Operating systems

Workshop 1

Exercise 1

Open a documentation for `ps` command and read it cursorily. List tasks that belong to current session, are owned by current user, and all processes. Show which of the listed tasks are kernel threads. Display a tree-like list of all processes (it should give you ancestor-descendant relationship information). As a next step, add threads to the listing. Inspect `init` process and find its parent. Is it possible to terminate this process?

Exercise 2

Find `pid` identifier of any of your processes. List following directory: `/proc/$pid`. Familarise yourself with `proc` virtual file system. Display `maps` file and analyse its contents. Within it find the location of heap, stack, text / data / bss segments and remnants of dynamic linker. Try to find other useful / interesting information in the directory of the process.

Exercise 3

Run arbitrarily chosen process and terminate it using `kill` command. Find an easier way of process termination that doesn't require you to know the `pid` of a process (eg. `xkill`, `pkill`). Read `pgrep` command manual. It serves as a process filter according to some criteria provided by a user. Which signal does `kill` command send by default? A process is allowed to intercept some signals. How to terminate a process that normally ignores / intercepts termination request? Find a signal that is normally used to reconfigure daemons (ie. processes that serve certain functionalities in background) ?

Exercise 4

Trace through list of ancestors of current process. What happens to a process that loses its parent (ie. becomes an orphan)? What is a session identifier and how is it related to controlling terminal? Display all tasks and identify processes which are session leaders. Find processes that cooperate in scope of single session or single process group. Can you tell why such discernment was introduced?

Exercise 5

Run a command and inspect its exit code. What happens if you terminate a process by sending a signal to it? What conditions have to be satisfied in order for a process to become a zombie? Use `ps` command to list all tasks, their state and where do they await (ie. kernel function) resumption.

Exercise 6

Most of the resources in Unix-like OSes have a semantics of a file. Read `lsof` tool manual. Launch an internet browser and show all files it has opened for use. Classify the resources into groups of regular files, directories, devices and network sockets. Analyse properties of a representative of each group.

Exercise 7

Familarise yourself with `strace` and `ltrace` commands. Run a simple program using mentioned tools, in order to get a trace of executed system / library calls. If possible, attach one of the tools to a running process and observe as it proceeds. How to trace an application composed of several processes or threads? Can you look up a facility of the system, that is used to implement the tools (ie. find a system call that enables tracing)?

Exercise 8

Locate two programs - one of them statically another one dynamically linked. Read the structure of the executable files using `readelf` or `objdump` tools. Obviously, the second file requires certain libraries in order to be launched, can you display the list of prerequisites? Use `ldd` command to achieve that. Note that `ldd` provides additional information - for each library it gives you an address to which the library would be relocated. Do the addresses change each time you execute `ldd`, if yes then why?