Lesson 3 – making process

Processes

Each time we need to perform a task, we write some code, and ask OS to run it.

In modern OS, multiple tasks/processes may run in parallel, even on one core CPU via time-sharing

In Linux Process = Thread = Task

Processes - Fork

There is no way to create a new process.
We are only allowed to copy existing one, and modify it.

We can copy a process by *fork* command.

Pid_t pid = fork();

This will make an exact copy of our process, and two of them will run im the system, from the same line of code !!

Processes - Exec

But what if we need another task ?

We have the ability, to change the current process code with another one by *exec* function

*char * args[2] = {"./friend",NULL};*
execvp(args[0],args);

Example !!

Processes - Clone

A more flexible way to make another process or a thread – *clone*

It alows to share some resources with the original process, or create them from scratch.

int id = clone(child,child_stack+STACK_SIZE,0,0);

This is the basic mechanism of creating Threads.

Example !

Processes - tools

Get my process ID: *echo $$*

Redirect output:
foo > stdout.txt 2> stderr.txt
foo > allout.txt 2>&1

Show running process – ps

ps – a : shows all processes
ps – xj : show more info/formated
ps –help all : get all options

Processes - tools

pstree – shows current tree of all processes in the system
pstree(pid) – shows a tree of processes and childs

Top: show CPU usage of each processes

Kill(pid) – kills a processes

Processes - Daemon

Often, we need some background task.
f.ex: file indexing.

There in no need of user interaction with this task.

Such a task called Deamon, and it runs independently of the process starting it.

full example:
https://github.com/pasce/daemon-skeleton-linux-c

Processes - Daemon

A few steps to make proper daemon

1) create it with fork()
2) change it’s dir to root , in order to release the filesystem
3) move it to a new session, so it will not be connected with starting process
4) close IO (stdin,stdout,stderr)
5) optionaly - make some other way to interact with it (socket/signals/logs/etc.)

A bit of syslog

To use system log, in code:

```
#include <syslog.h>
openlog ("myDeamon", LOG_PID, LOG_DAEMON);
syslog (LOG_NOTICE, "daemon started");
syslog (LOG_NOTICE, "daemon finished");
```

At Ubuntu, get the log by this:

grep myDeamon /var/log/syslog

Other Processes types

Zombie process – a proccess that finished, but nobody listening to it’s return code (parent is sleeping/busy)

Orphan – a process that finished, but there is nobody to get it’s return code (parent closed). will be adobpet by init process

https://www.geeksforgeeks.org/zombie-and-orphan-processes-in-c/