, output = sys.argv[1:]
if not pid.isdigit() or int(pid) < 1:
    raise SystemExit('PID·····')
count, interval = int(count_arg), float(interval_arg)
if count < 1 or not math.isfinite(interval) or interval <= 0:
    raise SystemExit('·····')
cg = Path(cg_arg)
if not (cg / 'memory.current').is_file():
    raise SystemExit('···memory.current···cgroup v2··')

def read(path):
    try:
        return Path(path).read_text()
    except OSError as err:
        return {'read_error': type(err).__name__}

def start_ticks():
    value = read(f'/proc/{pid}/stat')
    if not isinstance(value, str):
        return None
    fields = value.rsplit(')', 1)[-1].split()
    return fields[19] if len(fields) > 19 else None

identity = start_ticks()
if identity is None:
    raise SystemExit('·····/proc···')
with Path(output).open('x') as out:
    for index in range(count):
        if start_ticks() != identity:
            out.write(json.dumps({'time_unix': time.time(), 'status': 'process_ended_or_reused'}) + '\n')
            break
        row = {'time_unix': time.time(), 'pid': int(pid), 'starttime_ticks': identity,
               'status_raw': read(f'/proc/{pid}/status'),
               'smaps_rollup_raw': read(f'/proc/{pid}/smaps_rollup')}
        for name in ('memory.current', 'memory.max', 'memory.swap.current',
                     'memory.swap.max', 'memory.stat', 'memory.events', 'memory.pressure'):
            row[name] = read(cg / name)
        out.write(json.dumps(row) + '\n')
        out.flush()
        if index + 1 < count:
            time.sleep(interval)
```

·····RSS·vmstat -y ····· procps ·····
···pidstat · major faults ··· swap ···

```
set -euo pipefail
: "${PID:?·····PID}" "${CG:?····cgroup v2··}"
: "${WORK:?·····}"
[[ "$PID" =~ ^[1-9][0-9]*$ ]] || exit 2
test -r "/proc/$PID/status" && test -r "$CG/memory.current"
out=$(mktemp -d "$WORK/swap-window.XXXXXX")
date -Is > "$out/start.txt"
vmstat -w -t -y 1 60 > "$out/vmstat.txt" & vm_pid=$!
pidstat -r -u -p "$PID" 1 60 > "$out/pidstat.txt" & pidstat_pid=$!
cleanup() { kill "$vm_pid" "$pidstat_pid" 2>/dev/null || true; wait "$vm_pid" "$pidstat_pid" 2>/dev/null || true; }
trap cleanup EXIT
python memory_sample.py "$PID" "$CG" 60 1 "$out/memory.jsonl"
wait "$vm_pid"; wait "$pidstat_pid"
trap - EXIT
date -Is > "$out/end.txt"
printf 'Saved samples: %s\n' "$out"
```

··PSI·····

cgroup · memory.pressure ···· /proc/pressure/memory ··· some·full ·····some ·····
··full ····· total ···· avg ·····
·····

·····

····· swap ····· si/so ····cgroup PSI ·····
····· so ·····PSI ····· cgroup ·····memory.events ·····
·····

··· si/so ····· cgroup · memory.pressure ·····
·····memory.swap.current ·····
·····

·····
··· swapoff·····

·····ID·····cgroup·····si/so···PSI·····
····· swa
p ·····

· PID RSS ···· worker····· RSS ····· cgroup ·····
····· I/
O·CPU ·····

····

Linux··PSI·····cgroup·· https://docs.kernel.org/accounting/psi.html

Linux··cgroup v2···swap·· https://docs.kernel.org/admin-guide/cgroup-v2.html

Linux··/proc·····smaps_rollup https://docs.kernel.org/filesystems/proc.html