

Olimpiada de Informatică

Aplicată AcadNet 2026

Calculatoare 11-12

Soluții

§ 1 no-more-french-package

a. `sudo apt-get install wfrench`

`apt-cache depends wfrench`

Pachetul de dependență este debconf.

b. `find / -type f -name french 2>/dev/null`

c. `grep '^f[ae]r.*o$' /usr/share/dict/french`

d. `sudo apt-get purge wfrench`

§ 2 Automatizarea compilării cu Makefile

Un exemplu de fișier Makefile rezultat din urmarea cerințelor este:

```
CFLAGS = -Wall -Wextra

number_stat: main.o linked_list.o
    gcc $(CFLAGS) -o number_stat main.o linked_list.o

main.o: main.c
    gcc $(CFLAGS) -c main.c

linked_list.o: linked_list.c
    gcc $(CFLAGS) -c linked_list.c

debug: CFLAGS += -DDEBUG
debug: clean number_stat
```

```
run: number_stat
    ./number_stat $(ARGS)

clean:
    rm -f number_stat *.o
```

§ 3 Gestiunea concurenței prin Lockfiles

Se admite orice script bash ce indeplinește condițiile cerințelor (lockfile, argument, creare lanevoie, buclă infinită)

```
#!/bin/bash
if [ -z "$1" ]; then
    echo "utilizare: $0 <nume_fisier>"
    exit 1
fi

FILE_NAME=$1
LOCK_DIR=".${FILE_NAME}.lock"

cleanup() {
    rmdir "$LOCK_DIR" 2>/dev/null
    exit
}

# destul de important
trap cleanup SIGINT SIGTERM

# creare fisier daca nu exista
if [ ! -f "$FILE_NAME" ]; then
    touch "$FILE_NAME"
fi

if mkdir "$LOCK_DIR" 2>/dev/null; then
    # am dat lock cu succes
    while true; do
        sleep 1
    done
else
    # nu am dat lock
    exit 1
fi
```

§ 4 Arhitectul de pachete

Exemplu de astfel de script:

```
#!/bin/bash

START_DIR=$(pwd)
DEB_FILE=$(realpath "$1")
TMP=$(mktemp -d)

cd "$TMP" || exit 1

ar x "$DEB_FILE"
mkdir pkg_root control_ext

tar -xf data.tar.* -C pkg_root
tar -xf control.tar.* -C control_ext

PKGNAME=$(grep -Po '(?<=^Package: ).*' control_ext/control)
PKGVER=$(grep -Po '(?<=^Version: ).*' control_ext/control)
PKGDESC=$(grep -Po '(?<=^Description: ).*' control_ext/control)
DEBARCH=$(grep -Po '(?<=^Architecture: ).*' control_ext/control)

case "$DEBARCH" in
    amd64) ARCH="x86_64" ;;
    i386) ARCH="x86" ;;
    arm64) ARCH="aarch64" ;;
    *) ARCH="$DEBARCH" ;;
esac

cat <<EOF > pkg_root/.PKGINFO
pkgname = $PKGNAME
pkgver = $PKGVER
pkgdesc = $PKGDESC
arch = $ARCH
EOF

for s in preinst postinst prerm postrm; do
    if [ -f "control_ext/$s" ]; then
        case $s in
            preinst) f="pre_install" ;;
            postinst) f="post_install" ;;
            prerm) f="pre_remove" ;;
            postrm) f="post_remove" ;;
        esac

        echo "$f() {" >> pkg_root/.INSTALL
        tail -n +2 "control_ext/$s" >> pkg_root/.INSTALL
        echo "}" >> pkg_root/.INSTALL
    fi
done

cd pkg_root
tar --zstd -cf "$START_DIR/${PKGNAME}-${PKGVER}-${ARCH}.pkg.tar.zst" .
```

```
rm -rf "$TMP"
```

Funcțiile sale sunt destul de auto-evidente. Este esențială dezarhivarea și parsarea corectă a pachetului de debian, păstrarea coerenței arborelui de fișiere, și construcția corectă a pachetului de Arch Linux prin popularea corectă a fișierelor .PKGINFO și .INSTALL.

§ 5 Integritate și Socket-uri

a.

```
gpg --batch --generate-key <<EOF
Key-Type: RSA
Key-Length: 2048
Name-Real: student
Name-Email: student@upb.ro
%no-protection
%commit
EOF
```

```
lscpu > cpu_info.txt
```

b.

```
gpg --output cpu_info.sig --detach-sig cpu_info.txt
```

Acum în primul terminal:

```
nc -lU /tmp/transfer.sock > primite.data
```

Și în al doilea:

```
cat arhiva.tar | nc -U /tmp/transfer.sock
```

c.

```
gpg --verify cpu_info.sig cpu_info.txt > /dev/null 2>&1

if [ $? -eq 0 ]; then
    echo "Validare Reusita"
else
    echo "Eroare Integritate"
fi
```

§ 6 Compatibilitate

a.

```
$ git clone git://git.savannah.gnu.org/automake.git
```

b.

```
$ git checkout v1.16.4
```

c.

```
$ patch -p1 < ~/NTRU/compatibilitate_automake.patch
```

d.

```
$ ./bootstrap && ./configure --prefix=~/NTRU/am/ && make && make install
```

e.

```
$ ~/NTRU/am/bin/automake --version
```

Va prezenta versiunea 1.16.4-Acadnet

§ 7 Acadcore

a. `cp tc.iso new.iso`

```
fdisk new.iso
```

Apoi

```
e,+80M; a; n
```

b. `doas kpartx -a new.iso; mkfs.ext4 /dev/mapper/loop0p2`

c.

```
mkdir root
mount /dev/mapper/loop0p2 root
touch hydra.txt
sudo kpartx -d new.iso
```

În loc de kpartx se poate folosi și folosirea losetup.

d. `qemu-system-x86_64 -hda new.iso -nographic`

Oricum, dacă merge totul bine ar trebui să arate un mesaj zicând că pot lua punctele.

§ 8 Turnul Babel

a. `docker run -v /:/mnt -it ubuntu:22.04 /usr/bin/bash`

În acest moment, vom avea acces la un shell privilegiat în contextul containerului care are acces liber la sistemul de fișiere al gazdei în directorul /mnt.

b. `(în docker) chroot /mnt`

Am obținut un root shell pe gazdă.

c. `dpkg-reconfigure locales`

Se admit și soluțiile în care fișierele de configurare a locațiilor sunt direct modificate folosind un editor text existent pe container, fără schimbarea rădăcinii sistemului de fișiere virtual (chroot).

d. **Explicație tehnică**

Documentația oficială docker previne faptul că apartenența unui utilizator la grupul docker este echivalent cu privilegierea sa, dându-i acces de comunicare și control la daemonul docker, ce rulează cu privilegii maxime. Folosindu-ne de acesta, îl vom convinge prin montarea sistemului gazdă de fișiere (funcție folosită pentru sincronizarea directoarelor bazelor de date, fișierelor statice share, etc) să ne dea acces direct, privilegiat, la acesta. Fiind în genere foarte bine izolat, iar această funcție de montare directă

fiind adesea umbrită de docker images, multe sisteme suferă încă de această escaladare de privilegii, menționată clar în documentație, din cauza neatenției administratorului.

§ 9 Configurare rețea și servicii

```
(a)
node1:
go node1
ip link set eth0 up
ip addr add 12.45.21.50/24 dev eth0

node2:
go node2
ip link set eth0 up
ip addr add 12.45.22.50/24 dev eth0

(b)
node1:
go node1
ip route add default via 12.45.21.254

node2:
go node2
ip route add default via 12.45.22.254

(c)
node1:
go node1
echo "12.45.22.50      node2" >> /etc/hosts
echo "12.45.21.254    router-node1-intf" >> /etc/hosts
echo "12.45.22.254    router-node2-intf" >> /etc/hosts

node2:
go node2
echo "12.45.21.50      node1" >> /etc/hosts
echo "12.45.22.254    router-node2-intf" >> /etc/hosts
echo "12.45.21.254    router-node1-intf" >> /etc/hosts

(d)
node1:
go node1
su - flavia
ssh-keygen
ssh-copy-id mihaela@node2

node2:
go node2
su - mihaela
ssh-keygen
ssh-copy-id flavia@node1
```

§ 10 cgroupv2

a. `mkdir /sys/fs/cgroup/limiter && echo "echo $(pgrep CollatzProver)" >> /sys/fs/group/limiter/cgroup.procs`

b. `lsblk | grep 'sda ' | awk '{print $2}'`

Sau orice alta metoda corecta

c. `echo "8:0 wbps=2048" > /sys/fs/cgroup/limiter/io.max`

`echo "110000 100000" > /sys/fs/cgroup/limiter/cpu.max`

d. In script trebuie verificat ca se citeste maj:min automat, caci acesta se poate schimba intre rebooturi.

```
#!/bin/bash
/usr/bin/collatzProver &
mkdir -p /sys/fs/cgroup/limiter
majmin=$(lsblk | grep 'sda ' | awk '{print $2}')
echo "$majmin wbps=2048" > /sys/fs/cgroup/limiter/io.max
echo "110000 100000" > /sys/fs/cgroup/limiter/cpu.max
echo "$(pgrep collatzProver)" > /sys/fs/group/limiter/cgroup.procs
```

§ 11 Fantoma în Namespace

Folosind formula

$$\text{Container UID} = (\text{Host SubUID Start} + \text{Internal UID}) - 1$$

am identificat maparea corectă a utilizatorului din namespace-ul containerului pe sistemul gazdă.

Pentru a remedia accesul fără a schimba masca de permisiuni (755), se execută una dintre următoarele comenzi:

```
# Varianta 1: Calcul manual bazat pe /etc/subuid
chown -R 100032:100032 ./data

# Varianta 2: Folosind utilitarul podman pentru mapare automată
podman unshare chown -R 33:33 ./data
```

a. **Explicația tehnică:** Motivul pentru care eroarea apare în configurațiile **rootless** este legat de izolarea prin **User Namespaces**:

- **Maparea Identităților:** În mod implicit, UID-ul utilizatorului **student** de pe gazdă este mapat ca **root** (UID 0) în interiorul containerului. Restul utilizatorilor din container (precum utilizatorul **www-data** sau **apache**) sunt mapați folosind un offset definit în `/etc/subuid`.

- **Conflictul de Acces:** Deoarece webserver-ul din container încearcă să ruleze procesele cu UID 33, acesta este văzut de sistemul de fișiere de pe gazdă ca un utilizator complet diferit față de student.
- **Constrângerea:** Prin schimbarea owner-ului fișierelor către UID-ul mapat corespunzător de pe gazdă, procesul din container capătă drepturi de proprietar, permițând funcționarea serverului în timp ce menținem permisiunile stricte de 755.