Önálló laboratórium beszámoló

Távközlési és Médiainformatikai Tanszék

Készítette:	**Bánfalvi Gergő**
Neptun-kód:	**F7AGBY**
Ágazat:	**Felhő alapú hálózatok**
E-mail cím:	**banf.ger@gmail.com**
Konzulens(ek):	**Dr. Maliosz Markosz, Dr. Simon Csaba**
E-mail címe(ik):	**maliosz@tmit.bme.hu simon@tmit.bme.hu**

Téma címe: Function as a Service megvalósítási alternatívák hatékonysága – Linux process izoláció

Feladat

A feladat a Linux operációs rendszeren belül a folyamatok izolációjának alapvető lehetőségeinek tesztelése és mérése. Ez egyben a Linux névterek (namespace) beállítását és azok felkonfigurálásának mérését jelenti.

2017/2018. 2. félév

1. A laboratóriumi munka környezetének ismertetése, a munka előzményei és kiindulási állapota

1.1 Bevezető

Function as a Service (FaaS) azt jelenti, hogy a fejlesztő a nélkül tud futtatni egy feladatot megvalósító programkódot, hogy ismernie vagy szerveznie kellene az infrastruktúrát. Ez nagyban megkönnyíti a fejlesztést, mert másra nem kell figyelni. Azonban sok kérdés is felmerül, például a hatékonyság milyensége. Ezért készült a mérés, hogy később meg lehessen vizsgálni milyen tényezők játszanak fontos szerepet az indulási/végrehajtási idők nagyságában.

A legalapvetőbb FaaS alternatíva, ha a végrehajtandó függvényt a Linux névtereinek felhasználásával izoláljuk és futtatjuk. Kernel szintű műveletek miatt rendkívül gyors megoldás, ezért pontos méréseket lehet vele végezni.

1.2 Elméleti összefoglaló

A Linux névtereknek jelenleg 7 fajtája van. Ezek használatával a folyamatok a kernel erőforrásaiból és beállításaiból mást látnak, és máshoz tudnak hozzáférni. Így a helyes felkonfigurálással elérhetjük, hogy egy folyamat ne lássa az eredeti rendszert, és úgy érzékelje, hogy ő az init process[1]. Ezáltal lehet futtatni például egy rendszer image-et elszeparáltan.

Fontos dolog, hogy az init process alapból rendelkezik mindegyik névtérrel rendszerszinten beállítva, és minden létrehozott új folyamattal a névterek is tovább öröklődnek.

1.2.1 Process ID (pid)

Rendszeren belül a folyamatok azonosítóval rendelkeznek, ez a process ID (pID). Ezek a pID-k fa struktúrában helyezkednek el, a gyökér elem 1, ami az init process pID-ja. Minden más folyamat az init-ből, vagy annak leszármazottjából származik.

A pid névtér elkülöníti a folyamatok pID-ját. A szülő (parent) folyamatból[2] nem látszik változás, ha a gyerek (child) másik pid névtérbe kerül, azonban a child process nem látja az ősét, és magát init, azaz 1-es folyamat azonosítóval rendelkezőnek tekinti. Ezt ábrázolja az 1. ábra.

Tömören összefoglalva, egy pid névtérbeli folyamatok csak azokat a további folyamatokat látják, amik ugyanabban, vagy a sajátjukból leszármazott pid névtérben vannak. Ezeket el is tudják érni. A pid névterek lehetnek beágyazottak, azaz az ős pid névtérből létre lehet hozni újat, és az újban is továbbiakat.

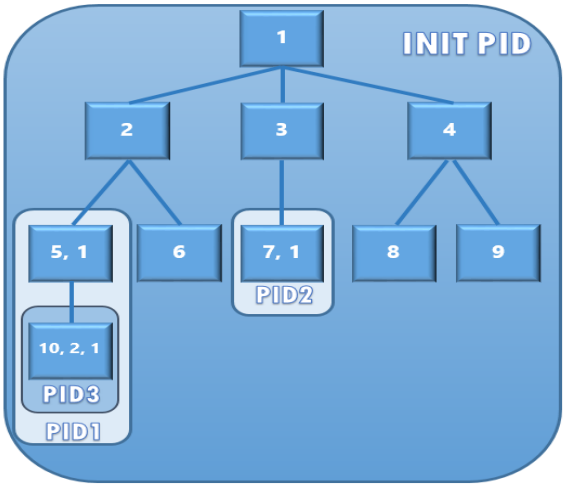
A megoldás így nem teljes, mert egy új pid névtérbe helyezett folyamatból indítva a *ps –aux*[3] parancsot, az init process szemszögéből továbbra is megkapjuk az ős folyamatok azonosítóit. Ennek elkerüléséhez majd a Mount névtér is kell.

Részletesebb leírás a hivatalos Ubuntu oldalon [1]-ben található.

[1] Az első folyamat, ami elindul boot-olásnál és fut, míg le nem áll a gép.

[2] Az a folyamat, amelyik létrehozta az új folyamatot.

[3] Linux rendszereken parancs arra, hogy kilistázza az aktív folyamatokat. A ’-’ utáni rész csak további szűrést végez az adatokon.



1. ábra. Fa struktúra több pid névtér egyszerre létezése esetén, mely megmutatja, hogy melyik folyamat mit lát.

1.2.2 UTS

Az UTS (UNIX Time Sharing) névtér két kernelszintű azonosítónak biztosít szeparációt, a **hostname**-nek és a **domainname**-nek. A domainname manapság már nincs használva, így főleg a hostname miatt fontos ez a névtér. Így a rendszer azonosítására használt névtől különböző névvel rendelkezhet egy másik UTS névtérben levő folyamat.

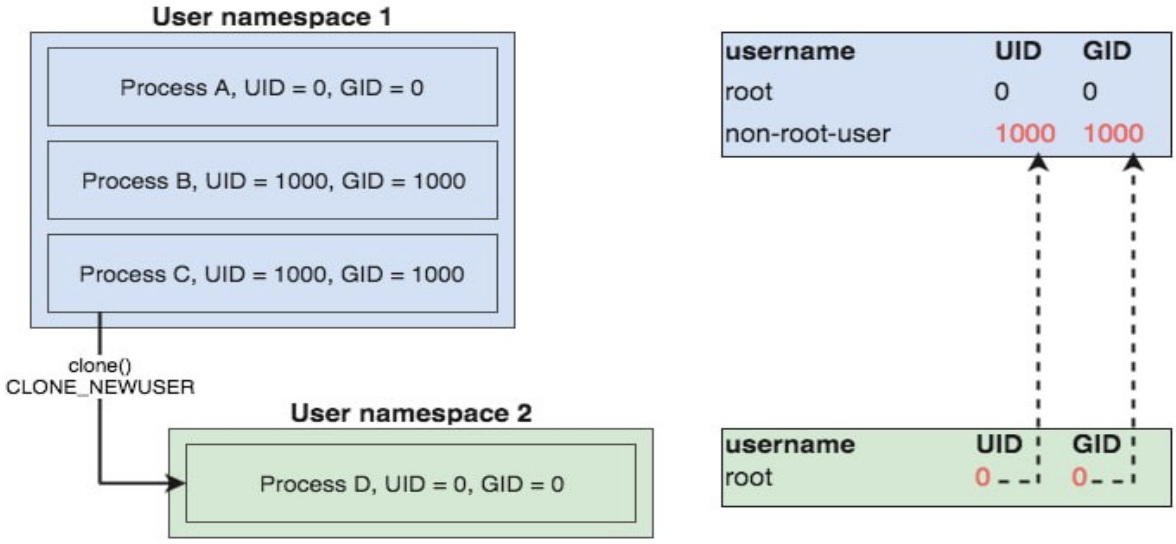
1.2.3 User (user)

A User, azaz felhasználói névtér biztonsági azonosítókat és tulajdonságokat izolál, pontosabban ezeket: Keyrings, Capabilites, Credentials. Most csak a Credentials, azaz folyamatazonosítók (nem összekeverendő a pID-val) résszel foglalkozunk. A többiről bővebben a [2]-ben lehet olvasni.

A folyamatazonosítók közül itt a **userID** és **groupID**, azaz felhasználó - és csoport azonosítóval dolgozunk. Ezek határozzák meg, hogy milyen erőforrásokhoz férhet hozzá az adott felhasználói névtérben lévő process. A létrehozott user névtérben hozzá lehet rendelni ezeket az azonosítókat az ős névtérben levő felhasználókhoz, így ellehet érni, hogy root[4] jogosultságai legyenek az új névtérben egy folyamatnak. Az összerendelés szemléltetése a 2. ábra.

Beállítás a folyamathoz tartozó uid_map és gid_map fájlok tartalmának átírásával lehetséges. A különböző user névterekben lévő folyamatok más értékeket látnak ezekben a fájlokban (beállítástól függően). Amennyiben nem változtatjuk az állományok tartalmát, akkor egy folyamat által meghívott *whoami*[5] parancs, ami új user névtérbe van, azt kapjuk, hogy nobody (senki) vagyunk. Ez később látható a 6. ábrán.

[4] A root felhasználó mindenhez jogosultságot szerez az operációs rendszerben.
[5] Megadja a felhasználónevét annak, aki meghívja a parancsot.



2. ábra. A különböző user namespace-beli folyamatok által látott felhasználó- és csoportazonosítók, illetve az azonosító hozzárendelés működése ([3] : Namespaces in Go – User)

Ezekben a fájlokban a sorok száma 340 lehet, egy sorban pedig 3 adat található:

1) : A létrehozott user namespace-ben lévő userID-k intervallumának kezdete.
2) : A létrehozó user namespace-ben lévő userID-k itervallumának kezdete.
3) : Az intervallum mérete.

Az alábbi formátumban:

ID_inside-ns ID-outside-ns length

Például: **0 getuid()[6] 1**

Ez azt jelenti, hogy létrehozott user névtérben lévő folyamatnál a root-ot (0) hozzárendeljük az ős user namespace-beli uID-hoz, ami létrehozta a child-ot.

1.2.4 Network (net)

A Network, azaz hálózati névtér az internet és hálózat használattal kapcsolatos erőforrásoknak és beállításoknak biztosít izolációt, mint például az interfészek, tűzfal beállítások, stb..

Egy fizikai hálózati interfész csak egy hálózati névtérben lehet, ha ez a névtér megszűnne, akkor átkerül az ős névtérbe. Virtuális interfészek azonban nyugodtan rakhatók bele. Egy ilyen névtérrel létrehozhatunk egy elszigetelt hálózatot a folyamatainknak.

1.2.5 Mount (mnt)

A mount névtér a moint point[7]-ok elkülönítését végzi, azaz hogy mit látnak a folyamatok a fájlhierarchiában. Amit az új névtérben mount-olunk, az nincs hatással az ős névtérre. Alapvetően nem teljes az elszeparálás, ugyanis az új névtérbe minden lemásolódik az ősből, így teljes rálátása van a rendszerre.

[6] Függvény, amely visszaadja a userID-ját annak a folyamatnak, amelyik meghívta.
[7] Egy könyvtár a fájlrendszerben, amit logikai úton csatolunk az operációs rendszer partíciójához.

1.2.6 Interprocess Communication (ipc)

IPC névtér folyamatok közötti kommunikációs (ipc) erőforrásokat különít el, nevezetesen System V IPC objektumokat és POSIX üzenet sorokat.

1.2.7 Control group (cgroup)

Ez a névtér a process-ek fizikai erőforrás elérését korlátozhatja, mint például, RAM, CPU. A cgroup a legújabb névtér jelenleg.

A következő részek a C nyelven íródott program főbb függvényeit, részleteit mutatják be elméleti szinten, melyek elengedhetetlenek a cél eléréséhez és a program megértéséhez.

1.2.8 Clone függvény

A *clone(2)*[8] függvénnyel lehet egy új folyamatot létrehozni, amelynek meg is adhatjuk milyen új névtereket kapjon. A futásához root jogosultság szükséges.

*int clone(int (*fn)(void *), void *child_stack, int flags, void *arg, ... /* pid_t *ptid, void *newtls, pid_t *ctid */);*

A visszatérési érték egy egész szám, amely a létrehozott folyamat pID-ját adja. -1, ha sikertelen a művelet. A pID a létrehozó folyamat PID névterének a szemszögéből adódik.

Fontos paraméterek:

- **int (*fn)(void*):** A child process függvényének pointer-e. A függvény visszatérési értéke **integer** kell, hogy legyen.
- **void *** child**_stack:** A parent process-nek létre kell hozni egy vermet a child process-nek, majd az erre mutató pointer-t átadni a clone() függvénynek. A veremnek a tetejére kell mutatnia, mert szinte minden processzoron, ami Linuxot futtat, a vermek lefelé nőnek.
- **int flags:** CLONE_* flag[9]-ek megadása egy **integer**ben. Ezek mutatják, hogy milyen új namespace-t kell létrehozni.
- **void *arg**: A child process-nek átadandó argumentumok.

További függvények is rendelkezésre állnak a folyamatok és névterek kezelésére:

- unshare() : A hívó process-t helyezi át a flag-ekben megadott új namespace-ekbe.
- setns() : A hívó process-t már létező namespace-kbe lehet áthelyezni.

1.3 A munka állapota, készültségi foka a félév elején

Korábban volt már konténeres technológiákkal dolgom, de ilyen alap kernel szintű feladatban nem volt tapasztalatom. A program C nyelven íródott, abban voltak előzetes ismereteim. Legtöbbet a hivatalos dokumentációk és pár segédlet (ezekről később lesz szó) segített.

[8] A zárójelben lévő szám jelzi, hogy a dokumentációban melyik kategóriában található meg a parancs. A 2-es a rendszerhívást jelenti.

[9] Kétállású állapottárolást biztosít, egy jelzőbitet jelent.

2. Az elvégzett munka és eredmények ismertetése

2.1 Kiindulási alap

A fő cél az, hogy létrehozzunk egy új process-t a megadott új névterekkel együtt, azokat beállítsuk, és ennek az egésznek az időtartamát lemérjük. Először is szükség van egy main függvényre, amely meghívja a *clone(2)*-t, és mellette egy másikra, amelyiket majd az új névterekben létrehozunk. Ez legyen child_fn. A fentebb leírt szükséges argumentumok a child_fn-re mutató pointer, a child_fn által használt memóriaterületre (1MB) mutató pointer, illetve a felhasználó által programindításkor megadott névtér flag-ek. Ezekhez a jelzőkhöz még társul a SIGCHILD, ami azt adja meg, hogy amikor a child_fn futása a végére ér és terminálódik, akkor jelezzen a szülőjének. Ezután a *waitpid(2)* függvénnyel bevárjuk a child_fn-t, hogy ne tudjon addig a szülő folyamat se végezni, így tudunk mérni.

A child_fn létrehozása előtt és a megszűnése után mérünk időt. A pontos időt a *clock_gettime(2)*-mal mérjük le, aminek átadjuk a CLOCK_REALTIME flag-et, így egy rendszerszintű valós idejű óra jelenlegi értékét kapjuk vissza.

Az időmérés formátuma:

start:[sec].[nanosec]
ready:[sec].[nanosec]

A programnak a névtereket 3. ábrában lévő betűjelek szerint kell megadni. A konstansok a *clone(2)* függvény számára szóló flag-ek.

Sign	Namespace	Constant
c	Cgroup	CLONE_NEWCGROUP
i	IPC	CLONE_NEWIPC
n	Network	CLONE_NEWNET
m	Mount	CLONE_NEWNS
p	PID	CLONE_NEWPID
U	User	CLONE_NEWUSER
u	UTS	CLONE_NEWUTS

3. ábra. A táblázat a névterekhez tartozó flag-ek nevét mutatja, illetve a Sign (jel) azt a betűjelet, amit a programhoz kell majd megadni.

A jelen dokumentáció és program négy darab segédleten alapul a hivatalos dokumentációk mellett, ezek [3] [4] [5] [6] –ban megtalálhatók. [3] és [4] a mount névtérhez adott útmutatást, [5] a hálózati beállításokhoz, [6] pedig egy alap általános áttekintést nyújtott.

2.2 Névterek konfigurációja

2.2.1 PID

Az első és legegyszerűbb a pid névtér, itt ugyanis nincs semmi dolgunk, maximum a mount-nál lesz később. Egyedül a hozzá tartozó flag-et kell odaadni a clone-nak és kész is. Ellenőrzésképpen és az elméleti részben leírtakat alátámasztón a 4. ábrán látható a példakimenet.

```
Types of namespaces:p
start: 1525782250.347519195
Child process pID (parent) : 1630
Child process pID (child)  : 1
ready: 1525782250.348356454
```

4. ábra. A képen az látszik, hogy a gyermek folyamatot csak új pid névtérbe helyeztük, a többi marad az eredeti. A szülő folyamat is kiírja, hogy a gyereknek mit lát pID-nak, illetve a gyerek is kiírja, hogy mit tart a sajátjának. A mérési kimenetek is megjelennek.

2.2.2 UTS

A szintén egyszerű, de csekély beállítást már igénylő uts névtér a következő. A speciális nevek átírása a child_fn-ben történik a *sethostname(2)* és *setdomainname(2)*-vel. Tetszőleges nevet megadhatunk. A teszt nem túl látványos, mert a lekérdezés is a child_fn-ben történik, így egy sima lekérdezés-átírás-lekérdezés kombináció. A beállításokat szemléltető kimenet látható az 5. ábrán.

```
Types of namespaces:u
start: 1525782319.907095853
banfalviPC
(none)
new-hostname
new-domainname
ready: 1525782319.907799828
```

5. ábra. A child process-t új uts névtérbe helyezve először lekérjük az eredeti neveket. Jól látszik, hogy a domainname az eredetileg már nincs is semmire se állítva. A következő páros egyértelműen az új neveket mutatja.

2.2.3 User

A user névtér már sokkal bonyolultabb mint az előzőek. Egyrészt fájlokat kell módosítani, másrészt meg kell oldani egy kommunikációs problémát. Ugyanis ezeket a fájlokat, amik a gyerek folyamathoz tartoznak, a szülőből kell módosítani. Azonban ezt a szerkesztést a gyereknek meg kellene várnia, de miután különböző memóriaterületen dolgoznak, így ez nem egyszerű.

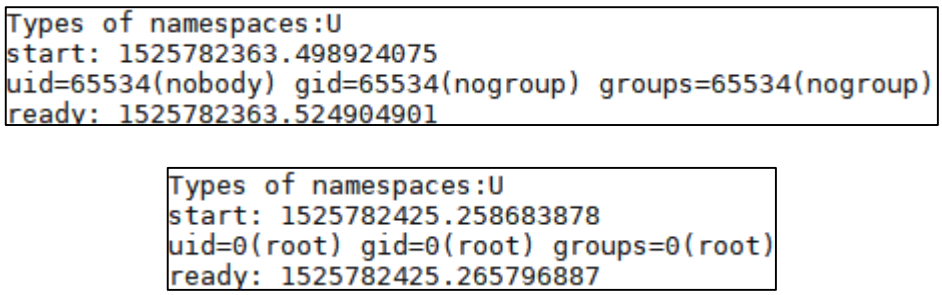
Először is amikor létrejön a gyerek folyamat, a szülő rendelkezik a gyerek pID-jával. Ezután a /proc/[pID]/uid_map és /proc/[pID/gid_map állományokat kell módosítani. A /proc könyvtár tartalmazza a process-eket. Módosítás utáni és módosítás nélküli kimenet a 6. ábrán látható.

Az uid_map –ba a következő sor kerül: 0 [*getuid(2)*] 1 , ahol a *getuid(2)* visszaadja a hívó process user ID-ját, 2. ábrában a UID.

Ugyanez lenne a feladat a gid_map –nál is, csak ott a *getuid(2)* helyett *getgid(2)* –vel. Ellenben a gid_map –ot nem lehet csak úgy szerkeszteni, mert egy biztonsági újítás miatt ez tiltott. Feloldhatjuk a tiltást, ha a /proc/[pID]/setgroups fájlban az eredetileg benne levő ’allow’

szót kicseréljük ’deny’ –ra. Ezután szabad a szerkesztés.

A kommunikációs problémát a *pipe(2)* oldja meg. Ez használható folyamatok közötti üzenetküldésre, és az sem baj hogy csak egyirányú, hiszen csak a szülőtől kell jelezni a gyerek felé.



6. ábra. Az első képen az látható, hogy a fájlok módosítása nélkül miként azonosítja magát a gyermek folyamat. A második képen pedig a beállítások után ugyanez. Jól megfigyelhető, hogy itt már elértük a célt, és root-ként viselkedik. (Az azonosítókat a *whoami(1)* parancs adta)

A user névtér beállításaihoz példakódot és útmutatást a [2]-es dokumentáció nyújtott.

2.2.4 Network

A net névtérnél a cél az, hogy a gyerek folyamatból is elérjük az internetet. Ennek a hálózati beállításaihoz az [5] nyújt segítséget. Miután egy fizikai hálózati interfész csak egy net névtérben lehet, ezért az marad az ősben, és megpróbálunk kapcsolatot létesíteni felé az újban.

Mind a két folyamatban, szülőben és gyerekben is konfigurálni kell. A *system(3)* függvényt fogjuk használni, ami a paraméterként megadott shell parancsokat futtatja.

A megoldás az, hogy a parent process-ben létrehozunk egy virtuális ethernet párt, aminek az egyik végét áthelyezzük a már létrehozott új net névtérbe a child process pID-ját használva (a két vég a veth0 és veth1, amiből a veth1 kerül át az új névtérbe):

```
ip link add name veth0 type veth peer name veth1 netns [pid]
```

Miután ezt a parent process-be csináltuk, ezért először itt konfigurálunk. A veth0 interfésznek IP címet kell adni, engedélyezni kell az ip forwardot (ip továbbítás), utána töröljük az iptables[10]-ből a már meglévő továbbítási szabályokat, és hozzáadjuk a mieinket. (Amennyiben vannak már fontos meglévő szabályok, célszerű nem törölni.)

```
ip addr add 10.200.1.1/24 dev veth0
ip link set veth0 up
echo 1 > /proc/sys/net/ipv4/ip_forward

iptables -P FORWARD DROP
iptables -F FORWARD
iptables -t nat -F
iptables -t nat -A POSTROUTING -s 10.200.1.2/24 -o enp0s7 -j MASQUERADE
iptables -A FORWARD -i enp0s7 -o veth0 -j ACCEPT
iptables -A FORWARD -o enp0s7 -i veth0 -j ACCEPT
```

[10] Olyan Linux program, mely lehetővé teszi a TCP/IP hálózati csomagok továbbításának szabályozását.

A fentebb leírt shell parancsok közül az alsó három kis kifejtésre szorul. Az enp0s7 az a fizikai hálózati interfész azonosítója. Minden gépen más lehet, ezért a fő programban ezt az információt az adott gépről nyerjük ki. A MASQUERADE-t tartalmazó sor azt állítja be, hogy a 10.200.1.2/24-es IP címről (ez lesz a veth1 címe) érkező forgalmat ”rejtse” a fizikai interfész hálózati forgalma alá, azaz cserélje ki a küldő IP címét (veth1) a fizikai interfész címére.

A child process-ből pedig a veth1 interfészt szabjuk testre. Kap IP címet, loopback interfészt, és egy útvonalat, ami azt jelenti, hogy a veth0-n keresztül érheti el az internetet, mint ahogy a 7. ábra mutatja.

```
ip addr add 10.200.1.2/24 dev veth1
ip link set dev veth1 up
ip link set dev lo up
ip route add default via 10.200.1.1
```

```
Types of namespaces:n
start: 1525782473.745996523
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.200.1.2  netmask 255.255.255.0  broadcast 0.0.0.0
        inet6 fe80::fc4e:97ff:fec0:74c9  prefixlen 64  scopeid 0x20<link>
        ether fe:4e:97:c0:74:c9  txqueuelen 1000  (Ethernet)
        RX packets 2  bytes 152 (152.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1  bytes 90 (90.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=56 time=25.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=56 time=25.0 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=56 time=24.2 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=56 time=24.7 ms

--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 24.262/24.865/25.399/0.443 ms
ready: 1525782477.173461458
```

7. ábra. A képen megjelenik a child process hálózati interfészeinek a tulajdonságai, alattuk pedig egy teszt az internet elérésére, ami sikeres.

A user névtérnél használt kommunikációs csatorna itt is hasznos, mert akkor nem csak az ottani beállításokat kell megvárni a gyerek folyamatnak, hanem a hálózatit is. Hozzá jön még az, hogy a child_fn visszatérése után törölni kell az iptables-hez adott szabályokat.

A mérés ismételhetőségéhez, ha új net névtér is használatra kerül, akkor visszatérés után a prorgam vár egy másodpercet, mert különben a tesztek során „RTNETLINK answers: File exists” hiba jött elő, amit legkönnyebben csak számítógép újra indításával lehet megoldani.

2.2.5 Mount

A helyes beállításokhoz a [3] és [4] segédletek nyújtottak segítséget. A cél az, hogy egy alap root könyvtárat biztosítsunk a child process-nek, amiből nem lát ki a valós rendszer felé. Itt jön elő a korábban említett gond a pid névtérnél, ugyanis egy új root könyvtárral már nem lesznek láthatóak a rendszer folyamatai, csak amiket child process láthat. A jogosultságok és a pid névtérre való hatás miatt csak a user és pid névtérrel együtt jöhet létre.

Először is szükség van a dokumentáció alján megadott rootfs könyvtárra. Ez lesz a child process root könyvtára. A következőkben használt függvények:

mount(2) - A megadott elérési úton lévő könyvtárat hozzácsatlakoztatja a cél eléréshez.

mkdir(2) -Új mappát hoz létre a megadott útvonalon, és a megadott jogosultságokkal.

syscall(2) - Olyan funkció meghívására szolgál, amire nincs megfelelő C nyelvű függvény implementálva

umount2(2) - Le tudja a csatlakoztatni a könyvtárhoz adott másik könyvtárat, és még bizonyos flag-eket is hozzá lehet adni.

remove(2) - Törli a megadott útvonalon lévő mappát.

pivot_root – A root könyvtár megváltoztatására szolgál.

procPath: rootfs mappán belüli proc mappa elérési útvonala
path: rootfs mappa elérési útvonala
oldPath: rootfs mappán belüli .pivot_root mappa elérési útvonala, umount-nál már csak "/.pivot_root" lesz

Kezdésnek a rootfs-beli /proc könyvtárat mount-oljuk, így a child_fn szemszögéből megfelelő folyamatok fognak csak látszódni.

```
mount("proc", procPath, "proc", 0, NULL)
```

Ezután a rootfs könyvtárat saját magához mount-oljuk, ami azért szükséges, mert a pivot_root-nak van olyan előfeltétele, amit így lehet elkerülni. (Az új root könyvtár és a régi root könyvtárnak szánt hely nem lehet ugyanazon a fájlrendszeren, mint a mostani root könyvtár.)

```
mount(path, path, "", MS_BIND | MS_REC, NULL)
```

A régi root könyvtárnak létrehozunk a rootfs-en belül egy. pivot_root (rejtett) mappát.

```
mkdir(oldPath, 0777)
```

Megváltoztatjuk a root könyvtárat. Az új a rootfs lesz, a régi a .pivot_root mappába kerül. Ahhoz, hogy a pivot_root funckiót meglehessen hívni, a *syscall(2)*-t kell alkalmazni.

```
syscall(SYS_pivot_root, path, oldPath)
```

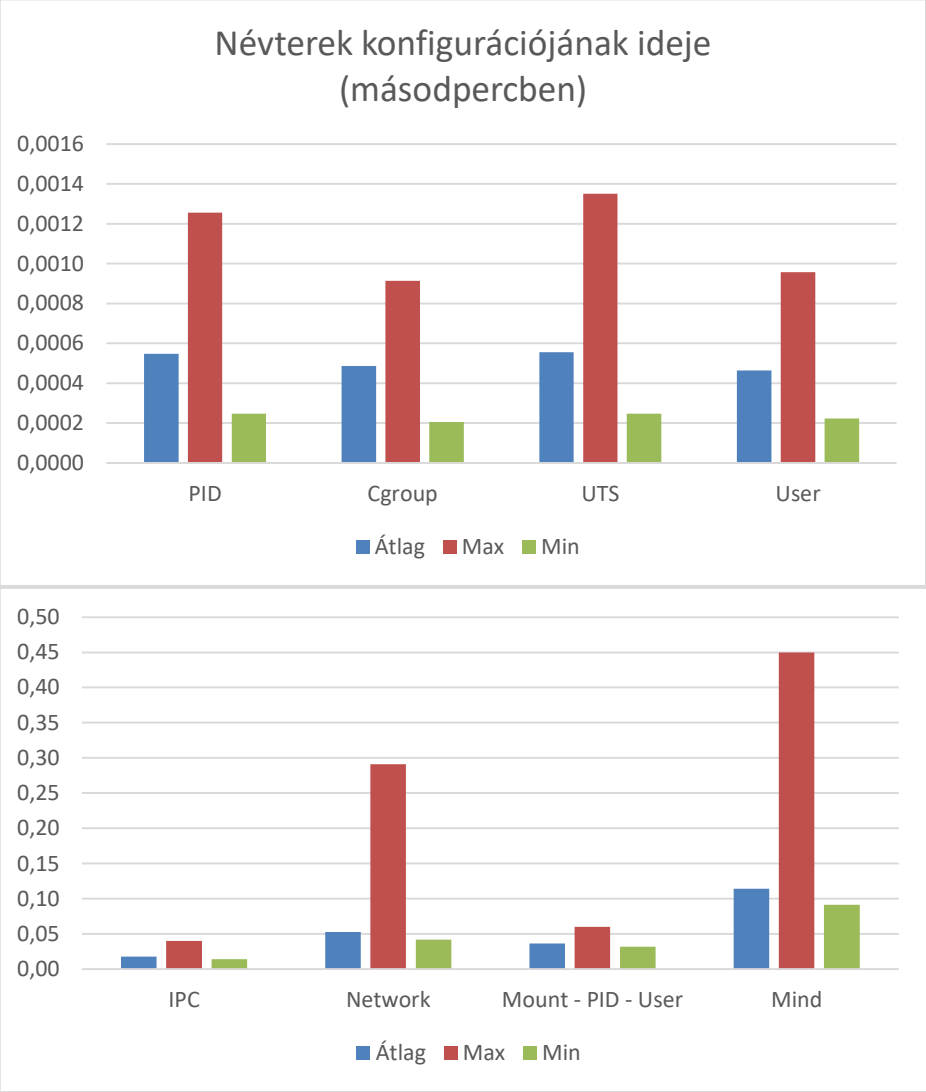
chdir("/") segítségével átváltjuk az aktuális könyvtárat a root-ba.

Végül leválasztjuk a .pivor_root –ot a régi root könyvtárral, majd töröljük a mappát.

```
umount2(oldPath, MNT_DETACH)
remove(oldPath)
```

2.3 Mérési eredmények

A mérések egy Ubuntu 17.10-es rendszerű számítógépen, 4.13.0-39-generic kernel verzió mellett készültek. A nagyságrendi különbségek miatt több grafikon szerepel. Mindegyik variációra a statisztika 1000 tesztesetből készült.
Külön-külön le lett mérve az össze névtér, kivéve a mount, az egyben van a pid-del és a user-rel. A 'Mind' azt jelenti, hogy mind a 7 névtér új és konfigurált (a cgroup és ipc nem konfigurált).



Tisztán látszik, nagyságrendbeli különbségek vannak. Az egyik meglepő eredmény, hogy a User névtér leválasztása is rövid ideig tart, pedig ott fájlokat módosítottunk. Ez mutatja, hogy sok minden történhet a háttérben a többi névtérnél is, amit mi nem észlelhetünk. Továbbá az IPC névtér esetén, csupán új lett létrehozva a gyerek folyamat részére, mégis egészen sok időt, átlagosan kb. 0.017 másodpercet vesz igénybe. A Mount-PID-User hármas átlagosan a dupláját, míg egyénileg a Network az, ami a legtöbb időt igényli. Nem is csoda, hiszen ott rengeteg shell parancshívás történt. A maximumok egészen szélsőséges magasságokba tudnak nyúlni, különösen a Network-nél. A minden névtérből újat használó process pedig átlagosan 0.11 másodperc alatt készül el.

2.4 Összefoglalás

A névterek használatával egy függvényt/funkciót sikerült a rendszertől elszigeteltté tenni. A Cgroup és IPC névterekkel nem foglalkoztam részletesebben, azonban később velük is lehet kísérletezni.

Ez az elszigeteltség pedig azért jó, mert, ahogy a dokumentáció elején meg lett említve, a FaaS arról szól, hogy egy függvényt az infrastruktúra ismerete nélkül lehessen szabadon futtatni. Megtévesztő lehet, hogy a child_fn-ből ismernünk kellett a könyvtárrendszert, de a valóságban a child_fn lehetne csak az előkészítő, ami létrehozza a futtatandó függvény részére az izolált környezetet, és szabályozza esetleg azt. Miután ezek mind kernel szintű műveletek, a mérések is bizonyítják, hogy egy ilyen környezet létrehozása nagyon gyors, azonban az implementáláson sok múlik, és a kiugró maximum értékekből pedig leszűrhetjük, hogy a háttérben valami befolyásolhatja úgy az egész folyamatot, hogy többszörösére nő a végrehajtási idő.

3. Irodalom, és csatlakozó dokumentumok jegyzéke

A tanulmányozott irodalom jegyzéke:

[1] Ubuntu manuals – PID namespace, Megtekintés ideje: 2018.05.04. http://manpages.ubuntu.com/manpages/bionic/man7/pid_namespaces.7.html

[2] Ubuntu manuals – User namespace, Megtekintés ideje: 2018.05.04. http://manpages.ubuntu.com/manpages/bionic/man7/user_namespaces.7.html

[3] Linux Namespaces, https://medium.com/@teddyking/linux-namespaces-850489d3ccf, szerk.: Ed King, 2016. december 10. Megtekintés ideje: 2018.05.04.

[4] Unprivileged containers in Go, Part3: Mount namespace https://lk4d4.darth.io/posts/unpriv3/ Megtekintés ideje: 2018.05.04.

[5] Setup a network namespace with Internet access https://gist.github.com/dpino/6c0dca1742093346461e11aa8f608a99 szerk.: Diego Pino, Megtekintés ideje: 2018.05.04.

[6] Separation Anxiety: A Tutorial for Isolating Your System with Linux Namespaces https://www.toptal.com/linux/separation-anxiety-isolating-your-system-with-linux-namespaces szerk.: Mahmud Ridwan, Megtekintés ideje: 2018.05.04.

Csatlakozó egyéb elkészült dokumentációk / fájlok / stb. jegyzéke:

A teljes kommentezett program és a hozzátartozó egyéb fájlok megtalálhatóak ezen a linken:

https://github.com/banfger/onlab/tree/master/linuxproc

A rootfs állományrendszer eredeti verziója pedig itt érhető el:
wget https://github.com/jpetazzo/docker-busybox/raw/master/rootfs.tar