TRƯỜNG ĐẠI HỌC BÁCH KHOA – ĐẠI HỌC QUỐC GIA TP.HCM
TRUNG TÂM KỸ THUẬT ĐIỆN TOÁN
268 Lý Thường Kiệt, Phường 14, Quận 10, TP.Hồ Chí Minh
Điện thoại: 84-8-3864 7256 ext 5371 – Fax: 84-8-3865 8687
Website: **www.cce.hcmut.edu.vn**
E-mail: **dientoan@hcmut.edu.vn**

Lab 8 - System call

1 Background

1.1 System call

We have worked with a variety of functions that your program can invoke to perform the system related functionality. To be more details, we have been introduced in theory that these function fall into two categories:
- A library function is an ordinary function that resides in a library external to your program.
- A system call is implemented in the Linux kernel.

Linux currently provides about 200 different system calls. In this lab, we have a practice to use these system calls or the libraries that in their internal code call these system calls to interract with Linux kernel.

1.2 System manipulation to work with processes's problem

1.2.1 Memory leak

- General advice for memory leaks:
  - Make sure your dynamically allocated memory does in fact get freed.
  - Don't allocate memory and forget to assign the pointer.
  - Don't overwrite a pointer with a new one unless the old memory is freed.
- General advice for memory errors:
  - Access and write to addresses and indices you're sure belong to you. Memory errors are different from memory leaks; they're often just IndexOutOfBoundsException type problems.
  - Don't access or write to memory after freeing it.
- Sometimes your leaks/errors can be linked to one another then the compiler treat these chain of problem inteferently. In such case, these ideas may give help:
  - List out the functions in your code that depend on/are dependent on the "offending" code that has the memory error. Follow the program's execution (maybe even in gdb perhaps), and look for precondition/postcondition errors. The idea is to trace your program's execution while focusing on the lifetime of allocated memory.
  - Try commenting out the "offending" block of code (within reason, so your code still compiles). Some helper tool Valgrind may bypass these code and don't show up the warning.
  - If all else fails, try looking it up. All helper tool as Valgrind, GDB etc. has further document, try out by yourself.
- To simulate this pratice, you will have an exercise of self-reading and trying a new command based on a provided documentation (man or help from command).

1.2.2 Deadlock

Deadlock can arise if four conditions hold simultaneously

- Mutual exclusion
- Hold and wait
- No preemption
- Circular wait

In a situation when the process cannot execution or get stuck in somewhere. We can do some analysis in the two following phase to gain more information of the problem:

- deadlock detection: figure out that deadlock occurred
- deadlock resolution: do something to resolve it

Strategies once Deadlock is detected:

- Abort all deadlocked processes
- Back up each deadlocked process to some previously defined checkpoint, and restart all process
  - Original deadlock may reoccur
- Successively abort deadlocked processes until deadlock no longer exists
- Successively preempt resources until deadlock no longer exists

In the previous stategies, someo of them require to select a process to do the policy. There is many criteria to select the victim one, for example:

- Least amount of processor time consumed so far
- Least number of lines of output produced so far
- Most estimated time remaining
- Least total resources allocated so far
- Lowest priority

2 Programming Interface of synchronization tools

2.1 Perf

2.1.1 Counting events

```
$ perf stat ./hello_thread_wait
Performance counter stats for './old_exp/synchronization/intlogger/intlog':

          8.682604      task-clock (msec)         #    0.076 CPUs utilized
               199      context-switches          #    0.023 M/sec
                 0      cpu-migrations            #    0.000 K/sec
               194      page-faults               #    0.022 M/sec
   <not supported>      cycles
                 0      stalled-cycles-frontend   #    0.00% frontend cycles idle
   <not supported>      stalled-cycles-backend
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

       0.114413049 seconds time elapsed
```

We can count the different event of context-switching, it show some statistic of CPU counter, but in the end the information have not given enough the whole big picture yet

```
Performance counter stats for 'mutex':

       4393.202436      task-clock (msec)
             1,100      context-switches
                 0      cpu-migrations
                64      page-faults
   <not supported>      cycles
                 0      stalled-cycles-frontend
   <not supported>      stalled-cycles-backend
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

     12.399297399 seconds time elapsed
```

```
Performance counter stats for 'mutex_c/main_mutex':

       4392.761154      task-clock (msec)
                33      context-switches
                 0      cpu-migrations
                65      page-faults
   <not supported>      cycles
                 0      stalled-cycles-frontend
   <not supported>      stalled-cycles-backend
   <not supported>      instructions
   <not supported>      branches
   <not supported>      branch-misses

      4.398933548 seconds time elapsed
```

We can get the statistic

```
# CPU counter statistics for the specified command:
perf stat command

# Detailed CPU counter statistics (includes extras) for the specified command:
perf stat -d command
```

```
# CPU counter statistics for the specified PID, until Ctrl-C:
perf stat -p PID

# Count system calls by type for the specified PID, until Ctrl-C:
perf stat -e 'syscalls:sys_enter_*' -p PID

# Count scheduler events for the specified PID, until Ctrl-C:
perf stat -e 'sched:*' -p PID
```

```
# Various CPU level 1 data cache statistics for the specified command:
perf stat -e L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores command

# Various CPU data TLB statistics for the specified command:
perf stat -e dTLB-loads,dTLB-load-misses,dTLB-prefetch-misses command

# Various CPU last level cache statistics for the specified command:
perf stat -e LLC-loads,LLC-load-misses,LLC-stores,LLC-prefetches command
```

```
# Sample on-CPU functions for the specified command, at 99 Hertz:
perf record -F 99 command


sudo perf record -F 99 ./hello_thread_spin
perf report
```

```
Samples: 577  of event 'cpu-clock', Event count (approx.): 5828282770
Overhead  Command          Shared Object       Symbol
  90.12%  hello_thread_sp  libpthread-2.19.so  [.] pthread_spin_lock
   6.76%  hello_thread_sp  hello_thread_spin   [.] _Z7f_countPv
   1.91%  hello_thread_sp  hello_thread_spin   [.] pthread_spin_lock@plt
   0.87%  hello_thread_sp  libpthread-2.19.so  [.] pthread_spin_unlock
   0.35%  hello_thread_sp  hello_thread_spin   [.] pthread_spin_unlock@pl
```

```
Samples: 1K of event 'cpu-clock', Event count (approx.): 20121211920
Overhead  Command          Shared Object       Symbol
  25.10%  hello_thread_mu  libpthread-2.19.so  [.] pthread_mutex_unlock
  20.53%  hello_thread_mu  libpthread-2.19.so  [.] pthread_mutex_lock
  17.77%  hello_thread_mu  [kernel.kallsyms]   [k] entry_SYSCALL_64_after_swapgs
   7.33%  hello_thread_mu  libpthread-2.19.so  [.] __lll_unlock_wake
   6.98%  hello_thread_mu  libpthread-2.19.so  [.] __lll_lock_wait
   2.66%  hello_thread_mu  [kernel.kallsyms]   [k] get_futex_key
   2.66%  hello_thread_mu  hello_thread_mutex  [.] _Z7f_countPv
   2.21%  hello_thread_mu  [kernel.kallsyms]   [k] do_futex
   1.96%  hello_thread_mu  [kernel.kallsyms]   [k] futex_wake
   1.81%  hello_thread_mu  [kernel.kallsyms]   [k] futex_wait_setup
   1.61%  hello_thread_mu  [kernel.kallsyms]   [k] entry_SYSCALL_64_fastpath
   1.56%  hello_thread_mu  [kernel.kallsyms]   [k] sys_futex
```

In general, we have some shortlist of perf capabilities

```
# perf

 usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

 The most commonly used perf commands are:
```

```
   annotate        Read perf.data (created by perf record) and display …
   archive         Create archive with object files with build-ids …
   bench           General framework for benchmark suites
   buildid-cache   Manage build-id cache.
   buildid-list    List the buildids in a perf.data file
   config          Get and set variables in a configuration file.
   data            Data file related processing
   diff            Read perf.data files and display the differential profile
   evlist          List the event names in a perf.data file
   inject          Filter to augment the events stream with additional information
   kmem            Tool to trace/measure kernel memory properties
   kvm             Tool to trace/measure kvm guest os
   list            List all symbolic event types
   lock            Analyze lock events
   mem             Profile memory accesses
   record          Run a command and record its profile into perf.data
   report          Read perf.data (created by perf record) and display the profile
   sched           Tool to trace/measure scheduler properties (latencies)
   script          Read perf.data (created by perf record) and display trace output
   stat            Run a command and gather performance counter statistics
   test            Runs sanity tests.
   timechart       Tool to visualize total system behavior during a workload
   top             System profiling tool.
   probe           Define new dynamic tracepoints
   trace           strace inspired tool

 See 'perf help COMMAND' for more information on a specific command.
```

Example

```
$ sudo perf stat -e L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores
./hello_thread_mutex
Thread 2: holding 1998862895
Thread 1: holding 2000000000

 Performance counter stats for './hello_thread_mutex':

                 0      L1-dcache-loads
                 0      L1-dcache-load-misses     #    0.00% of all L1-dcache hits
                 0      L1-dcache-stores

       4.391238522 seconds time elapsed

oslab@ubuntu:~$ sudo perf stat -e dTLB-loads,dTLB-load-misses,dTLB-prefetch-misses
/hello_thread_spin
Thread 1: holding 1995380238
Thread 2: holding 1998384682

 Performance counter stats for './old_exp/synchronization/mutex/mutex_c/mutex_c':
```

```
                 0      dTLB-loads
                 0      dTLB-load-misses          #    0.00% of all dTLB cache hits
   <not supported>      dTLB-prefetch-misses

       4.633764742 seconds time elapsed
```

```
$ perf list | grep L1-dcache
  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-prefetch-misses                          [Hardware cache event]
  L1-dcache-store-misses                             [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
```

```
perf stat -e L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores
 ./hello_thread_mutex
old_exp/synchronization/diningPhilosophy/din_c/dinPhl_cond_deadlock: Interrupt

 Performance counter stats for
'old_exp/synchronization/diningPhilosophy/din_c/dinPhl_cond_deadlock':

                 0      L1-dcache-loads
                 0      L1-dcache-load-misses     #    0.00% of all L1-dcache hits
                 0      L1-dcache-stores

      12.564236614 seconds time elapsed
```

2.1.2 Timed Profiling

```
perf record -F 99 -a -g -- sleep 30
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.132 MB perf.data (297 samples) ]
$ perf report --stdio
[kernel.kallsyms] with build id 3eccf666c1854c1d0e5c3b730ca148a1fc1a74ed not found,
continuing without symbols
# To display the perf.data header info, please use --header/--header-only options.
#
#
# Total Lost Samples: 0
#
# Samples: 297  of event 'cpu-clock'
# Event count (approx.): 2999999970
#
# Children      Self  Command      Shared Object      Symbol
# ........  ........  ...........  .................  ......................
#
    99.66%     0.00%  swapper      [kernel.kallsyms]  [k] 0xffffffff81037b2e
               |
               ---0xffffffff81037b2e
                  0xffffffff81064756

    99.66%     0.00%  swapper      [kernel.kallsyms]  [k] 0xffffffff810386b5
               |
               ---0xffffffff810386b5
                  0xffffffff81037b2e
                  0xffffffff81064756

    99.66%     0.00%  swapper      [kernel.kallsyms]  [k] 0xffffffff810c54da
               |
```

2.1.3 Event profiling

```
$ perf list | grep L1-dcache
  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-prefetch-misses                          [Hardware cache event]
  L1-dcache-store-misses                             [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
```

```
perf stat -e L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores
 ./hello_thread_mutex

 Performance counter stats for
'old_exp/synchronization/diningPhilosophy/din_c/dinPhl_cond_deadlock':

                 0      L1-dcache-loads
                 0      L1-dcache-load-misses     #    0.00% of all L1-dcache
hits
                 0      L1-dcache-stores

      12.564236614 seconds time elapsed
```

2.1.4 Static kernel tracing

In this indicator, we explain

```
# dd if=/dev/zero of=/dev/null bs=512 count=10000k
5242880000 bytes (5.2 GB) copied, 8.62452 s, 608 MB/s

# sudo perf stat -e 'syscalls:sys_enter_*' dd if=/dev/zero of=/dev/null bs=512
count=10000k
5242880000 bytes (5.2 GB) copied, 11.6846 s, 449 MB/s


# strace -c dd if=/dev/zero of=/dev/null bs=512 count=10000k
5242880000 bytes (5.2 GB) copied, 218.915 s, 23.9 MB/s
```

2.1.5 Static user tracing

```
/* file(s) to add */
$ perf buildid-cache --add ./hello_thread_mutex

/* path(s) to remove */
$ sudo perf buildid-cache --purge ./hello_thread_mutex

/* file(s) to update */
$ sudo perf buildid-cache --update ./ hello_thread_mutex
```

2.1.6 Dynamic tracing

Tracing user routine libc::malloc

```
$ sudo perf probe -x /lib/x86_64-linux-gnu/libc-2.19.so --add malloc

sudo perf record -e probe_libc:malloc -aR /home/oslab/aggsum/main 10000 10

$ sudo perf script
            perf  3683 [000]  7182.843866: probe_libc:malloc: (7f9ae7642a80)
            main  3686 [000]  7182.845379: probe_libc:malloc: (7fc84d073a80)
            main  3686 [000]  7182.845425: probe_libc:malloc: (7fc84d073a80)
            main  3686 [000]  7182.846009: probe_libc:malloc: (7fc84d073a80)
            sshd  1893 [000]  7182.908613: probe_libc:malloc: (7fc3cdb58a80)
            sshd  1893 [000]  7182.961814: probe_libc:malloc: (7fc3cdb58a80)
            main  3686 [000]  7183.007181: probe_libc:malloc: (7fc84d073a80)
            main  3686 [000]  7182.845425: probe_libc:malloc: (7fc84d073a80)
            main  3686 [000]  7182.846009: probe_libc:malloc: (7fc84d073a80)
            sshd  1893 [000]  7182.908613: probe_libc:malloc: (7fc3cdb58a80)
            sshd  1893 [000]  7182.961814: probe_libc:malloc: (7fc3cdb58a80)

$ sudo perf report
Samples: 7  of event 'probe_libc:malloc', Event count (approx.): 7
Overhead  Command  Shared Object  Symbol
  57.14%  main     libc-2.19.so   [.] malloc
  28.57%  sshd     libc-2.19.so   [.] malloc
  14.29%  perf     libc-2.19.so   [.] malloc

$ sudo perf sched latency
```

Task	Runtime ms	Switches	Average delay ms	Maximum delay ms	Maximum delay at
perf:3728	24.686 ms	2	avg: 4.729 ms	max: 9.455 ms	max at: 7534.472857 s
rcu_sched:7	0.059 ms	4	avg: 0.076 ms	max: 0.274 ms	max at: 7534.466233 s
sshd:1893	120.897 ms	3070	avg: 0.051 ms	max: 3.602 ms	max at: 7534.546001 s
main:(12)	98.079 ms	8723	avg: 0.012 ms	max: 3.961 ms	max at: 7534.787900 s
ksoftirqd/0:3	0.208 ms	16	avg: 0.011 ms	max: 0.045 ms	max at: 7534.699788 s
kworker/u256:1:3274	99.403 ms	17419	avg: 0.003 ms	max: 2.991 ms	max at: 7534.545985 s
TOTAL:	343.333 ms	29234			

2.1.7 Scheduler analysis

```
$ sudo perf sched record -- ./hello_thread_mutex 10000 10

[ perf record: Woken up 9 times to write data ]
[ perf record: Captured and wrote 16.981 MB perf.data (158086 samples) ]

$ sudo perf script
          perf  3728 [000]  7534.463392: sched:sched_stat_runtime: comm=perf pid=3728 runtime=1049251 [ns] vruntime=316541184427 [ns]
          perf  3728 [000]  7534.463395: sched:sched_stat_sleep: comm=perf pid=3731 delay=21049733 [ns]
          perf  3728 [000]  7534.463396: sched:sched_wakeup: comm=perf pid=3731 prio=120 target_cpu=000
          perf  3728 [000]  7534.463400: sched:sched_stat_wait: comm=perf pid=3731 delay=0 [ns]

$ sudo perf report
Available samples
37K sched:sched_switch
37K sched:sched_stat_wait
20K sched:sched_stat_sleep
0 sched:sched_stat_iowait
41K sched:sched_stat_runtime
11 sched:sched_process_fork
20K sched:sched_wakeup
11 sched:sched_wakeup_new
22 sched:sched_migrate_task
```

2.1.8 Gathering statistic info

Record the event

```
$ sudo perf record -e 'cpu-clock' ./hello_thread_mutex

$ sudo perf script
          hello_thread_m  2213   910.724886:     250000 cpu-clock:
7f4164763185 __init_cpu_features (/lib/x86_64-linux-gnu/libc-2.19.so)
          hello_thread_m  2213   910.725308:     250000 cpu-clock:
7f4164b1bbed __get_cpu_features (/lib/x86_64-linux-gnu/libpthread-2.19.so)
          hello_thread_m  2213   910.725567:     250000 cpu-clock:
7f41647e526b intel_check_word (/lib/x86_64-linux-gnu/libc-2.19.so)
          hello_thread_m  2213   910.725857:     250000 cpu-clock:
ffffffff8182b70b _raw_spin_unlock_irqrestore ([kernel.kallsyms])
          hello_thread_m  2213   910.725567:     250000 cpu-clock:
ffffffff8182be5a fentry_SYSCALL_64_after_swapgs ([kernel.kallsyms])
          hello_thread_m  2213   910.725567:     250000 cpu-clock:
ffffffff81101b40 get_futex_key ([kernel.kallsyms])
```

TRƯỜNG ĐẠI HỌC BÁCH KHOA – ĐẠI HỌC QUỐC GIA TP.HCM
TRUNG TÂM KỸ THUẬT ĐIỆN TOÁN
268 Lý Thường Kiệt, Phường 14, Quận 10, TP.Hồ Chí Minh
Điện thoại: 84-8-3864 7256 ext 5371 – Fax: 84-8-3865 8687
Website: **www.cce.hcmut.edu.vn**
E-mail: **dientoan@hcmut.edu.vn**

2.2 Monitoring – Sysstat utilities

iostat	Used for CPU statistics and input/output statistics for the block devices and partitions and generate report.
mpstat	Used for processor related statistics and reports.
pidstat	Used for I/O, CPU, memory statistics for Linux processes and generate report.
tapestat	Used for the statistics for tape drives attached to Linux system.
cifsiostat	Used for generating reports CIFS statistics.
sar	Used for collects and saves all the system activities and report.

To get the command mannual page:

$ man command_name

For example:

```
$ man pidstat
NAME
       pidstat - Report statistics for Linux tasks.

$ man sar
NAME
       sar - Collect, report, or save system activity information.
```

$ sudo apt-get install sysstat

An illu\stration of iostat usage

```
$ iostat -x 1
Linux 4.4.0-142-generic (ubuntu)        10/19/2022      _x86_64_        (3 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           3.39    0.00    1.95    0.14    0.00   94.52
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s …      %util
sda               0.00     3.80   22.02    2.28   492.59   155.43    …    0.73
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    0.00    0.00    0.00  100.00
```

Try all other ultilities in the sysstat package. Depend on the measurment target and the availability of the associated devices on your own system, not all tools are applicable, i.e. tape or CIFS devices is missing on some systems.

```
$ mpstat -P ALL 1 2
Linux 4.4.0-142-generic (ubuntu)        10/21/2022      _x86_64_        (3 CPU)

11:53:20 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
11:53:21 PM  all    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
11:53:21 PM    0    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
11:53:21 PM    1    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
11:53:21 PM    2    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00

11:53:21 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
11:53:22 PM  all    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
11:53:22 PM    0    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
11:53:22 PM    1    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
11:53:22 PM    2    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
```

```
Average:      CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
Average:      all    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
Average:        0    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
Average:        1    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
Average:        2    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00  100.00
```

```
$ sar -P ALL 1 2
Linux 4.4.0-142-generic (ubuntu)        10/22/2022      _x86_64_        (3 CPU)

12:25:35 AM      CPU     %user     %nice   %system   %iowait    %steal     %idle
12:25:36 AM      all      0.00      0.00      0.00      0.00      0.00    100.00
12:25:36 AM        0      0.00      0.00      0.00      0.00      0.00    100.00
12:25:36 AM        1      0.00      0.00      0.00      0.00      0.00    100.00
12:25:36 AM        2      0.00      0.00      0.00      0.00      0.00    100.00

12:25:36 AM      CPU     %user     %nice   %system   %iowait    %steal     %idle
12:25:37 AM      all      0.00      0.00      0.00      0.00      0.00    100.00
12:25:37 AM        0      0.00      0.00      0.00      0.00      0.00    100.00
12:25:37 AM        1      0.00      0.00      0.00      0.00      0.00    100.00
12:25:37 AM        2      0.00      0.00      0.00      0.00      0.00    100.00

Average:         CPU     %user     %nice   %system   %iowait    %steal     %idle
Average:         all      0.00      0.00      0.00      0.00      0.00    100.00
Average:           0      0.00      0.00      0.00      0.00      0.00    100.00
Average:           1      0.00      0.00      0.00      0.00      0.00    100.00
Average:           2      0.00      0.00      0.00      0.00      0.00    100.00
```

2.3 Address santinizer

Xxx g++ -O0 -g -fsanitize=address -fno-omit-frame-pointer

```
$ gcc call_deleted_mem.c -o call_deleted_mem -fsanitize=address -static-libasan -g


$ ./call_deleted_mem
=================================================================
==48263== ERROR: AddressSanitizer: heap-use-after-free on address 0x60160000af90 at
pc 0x41ab67 bp 0x7ffc4fc98c00 sp 0x7ffc4fc98bf8
WRITE of size 1 at 0x60160000af90 thread T0
    #0 0x41ab66 (/home/oslab/exp/leakmem/call_deleted_mem+0x41ab66)
    #1 0x7f06c8e3bf44 (/lib/x86_64-linux-gnu/libc-2.19.so+0x21f44)
    #2 0x402166 (/home/oslab/exp/leakmem/call_deleted_mem+0x402166)
0x60160000af90 is located 0 bytes inside of 100-byte region
[0x60160000af90,0x60160000aff4)
freed by thread T0 here:
    #0 0x41200a (/home/oslab/exp/leakmem/call_deleted_mem+0x41200a)
```

```
    #1 0x41aac4 (/home/oslab/exp/leakmem/call_deleted_mem+0x41aac4)
    #2 0x7f06c8e3bf44 (/lib/x86_64-linux-gnu/libc-2.19.so+0x21f44)
previously allocated by thread T0 here:
    #0 0x4120ea (/home/oslab/exp/leakmem/call_deleted_mem+0x4120ea)
    #1 0x41aab4 (/home/oslab/exp/leakmem/call_deleted_mem+0x41aab4)
    #2 0x7f06c8e3bf44 (/lib/x86_64-linux-gnu/libc-2.19.so+0x21f44)
Shadow bytes around the buggy address:
  0x0c033fff95a0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c033fff95b0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c033fff95c0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c033fff95d0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c033fff95e0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x0c033fff95f0: fa fa[fd]fd fd fd fd fd fd fd fd fd fd fd fa
  0x0c033fff9600: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c033fff9610: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c033fff9620: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c033fff9630: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c033fff9640: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:     fa
  Heap righ redzone:     fb
  Freed Heap region:     fd
  Stack left redzone:    f1
  Stack mid redzone:     f2
  Stack right redzone:   f3
  Stack partial redzone: f4
  Stack after return:    f5
  Stack use after scope: f8
  Global redzone:        f9
  Global init order:     f6
  Poisoned by user:      f7
  ASan internal:         fe
==48263== ABORTING
```

2.4 Strace

Trace system call and signal

```
$ strace -c dd if=/dev/zero of=/dev/null bs=512 count=10000k
10240000+0 records in
10240000+0 records out
5242880000 bytes (5.2 GB) copied, 399.553 s, 13.1 MB/s
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 51.50    0.342132           0  10240003           read
 48.50    0.322166           0  10240003           write
```

```
  0.00    0.000000           0        10         3 open
  0.00    0.000000           0        10           close
  0.00    0.000000           0         5           fstat
  0.00    0.000000           0         1           lseek
  0.00    0.000000           0         9           mmap
  0.00    0.000000           0         4           mprotect
  0.00    0.000000           0         2           munmap
  0.00    0.000000           0         3           brk
  0.00    0.000000           0         4           rt_sigaction
  0.00    0.000000           0         3         3 access
  0.00    0.000000           0         2           dup2
  0.00    0.000000           0         1           execve
  0.00    0.000000           0         1           arch_prctl
------ ----------- ----------- --------- --------- ----------------
100.00    0.664298              20480061         6 total
```

2.5 Htop

Htop is a free (GPL) ncurses-based process viewer for Linux.

$ htop
The program 'htop' is currently not installed. You can install it by typing:
$ sudo apt-get install htop

```
$ htop

  1  [                                                      0.0%]     Tasks: 33, 4 thr; 1 running
  2  [                                                      0.0%]     Load average: 0.02 0.02 0.00
  3  [|                                                     1.3%]     Uptime: 2 days, 16:00:17
  Mem[||||||||||||||||||||||||||||||||||||||||||||||||      154/974MB]
  Swp[                                                      0/1021MB]

    PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
      1 root       20   0 33524  4036  2668 S  0.0  0.4  0:02.12 /sbin/init
  46538 root       20   0 23664  2284  2036 S  0.0  0.2  0:00.04 ├─ cron
  46390 root       20   0  173M  4328  3924 S  0.0  0.4  0:00.00 ├─ /usr/lib/x86_64-lin…
  46392 root       20   0  173M  4328  3924 S  0.0  0.4  0:00.00 │  └─ /usr/lib/x86_64-…
  45945 root       20   0 48816  4508  3760 S  0.0  0.5  0:00.07 ├─ /lib/systemd/systemd --system
  45976 root       20   0  127M 60864 60476 S  0.0  6.1  0:00.44 │  └─ /lib/systemd/systemd-journald
   4061 root       20   0 15828  2112  1968 S  0.0  0.2  0:00.00 ├─ /sbin/getty -8 38400 tty1
    927 root       20   0 61392  5564  4892 S  0.0  0.6  0:00.12 ├─ /usr/sbin/sshd -D
  47559 root       20   0  100M  6592  5612 S  0.0  0.7  0:00.02 │  ├─ sshd: oslab [priv]
  47581 oslab      20   0  100M  4200  3228 S  0.0  0.4  0:00.09 │  │  └─ sshd: oslab@pts/5
  47582 oslab      20   0 22432  5188  3344 S  0.0  0.5  0:00.09 │  │     └─ -bash
  47514 root       20   0  100M  6656  5668 S  0.0  0.7  0:00.02 │  ├─ sshd: oslab [priv]
  47540 oslab      20   0  100M  4152  3176 S  0.0  0.4  0:00.23 │  │  └─ sshd: oslab@pts/1
  47541 oslab      20   0 22540  5424  3472 S  0.0  0.5  0:00.28 │  │     └─ -bash
  48010 oslab      20   0 26292  4116  3032 R  0.7  0.4  0:00.36 │  │        └─ htop
  47358 root       20   0  100M  6544  5572 S  0.0  0.7  0:00.00 │  ├─ sshd: oslab [priv]
  47380 oslab      20   0  100M  4284  3320 S  0.0  0.4  0:00.10 │  │  └─ sshd: oslab@pts/4
  47381 oslab      20   0 22432  5252  3412 S  0.0  0.5  0:00.08 │  │     └─ -bash
  47232 root       20   0  100M  6544  5560 S  0.0  0.7  0:00.04 │  └─ sshd: oslab [priv]
```

```
  47254 oslab      20   0  100M  4064  3092 S  0.0  0.4  0:00.10 |     `- sshd: oslab@pts/3
  47255 oslab      20   0 22432  5252  3412 S  0.0  0.5  0:00.10 |        `- -bash
    907 root       20   0 15828  2096  1952 S  0.0  0.2  0:00.00 |- /sbin/getty -8 38400 tty6
    904 root       20   0 15828  2060  1916 S  0.0  0.2  0:00.00 |- /sbin/getty -8 38400 tty3
    903 root       20   0 15828  2048  1904 S  0.0  0.2  0:00.00 |- /sbin/getty -8 38400 tty2
    898 root       20   0 15828  2100  1952 S  0.0  0.2  0:00.00 |- /sbin/getty -8 38400 tty5
    896 root       20   0 15828  2136  1988 S  0.0  0.2  0:00.00 |- /sbin/getty -8 38400 tty4
    814 root       20   0 15536  2040  1556 S  0.0  0.2  0:00.06 |- upstart-socket-bridge --daemon
    686 root       20   0 10240  4260  1964 S  0.0  0.4  0:00.53 |- dhclient -1 -v -pf …
    577 root       20   0 15812  2220  1504 S  0.0  0.2  0:00.13 |- upstart-file-bridge --daemon
    522 syslog     20   0  249M  2940  2520 S  0.0  0.3  0:01.84 |- rsyslogd
    525 syslog     20   0  249M  2940  2520 S  0.0  0.3  0:01.05 |  |- rsyslogd
    524 syslog     20   0  249M  2940  2520 S  0.0  0.3  0:00.06 |  |- rsyslogd
    523 syslog     20   0  249M  2940  2520 S  0.0  0.3  0:00.70 |  `- rsyslogd
    516 root       20   0 43464  3300  2936 S  0.0  0.3  0:00.17 |- /lib/systemd/systemd-logind
    505 messagebu  20   0 39388  2880  2356 S  0.0  0.3  0:00.51 |- dbus-daemon --system --fork
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

2.6 mtrace

The mtrace interprets the output from when the MALLOC_TRACE environment variable is set.

```
$ mtrace helloleak.c
No memory leaks.
```

2.7 Tracing running process with Linux native interface

Linux comes up with some interface to access running process.
Read executable binary file

```
objdump -d hello > hello_asm_dump
objdump -s hello > hello_hex_dump
```

Process listing

ps aux

Information

CODE	NORMAL	HEADER
%C	pcpu	%CPU
%G	group	GROUP
%P	ppid	PPID
%U	user	USER
%a	args	COMMAND
%c	comm	COMMAND
%g	rgroup	RGROUP
%n	nice	NI
%p	pid	PID
%r	pgid	PGID

```
        %t      etime     ELAPSED
        %u      ruser     RUSER
        %x      time      TIME
        %y      tty       TTY
        %z      vsz       VSZ
```

Example
ps –eo "%p %y %x %c %a"

Process memory layout

```
cat /proc/${pid}/maps
```

Process status

```
cat /proc/${pid}/status
cat /proc/${pid}/syscall
cat /proc/${pid}/stack
```

Process environment

```
xargs -0 -L1 -a /proc/${pid}/environ
```

2.8 Tracing running process with debugger GDB

GDB attach to a running process (with pid). By default, we can use an option

```
$ ./any_program

# Using the instruction from the previous section to retrieve the process ID (PID)
$ cat /proc/<PID>/maps

$ sudo gcore -o process <PID>

# Remember to run in super-previledge with sudo or root account
$ sudo gdb -p <PID>

Use GDB command

quit
```

In the case you forgot the option or you want to change the target process

```
(gdb) attach 48046
Attaching to process 48046
Reading symbols from /home/oslab/exp/helloworldwait/a.out...done.
Reading symbols from /usr/lib/x86_64-linux-gnu/libasan.so.0...(no debugging symbols
found)...done.
Loaded symbols for /usr/lib/x86_64-linux-gnu/libasan.so.0
Reading symbols from /lib/x86_64-linux-gnu/libpthread.so.0...Reading symbols from
/usr/lib/debug//lib/x86_64-linux-gnu/libpthread-2.19.so...done.
```

```
done.
[New LWP 48048]
[New LWP 48047]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Loaded symbols for /lib/x86_64-linux-gnu/libpthread.so.0
Reading symbols from /lib/x86_64-linux-gnu/libc.so.6...Reading symbols from
/usr/lib/debug//lib/x86_64-linux-gnu/libc-2.19.so...done.
done.
Loaded symbols for /lib/x86_64-linux-gnu/libc.so.6
Reading symbols from /lib/x86_64-linux-gnu/libdl.so.2...Reading symbols from
/usr/lib/debug//lib/x86_64-linux-gnu/libdl-2.19.so...done.
done.
Loaded symbols for /lib/x86_64-linux-gnu/libdl.so.2
Reading symbols from /lib64/ld-linux-x86-64.so.2...Reading symbols from
/usr/lib/debug//lib/x86_64-linux-gnu/ld-2.19.so...done.
done.
Loaded symbols for /lib64/ld-linux-x86-64.so.2
Reading symbols from /lib/x86_64-linux-gnu/libgcc_s.so.1...(no debugging symbols
found)...done.
Loaded symbols for /lib/x86_64-linux-gnu/libgcc_s.so.1
(gdb) help
List of classes of commands:

aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
```

Try by your self help cmd to use the GDB features. For example

```
(gdb) help stack
...

List of commands:

backtrace -- Print backtrace of all stack frames
bt -- Print backtrace of all stack frames
```

```
down -- Select and print stack frame called by this one
frame -- Select and print a stack frame
return -- Make selected stack frame return to its caller
select-frame -- Select a stack frame without printing anything
up -- Select and print stack frame that called this one
```

```
(gdb) info stack
#0  0x00007f3379c5e65b in pthread_join (threadid=139859051591424,
thread_return=0x0) at pthread_join.c:92
#1  0x0000000000400c8b in main (argc=1, argv=0x7ffeec63cb48) at hello_thread.c:35
```

2.9 Valgrind

Valgrind is a flexible program for debugging and profiling Linux executables. It consists of a core, which provides a synthetic CPU in software, and a series of debugging and profiling tools. The architecture is modular, so that new tools can be created easily and without disturbing the existing structure.

```
$ valgrind --leak-check=full ./memleak_3sec
==46987== Memcheck, a memory error detector
==46987== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==46987== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==46987== Command: ./memleak_3sec
==46987==
==46987==
==46987== HEAP SUMMARY:
==46987==     in use at exit: 4 bytes in 1 blocks
==46987==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==46987==
==46987== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==46987==    at 0x4C2AB80: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-
linux.so)
==46987==    by 0x40058E: main (in /home/oslab/memleak/memleak_3sec)
==46987==
==46987== LEAK SUMMARY:
==46987==    definitely lost: 4 bytes in 1 blocks
==46987==    indirectly lost: 0 bytes in 0 blocks
==46987==      possibly lost: 0 bytes in 0 blocks
==46987==    still reachable: 0 bytes in 0 blocks
==46987==         suppressed: 0 bytes in 0 blocks
==46987==
==46987== For counts of detected and suppressed errors, rerun with: -v
==46987== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

2.10 iperf

Iperf is tool used for generating the network traffic and measuring the traffic performance while it transfer the generated data between its client and its server.
To start a measurement experiment, we need 2 site act as server and client.

To start an iperf server

```
# Replace the IP address by your own IP address
$ iperf -s -B 192.168.232.128
------------------------------------------------------------
Server listening on TCP port 5001
Binding to local address 192.168.232.128
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

It needs a server existed to create a client to connect to that server and do the measurement

```
$ iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.232.128 port 5001 connected with 192.168.232.128 port 44508
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  32.6 GBytes  27.9 Gbits/sec
```

3 System analysis

3.1 Legacy analysis

3.1.1 Statistic analysis and snapshot analysiss

A snapshot sample is create by choosing a checkpointing status (a fixed point in chronological time) with ensuring the systematic consistency. It can be achieved by sampling only the subject where the initial event has occurred but the subsequent event has yet to occur (active subjects), or by sampling only subjects where both the initial and subsequent events have occurred (inactive subjects).
A snapshot tool translate the methodology for assessing statistical quality into a series of clear question with pre-defined answer. The results are easy to understand and use. Before getting into the assessment, it is necessary to draw a map of the statistical system. This initial step is important to identify which institutions/actors are involved in the statistical production.
How do we find the information to fill in the 'snapshot':

- A development partner could fill in the tool, based on available documents, exchanges of views within the development partner communit.
- The tool could initially be filled in by an external consultant. If the tool is used in an periodical monitoring exercise, the effort required to update the information is substantially less
- An alternative is that the consultant and the local data producer work together through a small and targeted workshop, dedicated to filling a checklist.
- A promising option would be that the tool fills it in and share the outcomes within development partner. We have seen these collaboration community in our technical sector, i.e. Intel dominated OpenStack, Google and Facebook backing of TensorFlow or Torch and even those approach are collaborative each orther.

In general, we can have the outcomes in an intuitive representation: checklist or table of mapping. But in genneral, they have a deterministic (know or predefine) factor. It has lack of support non-deterministic contex.
An illustration is Sanity testing[1] which is a subset of regression testing. Sanity testing is performed to ensure that the code changes that are made are working as properly. Sanity testing is a stoppage to check whether testing for the build can proceed or not. The focus of the team during sanity testing process is to validate the functionality of the application and not detailed testing. Sanity testing is generally performed on build where the production deployment is required immediately like a critical bug fix.:

- Subset of Regression Testing: Sanity testing is a subset of regression testing and focuses on the smaller section of the application.
- Unscripted: Most of the times sanity testing is not scripted.
- Not documented: Usually sanity testing is undocumented.
- Narrow and deep: Sanity testing is narrow and deep approach of testing where limited functionalities are covered deeply.
- Performed by testers: Sanity testing is normally performed by testers.

[1] https://www.geeksforgeeks.org/sanity-testing-software-testing/

An advantages of Sanity Testing is help in quickly identify defects in the core functionality. It can be carried out in lesser time as no documentation is required for sanity testing. In contrast, It focuses only on the functions and commands of the system application. Meanwhile, it is not possible to cover all the test cases in test scenarios.

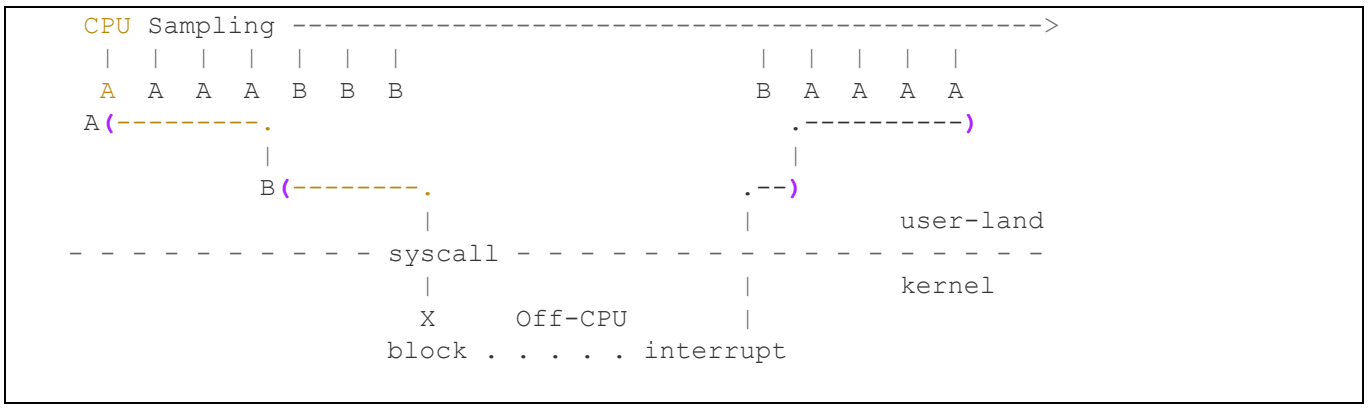
3.1.2 Benchmark

In short, benchmark is any standard or reference by which others can be measured or judged.
A simple example is the referenced product price. There are different types of benchmarking: internal, external, performance, and practice.

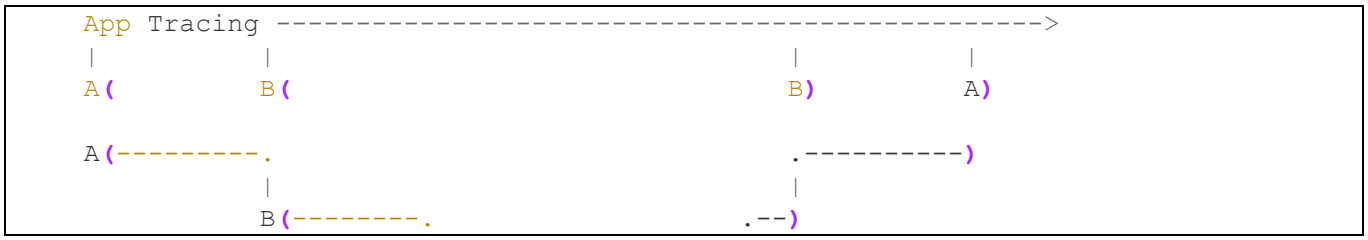
In our system development, we have some circumstance of cross-Platform Benchmark. The cross platform method help overcome the lack of environment or infrastructure when push comes to crunch to provide a comparison to the newest devices on the market. In some system development model, this technique helps enhance the pipeline production model in somewhere the future produce which is not existed yet or under producing can be taken in action. For example, we run a verification of ASIC design on a gate-library system.

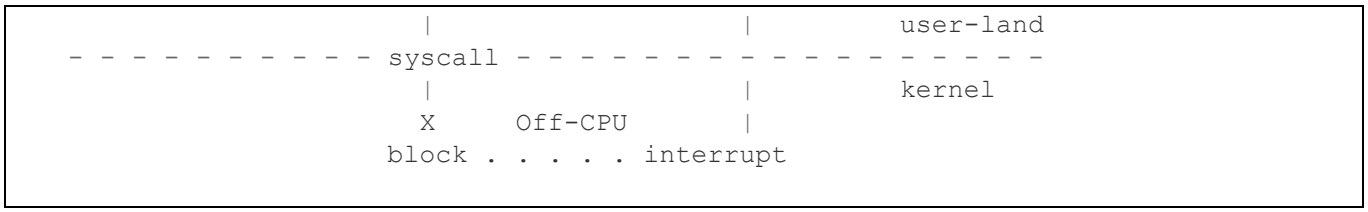
3.2 Off-CPU analysis

Consider application function A() call function B(), which makes a blocking system call.

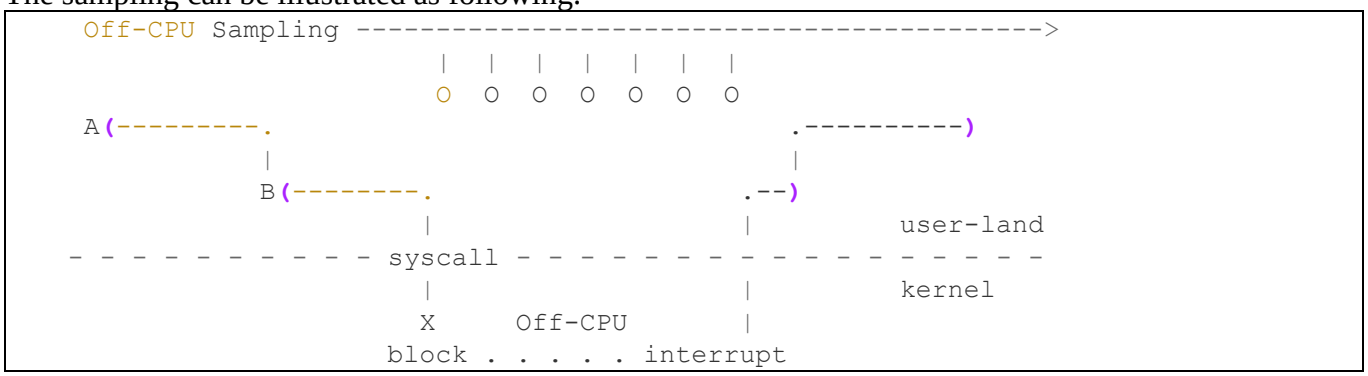


The approach of application tracing will consume an amount of computation that may trace the border of application and use some estimation methode to deduct the target measurement time. But in fact, a disadvantage is that you either trace all application function, which can significantly impact the performance and somehow it may change the application behavior far from the original.





Instead, ***Off-CPU tracing capture all wait event for any application***
The sampling can be illustrated as following:



In short, it captures the syscall of wait event, then we can deduct the amount of time when all process are really under working. This method has an zero impact on the normal operation of application function but it still has a limitation since it is the amount which is effected by all running applications.

```
$ sudo perf stat -e cycles ./hello_thread_spin

 Performance counter stats for './ comes to crunch':

         12.249127      task-clock:u (msec)       #    0.002 CPUs utilized

       5.210557018 seconds time elapsed

$ perf report --sort cpu
```

3.3 Active benmarking

In general. benchmarking provides such a tool to work with the power of the system we want measure without understanding what they are testing or checking that the results are valid. Therefor, we end up with a poor development of architecture choices . This situation can be illustration as following state: you want to benchmark an object A but what you are really doing measurement is belong to object B and in the end of the day, you make a conclusion that you have measure object C, that a bad result since the accuracy of the benchmarking results may act as an important factor for future work.
Benchmarks are usually configured to run and are ignored until they have completed. That is passive benchmarking where we collect the data of the system behavior but they are not fully an information provider. An alternative approach employs the on-site analysis the performance while the benchmark are

still running. To perform such an analysis, you may have equipped some performance analysis tools includes: vmstat, iostat, top, tcpdump/snoop, perf, gdb, strace.
One suggested framework use in performance analysis, this list is made based on personal opinion only:

1. Ad-hoc checklist method, i.e. Run A if B do C
2. Problem statement method: identify is problem characteristic
3. Read the fine manual: study performance tool and metrics
4. Scientific statement method: make a hypothesis or prediction to accept/reject
5. Continous improvement method: observe/orient/decice/act (OoCA)
6. Work load characterization method : root of load, the type and its stability
7. Device and conqueue method or elimination in procedure: (DiCPA)
    a. Divide big problem to subproblem
    b. Choose a test
    c. Perform test
    d. Analysis, decide to go back (b) if the test is not solid or (a) for wrong problem identification.
8. Time division method: split timeslot/synchronization epoch and measure it
9. Diversify the direction: measure by different layers, tools, indicator (ultilization, saturation, errors), collaboration (request, response), dynamicity (off-cpu, dynamic tracing/benchmarking)

4 Tool in analyzing

4.1 Monitor memory leak

4.1.1 System statistic info

The Linux system provide some statistic attach to a living process with <PID>. It provides much information in regard to answering the question "How much memory are applications really using?"
We can use the info of the "proportional set size" (PSS) of a process which is the count of pages it has in memory, where each page is divided by the number of processes sharing it.[2]
The /proc/PID/smaps is an extension based on maps, showing the memory consumption for each of the process's mappings.

Capture the information

```
$ sudo cat /proc/409/smaps
55c798fd9000-55c798fea000 r-xp 00000000 08:01 1245 /sbin/upstart-udev-bridge
Size:                  4 kB
Rss:                   0 kB
Pss:                   0 kB
Shared_Clean:          0 kB
Shared_Dirty:          0 kB
Private_Clean:         0 kB
Private_Dirty:         0 kB
Referenced:            0 kB
Anonymous:             0 kB
AnonHugePages:         0 kB
Shared_Hugetlb:        0 kB
Private_Hugetlb:       0 kB
Swap:                  0 kB
SwapPss:               0 kB
KernelPageSize:        4 kB
MMUPageSize:           4 kB
Locked:                0 kB
```

There are some important indices:

- (RSS) the amount of the mapping that is currently resident in RAM
- (PSS) the process' proportional share of this mapping
- I do Shared_Clean + Shared_Dirty + Private_Clean + Private_Dirty for every smaps entry (process may have multi-thread) for the process then I must get the RSS [3]

```
$ ps -o pid,rss
```

Compare the captured information ca

Remind the command to capture the information

```
$ cat /proc/<PID>/smaps > BeforeMemInc.txt
```

[2] https://lwn.net/Articles/230975/ ELC: How much memory are applications really using?
[3] https://unix.stackexchange.com/questions/33381/getting-information-about-a-process-memory-usage-from-proc-pid-smaps

```
$ ./run_mem_inc_program
$ cat /proc/<PID>/smaps > AfterMemInc.txt
```

To compare the 2 output file

```
$ diff -u BeforeMemInc.txt AfterMemInc.txt
```

BeforeMemInc.txt		AfterMemInc.txt		
55c798fd9000-55c798fea000 r-xp 000...		55c798fd9000-55c798fea000 r-xp 0000...		
Size:	4 kB	Size:	4 kB	
Rss:	0 kB	Rss:	0 kB	
Pss:	0 kB	Pss:	0 kB	
Shared_Clean:	0 kB	Shared_Clean:	0 kB	
Shared_Dirty:	0 kB	Shared_Dirty:	0 kB	
Private_Clean:	0 kB	Private_Clean:	0 kB	
Private_Dirty:	0 kB	Private_Dirty:	0 kB	
Referenced:	28 kB	Referenced:	36 kB	
Anonymous:	28 kB	Anonymous:	36 kB	#INCREASE MEM
AnonHugePages:	0 kB	AnonHugePages:	0 kB	

If you figure out some interesting information, you can go further investigating with GDB

4.1.2 Work with GDB dumpmem

In this section, we will analyze more details inside the process (meaning in executing) memory. The step will try to attach GDB to the process and use the memory dump to retrieve the data (require the root to access other user process memory region). Finally, we use the mapping "string" to revese map the content to human-readable string (aka function name).

```
$ ./hello_thread

$ cat /proc/46660/maps

$ sudo gcore -o process <PID>

$ sudo gdb -p <PID>

dump memory ./dump_out 0x00400000 0x00401000

quit

$  strings dump_out
```

4.1.3 Work with Address santinizer

```
$ gcc heap_overflow.c -o heap_overflow -fsanitize=address -static-libasan -g

$ ./heap_overflow
=================================================================
==48314== ERROR: AddressSanitizer: heap-buffer-overflow on address 0x60040000dffc
at pc 0x41ab93 bp 0x7ffd4fcfaca0 sp 0x7ffd4fcfac98
WRITE of size 1 at 0x60040000dffc thread T0
    #0 0x41ab92 (/home/oslab/exp/leakmem/heap_overflow+0x41ab92)
```

```
    #1 0x7f38ce8c0f44 (/lib/x86_64-linux-gnu/libc-2.19.so+0x21f44)
    #2 0x402166 (/home/oslab/exp/leakmem/heap_overflow+0x402166)
0x60040000dffc is located 0 bytes to the right of 12-byte region
[0x60040000dff0,0x60040000dffc)
allocated by thread T0 here:
    #0 0x4120ea (/home/oslab/exp/leakmem/heap_overflow+0x4120ea)
    #1 0x41aab4 (/home/oslab/exp/leakmem/heap_overflow+0x41aab4)
    #2 0x7f38ce8c0f44 (/lib/x86_64-linux-gnu/libc-2.19.so+0x21f44)
Shadow bytes around the buggy address:
  0x0c00ffff9ba0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c00ffff9bb0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c00ffff9bc0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c00ffff9bd0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c00ffff9be0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
=>0x0c00ffff9bf0: fa fa fa fa fa fa fa fa fa fa fa fa fa fa 00[04]
  0x0c00ffff9c00:fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c00ffff9c10: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c00ffff9c20: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c00ffff9c30: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c00ffff9c40: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:     fa
  Heap righ redzone:     fb
  Freed Heap region:     fd
  Stack left redzone:    f1
  Stack mid redzone:     f2
  Stack right redzone:   f3
  Stack partial redzone: f4
  Stack after return:    f5
  Stack use after scope: f8
  Global redzone:        f9
  Global init order:     f6
  Poisoned by user:      f7
  ASan internal:         fe
==48314== ABORTING
```

4.1.4 Work with valgrind

In order to gaining analytical information, the valgrind has an option “--verbose” to make it more internal announcement.

```
$ valgrind --leak-check=full --verbose ./memleak_3sec
==47003== Memcheck, a memory error detector
==47003== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.
==47003== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==47003== Command: ./memleak_3sec
==47003==
--47003-- Valgrind options:
```

```
--47003--    --leak-check=full
--47003--    --verbose
...
--47003-- Reading syms from /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so
--47003--   Considering /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so ..
--47003--   .. CRC mismatch (computed 3df18bf1 wanted 14fefe1c)
--47003--    object doesn't have a symbol table
==47003== WARNING: new redirection conflicts with existing -- ignoring it
--47003--     old: 0x04019d70 (strlen              ) R-> (0000.0) 0x380764b1 ???
--47003--     new: 0x04019d70 (strlen              ) R-> (2007.0) 0x04c2e1a0 strlen
--47003-- REDIR: 0x4019b20 (ld-linux-x86-64.so.2:index) redirected to 0x4c2dd50
...
--47003-- REDIR: 0x4eba120 (libc.so.6:free) redirected to 0x4c2bd80 (free)
==47003==
==47003== HEAP SUMMARY:
==47003==     in use at exit: 4 bytes in 1 blocks
==47003==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==47003==
==47003==
==47003== LEAK SUMMARY:
==47003==    definitely lost: 4 bytes in 1 blocks
==47003==    indirectly lost: 0 bytes in 0 blocks
==47003==      possibly lost: 0 bytes in 0 blocks
==47003==    still reachable: 0 bytes in 0 blocks
==47003==         suppressed: 0 bytes in 0 blocks
==47003==
==47003== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
==47003== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

4.2 Stack tracing coredump

Using the GDB to tracking the coredump

```
$ g++ -std=c++11 -g -o buggy buggy.c

$ ./buggy
i is 0
i is 1
i is 2
i is 3
i is 4
Segmentation fault (core dumped)

# Verify the dump file is automatically generated
$ file core
core: ELF 64-bit LSB  core file x86-64, version 1 (SYSV), SVR4-style, from
'./buggy'

$ gdb buggy core
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.3) 7.7.1
```

```
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from buggy...done.
Core was generated by `./buggy'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x0000000000400848 in main () at buggy.c:13
13              * noptr = i;
(gdb)
```

The debugger has more utilities to get more detailed on the error.

```
(gdb) backtrace
#0  0x0000000000400848 in main () at buggy.c:13
(gdb) backtrace full
#0  0x0000000000400848 in main () at buggy.c:13
        i = 5
        num = 0
        ptr = 0x1d32010
        noptr = 0x0
(gdb) info threads
  Id   Target Id         Frame
* 1    LWP 48485         0x0000000000400848 in main () at buggy.c:13
(gdb) thread apply all bt

Thread 1 (LWP 48485):
#0  0x0000000000400848 in main () at buggy.c:13
(gdb) thread apply all bt full

Thread 1 (LWP 48485):
#0  0x0000000000400848 in main () at buggy.c:13
        i = 5
        num = 0
        ptr = 0x1d32010
        noptr = 0x0
```

5 Exercise

PROBLEM 1

Write a program with a function print_val() to print to the screen a number (i.e. value = 1) store in variable named "val" and create a loop to call this function every 3-5 seconds.

```
...
    while(1) {
       print_val();

       sleep(3);
    }
...
```

- Using GDB to attach to the process running the written program.
- Add a break point to your defined function print_val() (using help break to read the instruction)
- Using continue or next to keep the process runing to next breakpoint or next command.
- Using command p and s to display and set the new value print out to the screen