Operating Systems
Spring 2018

S02

Professor : Philippe Cudré-Mauroux
Assistant : Ines Arous

Submitted by Sylvain Julmy

Exercice 2

For the exercice, I have choose the following commands :

- `perf bench`
- `top`
- `kill`

`perf bench`

`perf bench` is a command that can launch a set of multi-threaded benchmarks to exercice various subsystems in the Linux kernel and system calls.
This command allow any devolopper to easely create benchmarks and run them in Linux. It is very usefull to test some very specific instructions of the microprocessor like **Compare-And-Swap** (CAS).
The figure 1 shows an example of an execution of the command. The system perform benchmarks for memory test.

`top`

The `top` command provides a dynamic real-time view of a running system. It can display system summary information as well as a list of processes or threads currently being managed by the Linux kernel[1].
The figure 2 shows the run of the command, we can show all the information related before (and in series 01).

`kill`

The command `kill` is used to send specific signal to the specified processes or process groups. A special case of this command is `kill -9` where the process is simply killed by the command, this signal cannot be caught or ignored, and the receiving process cannot perform any clean-up upon receiving this signal.
There are some exceptions[2] :

[1] `https://perf.wiki.kernel.org/index.php/Tutorial`
[2] `https://en.wikipedia.org/wiki/Signal_(IPC)`

```
snipy@snipy-anarchy ~ perf bench mem all
# Running mem/memcpy benchmark...
# function 'default' (Default memcpy() provided by glibc)
# Copying 1MB bytes ...

      16.276042 GB/sec
# function 'x86-64-unrolled' (unrolled memcpy() in arch/x86/lib/memcpy_64.S)
# Copying 1MB bytes ...

      10.850694 GB/sec
# function 'x86-64-movsq' (movsq-based memcpy() in arch/x86/lib/memcpy_64.S)
# Copying 1MB bytes ...

      14.361213 GB/sec
# function 'x86-64-movsb' (movsb-based memcpy() in arch/x86/lib/memcpy_64.S)
# Copying 1MB bytes ...

      12.056327 GB/sec

# Running mem/memset benchmark...
# function 'default' (Default memset() provided by glibc)
# Copying 1MB bytes ...

      17.132675 GB/sec
# function 'x86-64-unrolled' (unrolled memset() in arch/x86/lib/memset_64.S)
# Copying 1MB bytes ...

      11.909299 GB/sec
# function 'x86-64-stosq' (movsq-based memset() in arch/x86/lib/memset_64.S)
# Copying 1MB bytes ...

      17.438616 GB/sec
# function 'x86-64-stosb' (movsb-based memset() in arch/x86/lib/memset_64.S)
# Copying 1MB bytes ...

      17.438616 GB/sec
```

Figure 1: Example of the execution of the `perf bench mem all` command, that is lauching all the benchmarks related to memory.

- Zombie processes cannot be killed since they are already dead and waiting for their parent processes to reap them.
- Processes that are in the blocked state will not die until they wake up again.
- The init process is special: It does not get signals that it does not want to handle, and thus it can ignore SIGKILL. An exception from this exception is while init is ptraced on Linux.
- An uninterruptibly sleeping process may not terminate (and free its resources) even when sent SIGKILL. This is one of the few cases in which a UNIX system may have to be rebooted to solve a temporary software problem.

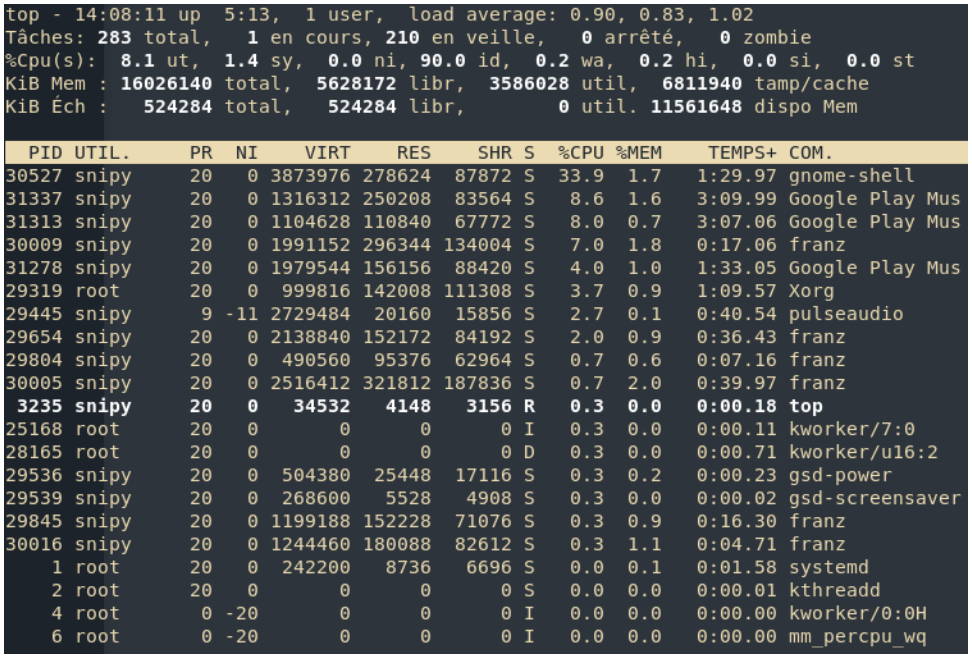
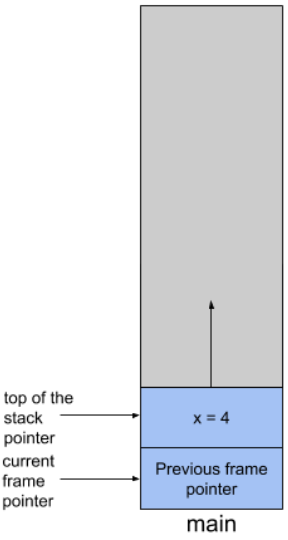


Figure 2: Execution of the `top` command, relevant information like Cpu usage, memory or processes are display.

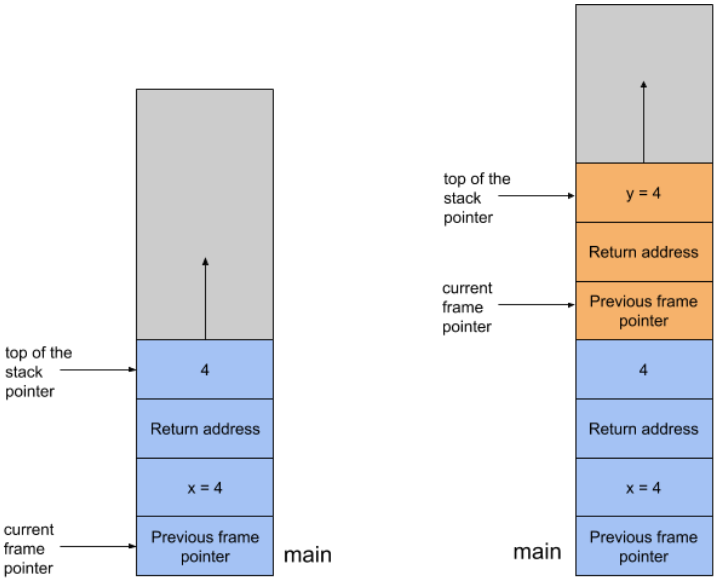
Exercice 3

```
1  int f(int x) {
2    int y = 4;
3    return x + y + 2;
4  }
5  int main(int argc, const char * argv[]) {
6    int x = 4;
7    f(x);
8    return 0;
9  }
```

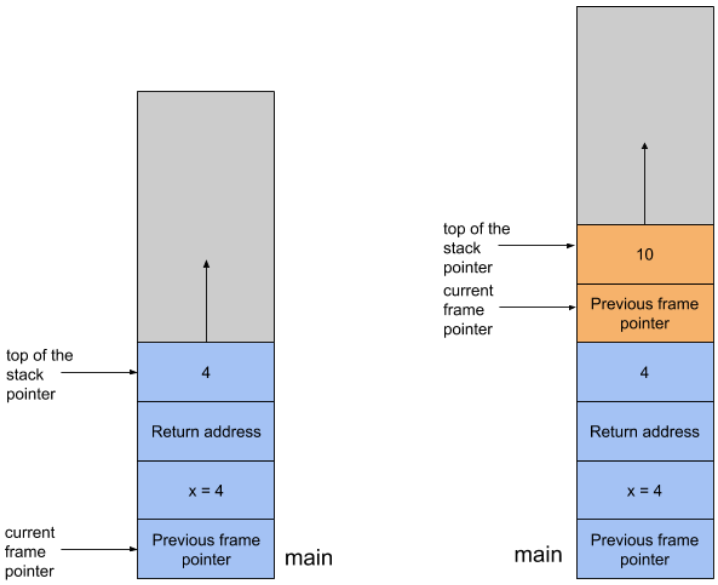
End of line 6



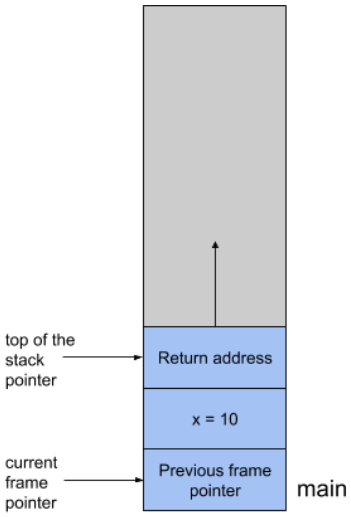
End of line 2



End of line 3



End of line 8



Exercice 4

Location of referenced word	Probability	Total time for access in ns
Cache	0.9	30
Not in cache but in main memory	$0.1 * 0.7 = 0.07$	$30 + 70 = 100$
Not in cache or in main memory	$0.1 * 0.3 = 0.03$	$100 + 22ms = 22'000'100$

The average access time is :

$$avg = 0.9 * 20 + 0.06 * 100 + 0.04 * 22'000'100 = 880'028ns$$

Exercice 5

Yes, if the stack is only used to store the return address, then the program counter can be eliminated. In the case where the stack is also used to store the parameters, then, at a certain point in time, the processor would need both a parameter and the program counter on top of the stack at the same time.