Process and Thread Monitoring in Linux

Overview

This exercise demonstrates how to monitor processes and threads in Linux, focusing on system parameters important for real-time systems, particularly understanding minor page faults and their impact.

Test Program Execution

The following `load` program was executed to generate CPU load for monitoring:

```
zaka@zakaBouj:/mnt/c/Users/zb200/Documents/GitHub/RTS_SoSe_25_Weronek/Linux/05_Execution_Times_2$ ./load MyLabel 10 15 2000000000
./load MyLabel   0 1750776720 506046606
./load MyLabel    1 1750776721 631641045
./load MyLabel    2 1750776722 745157413
./load MyLabel    3 1750776723 856734379
./load MyLabel    4 1750776724 969497903
./load MyLabel    5 1750776726  75214369
./load MyLabel    6 1750776727 183575649
./load MyLabel    7 1750776728 292967006
./load MyLabel    8 1750776729 415773304
./load MyLabel    9 1750776730 530860335
./load MyLabel   10 1750776731 643975887
```

Program Parameters:

- Label: `MyLabel`
- Number of slices: `10`
- Nice value: `15` (lower priority)
- Load per slice: `2000000000` iterations

Exercise 1a: Detailed Process Information

Command Used:

```
ps -u $(whoami) -o ppid,pid,psr,sgi_p,pcpu,comm,policy,rtprio,pri,nice,time,c,f,wchan,cmd,pmem,maj_flt,min_flt,sz
```

Output:

```
   PPID     PID PSR P %CPU COMMAND         POL RTPRIO PRI  NI     TIME  C F WCHAN
CMD                         %MEM  MAJFL  MINFL    SZ
    405     406  14 *  0.0 bash            TS       -  19   0 00:00:00  0 4 do_wai
-bash                        0.0      4   3922  1715
      1     460  15 *  0.0 systemd         TS       -  19   0 00:00:00  0 4 -
/lib/systemd/systemd --user  0.0      0   1815  4301
    460     464  11 *  0.0 (sd-pam)        TS       -  19   0 00:00:00  0 5 -
(sd-pam)                     0.0      0     52 42305
    460     474   0 *  0.0 pipewire        TS       -  19   0 00:00:00  0 0 -
/usr/bin/pipewire            0.0     39    667  8564
    460     475  13 *  0.0 pipewire-media- TS       -  19   0 00:00:00  0 0 -
/usr/bin/pipewire-media-ses  0.0     39    664  4607
    407     478   4 *  0.0 bash            TS       -  19   0 00:00:00  0 4 core_s
-bash                        0.0      0   1473  1564
    460     495   0 *  0.0 dbus-daemon     TS       -  19   0 00:00:00  0 0 -
/usr/bin/dbus-daemon --sess  0.0      0    334  2075
    406    3579   3 *  0.2 load            TS       -   4  15 00:00:01  0 0 do_sig
./load MyLabel 10 15 200000  0.0      1     86   694
   3913    3919  15 *  0.0 bash            TS       -  19   0 00:00:00  0 4 do_wai
-bash                        0.0      0   3052  1586
    406    7140   4 4 89.5 load            TS       -   4  15 00:00:05 89 0 -
./load MyLabel 10 15 200000  0.0      0     84   694
   3919    7163   8 8  0.0 ps              TS       -  19   0 00:00:00  0 0 -
ps -u zaka -o ppid,pid,psr,  0.0      0    192  1871
```

Column Explanations:

Column	Description	Example Value	Significance
PPID	Parent Process ID	406	Shows process hierarchy
PID	Process ID	7140	Unique identifier for the process
PSR	Processor (CPU core)	4	Which CPU core is executing the process
P	Processor last used	4	Last CPU core that ran this process
%CPU	CPU usage percentage	89.5	Shows high CPU usage for active load process
COMMAND	Command name	load	Truncated command name
POL	Scheduling Policy	TS	Time Sharing (normal scheduling)
RTPRIO	Real-time priority	-	Not a real-time process
PRI	Priority	4	Lower number = higher priority
NI	Nice value	15	Our specified nice value (lower priority)

Column	Description	Example Value	Significance
TIME	CPU time used	00:00:05	Total CPU time consumed
C	Processor utilization	89	For scheduler (obsolete)
F	Process flags	0	Process state flags
WCHAN	Wait channel	do_sig	Kernel function where process is sleeping
CMD	Full command	./load MyLabel...	Complete command with arguments
%MEM	Memory usage %	0.0	Percentage of physical memory
MAJFL	Major page faults	0-1	Faults requiring disk I/O
MINFL	Minor page faults	84-86	Faults without disk I/O
SZ	Size in pages	694	Virtual memory size in pages

Key Observations:

- **Load processes (PIDs 3579, 7140)** show:
  - High CPU usage (89.5% for the active one)
  - Nice value of 15 (lower priority as specified)
  - **Minor page faults: 84-86** (important for real-time analysis)
  - No major page faults (good for performance)
  - Running on CPU cores 3 and 4

Exercise 1b: Process Tree Visualization

Command Used:

```
pstree -acghlpsUu
```

Options explained:

- `-a`: Show command line arguments
- `-c`: Don't compact identical subtrees
- `-g`: Show process group IDs
- `-h`: Highlight current process
- `-l`: Long lines (don't truncate)
- `-p`: Show PIDs
- `-s`: Show parent processes
- `-U`: Use UTF-8 line drawing
- `-u`: Show user transitions

Output (relevant section):

```
systemd,1,1
  ├─init-systemd(Ub,2,
  │   ├─SessionLeader,404,404
  │   │   └─Relay(406),405,404
  │   │       └─bash,406,406,zaka
  │   │           ├─load,3579,3579 MyLabel 10 15 2000000000
  │   │           └─load,7500,7500 MyLabel 10 15 2000000000
  │   ├─SessionLeader,3912,3912
  │   │   └─Relay(3919),3913,3912
  │   │       └─bash,3919,3919,zaka
  │   │           └─pstree,7523,7523 -acghlpsUu
```

Process Hierarchy Analysis:

- The `load` processes (3579, 7500) are children of bash (406)
- Shows the complete process ancestry from systemd (PID 1)
- Multiple terminal sessions visible (different bash instances)
- Clear parent-child relationships with PIDs and process group IDs

---

Exercise 1c: Filtered Process List

Command Used (showing all user processes):

```
ps -ef | grep $(whoami)
```

Output:

```
zaka         406     405  0 16:34 pts/0    00:00:00 -bash
zaka         460       1  0 16:34 ?        00:00:00 /lib/systemd/systemd --user
zaka         464     460  0 16:34 ?        00:00:00 (sd-pam)
zaka         474     460  0 16:34 ?        00:00:00 /usr/bin/pipewire
zaka         478     407  0 16:34 pts/1    00:00:00 -bash
zaka         495     460  0 16:34 ?        00:00:00 /usr/bin/dbus-daemon --session
--address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
zaka        3579     406  0 16:44 pts/0    00:00:01 ./load MyLabel 10 15
2000000000
zaka        3919    3913  0 16:45 pts/2    00:00:00 -bash
zaka        7961     406 97 16:58 pts/0    00:00:04 ./load MyLabel 10 15
2000000000
zaka        7982    3919  0 16:58 pts/2    00:00:00 ps -ef
zaka        7983    3919  0 16:58 pts/2    00:00:00 grep --color=auto zaka
```

More Specific Command (filtering for load processes only):

```
ps -ef | grep load
```

Output:

```
zaka        3579     406  0 16:44 pts/0    00:00:01 ./load MyLabel 10 15
2000000000
zaka        8135     406 91 16:59 pts/0    00:00:08 ./load MyLabel 10 15
2000000000
zaka        8170    3919  0 16:59 pts/2    00:00:00 grep --color=auto load
```

Key Information:

- The first command shows all processes owned by user `zaka`
- The second command specifically filters for `load` processes
- Two `load` processes visible (PIDs 3579, 8135)
- Process 8135 using 91% CPU (actively running)
- Shows cumulative CPU time (00:00:08 for the active process)
- The grep command itself appears in the output (PID 8170)

Analysis:

The specific `grep load` command provides a cleaner view focused on our test processes:

- **PID 3579**: First load process, mostly idle (0% CPU)
- **PID 8135**: Second load process, actively consuming CPU (91%)
- Clear progression of CPU time from 1 second to 8 seconds
- Both processes running in the same terminal (pts/0)

Understanding Minor Page Faults

What is a Minor Page Fault?

A **minor page fault** occurs when:

1. A process accesses a memory page that is valid but not currently in the CPU's page table
2. The page is already in RAM (no disk I/O required)
3. The kernel only needs to update the page table mappings

Why Minor Page Faults Matter for Real-Time Systems:

1. **Latency Impact**: Even without disk I/O, they introduce unpredictable delays
2. **Jitter**: Cause timing variations that affect deterministic behavior
3. **CPU Overhead**: Kernel must handle the fault, interrupting normal execution
4. **Unpredictability**: Can occur at any time during execution