Operating Systems
Jacobs University Bremen
Dr. Jürgen Schönwälder

Module: CO-562
Date: 2020-11-26
Due: 2020-12-03

OS 2020 Problem Sheet #12

Warning: Whenever you prefix a shell command with sudo, make sure you know what you are doing. And never work as root unless you know what you are doing.

Problem 12.1: *union mount file systems* (1+1+1+1+1+1 = 6 points)

Union mount file systems overlay file systems in such a way that content of the layers is merged into a single view. The Linux kernel supports union mount file systems, one of them being the `overlay` file system.

Create an empty directory and change into it. Then execute the following shell commands:

```
mkdir /lower /upper /work /merged
sudo mount -t overlay overlay \
     -olowerdir=/lower,upperdir=/upper,workdir=/work /merged
```

a) Create a file `merged/top`. Where is the file `merged/top` actually stored? Create a file `lower/low`. What happens if you append data to `merged/low`, i.e., where is the data actually stored?

b) What happens if you unlink `merged/low`? (How does the file system remember that `merged/low` got unlinked?)

c) Create a file `lower/lo`. What happens if you change the permissions of `merged/lo`?

d) Describe at least two use cases for overlay file systems.

e) Does the overlay file system always copy data when metadata of a file in the lower layer is changed? Explain.

f) Is it possible to stack multiple lower layers? Why would this be useful? Explain.

Problem 12.2: *namespaces and chroot* (1+1+1+1 = 4 points)

Install a statically linked version of `busybox` on your machine. On Debian/Ubuntu, you can do this via `apt`:

```
sudo apt-get install busybox-static
```

Create an empty directory called `root` and then run the following commands:

```
mkdir root/bin root/dev root/etc root/lib root/proc root/tmp root/usr
cp /bin/busybox root/bin
for t in sh ls pwd ps top mount umount vi; do
    ln -s busybox root/bin/$t;
done
```

a) The `chroot` shell command invokes the `chroot()` system call to change the root of the file system. Execute the following command:

```
sudo chroot root /bin/sh
```

What happens if you type `ps` or `top`? Explain. If there is a problem, how can it be fixed?

Finally, leave the chroot shell to return to your regular shell.

b) Compile and run the following program with root permissions.

```
/*
 * nssh.c --
 *
 *      A new process identifier namespaces can be created using the
 *      clone() system call. (The clone() system call is the Linux
 *      backend of the fork() system call).
 */

#define _GNU_SOURCE

#include <sched.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <unistd.h>

#define STACK_SIZE      1048576
static char stack[STACK_SIZE];

static int child(void *ignored)
{
    execl("/bin/sh", "sh", NULL);
    perror("execl");
    return 1;
}

int main(void)
{
    pid_t pid;

    pid = clone(child, stack+STACK_SIZE, CLONE_NEWPID | SIGCHLD, NULL);
    if (pid < 0) {
        perror("clone");
        return 1;
    }
    (void) waitpid(pid, NULL, 0);
    return 0;
}
```

What is the process identifier of the resulting shell that the shell sees? Explain what you observe. (Find out how to obtain the shell's process identifier from within the shell or modify `nssh.c` to write the process identifier to the terminal before the `execl()` call.)

c) Read the manual page of the `chroot()` system call and modify the program such that the child process changes its file system root to the `root` directory. Run the program and make sure you can run the `ps` and `top` commands. What do `ps` and `top` show? Explain.

d) The `unshare()` system call disassociates parts of the process execution context (namespaces). Extend the program to unshare the networking namespace. Rerun the program. Which network interfaces do you see?