# Lab 1. Container by your hands

Total Virtualization

Instructor: Anna Melekhova Assistant: Vadim Reutskiy

Docker is not a wonder. You can create container in Linux in 2 hours.

# Agenda

- *fork()*
- *clone()*
- Linux Namespaces
- *loop* mechanism
- LXC
- Docker
- **Assignment**

But first:
https://goo.gl/mGqhJF

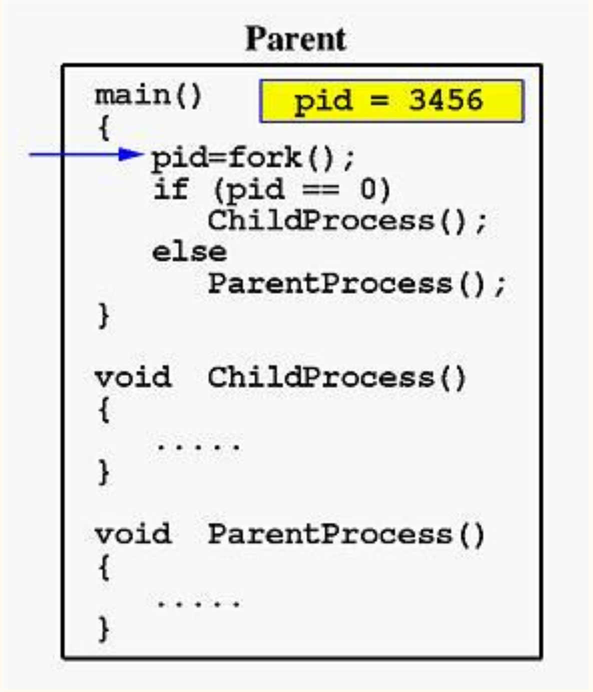
## System call: *fork()*

## fork() - system call to duplicate process

*fork()* creates copy of current process with all address space => new process is created with same values of all variables etc, but further changes will be different (except cases when they synchronized explicitly or accidentally)

For parent process *fork()* returns ID of new process, while for child process it returns 0 (zero)

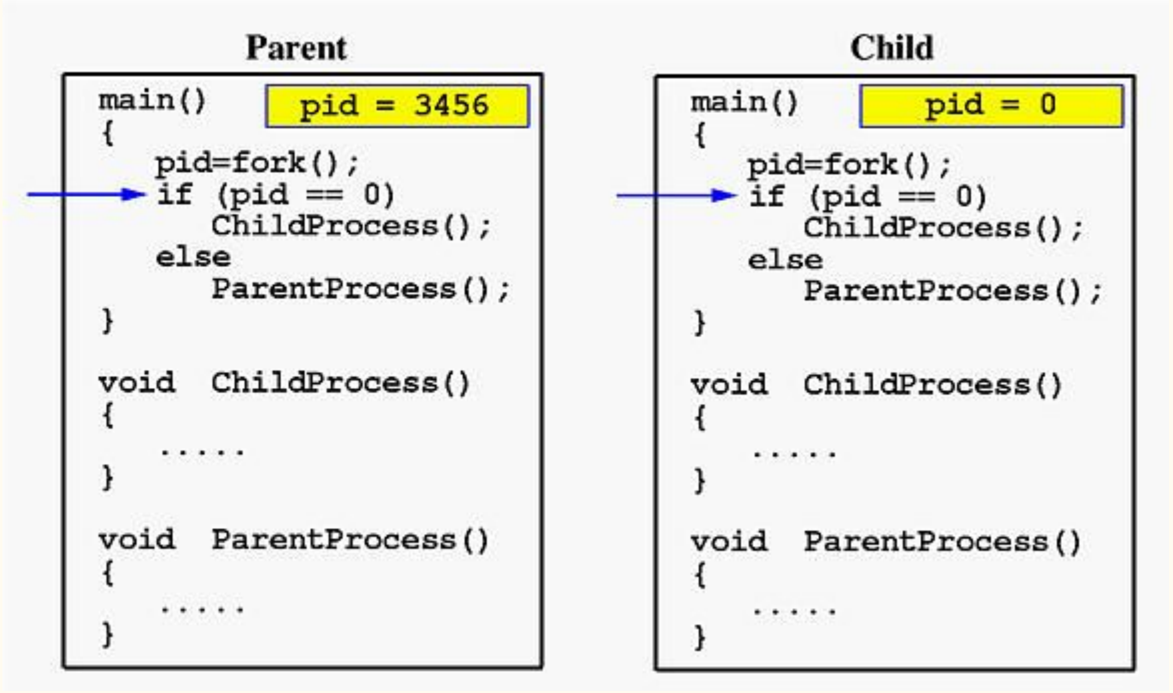
## fork() illustrated



```
main()
{
   pid=fork();
   if (pid == 0)
      ChildProcess();
   else
      ParentProcess();
}

void  ChildProcess()
{
   .....
}

void  ParentProcess()
{
   .....
}
```

Source: http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html

## fork() illustrated

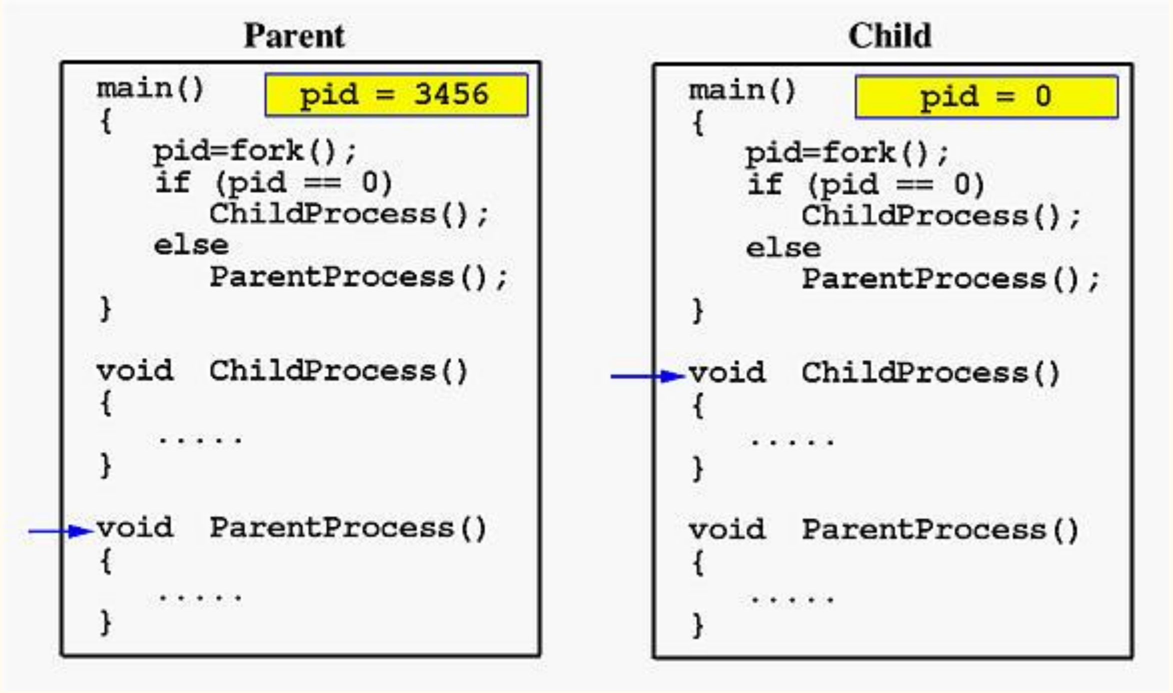


```
main()
{
   pid=fork();
   if (pid == 0)
      ChildProcess();
   else
      ParentProcess();
}

void  ChildProcess()
{
   .....
}

void  ParentProcess()
{
   .....
}
```

```
main()
{
   pid=fork();
   if (pid == 0)
      ChildProcess();
   else
      ParentProcess();
}

void  ChildProcess()
{
   .....
}

void  ParentProcess()
{
   .....
}
```

Source: http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html

## fork() illustrated



```
main()
{
   pid=fork();
   if (pid == 0)
      ChildProcess();
   else
      ParentProcess();
}

void  ChildProcess()
{
   .....
}

void  ParentProcess()
{
   .....
}
```

Source: http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html

## System call: *clone()*

## clone() - improved version of fork()

Using *clone()* child process can share some system resources with parent (depending on parameters passed to *clone()*).

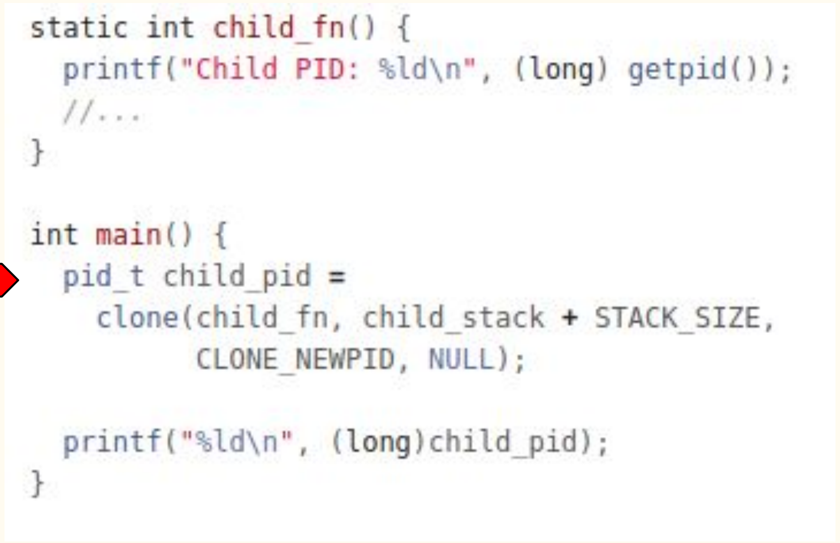
Execution continues from provided address to function (in parameters to *clone()*)

## clone() - what resources can be configured

- Clone parent PID or use current process PID as parent
- Share the same file system information or not (relevant for *chroot()*, *chdir()*, *umask()* commands)
- Share the same file descriptor table or not
- Start child process in new mount / network / PID/ user / etc. namespaces or inherit them from parent
- More details: *man clone*

# clone() - example, step 1

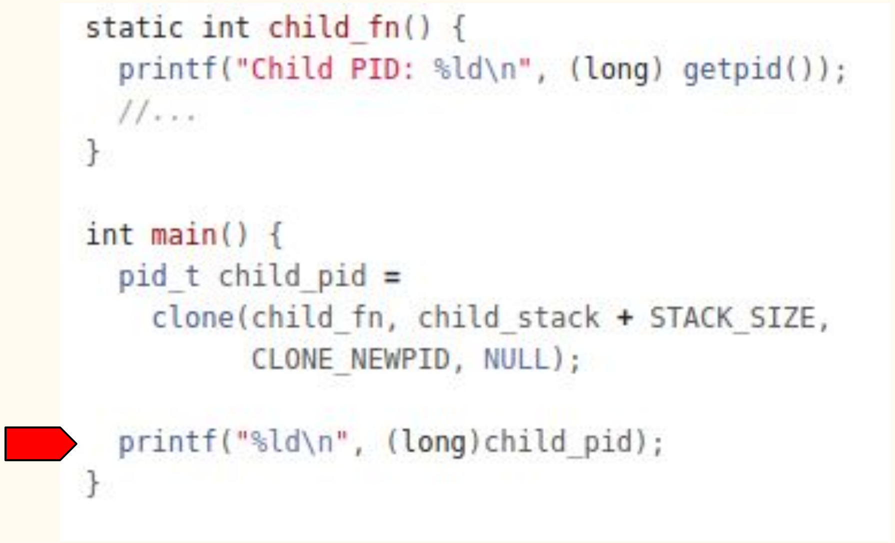
## Parent process



```c
static int child_fn() {
  printf("Child PID: %ld\n", (long) getpid());
  //...
}

int main() {
  pid_t child_pid =
    clone(child_fn, child_stack + STACK_SIZE,
          CLONE_NEWPID, NULL);

  printf("%ld\n", (long)child_pid);
}
```

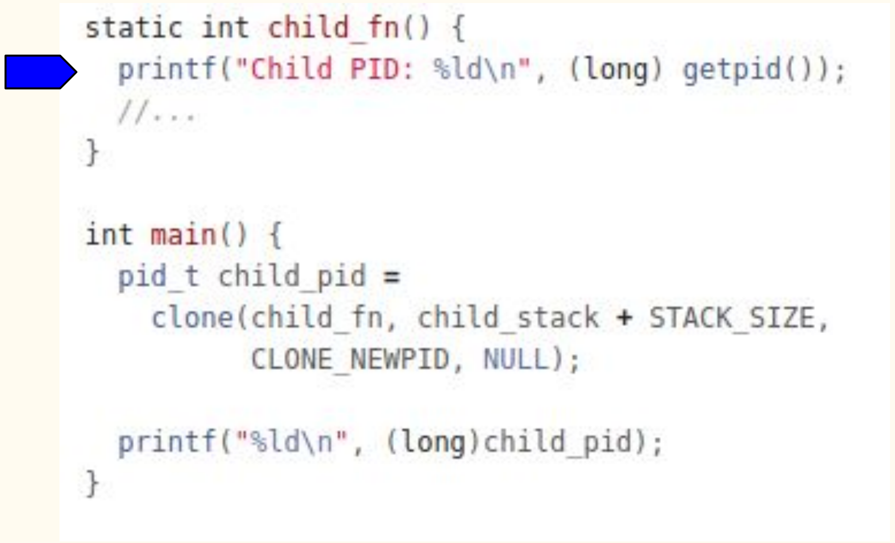
# clone() - example, step 2

## Parent process



```
static int child_fn() {
  printf("Child PID: %ld\n", (long) getpid());
  //...
}

int main() {
  pid_t child_pid =
    clone(child_fn, child_stack + STACK_SIZE,
          CLONE_NEWPID, NULL);

  printf("%ld\n", (long)child_pid);
}
```

## Child process

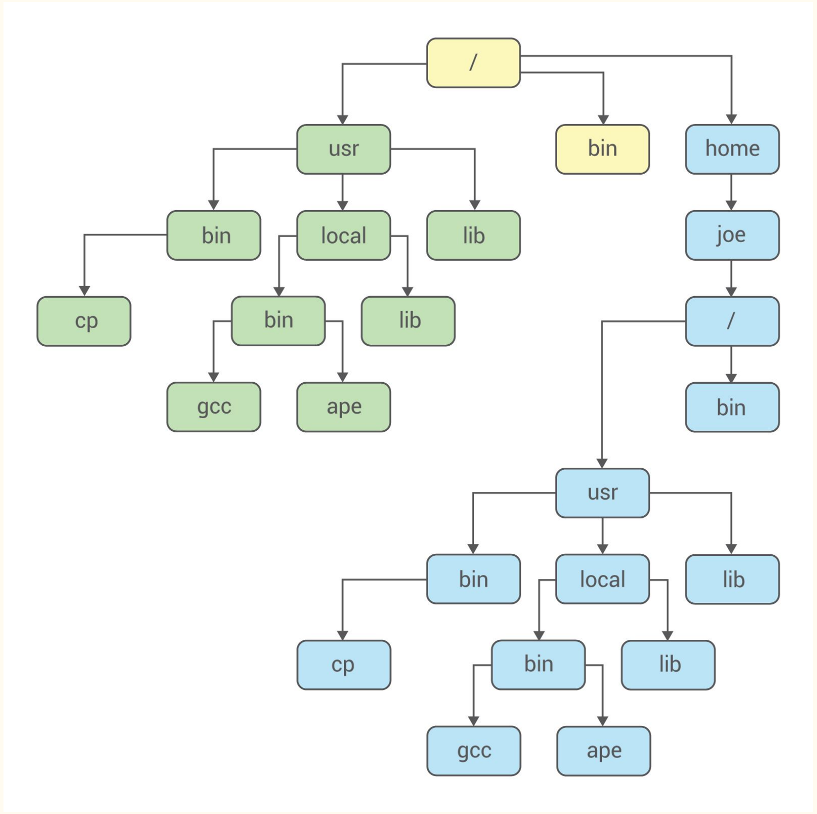


```
static int child_fn() {
  printf("Child PID: %ld\n", (long) getpid());
  //...
}

int main() {
  pid_t child_pid =
    clone(child_fn, child_stack + STACK_SIZE,
          CLONE_NEWPID, NULL);

  printf("%ld\n", (long)child_pid);
}
```

## Linux Namespaces

# Predecessor: chroot

- Mechanism of changing root directory for particular process.
- First attempt of software virtualization for file system (implemented in 1979)
- Not comfortable and not secure (easy to reset new root)



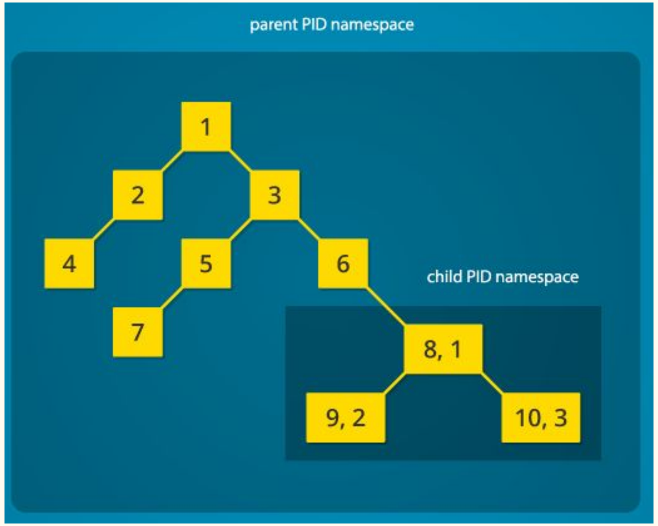
Source: https://blog.selectel.ru/mexanizmy-kontejnerizacii-namespaces/

## Linux Namespaces - kernel isolation mechanism

Linux Namespaces (do not mix up with Java ones) allows to isolate system from any process without need to virtualize all environment

This mechanism is successor of *chroot*

Utilized by: LXC, Docker, HackerRank, Heroku, etc. (many of them)



Source: https://www.toptal.com/linux/separation-anxiety-isolating-your-system-with-linux-namespaces

## Linux Namespaces - types

- Mount
- Process ID
- Network
- Interprocessor Communication
- UTS (hostnames)
- User ID
- Control Group (hardware resource usage)

## Linux Namespaces - types

- **Mount**
- **Process ID**
- **Network**
- Interprocessor Communication
- UTS (hostnames)
- User ID
- Control Group (hardware resource usage)

## *loop* mechanism

## *loop* - tool for virtual storage device operation

In Linux you can associate virtual storage device with any file using loop mechanism (you can even mount ISO image with it)

This mechanism can be used for isolation of file system from guest processes.

Example:

```
losetup /dev/loop0 image.img
mount -t ext4 /dev/loop0 /home/username
```

## LXC

## LXC - LinuX Containers

Origin of container evolution, solution for creation flyweight isolated environments w/o overhead of full virtualization

“Powerful chroot”, predecessor of Docker

AN x64 PROCESSOR IS SCREAMING ALONG AT BILLIONS OF CYCLES PER SECOND TO RUN THE XNU KERNEL, WHICH IS FRANTICALLY WORKING THROUGH ALL THE POSIX-SPECIFIED ABSTRACTION TO CREATE THE DARWIN SYSTEM UNDERLYING OS X, WHICH IN TURN IS STRAINING ITSELF TO RUN FIREFOX AND ITS GECKO RENDERER, WHICH CREATES A FLASH OBJECT WHICH RENDERS DOZENS OF VIDEO FRAMES EVERY SECOND
BECAUSE I WANTED TO SEE A CAT JUMP INTO A BOX AND FALL OVER.
I AM A GOD.

Source: https://xkcd.com/676/

## Docker

## Docker - step up over LXC (initially)

Docker is not only container virtualization tool , but also bunch of different tools and services for developers:
layered containers,
cloud registry of containers,
container build tools,
tools for running multi container (Compose) and/or clustered (Swarm) applications,
etc.

## Assignment

## Assignment

1. Create your own container that is able to run processes via command line using **clone()**.
2. Isolate created container by creating new **namespaces** for PID, mount and networking.
3. Set up a **network connection** between parent namespace and child namespace.
4. Put container filesystem in a file by using **loop** and try to create file in your new mount point.
5. Run processor listing, network information and benchmark in **your container, LXC and Docker** guest containers.
6. **Compare results** and write a brief report about comparison results.