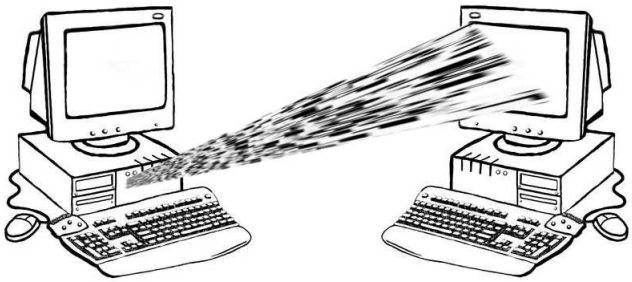
RADBOUD UNIVERSITEIT

Hardware Onafhankelijk Besturingssysteem

Studentnummer: 0712744



Auteur:
Mehdi
Aqadjani Memar

Supervisor:
Prof. Dr. Frits Willem
Vaandrager

29 juni 2012

Inhoudsopgave

1 Probleemstelling

Door de globalisering van de economie moeten we vaak voor ons werk reizen. Wie een zakelijke reis maakt heeft (bijna) altijd een computer nodig. Laptops zijn ontworpen om aan de behoefte van de mobiele computergebruiker tegemoet te komen. Ze zijn echter niet altijd gebruiksvriendelijk. Om e-mails te versturen en te ontvangen of een pdf-bestand te lezen zijn ze goed genoeg, maar voor langdurig werk en/of het uitvoeren van een applicatie waarvoor veel rekenkracht nodig is, zijn ze niet geschikt. Een programmeur bijvoorbeeld, kan niet werken zonder een los toetsenbord en een groot scherm. Een grafisch ontwerper heeft naast een groot scherm ook accessoires nodig waarmee hij kan tekenen en een video-editor kan niet zonder een computer met veel rekenkracht. Een goede oplossing voor elk van deze personen zou zijn dat ze, waar ook ter wereld, over hun eigen computer kunnen beschikken. Laten we het belang van deze mogelijkheid illustreren aan de hand van het volgende scenario.

Scenario

> Jelle is een webapplicatie-programmeur en werkt bij een middelgroot ICT-bedrijf. Hij schrijft applicaties in verschillende talen, zoals PHP, Prel, JavaScript, Mysql, Solr, HTML en CSS. Voor een website bewerkt hij regelmatig foto's en maakt hij af en toe animaties. Op zijn Linux-PC heeft hij compilers en applicaties geïnstalleerd die hij gebruikt voor zijn werk. Op de computer is een scanner aangesloten waarmee hij foto's scant. Codes, bewerkte foto's en gemaakte animaties worden lokaal op zijn computer opgeslagen. Iedere dag voordat hij naar huis gaat stuurt hij zijn werk van die dag naar een "Git-server".
>
> Op een dag vertelt zijn leidinggevende dat er drie ICT-beurzen in de VS gehouden zullen worden en dat het bedrijf van plan is om iemand naar de beurzen te sturen. De beurzen gaan over de software die het bedrijf gebruikt op verschillende afdelingen. De leidinggevende wil weten wat de nieuwe functies van de software precies inhouden alvorens er wordt besloten om het aan te schaffen. De beurzen worden in drie verschillende steden gehouden verspreid over twee maanden. De leidinggevende vraagt of Jelle naar de beurzen wil gaan. Er wordt een appartement voor hem geregeld waar hij in die periode kan verblijven. Naast zijn salaris krijg hij nog een extra bonus en alle onkosten worden vergoed. Voorwaarde is wel dat hij in die peri-

ode minimaal 3 dagen per week aan projecten besteedt waar hij op dit moment mee bezig is. Daarvoor wordt een computer beschikbaar gesteld in het appartement met alle randapparatuur die Jelle nodig heeft. Hij moet alleen de software die hij gebruikt zelf mee nemen.
Jelle vindt het een ideale kans en accepteert het aanbod. De dag voor zijn vertrek sluit hij zijn USB-stick, waar hij eerder een speciaal programma op heeft geïnstalleerd, aan op zijn computer en start de computer op. De computer start vanuit de USB en de vraag of er een kloon van de computer gemaakt moet worden verschijnt op het scherm. Jelle beantwoord de vraag met "ja" en na ongeveer vijf minuten verschijnt het bericht dat de kloon gemaakt is en dat de USB los gekoppeld mag worden. Jelle haalt de USB-stick uit de computer en drukt op "Enter". De computer start normaal op. Jelle zet de computer uit en gaat naar huis om de volgende dag naar de VS te vliegen.

Bij de aankomst wil Jelle zijn computer geschikt maken voor zijn werk. Hij stopt de USB-stick, die hij voor zijn vertrek had gebruikt, in de computer die in het appartement voor hem klaar staat en zet de computer aan. De computer start vanuit de USB op en vraagt of de beschikbare kloon op de computer gezet mag worden. Jelle beantwoordt de vraag met "ja" en na ongeveer zeven minuten verschijnt er een bericht op het scherm dat de kloon met succes op de computer is gezet en dat de USB-stick losgekoppeld moet worden. Hij volgt de instructies, start de computer opnieuw op en ziet dat de computer opstart zoals de computer op zijn werk. Hij logt in en vindt alle applicaties en data van de computer op kantoor. Hij ziet dat de aangesloten printer al geïnstalleerd is maar de scanner niet. Hij zoekt op het Internet naar de scanner en ontdekt dat er geen Linux drivers voor de scanner bestaan.

Het is belangrijk om op te merken dat als we het hier over een computer hebben, we het fysieke instrument bedoelen dat uit verschillende onderdelen bestaat. Onder de software verstaan we hier de applicaties en de bijbehorende data waar de gebruiker mee werkt, inclusief het besturingssysteem.

Terug naar de oplossing voor de mobiele computer gebruiker: door hardware en software van elkaar onafhankelijk te maken kunnen we ervoor zorgen dat de gebruiker zijn software op iedere computer die aan een minimum aantal eisen voldoet, waar dan ook ter wereld, kan plaatsen en kan gebruiken.

De computer kan worden gezien als een kantoor in een bedrijf of op een universiteit en de software staat voor het interieur van het kantoor. Medewerker

A richt het kantoor in naar zijn eigen smaak: een bureau bij het raam, een vergadertafel met zes stoelen in het midden en een familiefoto aan de muur. Als medewerker A naar een ander kantoor verhuist en medewerker B naar het betreffende kantoor betrekt, verandert hij de inrichting: een bureau tegen de muur, vier stoelen met een lage tafel in de hoek en een schilderij aan de muur.

Bij het verplaatsen van een besturingssysteem van de ene naar de andere computer is de kans groot dat de onderliggende hardware enigszins van elkaar verschilt. Als we ervoor zorgen dat het geconfigureerde besturingssysteem aangepast wordt aan de nieuwe hardware, draait het besturingssysteem direct op de *Instruction Set Architecture*. Deze methode kan gezien worden als een vorm van *"disk cloning"*. Een andere methode om het besturingssysteem van de ene op de andere computer over te zetten zonder hardware onverenigbaarheidsproblemen, is het creëren van een nieuwe laag tussen *ISA* en het besturingssysteem. Deze laag zorgt ervoor dat het besturingssysteem met de hardware communiceert. Waar de eerste methode een raakvlak heeft met "disk cloning" kunnen we hier een vergelijking trekken met hardware-virtualisatie.

In dit onderzoek gaan we na hoe een gekloonde harde schrijf op een andere computer met andere hardware gekopieerd kan worden zodat het besturingssysteem zonder problemen functioneert. Het idee is dat de gebruiker de totale software van zijn computer op een andere computer kan plaatsen. De verplaatsing van software kan op een aantal problemen stuiten. Men kan denken aan verschillende opslagruimtes op twee harde schijven of verschillende *motherboards*, CPU's, videokaart en etc. Het is belangrijk dat de verplaatsing geen technische kennis van de gebruiker vereist. De gebruiker moet niet belast worden met het installeren van drivers of oplossen van de hardware-onverenigbaarheid. De centrale vraag in dit onderzoek luidt als volgt: "hoe kan de kloon van een harde schijf met een Linux-besturingssysteem op iedere computer draaien?"

2 Verantwoording

Software die ervoor zorgt dat een besturingssysteem, applicaties en data van een bepaalde computer op iedere computer kunnen draaien, geeft de mobiele computergebruiker vrijheid. De gebruiker kan zoals we in het voorbeeld van Jelle hebben gezien, zijn software op een USB-stick zetten, deze makkelijk verplaatsen en overzetten naar een andere computer.
Het is ook mogelijk om in plaats van een *USB-stick* een opslagmedium op een netwerk te gebruiken, bijvoorbeeld een data-server aangesloten op het Internet. De gebruiker kan zo makkelijk zijn data via de Internet-verbinding naar de data server sturen en ze van hieruit op een andere computer overzetten.

Het is ook mogelijk om een opslagmedium op een netwerk te gebruiken, zoals een "data server" aangesloten op het Internet. In dit geval vervangt de server de *USB-stick*. De software kopiëert dan de data van de *source computer* op de server en schrijft vervolgens de data van de server op de *target computer*. Aangezien het gebruik van Linux de laatste jaren toegenomen is kan deze software zeer nuttig zijn. Met name voor studenten, die een grote groep vormen binnen de Linux-gebruikers, is dit interessant.

De oplossingen voor de problemen van de mobiele gebruiker kunnen we in drie categorieën indelen: draagbare hardware, connectie door een netwerkverbinding en het gebruik van virtualisatie.

Laptops, notebooks en mobiele telefoons die steeds meer op computers lijken vallen onder de eerste categorie. We hebben al uitgelegd dat deze computers echter niet voldoen aan alle behoeftes van een gebruiker en dus geen echte oplossing bieden.

SSH, remote desktop, VPN etc. vallen in de tweede categorie. Deze oplossingen zijn bruikbaar, maar afhankelijk van een aantal voorwaarden en ze hebben bovendien beperkingen. Voorwaarden zijn het beschikken over een netwerkverbinding en geschikte programma's op communicerende computers. Een beperking van deze methodes is dat de randapparatuur van de computer door de fysieke afstand niet (direct) gebruikt kunnen worden. Bijvoorbeeld, een uitgeprint document kan niet gebruikt worden.

De derde categorie, virtualisatie, biedt wel uitkomst voor de mobiele gebruiker. Hardware virtualisatie kan, namelijk de onafhankelijkheid tussen hardware en software realiseren. Deze onafhankelijkheid is geen doel op zich maar een middel om het uiteindelijke doel te bereiken, namelijk het draaien van meerdere virtuele machines op één hardware platform. In deze techniek komt er een laag, genoemd *"hypervisor"*, tussen de *"Instruction Set Architecture"* en

het besturingssysteem. De *hypervisor* zorgt voor een standaard procedure voor ieder besturingssysteem en hierdoor kunnen besturingssystemen gemakkelijk naar een andere computer, die dezelfde laag heeft, verplaatst worden en probleemloos functioneren. Als er sprake is van één besturingssysteem is het nadeel van deze methode voor de hardware-onafhankelijkheid de overhead van de *hypervisor*. Ook al hebben er de laatste jaren belangrijke ontwikkelingen plaatsgevonden en zijn er slimme manieren uitgevonden om de efficiëntie van *hypervisors* te verbeteren deze methode is nog altijd minder efficiënt dan een besturingssysteem dat direct op de hardware draait.

De x86 heeft historisch geen hardware-ondersteuning voor virtualisatie. Om die reden heeft VMWare twee soorten virtualisatie ontwikkeld. Eén soort is virtualisatie voor x86, MVWare-software genoemd, en het andere soort is voor hardware-ondersteunde CPU's (VT en AMD-V), VMWare-hardware genoemd. In [Adams Agessen, 2006] is een test gedaan om de *performance* van zowel VMWare-software als VMWare-hardware met een native systeem te vergelijken. In deze test worden er twee webservers gebruikt die op een virtuele laag draaien. De ene webserver is een *Linux based* webserver en de andere een *Windows based* webserver. Deze test wordt zowel op VMWare-software als op VMWare-hardware uitgevoerd. De verhouding van de *performance* van de virtuele webservers op de *performance* van de *native* webserver is als volgt:

	VMWare-software	VMWare-hardware
Windows	67%	53%
Linux	45%	38%

Tabel 1: Performance van WMWare-virtualisatie

De ideale oplossing is dat de gebruiker zijn besturingssysteem, applicaties en data direct op ieder andere computer kan installeren en er zonder probleem mee kan werken. Er zijn verschillende commerciële producten die de gebruiker in staat stellen om systeeminstellingen van een computer(*source*) op een andere computer(*target*) te zetten . Acronis maakt een hardware onafhankelijk image van een computer door drivers van de nieuwe hardware te installeren op het moment dat het image op de nieuwe computer uitgepakt wordt. De software vraagt de gebruiker om de locatie van de hardware drivers in te voeren.

In dit onderzoek gebruiken we hetzelfde principe, met het verschil dat de software de drivers zelf ophaalt en installeert zonder de gebruiker te belasten. Acronis vraag de gebruiker om input. Deze input kan variëren van drivers voor

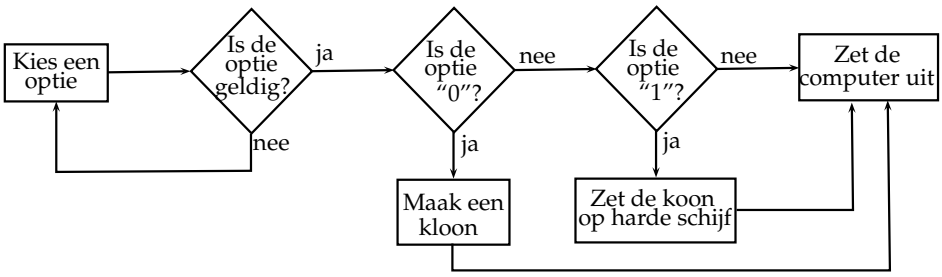
de hardware tot de partitionering van de harde schijf. In ons model moet een gebruiker met weinig of helemaal geen technische kennis in staat zijn om de software te gebruiken.

3 Theoretische kader

In de onderzoeksvraag hebben we het over de kloon van de harde schijf. De "kloon van de harde schijf" verwijst naar een methode waarmee een identieke versie van de harde schijf gemaakt wordt. Met "identiek zijn van de harde schijf" bedoelen we dat de inhoud van twee harde schijven aan elkaar gelijk is, uitgezonderd hardware drivers. Dus als de opslagruimte op de *target* kleiner of groter is dan die op de *source* moet ons model een oplossing bieden om alsnog de kloon van de *source* op de *target* te zetten. Het is vanzelfsprekend dat een kleine harde schijf op de *target computer* groot genoeg moet zijn voor alle data van de *source computer*.

Om het onderzoek in deelvragen op te splitsen lopen we door het scenario van Jelle uit de sectie "Probleemstelling". Aan de hand van het scenario leggen we uit wat onderzocht moet worden en welke kennisgebieden nodig zijn voor het onderzoek. Bovendien vormen de deelvragen de basis voor de sectie methode en de planning.

In het voorbeeld gebruikt Jelle een USB-stick. De keuze van een USB-stick is een vrije keuze die los staat van het centrale probleem van dit onderzoek, namelijk het draaien van het besturingssysteem, de applicaties en de data van een computer op een andere computer. We lichten de reden voor deze keuze aan het eind van deze sectie toe.



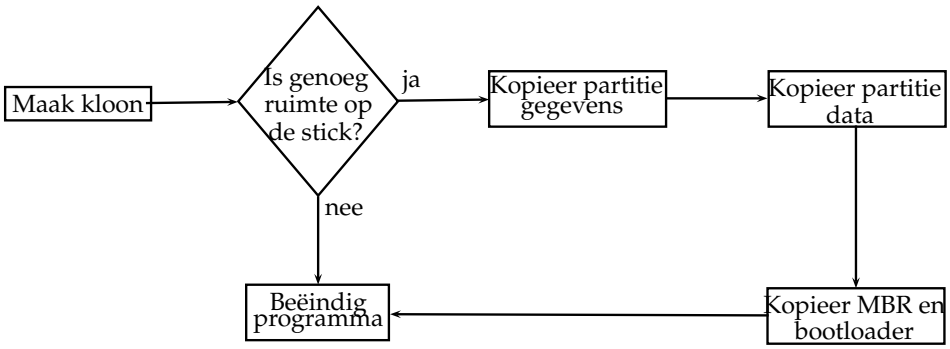
Figuur 1: Verloop van het hele proces

Jelle plugt een USB-stick in een USB-poort van zijn computer op kantoor en start de computer op. De computer start vanuit de USB-stick op. We nemen aan dat de Bios het starten van de computer vanuit een USB-stick toestaat. Op de USB-stick staat een programma, geschreven in "C", dat automatisch uitgevoerd wordt als de computer vanuit de USB-stick opstart. De computer start op en er verschijnen drie opties op het scherm. Optie "0" is het maken van een kloon van de harde schijf, optie "1" is het schrijven van de kloon op de harde schijf en optie "q" is het beëindigen van het programma (figuur 1). Het

probleem bij deze stap is het automatisch uitvoeren van het C-programma. Dit programma maakt gebruik van *services* die de *Linux Kernel* beschikbaar stelt, zoals “fdisk” , “tar” etc. De USB-stick moet dus ook de *Linux Kernel* bevatten. De deelvraag, “hoe wordt een programma automatisch vanuit een *bootable* USB-stick uitgevoerd?” leidt tot de oplossing van dit issue.

3.1 Maak de kloon

Jelle kiest voor optie “0” en het programma begint een kloon van de harde schijf te maken (figuur 2). Allereerst worden de gebruikte opslagruimte op de harde schijf en de beschikbare ruimte op de USB-stick met elkaar vergeleken. De opslagruimte op de USB-stick moet gelijk zijn aan of groter zijn dan de grootte van de data op de harde schijf. Als dit niet het geval is dan geeft het programma de melding dat de opslagruimte op de USB-stick te klein is voor de kloon en het programma wordt beëindigd. Als de stick genoeg ruimte heeft voor de data slaat het programma eerst de partitie-gegevens van de harde schijf op. Een probleem dat we tegen kunnen komen bij deze stap heeft te maken met de partitie-gegevens. De deelvraag “welke informatie is nodig om partities van een harde schijf na te kunnen bouwen?” moet voor deze stap beantwoord worden.



Figuur 2: Proces van het make van de kloon

Na het opslaan van de partitie-gegevens slaat het programma de inhoud van iedere partitie apart op. We gebruiken het programma “tar”om de inhoud van iedere partitie op te slaan. Deze data bevatten de bestanden en mappen die met de hardware te maken hebben niet. Deze mappen en bestanden moeten geïdentificeerd worden. De deelvraag, “welke bestanden en mappen in Linux zijn hardware afhankelijk?” leidt tot een lijst die bij het opslaan van de data gebruikt wordt. Nu de data opgeslagen zijn moeten we ook de *Master Boot*

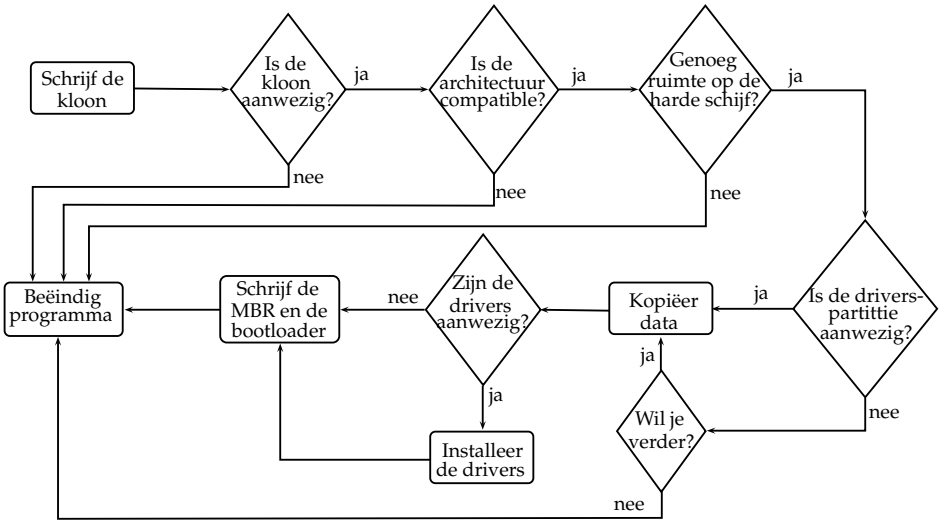
Record opslaan. Dit issue moet worden opgelost aan de hand van de vraag "hoe kunnen we de *Master Boot Record* en de *Bootloader* van een harde schijf opslaan?".

Het opslaan van de *Master Boot Record* is de laatste stap van het maken van een kloon. Hierna bereidt het programma de USB-stick voor op de loskoppeling en laat het weten dat de kloon succesvol gemaakt is. Er verschijnt een bericht op het scherm om de gebruiker te laten weten dat hij de USB-stick los mag koppelen als de computer uit is. Vervolgens gaat de computer uit.

3.2 Schrijf de kloon op de harde schijf

In dit stuk beschrijven we het proces waardoor de kloon op de harde schijf geschreven wordt. We houden nog het scenario van Jelle aan en leggen uit welke deelvragen beantwoord moeten worden. Aan het eind van dit stuk geven we een pseudocode om dit proces overzichtelijker te maken.

Als Jelle in zijn appartement in de VS is, plugt hij de USB-stick in op de computer en zet hij de computer aan. De computer start vanuit de USB-stick op en er verschijnen wederom drie opties op het scherm. Hij kiest optie "1" om de kloon op de harde schijf te schrijven. Het programma controleert of er een kloon op de USB-stick aanwezig is (figuur 3).



Figuur 3: Het proces van het schrijven van de kloon

De controle moet voorkomen dat het programma andere data aanziet voor de kloon. De deelvraag "wat moet gecontroleerd worden voor de aanwezigheid

van de kloon?" leidt tot het ontwerp van deze test. Als het programma de kloon niet vindt, verschijnt het bericht dat er geen kloon op de USB-stick staat en wordt het programma beëindigd. Als het programma de kloon wel vindt gaat het verder met de architectuurcontrole. Het programma controleert in deze stap of de architectuur van de kloon *compatible* is met de architectuur van de computer. In tabel 2 is de *compatibility* van de architecturen weergegeven.

	32x-hardware	64x-hardware
32x-kloon	✓	✓
64x-kloon	×	✓

Tabel 2: Verenigbaarheid van de kloon- en hardwarearchitectuur

Als de architectuur van de computer en de architectuur van de kloon niet *compatible* zijn dan bericht het programma de gebruiker over dit probleem en wordt het programma beëindigd. In het geval dat de architecturen wel *compatible* zijn controleert het programma de capaciteit van de harde schijf en de grootte van de kloon. Als de kloon groter is dan de harde schijf, verschijnt het bericht op het scherm dat de harde schijf van de computer te klein is voor de kloon en wordt het programma beëindigd. Als de harde schijf groot genoeg is gaat het programma verder. Vóór het programma de kloon op de harde schijf schrijft wordt er gecontroleerd of er een speciale partitie op de harde schijf aanwezig is.

We introduceren hier een partitie waarop de drivers van de computer opgeslagen zijn. Het programma gebruikt deze partitie om de drivers en andere informatie over de hardware van de computer waar de kloon op geschreven wordt, te vinden. Het antwoord op de deelvraag "welke drivers en hardware-informatie heeft het besturingssysteem nodig?" laat zien wat er in deze partitie opgeslagen moet worden. We noemen deze partitie de "drivers-partitie".

Als deze partitie niet bestaat laat het programma weten dat de drivers van de computer niet beschikbaar zijn en dat de kloon mogelijk niet optimaal met de hardware zal werken. De gebruiker moet aangeven of hij door wil gaan met het schrijven van de kloon op de harde schijf of niet. Als hij niet verder wil gaan wordt het programma beëindigd. Maar als hij verder gaat of het programma de partitie al gevonden heeft verwijdert het programma alle partities van de harde schijf behalve de drivers-partitie (als deze aanwezig is). Vervolgens maakt het programma de nieuwe partities op basis van de partitie-gegevens van de *source computer*. De deelvraag "hoe partitioneren we de harde schijf van de *target computer*?" moet leiden tot een wiskundig model

waarmee het programma de harde schijf partitioneert. Dit model berekent de grootte van de nieuwe partities op basis van de omvang van de partities op de *source computer*, de partitiedata en de capaciteit van de harde schijf. Nadat de partities gemaakt zijn *mount* het programma de partities op basis van de opgeslagen mount-informatie. Daarna worden de data van iedere partitie op de harde schijf geschreven. Vervolgens worden de Master Boot Record en de bootloader op de boot-partitie geschreven. Tenslotte worden de mappen en bestanden die met de hardware te maken hebben op de harde schijf aangemaakt en worden de drivers geïnstalleerd, indien de drivers partitie aanwezig is. De installatie van de drivers moet zo eenvoudig mogelijk zijn. De deelvraag die hierbij gesteld kan worden luidt: "hoe kunnen de hardware-drivers het best opgeslagen worden: in *binaries*, in *source code* of als een combinatie van beide?". Het antwoord op deze vraag geeft inzicht in de structuur van de opgeslagen data.

Na de installatie van de drivers verschijnt het bericht op het scherm dat de kloon op de harde schijf geschreven is, de USB-stick losgekoppeld kan worden en de computer opnieuw opgestart mag worden.

Als de computer opgestart is komt hij vanuit de harde schijf op en het besturingssysteem wordt geladen. De computer start zoals de *source computer*.

3.2.1 Pseudocode

Algorithm 1 Schrijven van de kloon op de harde schijf

```
if the clone is not present then
    Show "The clone does not exist"
    exit
else if the architecture is not compatible then
    Show "The architecture is not compatible"
    exit
else if There is not enough space on the HDD then
    Show "There is not enough space on the HDD"
    exit
else if The drivers partition does not exist then
    Show "The drivers are not accessible"
    if the user does not want to continue then
        exit
    end if
end if
write data
if the drivers partition exists then
    install drivers
end if
write MBS and bootloader
exit
```

3.3 Keuze USB-stick

We hebben voor de USB-stick gekozen omdat die makkelijk mee te nemen is en een ruime opslagruimte heeft. Op dit moment zijn er USB-sticks van 500G. Ze zijn weliswaar duur, maar met de snelle ontwikkelingen op het gebied van opslagruimte zullen ze over een paar jaar aanzienlijk goedkoper worden. De USB-stick zou in dit model ook vervangen kunnen worden door een externe harde schijf, CD of DVD. In het beoogde model wordt de kloon van de harde schijf op dezelfde USB-stick opgeslagen als waar het programma waarmee de kloon gemaakt is wordt opgeslagen. Als men een CD of DVD gebruikt i.p.v. een USB-stick is er een derde medium nodig om de kloon op te slaan. De rede hiervoor is dat de capaciteit van een CD of DVD ten opzichte van een harde schijf relatief klein is.

4 Methode

Dit onderzoek wil aantonen dat de kloon van een computer op een andere computer geplaatst kan worden zonder tussenkomst van de gebruiker. Met andere woorden, de gebruiker hoeft geen technische kennis te hebben om een kloon van een computer te maken of om de gemaakte kloon op een computer te schrijven. De gebruiker kiest de optie “maken van de kloon” of “schrijven van de kloon” en daarna hoeft hij alleen een aantal vragen met “ja” of “nee” te beantwoorden. We noemen het overzetten van de kloon op deze manier “automatische kloning”.

Automatische kloning moet mogelijk maken dat een kloon van een computer op iedere willekeurige computer die aan een aantal eisen voldoet (zie Theoretisch kader) geschreven wordt en correct functioneert. We brengen een hiërarchische structuur aan in het onderzoek. In dit onderzoek doen we een aantal aannames die het beschreven model in het theoretische kader afbakenen. We noemen dit afgebakende model het basismodel. Aan de had van het basismodel maken we een prototype. In een vervolgonderzoek kan het prototype verder ontwikkeld worden door de aannames een voor een weg te strepen. Hieronder sommen we de aannames voor het basismodel op:

1. De hardware van de *source computer* en de *target computer* zijn identiek (zelfde computer)
2. De *source computer* heeft alleen één harde schijf
3. De hardware drivers zijn niet nodig
4. De data op de draagbare opslagruimte is niet corrupt
5. Het *bootloader*-programma is “Grub”
6. Opstarten van de computers gaat via *Master Boot Record*. Met andere worden is de partitie-tabel van de harde schijf de traditionele “msdos” tabel.
7. De volgende *filesystems* worden ondersteund: “ext4”, “ext3”, “ext2” en “swap”

Aan de hand van de deelvragen die in het stuk “Theoretisch kader” gesteld zijn en van de aannames voor het basismodel bestuderen we de nodige literatuur, maken we, waar nodig, een prototype dat antwoord geeft op de deelvraag en schrijven we het resultaat op. Aan het eind van het onderzoek laten we zien of automatische kloning voor het basismodel mogelijk is. Als het mogelijk is bewijzen we dit aan de hand van het gemaakte prototype. Als blijkt dat het niet mogelijk is laten we zien wat de obstakels zijn en hoe het model eventueel

aangepast kan worden om de automatische kloning wel mogelijk te maken.

We behandelen iedere deelvraag in een paragraaf en beschrijven het probleem, de manier waarop we het probleem hebben benaderd en de oplossing.

4.1 Autorun programma vanuit *bootable USB-stick*

In het beschreven model gebruiken we een *USB-stick* als draagbare opslagruimte en als medium van waaruit de computer opstart met als doel om ofwel de kloon te maken ofwel de kloon op de harde schijf te schrijven. Dit betekent dat onze *USB-stick* een *bootable USB-stick* moet zijn. Bovendien start er vanuit de *USB-stick* automatisch een programma dat communiceert met de gebruiker om zijn opdracht, namelijke het maken van de kloon of het terugschrijven van de kloon, uit te voeren. Het is dus nodig om een *USB-stick* te bouwen die ons niet alleen in staat stelt om de computer vanuit de *USB-stick* op te starten, maar die ook ons programma automatisch opstart.

Een *bootable* CD of USB wordt in het Linux-jargon *live-CD* of *live-USB* genoemd. Een *live-USB* of *-CD* bevat een *live system* . *Live system* is een besturingssysteem dat zonder installatie op de harde schijf vanuit een CD of *USB-stick* opstart. Er zijn nog andere media van waaruit een *live system* op kan starten, zoals een netwerk of een externe harde schijf.

“Debian”, een van de Linux-distributies, vormt de basis voor een aantal andere bekende “Linux” distributies zoals “Ubuntu” en “Mint”. “Debian” heeft een project genaamd *Debian live*. Dit project bevat een aantal programma’s die het voor de gebruiker mogelijk maakt een *live system* op maat te maken. *Debian live* staat toe een (minimaal) debian-besturingssysteem te bouwen dat *scripts* van de gebruiker automatisch uitvoert. Wij gebruiken *debian live* als basis voor onze *bootable usb-stick*. Bovendien heeft *Debian live* een goede handleiding. Om deze redenen kiezen wij “Debian” als basis voor ons *live system*.

Een *live system* moet ons in staat stellen om *system commands* in de vorm van een script uit te voeren. Om te controleren of *debian live* aan onze eis voldoet bouwen we een debian-besturingssysteem dat vanuit een *usb-stick* opstart en ons “*Hello World*” programma uitvoert. We maken gebruik van de versie 3.0 $a47-1$ van *debian live*. Het bouwen van ons besturingssysteem wordt in twee fases opgesplitst, namelijk het configureren van het besturingssysteem en het bouwen daarvan. Tijdens het configureren kunnen we niet alleen bepalen welke programma’s op het besturingssysteem geïnstalleerd moeten worden maar kunnen we ook de taal, het toetsenbord, de *layout* en vele andere instellingen vastleggen. Deze fase is de fase waarin de gebruiker aan moet geven

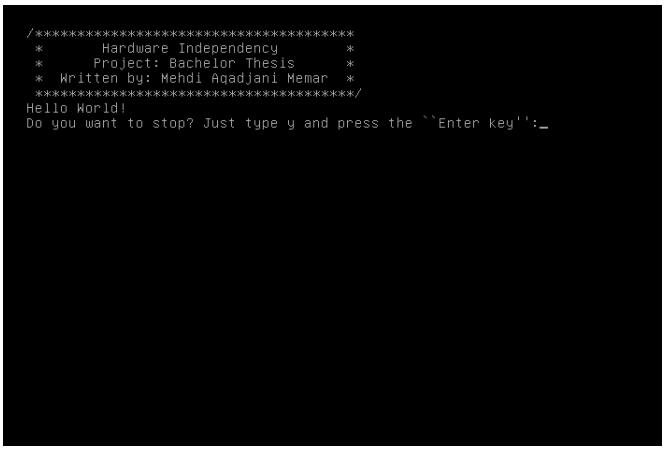
welke eigen programma's het besturingssysteem mag bevatten en welke programma's automatisch uitgevoerd moeten worden.

Om het besturingssysteem te laten weten dat een programma bij het opstarten automatisch uitgevoerd moet worden gebruiken we twee *bash scripts*. Het ene script laat het besturingssysteem weten dat het andere script automatisch bij het opstarten van het *live system* uitgevoerd moet worden . Het spreekt voor zich dat het andere script commando's bevat die wij automatisch willen uitvoeren.

De standaard *bootloader* van een *live system* is Syslinux. Deze *bootloader* heeft een paar standaard opties en een Debian-logo als achtergrond. Als een *live system* opstart ziet de gebruiker de *bootloader* en de opties waarvan hij er één moet kiezen om door te gaan. Voor onze toepassing is het verwarrend als de gebruiker met deze standaard opties van de *bootloader* geconfronteerd wordt. Daarom wijzigen we "Syslinux" zodat de gebruiker een juiste indruk krijgt van het programma. We wijzigen de *bootloader* aan de hand van een derde script. Het script wijzigt de titel van het menu, de opties van de *bootloader* en de achtergrond.

Hieronder twee *screen shots* van het gebouwde besturingssysteem dat ons *Hello World!* script automatisch uitvoert.

Figuur 4: Het *boot menu* voor het opstarten van het besturingssysteem



Figuur 5: Het uitgevoerde *Hello World* programma door het besturingssysteem

4.2 Partitie- en *mount*-gegevens van de harde schijf

Wanneer een computer door een *live system* opstart zijn mappen en bestanden op de harde schijf van de computer niet direct bereikbaar. Dit komt omdat de partities van de harde schijf niet gekoppeld zijn aan mappen op het *live system*. Het koppelen van een partitie aan een map wordt *mounting* genoemd. Dus we moeten eerst partities van de *source computer* mounten voordat we een kloon van de de data kunnen maken. De eerste partitie die gemount moet worden is de *boot*-partitie. De *boot*-partitie bevat systeem-informatie die we nodig hebben bij het mounten van andere partities. Het is ook belangrijk om op te merken dat partities niet alleen door hun naam maar ook door een unieke id, *universally unique identifier* oftewel "UUID" te geven aan geïdentificeerd kunnen worden.

In ons model heeft de *target computer* dezelfde partitie-indeling als die van de *source computer*. Het is ook mogelijk om één partitie op de *target computer* te maken en alle data van de *source computer* op die partitie te schrijven. Als wij de partitie-indeling van de *source computer* niet naar de *target computer* meenemen, zal de implementatie van het script dat deze taak uitvoert, makkelijker worden. Desalniettemin kiezen we voor het behoud van de partitie-indeling van de *source computer*. De reden hiervoor is dat meerdere partities meer bescherming bieden tegen eventuele beschadiging van de harde schijf.

Er is een aantal *system commands* dat informatie geeft over de harde schijf, partities, hun grootte etc. Deze informatie is gebaseerd op de partitie-tabel die in de volgende deelvraag aan de orde komt. We kunnen "fdisk", "sfdisk" en "parted" als voorbeeld van *system commands* noemen. We kiezen voor het

laatste omdat “parted” meer informatie geeft waardoor we de eerste harde schijf kunnen onderscheiden van andere aangesloten opslagmedia, zoals *USB-sticks* of externe harde schijven. De eerste harde schijf is de harde schijf waaruit de computer opstart. De informatie gegeven door “parted” wordt ook gebruikt om later de *target computer* te partitioneren. Dus de output van “parted” moet opgeslagen worden.

Het *bash script* voert “parted” uit en analyseert de output voor het vinden van de *boot*-partitie. Daarna *mount* het script de gevonden partitie. Na de *mounting* beschikt het script over de data van partities. De *mount*-informatie over alle partities zijn opgeslagen in het bestand `etc/fstab` van de *boot*-partitie. In dit bestand is iedere partitie, geïdentificeerd met zijn “UUID”, en zijn *mount*-locatie opgeslagen. De output van “parted” over partities is gebaseerd op de partitienamen. Dus we moeten de gevonden partities uit de output van “parted” met hun *mount*-locatie, opgeslagen in het bestand `fstab`, aan elkaar koppelen. Hiervoor gebruiken we het *system command* “blkid” . “blkid” laat attributen van de partities zien. Het script koppelt iedere partitie uit de output van “parted” aan zijn *mount*-locatie uit het bestand `fstab` met behulp van informatie uit het *system command* “blkid”.

Voordat we de genomen stappen in een algoritme beschrijven is het goed om op te merken dat de *system commands* in “Linux” output produceren volgens een vast patroon waardoor het makkelijk is om de output te analyseren. Ook voor de bestanden die systeem-informatie bevatten geldt een vast patroon.

Algorithm 2 Find the mount location of partitions

```
P =name of the disk partitions from the output of parted -l
L =UUID's and their mount locations from the file /etc/fstab
B =output of blkid
R = {}
while B has next element do
    E =UUID and partition name of the next element B
    while L has next element do
        F = next element L
        if E[UUID] = F[UUID] then
            while P has next element do
                N = next element P
                P = P without N
                if N[name] = E[name] then
                    R = R + {N[name], E[UUID], L[location]}
                end if
            end while
        end if
        L = L without F
    end while
    B = B without E
end while
return R
```

De output van van “blkid” wordt ook gebruikt voor het maken van *filesystems*

op de partities van de *target computer*. Voordat een besturingssysteem een partitie kan gebruiken, moet de partitie zijn voorzien van een een *filesystem*. Dit betekent dat we de partities van de *target computer* moeten voorzien van *filesystems*. Hiervoor gebruiken we het *system command* "mkfs" en de output van "blkid". De output bevat alle informatie die wij nodig hebben voor het maken van het *filesystem*. We slaan deze output op in het bestand `blkidData` op het draagbare opslagmedium. Deze data worden tijdens het overschrijven van de kloon op de *target computer* geanalyseerd en op basis daarvan worden er *filesystems* op de partities gemaakt.

Algorithm 3 Make filesystems on partitions

```
array = content of the file blkidData
COUNTER = 0
while ! end of array do
    if COUNTER %3 == 0 then
        if array[COUNTER] == ëxt4" then
            mkfs.ext4 -U array[COUNTER+1] array[COUNTER+2]
        else if array[COUNTER] == ëxt3" then
            mkfs.ext3 -U array[COUNTER+1] array[COUNTER+2]
        else if array[COUNTER] == ëxt2" then
            mkfs.ext2 -U array[COUNTER+1] array[COUNTER+2]
        else
            mkfswap -U array[COUNTER+1] array[COUNTER+2]
        end if
    end if
    COUNTER + = 1
end while
```

4.3 Kopiëren MBR en *bootloader*

Een *IBM PC compatible* systeem heeft een "Master Boot Record" , in het kort "MBR". Wanneer de computer opstart, voert de "BIOS" een aantal controles uit en vervolgens wordt de code in de "MBR" uitgevoerd. De "MBR" laadt de *kernel* en vanaf dat moment komt de controle over de hard- en software bij het besturingssysteem. Dus de *target computer* moet ook voorzien zijn van de "MBR". In het geval dat de harde schijf van de *source-* en de *target computer* niet identiek zijn moeten we de "MBR" op de *target computer* zelf maken. Daarvoor partitioneren we eerst de harde schijf op basis van informatie die "parted" heeft gegenereerd van de *source computer*. De grootte van de partities worden door een wiskundig model aangepast aan de harde schijf van de *target computer*. Aangezien we aangenomen hebben dat de *source-* en *target computers* identiek zijn laten we het ontwikkelen van het wiskundige model achterwege. Na het partitioneren van de harde schijf moeten we ook de *bootloader* installeren.

In het geval dat de computers identiek zijn mogen we exact dezelfde “MBR” data op beide computers, *source-* en *target computer*, hebben. Voor het maken van een kopie van “MBR” gebruiken we het *system command* “dd” . We maken een kopie van de eerste 512 *bytes* van de *boot*-harde schijf, op de *source computer*. Deze kopie kan later weer met hetzelfde commando, op de *target computer* geschreven worden.

Algorithm 4 Save MBR

```
A = name of primary hard disk
dd if=A of=/home/user/clone/mbr bs= 512 count= 1
```

4.4 Verplaatsen van data

In dit onderdeel beschrijven we twee onderwerpen die met het verplaatsen van data van de *source-* naar de *target computer* te maken hebben. Het eerste onderwerp is de keuze voor de methode waarmee we de data verplaatsen. Het tweede gaat over het verplaatsen van map- en bestandrechten.

De keuze voor de methode heeft invloed op de snelheid van het proces, de gebruikte opslagruimte op het draagbare opslagmedium en de kosten voor de aanschaf van het medium. Ons *live system* biedt een aantal *system commands* waarmee we een kopie van de data van de *source computer* kunnen maken. Met “data van de *source computer*” bedoelen we de partitie-data en niet de data op de “MBR”. De *System commands* “dd” en “cp” zijn niet geschikt voor het kopieëren van de data. De reden is dat deze programma’s de data niet comprimeren en de gekopiëerde data nemen evenveel of soms meer ruimte in beslag dan de originele data. Als wij “dd” gebruiken en alle *sectors* van de harde schijf kopiëren kopieert “dd” ook de lege *sectors*. In dit geval zijn de gekopiëerde data even groot als de harde schijf waar de de originele data op staan. Aangezien de data op een draagbaar opslagmedium opgeslagen worden is het belangrijk dat ze bij het kopiëren gecomprimeerd worden. “tar” is een *system command* dat comprimeren van data mogelijk maakt. “tar” biedt aan aantal compres-methodes die in snelheid en grootte van de gecomprimeerde data verschillen. Het comprimeren met “bz2” levert de kleinste gecomprimeerde data maar het comprimeren duurt lang. Het comprimeren met behulp van “gz” is een geschikte keuze voor ons model. Deze methode biedt een balans tussen de grootte van de gecomprimeerde data en de duur van het proces.

Alle bestanden en mappen in een “Linux” besturingssysteem hebben traditionele data-rechten. We kunnen deze rechten met traditionele kopie-/archief-

applicaties van overplaatsen. Er is echter ook andere informatie, de *Access Control List* of in het kort “ACL”, die niet door traditionele kopie-/archief-programma’s met gekopiëerde bestanden en mappen meegenomen kan worden. We moeten bij het kopiëren van data hiermee rekening houden. “Debian” stelt twee applicaties beschikbaar om een backup van “ACL” te maken en om de gemaakte backup terug te schrijven. We gebruiken deze applicaties, “getfacl” en ‘setfacl” om eventuele extra informatie die een map of een bestand bevat, mee te nemen naar de *target computer*. We gebruiken “getfacl” voor het maken van een bestand dat alle “ACL” informatie van de root-map en zijn sub-mappen en bestanden bevat. “setfacl” gebruikt de opgeslagen informatie in dit bestand om de rechten terug te zetten.

Algorithm 5 Save partition data

unmount all partition except boot partition
$A =$ mount location boot partition
$B =$ data location on the portable storage medium
$C =$ excluded directories
tar cps-f A –exclude C -C B *
mount all partition
$A =$ all partition except boot partition
while A has next element **do**
    $B =$ next element of A
    $D =$/home/user/mnt/"+ B[mount location]
    $L =$/home/user/clone/ + B[mount location] + ".tar"
    $A = A$ without next element
    $tarcps - f$L−CD∗
end while
cd /home/user/mnt/
getfacl * > /home/user/clone/acl

4.5 Terugschrijven van de kloon

Theoretisch gezien zal het terugschrijven van de kloon op de harde schijf, zoals beschreven in de sectie “Theoretisch kader”, zonder problemen plaats vinden. De praktijk heeft echter anders bewezen, namelijk dat een programma (bijna) altijd bugs heeft en de bugs zorgen ervoor dat het programma niet functioneert zoals het hoort. Bugs komen meestal aan het daglicht tijdens het testen. Aangezien we het programma op een productieve laptop willen testen, lopen we het risico om data te verliezen door het optreden van bugs. De kans op dataverlies is groot om dat we de “Master Boot Record” van de laptop overschrijven.

De belangrijkste opgave in deze fase is ervoor te zorgen dat we data van de laptop tijdens het testen van het programma niet verliezen. Als bugs tijdens het overzetten van de kloon op de harde schijf optreden, zouden ze tot het verlies

van de kloon en de data op de laptop leiden. Om deze reden kiezen we ervoor een *backup* te maken van de kloon op een tweede computer. Dit betekent dat we een kopie maken van alle bestanden en mappen, die bij het proces van "het maken van de kloon" op het draagbare opslagmedium gemaakt zijn. Deze kopie wordt op een productieve computer opgeslagen. Op deze manier kunnen we als het nodig, de gemaakte kloon handmatig weer op het draagbare opslagmedium kopiëren.

Hoewel de beschreven stap geen volledige garantie geeft om de data van laptop in alle omstandigheden terug te schrijven, is het wel belangrijk om deze stap te nemen. De reden van de onvolledige garantie is dat we niet weten of de gemaakte kloon al volledig is. Stel dat we een redeneringsfout hebben gemaakt in het proces van "het maken van de kloon" en dat we daardoor sommige belangrijke data niet opgeslagen hebben bij het maken van de kloon, dan is de kloon zelf onvolledig. In deze situatie leidt het overschrijven de kloon op de laptop tot het verlies van de data op de laptop, of er nu bugs optreden of niet.

5 Test

De ontwikkelde applicatie is geschreven in “Bash scripting” en bestaat uit een code van 1450 regels. Deze applicatie is op een “Debian *Live USB-stick*” geplaatst en wordt automatisch uitgevoerd bij het opstarten van het besturingssysteem op de *USB-stick*. De benodigde ruimte op de *USB-stick* voor het besturingssysteem en de applicatie is 154MB.

Tijdens het testen bleek de applicatie de gemaakte kloon niet op de harde schijf te kunnen schrijven. Dit kwam omdat er geen rekening gehouden was met het *filesystem* van de partities. We hadden alleen de partitie-tabel van de harde schijf overgezet en geen *filesystem* gemaakt op de partities. Na deze ontdekking hebben we het antwoord op de tweede deelvraag aangevuld en de applicatie aangepast.

Voor het testen hebben we een *notebook* en een *USB-stick* 2.0 van 64 gigabyte gebruikt. Het *notebook* heeft de de volgende kenmerken:

harde schijf: 250GB

geheugen: 1GB

CPU: intel atom n450 1.66Ghz

CPU cache: 512KB

aantal partities: 3

grootte boot-partitie: 30GB

gebruikte ruimte op de boot-partitie: 7.8GB

grootte home-partitie: 218GB

gebruikte ruimte op de home-partitie: 7.5GB

grootte swap-partitie: 2GB

We hebben de applicatie alleen op een computer met “Debian” besturingssysteem getest, maar de applicatie zou op alle “Linux” distributies moeten werken. In de hele code hebben we één bestand van het besturingssysteem gebruikt en dit bestand is identiek in alle “Linux” distributies. We hebben de test drie keer uitgevoerd op het *notebook* met een 64× “Debian” besturingssysteem. Bij het opstarten en het stoppen van ieder proces, “het maken van de kloon” en “het overschrijven van de kloon” hebben we de tijd opgeslagen in een file. Hieronder de resultaten:

	Maken	Schrijven
Eerste test	41" 36’	8" 43’
Tweede test	43" 09’	9" 03’
Derde test	43" 00’	8" 51’

Tabel 3: De tijdsduur van het maken en het overschrijven van de kloon

De grootte van de gekloonde data op de *boot*-partitie en de *home*-partitie zijn respectievelijk 4.1GB en 4GB. Deze getallen komen overeen met ongeveer de helft van de gebruikte ruimte van de partities op het *notebook*.

6 Reflectie

We hebben in de secties “Probleemstelling” en “Verantwoording” gezien dat er twee methodes bestaan die de gebruiker in staat stellen om zijn besturingssysteem, geïnstalleerde applicaties en data van de ene naar de andere computer te verplaatsen. Een methode hebben we zelf geïntroduceerd en de andere methode is de methode die in de hardware-virtualisatie gebruikt wordt. De voordelen van onze methode ten opzichte van de andere methode hebben we onze methode hebben we op een rij gezet. Onze methode kent echter ook enkele nadelen.

Het grootste probleem van de onderzochte methode is dat we ervan uitgaan dat de drivers van de hardware van alle computers op een partitie opgeslagen zijn. Deze veronderstelling brengt de volgende vragen en moeilijkheden met zich mee: “hoe dwingen we de aanwezigheid van de driverspartitie af zodat “alle” computers de partitie hebben?”, “hoe wordt een driver geüpdatet op de driverpartitie als er een nieuwe versie van de driver uitgebracht is of als een hardware onderdeel van de computer door een ander vervangen wordt?” en “hoe zorgen we dat de aanwezige drivers installeerbaar zijn op alle besturingssystemen?”

Een ander probleem is de duur van het maken en schrijven van de kloon. Deze processen kosten relatief veel tijd als de data op de *source computer* groot zijn. De reden hiervoor is dat het programma “tar” relatief traag is in combinatie met het comprimeren.

Een ander nadeel van onze methode is dat de applicatie zoals deze nu ontworpen is, de data van de partities die niet *mounted* zijn niet meeneemt naar de *target computer*. Dit betekent dat als een gebruiker een partitie op zijn harde schijf heeft waarop *backup*-bestanden opgeslagen zijn en de partitie niet automatisch *mounted* wordt het huidige prototype deze data negeert.

Tenslotte is er geen aandacht besteed aan de veiligheid van de data die als kloon op het draagbare opslagmedium opgeslagen zijn. In het huidige ontwerp is het makkelijk om de kloon te manipuleren en daardoor worden de data corrupt. Bovendien is de efficiëntie van het prototype niet in acht genomen. De code van het prototype en het gebruikte *system live* kunnen efficiënter dan ze nu zijn.

Methodes in de hardware-virtualisatie bestaan al sinds lange tijd en in de loop van de jaren zijn ze verbeterd. Sommige van de methodes geven een minimum aan *overhead*. Onderzocht zou moeten worden of de aanpassingen van bestaande methodes voor de hardware-virtualisatie een acceptabele *overhead*

kunnen bieden voor het antwoord op onze onderzoeksvraag.

7 Conclusie

In de loop van het onderzoek hebben we ontdekt dat de hoofdvraag te omvangrijk was voor de beschikbare tijd voor dit onderzoek. Daarom hebben we besloten om uit te gaan van een aantal aannames. Het stellen van deze aannames heeft geleid tot een vereenvoudigde onderzoeksvraag passend bij de omvang van dit onderzoek.

Binnen de gestelde aannames hebben we een antwoord op de onderzoeksvraag gevonden. Het is mogelijk om een kloon van de harde schijf van een computer te maken en deze vervolgens op een identieke computer over te schrijven. De gemaakte kloon van de harde schijf moet zo klein mogelijk zijn zodat hij een minimum aan opslagruimte nodig heeft. We kunnen de maximale grootte van de kloon niet vaststellen aangezien de grootte van de data op iedere computer varieert. Bij het streven naar een zo klein mogelijke kloon moeten we de tijdsduur van het comprimeren in de gaten houden. Als het comprimeren te lang duurt is de applicatie niet bruikbaar.

Een andere bevinding in dit onderzoek is dat we een besturingssysteem in een aantal losse data pakketten kunnen opsplitsen waarmee we het besturingssysteem op een andere computer kunnen reconstrueren. We hebben deze pakketten zo gemaakt dat ze alleen de bouwstenen bevatten van het besturingssysteem. Met deze bouwstenen kunnen we de weggelaten informatie terughalen.

We hebben de hoofdvraag zoals gesteld in de probleemstelling niet kunnen beantwoorden, maar het resultaat van ons onderzoek vormt een goed uitgangspunt voor vervolg onderzoek naar de oorspronkelijke vraag.

8 Appendix

```
#!/usr/bin/env bash

### BEGIN INIT INFO
# Provides: hardwareIndependent
# Required-Start: $syslog
# Required-Stop: $syslog
# Default-Start: 1 2 3 4 5
# Default-Stop:
# Short-Description: Software
# Description: To make and write the clone
### END INIT INFO

#===============================================================================
#
#          FILE: hardwareIndependent
#
#         USAGE: ./hardwareIndependent
#
#   DESCRIPTION:
#
#       OPTIONS: ---
#  REQUIREMENTS: ---
#          BUGS: ---
#         NOTES: ---
#        AUTHOR: Mehdi Aqadjani Memar (-), m.aqadjanimemar@student.ru.nl
#  ORGANIZATION: -
#       CREATED: 05/06/2012 11:13:22 PM CEST
#      REVISION:  ---
#===============================================================================


# GLOBAL VARIABLES
declare -r LIBRARY="/lib/hind/"                         # programme library
declare -r MESSAGES=${LIBRARY}messages                  # programme messages
declare -r WORKING_DIRECTORY="/home/user/"              # directory in wich the
                                                        # script writes
declare -r MOUNT_LOCATION=${WORKING_DIRECTORY}mnt/      # mount directory
declare -r CLONE_LOCATION=${WORKING_DIRECTORY}clone/    # clone directory
declare -r LOCATION_PARTITION_DATA=${CLONE_LOCATION}partitionData/  #partition
                                                                   # data
declare -r MASTER_BOOT_RECORD=${CLONE_LOCATION}mbr      # location of mbr
declare -r ACCESS_CONTROL_LIST=${CLONE_LOCATION}acl     # location of acl data
declare -r BOOT_PARTITION_DATA=${CLONE_LOCATION}boot.tar.gz # boot partition
                                                           # data
declare -r SPECIAL_DIRECTORIES=("dev" "proc" "sys" "run" "tmp") # special
                                                                # directories
declare -i EXTRA_SPACE=5000
declare -r TIME_FILE=${WORKING_DIRECTORY}time     # location of time registration
declare -r BLKID_FILE=${WORKING_DIRECTORY}blkidData # location of blkid data

source ${LIBRARY}sharedLib
source ${LIBRARY}makeClone
source ${LIBRARY}writeClone

OPTION=-1

startProgramme        # function
dmesg -n 1

# user option is not valid
while [ $OPTION -lt 0 -o  $OPTION -gt 2 ]
do
 clear
 sed '/^[0-8]\s/!d; s/^\([0-8]\s\)\(.*\)/\2/' $MESSAGES
 echo
 getOption       # function
 OPTION=$?
done


case $OPTION in
0)                          # option 0 is chosen
  sed '/^10\s/!d; s/^\(10\s\)\(.*\)/\2/' $MESSAGES
  ;;
1)                          # option 1 is chosen
  date > $TIME_FILE
  makeClone
  ;;
2)                          # option 2 is chosen
  date > $TIME_FILE
  writeClone
  ;;
esac


FUNCTION_RESULT=$?

# option 1 is chosen and the clone is made
if [ $OPTION -eq 1 -a $FUNCTION_RESULT -eq 0 ]
then
  sed '/^11\s/!d; s/^\(11\s\)\(.*\)/\2/' $MESSAGES
# option 2 is chosen and the clone is written
elif [ $OPTION -eq 2 -a $FUNCTION_RESULT -eq 0 ]
then
  sed '/^13\s/!d; s/^\(13\s\)\(.*\)/\2/' $MESSAGES
fi


if [ $OPTION -ne 0 ]
then
  unmountPartitions      #function
  date >> $TIME_FILE      # set end time
fi




#!/usr/bin/env bash
#===============================================================================
```

```
#
#           FILE: makeClone
#
#          USAGE: Library
#
#    DESCRIPTION:
#
#        OPTIONS: ---
#   REQUIREMENTS: ---
#            BUGS: ---
#           NOTES: ---
#          AUTHOR: Mehdi Aqadjani Memar (-), m.aqadjanimemar@student.ru.nl
#   ORGANIZATION: -
#        CREATED: 05/07/2012 11:35:17 AM CEST
#       REVISION:  ---
#===============================================================================


# GLOBAL VARIALBLES
BOOT_PARITION_NAME=              # name of boot partition
FUNCTION_RESULT=                 # result of evaluated function
PARTITION_POINT_ARRAY=
COUNTER=
FSTAB_FILE=
USER_INPUT=
FREE_SPACE_PROTABLE_MEDIUM=
USED_SPACE_HARD_DISK=


# makes a clone on the portable storage medium
makeClone ()
{
  existClone                     # function
  FUNCTION_RESULT=$?             # result of evaluated function

  # clone exists
  if [ $FUNCTION_RESULT -eq 0 ]
  then
    askUserToContinue            # function
    FUNCTION_RESULT=$?

    # user wants to stop
    if [ $FUNCTION_RESULT -eq 1 ]
    then
      sed '/^19\s/!d; s/^\(19\s\)\(.*\)/\2/' $MESSAGES
      return 1

    # user wants to continue
    else
      rm -rf $CLONE_LOCATION
    fi

  fi

  mountHardDisk                  # function
  FUNCTION_RESULT=$?             # result of evaluated function

  # hard disk is not mounted
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^12\s/!d; s/^\(12\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  existEnoughSpacePortableMedium     # function
  FUNCTION_RESULT=$?   # result of evaluated function

  # Not enough space on the storage medium
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^12\s/!d; s/^\(12\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  mkdir $CLONE_LOCATION # make a clone location
  FUNCTION_RESULT=$?   # result of evaluated function

  # clone location is not made
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^35\s/!d; s/^\(35\s\)\(.*\)/\2/' $MESSAGES
    sed '/^12\s/!d; s/^\(12\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  unmountPartitions              # function
  FUNCTION_RESULT=$?             # return value of the function

  # partitions cannot be unmounted
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^221\s/!d; s/^\(221\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  copyMBR                        # Copy Master Boot Record
  FUNCTION_RESULT=$?
  # master boot record is not saved
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^12\s/!d; s/^\(12\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  mountBootPartition             # function
  FUNCTION_RESULT=$?
  # boot partition is not mounted
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^12\s/!d; s/^\(12\s\)\(.*\)/\2/' $MESSAGES
```

```
    return 1
  fi

  archiveBootPartitionData       # function
  # data of the boot partition is not archived
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^12\s/!d; s/^\(12\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  hasHardDiskMorePartitions        # function
  FUNCTION_RESULT=$?
  # an error has occurred
  if [ $FUNCTION_RESULT -eq 2 ]
  then
    sed '/^22\s/!d; s/^\(22\s\)\(.*\)/\2/' $MESSAGES
    return 1
  elif [ $FUNCTION_RESULT -eq 0 ]
  then
    mountOtherPartitions            # function
    FUNCTION_RESULT=$?
    # partitions are not mounted
    if [ $FUNCTION_RESULT -ne 0 ]
    then
      sed '/^22\s/!d; s/^\(22\s\)\(.*\)/\2/' $MESSAGES
      return 1
    fi

    archiveDataOtherPartitions                # function
    FUNCTION_RESULT=$?

    # partition data is not archived
    if [ $FUNCTION_RESULT -ne 0 ]
    then
      sed '/^12\s/!d; s/^\(12\s\)\(.*\)/\2/' $MESSAGES
      return 1
    fi
  fi

  makeAccessControlList              # function
  FUNCTION_RESULT=$?
  # access control list is not made
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^12\s/!d; s/^\(12\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  blkid > $BLKID_FILE
  return 0
}       # ---------- end of function makeClone ----------


# mounts the hard disk
mountHardDisk ()
{
  echo ">Start mountHardDisk"
  mountBootPartition     # function
  FUNCTION_RESULT=$?

  # boot partition is not mounted
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^26\s/!d; s/^\(26\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  hasHardDiskMorePartitions        # function
  FUNCTION_RESULT=$?
  # an error has occurred
  if [ $FUNCTION_RESULT -eq 2 ]
  then
    sed '/^22\s/!d; s/^\(22\s\)\(.*\)/\2/' $MESSAGES
    sed '/^26\s/!d; s/^\(26\s\)\(.*\)/\2/' $MESSAGES
    return 1
  elif [ $FUNCTION_RESULT -eq 0 ]
  then
    mountOtherPartitions            # function
    FUNCTION_RESULT=$?
    # partitions are not mounted
    if [ $FUNCTION_RESULT -ne 0 ]
    then
      sed '/^22\s/!d; s/^\(22\s\)\(.*\)/\2/' $MESSAGES
      sed '/^26\s/!d; s/^\(26\s\)\(.*\)/\2/' $MESSAGES
      return 1
    fi
  fi

  echo "<End End mountHardDisk"
  return 0
}       # ---------- end of function mountHardDisk ----------


# checks whether enough space is available on the storage medium
existEnoughSpacePortableMedium ()
{
  echo ">Start existEnoughSpacePortableMedium"
  set -o pipefail      # return pipe errors

  # set available space on the portable medium
  FREE_SPACE_PROTABLE_MEDIUM=$(df /home | awk '$4 ~ /[0-9]+/ {print $4}' \
                               2>/dev/null)
  FUNCTION_RESULT=$?

  # available space on the portable medium is not set
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^27\s/!d; s/^\(27\s\)\(.*\)/\2/' $MESSAGES
    return 1
```

```
  fi

  # set used space on the hdd boot partition
  USED_SPACE_HARD_DISK=`df $BOOT_PARITION_NAME 2>/dev/null | \
    awk '$3 ~ /[0-9]+/ {print $3}' 2>/dev/null`
  FUNCTION_RESULT=$?

  # used space on the hdd is not set
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^28\s/!d; s/^\(28\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  set +o pipefail      # unset pipefail

  # there are not any partitions, except boot partition and
  # portable medium has enough space
  if [ ${#PARTITION_POINT_ARRAY[*]} -eq 0 -a $FREE_SPACE_PROTABLE_MEDIUM -gt \
      $((USED_SPACE_HARD_DISK + $EXTRA_SPACE)) ]
  then
    echo "<End existEnoughSpacePortableMedium"
    return 0
  elif [ ${#PARTITION_POINT_ARRAY[*]} -gt 0 ]   # there exist more partitions
  then
    calculateUsedSpaceOtherPartitions    # function
    FUNCTION_RESULT=$?

    # used space on other partitions is not calculated
    if [ $FUNCTION_RESULT -ne 0 ]
    then
      sed '/^28\s/!d; s/^\(28\s\)\(.*\)/\2/' $MESSAGES
      return 1
    fi
  fi

  # enough space on the storage medium
  if [ $FREE_SPACE_PROTABLE_MEDIUM -gt $((USED_SPACE_HARD_DISK + \
                                          $EXTRA_SPACE)) ]
  then
    echo "<End existEnoughSpacePortableMedium"
    return 0
  fi

  sed '/^29\s/!d; s/^\(29\s\)\(.*\)/\2/' $MESSAGES
  return 1
}       # ----------  end of function existEnoughSpacePortableMedium  ----------


# calculates the used space on all partitions except boot partition
calculateUsedSpaceOtherPartitions ()
{
  echo ">Start calculateUsedSpaceOtherPartitions"
  # only boot partition
  if [ ${#PARTITION_POINT_ARRAY[*]} -eq 0 ]
  then
    sed '/^30\s/!d; s/^\(30\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  set -o pipefail   # return pipe error

  # through partitions
  for (( COUNTER = 0; COUNTER < ${#PARTITION_POINT_ARRAY[*]}; COUNTER++ ))
  do
    if [ $((COUNTER % 2)) -eq 0 ]   # counter is even
    then
      USED_SPACE_HARD_DISK=$((USED_SPACE_HARD_DISK+`df \
        ${PARTITION_POINT_ARRAY[$COUNTER]} 2>/dev/null | \
        awk '$3 ~ /[0-9]+/ {print $3}' 2>/dev/null`))
      FUNCTION_RESULT=$?

      # used space is not set
      if [ $FUNCTION_RESULT -ne 0 ]
      then
        set +o pipefail
        sed '/^31\s/!d; s/^\(31\s\)\(.*\)/\2/' $MESSAGES
        return 1
      fi

    fi
  done

  set +o pipefail   # unset pipefail
  echo "<End calculateUsedSpaceOtherPartitions"
  return 0
}       # ----------  end of function calculateUsedSpaceOtherPartitions  ----------


# copies the Mater Boot Record to the storage medium
copyMBR ()
{
  echo ">Start copyMBR"
  # copy master boot record
  dd if=${BOOT_PARITION_NAME:0:${#BOOT_PARITION_NAME}-1} \
        of=$MASTER_BOOT_RECORD bs=512 count=1 2>/dev/null
  FUNCTION_RESULT=$?

  # master boot record is not copied
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^32\s/!d; s/^\(32\s\)\(.*\)/\2/' $MESSAGES
    return 1
  else
    echo "<End copyMBR"
    return 0
  fi
}       # ----------  end of function copyMBR  ----------
```

```
# archives the data of the boot partition
archiveBootPartitionData ()
{
  echo ">Start archiveBootPartitionData"
  EXCLUDE=""        # excluded directories

  # go through special directories
  for dir in ${SPECIAL_DIRECTORIES[*]}
  do
    EXCLUDE+="--exclude $dir "
  done

  # make an archive of data excluded the special directories
  tar -czps --same-owner --atime-preserv $EXCLUDE -f $BOOT_PARTITION_DATA \
    -C ${MOUNT_LOCATION} . 2>/dev/null
  FUNCTION_RESULT=$?

  # archive is not made
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^33\s/!d; s/^\(33\s\)\(.*\)/\2/' $MESSAGES
    return 1
  else
    echo "<End archiveBootPartitionData"
    return 0
  fi
}       # ---------- end of function archiveBootPartitionData ----------


# archives the data of other partitions
archiveDataOtherPartitions ()
{
  echo ">Start archiveDataOtherPartitions"
  # there are some partitions
  if [ ${#PARTITION_POINT_ARRAY[@]} -eq 0 ]
  then
    sed '/^30\s/!d; s/^\(30\s\)\(.*\)/\2/' $MESSAGES
    sed '/^332\s/!d; s/^\(332\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  mkdir $LOCATION_PARTITION_DATA
  FUNCTION_RESULT=$?

  # directory is not made
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^38\s/!d; s/^\(38\s\)\(.*\)/\2/' $MESSAGES
    sed '/^332\s/!d; s/^\(332\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  for (( COUNTER = 0; COUNTER < ${#PARTITION_POINT_ARRAY[*]}; COUNTER++ ))
  do
    if [ $(($COUNTER % 2)) -eq 1 ]  # counter is odd
    then
      tar -czps --same-owner --atime-preserv \
        -f $LOCATION_PARTITION_DATA${PARTITION_POINT_ARRAY[$COUNTER]}.tar.gz \
        -C $MOUNT_LOCATION${PARTITION_POINT_ARRAY[$COUNTER]}/ . 2>/dev/null
      FUNCTION_RESULT=$?

      # archive is not made
      if [ $FUNCTION_RESULT -ne 0 ]
      then
        sed '/^331\s/!d; s/^\(331\s\)\(.*\)/\2/' $MESSAGES
        sed '/^332\s/!d; s/^\(332\s\)\(.*\)/\2/' $MESSAGES
        return 1
      fi
    fi
  done

  echo "<End archiveDataOtherPartitions"
  return 0
}       # ---------- end of function archiveDataOtherPartitions ----------


# asks user whether he wants to continue to make the clone
askUserToContinue ()
{
  sed '/^\(15\|16\)\s/!d; s/^\(\(15\|16\)\s\)\(.*\)/\3/' $MESSAGES
  read -e -r -p "$(sed '/^17\s/!d; s/^\(17\s\)\(.*\)/\2/' $MESSAGES) " \
    USER_INPUT

  # answer is not correct
  while [[ "$USER_INPUT" != "y" && "$USER_INPUT" != "n" ]]
  do
    printf "\033[1A\r\033[K"
    read -e -r -p "$(sed '/^18\s/!d; s/^\(18\s\)\(.*\)/\2/' $MESSAGES) " \
      USER_INPUT
  done

  if [[ "$USER_INPUT" = "y" ]]
  then
    return 0
  else
    return 1
  fi
}       # ---------- end of function askUserToContinue ----------


# makes a backup of the access control list of all directories and files
makeAccessControlList ()
{
  echo ">Start makeAccessControlList"
  cd $MOUNT_LOCATION 2>/dev/null     # goes to the directory
  FUNCTION_RESULT=$?

  # directory does not exist
  if [ $FUNCTION_RESULT -ne 0 ]
  then
```

```
    sed '/^49\s/!d; s/^\(49\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  EXCLUDE=""
  for dir in ${SPECIAL_DIRECTORIES[*]}
  do
    EXCLUDE+="${dir}|"       # make a list of excluded directories
  done

  # excluded directories from acl
  EXCLUDE="!(" `echo $EXCLUDE | sed 's/|$/)/'`

  shopt -s extglob
  getfacl -RP $EXCLUDE > $ACCESS_CONTROL_LIST   # make an access control list
  FUNCTION_RESULT=$?
  shopt -u extglob

  # getfacl has fired an error
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^49\s/!d; s/^\(49\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  cd $MOUNT_LOCATION
  echo "<End makeAccessControlList"
  return 0
}       # ----------  end of function makeAccessControlList  ----------
```

```
#!/usr/bin/env bash
#===============================================================================
#
#          FILE: writeClone
#
#         USAGE: Library
#
#   DESCRIPTION:
#
#       OPTIONS: ---
#  REQUIREMENTS: ---
#          BUGS: ---
#         NOTES: ---
#        AUTHOR: Mehdi Aqadjani Memar (-), m.aqadjanimemar@student.ru.nl
#  ORGANIZATION: -
#       CREATED: 05/07/2012 08:13:45 PM CEST
#      REVISION:  ---
#===============================================================================

# writes the clone to the hard disk
writeClone ()
{
  isCloneComplete     # function
  FUNCTION_RESULT=$?

  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^37\s/!d; s/^\(37\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  hasHardDiskEnoughSpace  # function
  FUNCTION_RESULT=$?

  # not enough space on the hard disk
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^37\s/!d; s/^\(37\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  writeMBR                  # function
  FUNCTION_RESULT=$?

  # master boot record is not saved
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^37\s/!d; s/^\(37\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  partprobe   # inform the os about partition changes

  makeFileSystems        # function
  FUNCTION_RESULT=$?

  # filesystem are not made
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^37\s/!d; s/^\(37\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi


  mountBootPartition        # function
  FUNCTION_RESULT=$?

  # boot partition is not mounted
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^37\s/!d; s/^\(37\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  writeDataBootPartition             # function
  FUNCTION_RESULT=$?
```

```
# data of the boot partition is not written
if [ $FUNCTION_RESULT -ne 0 ]
then
  sed '/^14\s/!d; s/^\(14\s\)\(.*\)/\2/' $MESSAGES
  return 1
fi

createSpecialDirectories          # function
FUNCTION_RESULT=$?
# special directories are not made
if [ $FUNCTION_RESULT -ne 0 ]
then
  sed '/^14\s/!d; s/^\(14\s\)\(.*\)/\2/' $MESSAGES
  return 1
fi

hasCloneMorePatitions          # function
FUNCTION_RESULT=$?
# clone has more than one partition
if [ $FUNCTION_RESULT -eq 0 ]
then
  setPartitionPointArray       # function
  FUNCTION_RESULT=$?
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^14\s/!d; s/^\(14\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  mountOtherPartitions          # function
  FUNCTION_RESULT=$?
  # partitions are not mounted
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^14\s/!d; s/^\(14\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  writeOtherPartitions          # function
  FUNCTION_RESULT=$?
  # partition data is not written
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^14\s/!d; s/^\(14\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

fi

setAccessControlList        # function
FUNCTION_RESULT=$?
if [ $FUNCTION_RESULT -ne 0 ]
then
  sed '/^14\s/!d; s/^\(14\s\)\(.*\)/\2/' $MESSAGES
  return 1
fi

updateGrub                # function
FUNCTION_RESULT=$?
# grub is not installed
if [ $FUNCTION_RESULT -ne 0 ]
then
  sed '/^14\s/!d; s/^\(14\s\)\(.*\)/\2/' $MESSAGES
  return 1
fi

return 0
}       # ---------- end of function writeClone ----------


# checks whether the clone exists and the content of the clone is complete
isCloneComplete ()
{
  echo ">Start isCloneComplete"
  existClone     # function
  FUNCTION_RESULT=$?

  if [ $FUNCTION_RESULT -ne 0 ] || [[ ! -f "$MASTER_BOOT_RECORD" || \
    ! -f "$BOOT_PARTITION_DATA" ]]
  then
    sed '/^39\s/!d; s/^\(39\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  hasCloneMorePatitions      # function
  FUNCTION_RESULT=$?

  if [[ -e "$LOCATION_PARTITION_DATA" ]] && [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^39\s/!d; s/^\(39\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  echo "<End isCloneComplete"
  return 0
}       # ---------- end of function isCloneComplete ----------


# checks whether the clone is made of more then one partition
hasCloneMorePatitions ()
{
  echo ">Start hasCloneMorePatitions"
  # directory of partition data exists
  if [[ -e "$LOCATION_PARTITION_DATA" ]]
  then
    cd $LOCATION_PARTITION_DATA 2>/dev/null
    # partition data do not exist and the output is a file
    if [ $( find -maxdepth 1 | grep -cv '^\.$' ) -eq 0 ]
    then
      cd $WORKING_DIRECTORY
```

```
      echo "<End hasCloneMorePatitions"
      return 1
    fi
  fi

  cd $WORKING_DIRECTORY
  echo "<End hasCloneMorePatitions"
  return 0
}       # ---------- end of function hasCloneMorePatitions ----------


# checks whether the hard disk has enough space for the clone
hasHardDiskEnoughSpace ()
{
  echo ">Start hasHardDiskEnoughSpace"
  set -o pipefail          # returns pipe errors
  DATA_SIZE=$(df /home | awk '$3 ~ /[0-9]+/ {print $3}')
  FUNCTION_RESULT=$?

  if [ $FUNCTION_RESULT -ne 0 ]    # an error has occurred
  then
    sed '/^42\s/!d; s/^\(42\s\)\(.*\)/\2/' $MESSAGES
    return 2
  fi

  HARD_DISK_SIZE=$(fdisk -l 2>/dev/null | awk -v condition=0 '/^Disk/ {
    if (condition == 0) {size=$5; condition=1}}
    END {print size}' 2>/dev/null)
  FUNCTION_RESULT=$?

  if [ $FUNCTION_RESULT -ne 0 ]    # an error has occurred
  then
    sed '/^41\s/!d; s/^\(41\s\)\(.*\)/\2/' $MESSAGES
    return 2
  fi
  set +o pipefail          # unset pipefail

  if [ $HARD_DISK_SIZE -gt $((DATA_SIZE * 2)) ]
  then
    echo "<End hasHardDiskEnoughSpace"
    return 0
  fi

  echo "<End hasHardDiskEnoughSpace"
  return 1
}       # ---------- end of function hasHardDiskEnoughSpace ----------


# writes the Master Boot Record to the hard disk
writeMBR ()
{
  echo ">Start writeMBR"
  setHardDiskName      # function
  FUNCTION_RESULT=$?

  # hard disk name is not set
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^44\s/!d; s/^\(44\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  dd if=$MASTER_BOOT_RECORD of=$HARD_DISK_NAME \
    bs=512 count=1 2>/dev/null
  FUNCTION_RESULT=$?

  # master boot record is not copied
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^44\s/!d; s/^\(44\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi
  echo "<End writeMBR"
  return 0
}       # ---------- end of function writeMBR ----------


# writes the data of the boot partition to the hard disk
writeDataBootPartition ()
{
  echo ">Start writeDataBootPartition"
  tar -zxps --same-owner --atime-preserv -f $BOOT_PARTITION_DATA -C \
    ${MOUNT_LOCATION} 2>/dev/null
  FUNCTION_RESULT=$?

  # data is not extracted in the boot partition
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^45\s/!d; s/^\(45\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  echo "<End writeDataBootPartition"
  return 0
}       # ---------- end of function writeDataBootPartition ----------


# creates special directories
createSpecialDirectories ()
{
  echo ">Start createSpecialDirectories"
  FUNCTION_RESULT=0
  for dir in ${SPECIAL_DIRECTORIES[@]}
  do
    # directory does not exist
    if [[ ! -e "${MOUNT_LOCATION}$dir" ]]
    then
      mkdir ${MOUNT_LOCATION}$dir    # make directory
      FUNCTION_RESULT=$?
```

```
      # an error has occurred
      if [ $FUNCTION_RESULT -ne 0 ]
      then
        sed '/^48\s/!d; s/^\(48\s\)\(.*\)/\2/' $MESSAGES
        return 1
      fi
    fi
  done

  echo "<End createSpecialDirectories"
  return 0
}       # ---------- end of function createSpecialDirectories ----------


# writes the content of partitions to the hard disk
writeOtherPartitions ()
{
  echo ">Start writeOtherPartitions"
  hasCloneMorePatitions     # function
  FUNCTION_RESULT=$?
  # there is no partition
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^30\s/!d; s/^\(30\s\)\(.*\)/\2/' $MESSAGES
    sed '/^47\s/!d; s/^\(47\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  cd $LOCATION_PARTITION_DATA
  FILES=$(ls -A)
  for FILE in $FILES
  do
    tar -zxps --same-owner --atime-preserv -f $FILE -C \
      $MOUNT_LOCATION${FILE%.tar*}/  2>/dev/null
    FUNCTION_RESULT=$?

    # content of a partition is not written
    if [ $FUNCTION_RESULT -ne 0 ]
    then
      sed '/^47\s/!d; s/^\(47\s\)\(.*\)/\2/' $MESSAGES
      return 1
    fi
  done

  echo "<End writeOtherPartitions"
  return 0
}       # ---------- end of function writeOtherPartitions ----------


# sets access control list of directories and files
setAccessControlList ()
{
  echo ">Start setAccessControlList"

  mkdir ${MOUNT_LOCATION}var/spool/cups/tmp
  cd $MOUNT_LOCATION 2>/dev/null    # goes to the mount directory
  FUNCTION_RESULT=$?

  # directory does not exist
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^50\s/!d; s/^\(50\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  setfacl --restore=$ACCESS_CONTROL_LIST
  FUNCTION_RESULT=$?

  # setfacl has fired an error
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^50\s/!d; s/^\(50\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  cd $MOUNT_LOCATION
  echo "<End setAccessControlList"
  return 0
}       # ---------- end of function setAccessControlList ----------


# sets the name of the hard disk
setHardDiskName ()
{
  echo ">Start setHardDiskName"
  set -o pipefail          # returns pipe errors

  # get the name of the boot partition
  HARD_DISK_NAME=$(fdisk -l 2>/dev/null | awk -v condition=0 '/^Disk/ {
    if (condition == 0) {name=substr($2,1,length($2)-1); condition=1}}
    END {print name}' 2>/dev/null)

  FUNCTION_RESULT=$?
  set +o pipefail          # unset pipefail

  if [ $FUNCTION_RESULT -ne 0 ]    # an error has occurred
  then
    sed '/^51\s/!d; s/^\(51\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  echo "<End setHardDiskName"
  return 0
}       # ---------- end of function setHardDiskName ----------


# makes partition filesystems
makeFileSystems ()
{
  echo ">Start makeFileSystems"
```

```
NEEDED_PARTITIONS=($(awk '/^\/dev\/sda/ {if ($3 ~ /ext4/) printf ( \
  "ext4 %s %s ", substr($2,7,length($2)-7), substr($1,1,length($1)-1));
  else if ($3 ~ /swap/) printf ("swap %s %s ", substr($2,7,length($2)-7), \
  substr($1,1,length($1)-1)); else if ($3 ~ /ext3/) printf (\
  "ext3 %s %s ", substr($2,7,length($2)-7), substr($1,1,length($1)-1));
  else if ($3 ~ /ext2/) printf ("ext2 %s %s ", substr($2,7,length($2)-7), \
  substr($1,1,length($1)-1))}' $BLKID_FILE))

for (( COUNTER=0; COUNTER < ${#NEEDED_PARTITIONS[@]}; COUNTER++ ))
do
  if [[ ${NEEDED_PARTITIONS[$COUNTER]} == "ext4" ]]
  then
    mkfs.ext4 -U ${NEEDED_PARTITIONS[$COUNTER+1]} \
      ${NEEDED_PARTITIONS[$COUNTER+2]} 2>/dev/null
    FUNCTION_RESULT=$?

    # filesystem is not made
    if [ $FUNCTION_RESULT -ne 0 ]
    then
      sed '/^52\s/!d; s/^\(52\s\)\(.*\)/\2/' $MESSAGES
      return 1
    fi

  elif [[ ${NEEDED_PARTITIONS[$COUNTER]} == "ext3" ]]
  then
    mkfs.ext3 -U ${NEEDED_PARTITIONS[$COUNTER+1]} \
      ${NEEDED_PARTITIONS[$COUNTER+2]} 2>/dev/null
    FUNCTION_RESULT=$?

    # filesystem is not made
    if [ $FUNCTION_RESULT -ne 0 ]
    then
      sed '/^52\s/!d; s/^\(52\s\)\(.*\)/\2/' $MESSAGES
      return 1
    fi

  elif [[ ${NEEDED_PARTITIONS[$COUNTER]} == "ext2" ]]
  then
    mkfs.ext2 -U ${NEEDED_PARTITIONS[$COUNTER+1]} \
      ${NEEDED_PARTITIONS[$COUNTER+2]} 2>/dev/null
    FUNCTION_RESULT=$?

    # filesystem is not made
    if [ $FUNCTION_RESULT -ne 0 ]
    then
      sed '/^52\s/!d; s/^\(52\s\)\(.*\)/\2/' $MESSAGES
      return 1
    fi

  elif [[ ${NEEDED_PARTITIONS[COUNTER]} == "swap" ]]
  then
    mkswap -U ${NEEDED_PARTITIONS[$COUNTER+1]} \
      ${NEEDED_PARTITIONS[$COUNTER+2]} 2>/dev/null
    FUNCTION_RESULT=$?

    # filesystem is not made
    if [ $FUNCTION_RESULT -ne 0 ]
    then
      sed '/^52\s/!d; s/^\(52\s\)\(.*\)/\2/' $MESSAGES
      return 1
    fi
  fi
done

echo "<End makeFileSystems"
return 0
}       # ---------- end of function makeFileSystems ----------


# installs Grub
updateGrub ()
{
  echo ">Start updateGrub"
  GRUB_DIRECTORIES=("dev" "proc" "sys")
  mountGrubDirectories    # function
  FUNCTION_RESULT=$?
  # directories are mounted
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^55\s/!d; s/^\(55\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  chroot $MOUNT_LOCATION grub-install $HARD_DISK_NAME
  FUNCTION_RESULT=$?
  # working directory is not chrooted
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^55\s/!d; s/^\(55\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  unmountGrubDirectories        # function
  FUNCTION_RESULT=$?
  # grub directories can not be unmounted
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^56\s/!d; s/^\(56\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  echo "<End updateGrub"
  return 0
}       # ---------- end of function updateGrub ----------


# mounts some directories to install Grub
mountGrubDirectories ()
{
  echo ">Start mountGrubDirectories"
```

```
  for dir in ${GRUB_DIRECTORIES[@]}
  do
    mount --bind /$dir $MOUNT_LOCATION$dir 2>/dev/null
    FUNCTION_RESULT=$?
    # directory is not mounted
    if [ $FUNCTION_RESULT -ne 0 ]
    then
      sed '/^53\s/!d; s/^\(53\s\)\(.*\)/\2/' $MESSAGES
      return 1
    fi
  done

  echo "<End mountGrubDirectories"
  return 0
}       # ----------  end of function mountGrubDirectories  ----------


# unmounts directories which were mounted to install Grub
unmountGrubDirectories ()
{
  echo ">Start unmountGrubDirectories"
  for dir in ${GRUB_DIRECTORIES[@]}
  do
    umount $MOUNT_LOCATION$dir 2>/dev/null
    FUNCTION_RESULT=$?
    # directory is not mounted
    if [ $FUNCTION_RESULT -ne 0 ]
    then
      sed '/^54\s/!d; s/^\(54\s\)\(.*\)/\2/' $MESSAGES
      return 1
    fi
  done

  echo "<End unmountGrubDirectories"
  return 0
}       # ----------  end of function unmountGrubDirectories  ----------
```

```
#!/usr/bin/env bash
#===============================================================================
#
#          FILE: sharedLib
#
#         USAGE: Library
#
#   DESCRIPTION:
#
#       OPTIONS: ---
#  REQUIREMENTS: ---
#          BUGS: ---
#         NOTES: ---
#        AUTHOR: Mehdi Aqadjani Memar (-), m.aqadjanimemar@student.ru.nl
#  ORGANIZATION: -
#       CREATED: 05/06/2012 11:33:12 PM CEST
#      REVISION: ---
#===============================================================================


# debug function
startProgramme ()
{
  clear
  stty -echo
  read -e -r -p "$(sed '/^36\s/!d; s/^\(36\s\)\(.*\)/\2/' $MESSAGES)" input
  echo $input | sha1sum | awk '{print $1}' > ${WORKING_DIRECTORY}pass
  stty echo
}       # ----------  end of function startProgramme  ----------


# prints options the user may choose and returns the user's input
getOption ()
{
 read -e -r -p "$(sed '/^9\s/!d; s/^\(9\s\)\(.*\)/\2/' $MESSAGES)" input

 if [ "$input" = "0" ]
 then
   return 0
 elif [ "$input" = "1" ]
 then
   return 1
 elif [ "$input" = "2" ]
 then
   return 2
 else
   return 3
 fi
}       # ----------  end of function getOption  ----------


# unmounts all partitions except boot partition
unmountOtherPartitions ()
{
  echo ">Start unmountOtherPartitions"
  # only one partition
  if [ ${#PARTITION_POINT_ARRAY[@]} -gt 0 ]
  then
    for (( COUNTER = 0; COUNTER < ${#PARTITION_POINT_ARRAY[*]}; COUNTER++ ))
    do
      if [ $(($COUNTER % 2)) -eq 1 ]  # counter is odd
      then
        umount $MOUNT_LOCATION${PARTITION_POINT_ARRAY[$COUNTER]} \
          2>/dev/null

        FUNCTION_RESULT=$?

        # a partion is not mounted
        if [ $FUNCTION_RESULT -ne 0 ]
```

```
        then
          sed '/^221\s/!d; s/^\(221\s\)\(.*\)/\2/' $MESSAGES
          return 1
        fi
      fi
    done
  fi

  echo "<End unmountOtherPartitions"
  return 0
}       # ---------- end of function unmountOtherPartitions ----------


# unmounts all partitions
unmountPartitions ()
{
  echo ">Start unmountPartitions"
  unmountOtherPartitions          # function
  FUNCTION_RESULT=$?              # result of evaluated function

  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^251\s/!d; s/^\(251\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  umount $MOUNT_LOCATION 2>/dev/null
  FUNCTION_RESULT=$?              # result of the command

  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^251\s/!d; s/^\(251\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  echo "<End unmountPartitions"
  return 0
}       # ---------- end of function unmountPartitions ----------


# mounts the boot partition
mountBootPartition ()
{
  echo ">Start mountBootPartition"
  setBootPartitionName            # function
  FUNCTION_RESULT=$?              # result of evaluated function

  # boot partition is not set
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^25\s/!d; s/^\(25\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  makeMountPoint                  # function
  FUNCTION_RESULT=$?              # result of evaluated function

  # mount location is not made
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^25\s/!d; s/^\(25\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  # mount boot partition
  mount $BOOT_PARITION_NAME $MOUNT_LOCATION 2>/dev/null
  RESULT=$?

  # boot partition is not mounted
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^25\s/!d; s/^\(25\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  echo "<End mountBootPartition"
  return 0
}       # ---------- end of function mountBootPartition ----------


# set the name of boot partition
setBootPartitionName ()
{
  echo ">Start setBootPartitionName"
  set -o pipefail          # returns pipe errors

  # get the name of the boot partition
  BOOT_PARITION_NAME=$(parted -l 2>/dev/null | awk '/^Model/ {
      condition=/ATA|IDE/} condition {
      if (/^Disk/) name = substr($2,1,length($2)-1)}
      condition {if (/boot$/) number = $1}
      END {print name number}' 2>/dev/null)

  FUNCTION_RESULT=$?
  set +o pipefail          # unset pipefail

  if [ $FUNCTION_RESULT -ne 0 ]    # an error has occurred
  then
    sed '/^21\s/!d; s/^\(21\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  echo "<End setBootPartitionName"
  return 0
}       # ---------- end of function setBootPartitionName ----------


# mounts all partitions except boot partition
mountOtherPartitions ()
{
```

```
  echo ">Start mountOtherPartitions"
  # there are not any other partition except boot partition
  if [ ${#PARTITION_POINT_ARRAY[@]} -eq 0 ]
  then
    sed '/^30\s/!d; s/^\(30\s\)\(.*\)/\2/' $MESSAGES
    sed '/^22\s/!d; s/^\(22\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  # go through all partition
  for (( COUNTER = 0; COUNTER < ${#PARTITION_POINT_ARRAY[*]}; COUNTER++ ))
  do
    # is counter even
    if [ $(($COUNTER % 2)) -eq 0 ]
    then
      mount ${PARTITION_POINT_ARRAY[$COUNTER]} \
        $MOUNT_LOCATION${PARTITION_POINT_ARRAY[$COUNTER + 1]} \
        2>/dev/null
      FUNCTION_RESULT=$?

      # a partion is not mounted
      if [ $FUNCTION_RESULT -ne 0 ]
      then
        sed '/^22\s/!d; s/^\(22\s\)\(.*\)/\2/' $MESSAGES
        return 1
      fi
    fi
  done

  echo "<End mountOtherPartitions"
  return 0
}       # ---------- end of function mountOtherPartitions ----------


# sets an array of partitions and their mount location
setPartitionPointArray ()
{
  echo ">Start setPartitionPointArray"
  # file containing mount info
  FSTAB_FILE=${MOUNT_LOCATION}etc/fstab
  local StartID=7         # start of uuid in the result blkid
  set -o pipefail         # returns pipe errors

  # set the array of partition names and their mount points
  PARTITION_POINT_ARRAY=($(blkid  2>/dev/null |
    awk -v start=$StartID -v counter=0 'FNR == NR {
      for (i = 1; i <= NF; i++) {
        if ($i ~ "UUID") {
          counter++; uuida[counter] = substr($i, 7, length($i) - start - 1);
          namea[counter]=$1;
        }
      }
    next }
    length ($1) > 20 {
      for (i=1; i <= counter; i++) {
        if ($2 != "/" && $2 != "none" && $0 ~ uuida[i]) {
          print substr (namea[i], 1, length(namea[i]) - 1), \
          substr ($2, 2, length($2)-1)
        }
      }
    }' - $FSTAB_FILE  2>/dev/null))

  FUNCTION_RESULT=$?
  set +o pipefail         # unsets pipefail

  # array is not set
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^23\s/!d; s/^\(23\s\)\(.*\)/\2/' $MESSAGES
    return 1
  else
    echo "<End setPartitionPointArray"
    return 0
  fi
}       # ---------- end of function setPartitionPointArray ----------


# makes a mount point on the storage medium
makeMountPoint ()
{
  echo ">Start makeMountPoint"
  # mount point exists
  if [ -e $MOUNT_LOCATION ]
  then
    COUNTER=$(ls -1 $MOUNT_LOCATION | wc -l)

    # mount point is not empty
    if [ $COUNTER -ne 0 ]
    then
      sed '/^24\s/!d; s/^\(24\s\)\(.*\)/\2/' $MESSAGES
      return 1
    else       # mount point is empty
      echo "<End makeMountPoint"
      return 0
    fi

  fi

  mkdir $MOUNT_LOCATION
  FUNCTION_RESULT=$?

  # mount point is not made
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^34\s/!d; s/^\(34\s\)\(.*\)/\2/' $MESSAGES
    return 1
  fi

  echo "<End makeMountPoint"
  return 0
```

```
}       # ----------  end of function makeMountPoint  ----------


# checks whether the clone already exists
existClone ()
{
  if [[ -e "$CLONE_LOCATION" ]]
  then
    return 0
  fi
  return 1
}       # ----------  end of function existClone  ----------


# checks whether the hard disk has more than one partition
hasHardDiskMorePartitions ()
{
  echo ">Start hasHardDiskMorePartitions"
  setPartitionPointArray            # function
  FUNCTION_RESULT=$?
  # partitions are not mounted
  if [ $FUNCTION_RESULT -ne 0 ]
  then
    sed '/^57\s/!d; s/^\(57\s\)\(.*\)/\2/' $MESSAGES
    return 2
  fi

  echo "<End hasHardDiskMorePartitions"
  if [ ${#PARTITION_POINT_ARRAY[@]} -eq 0 ]
  then
    return 1
  fi

  return 0
}       # ----------  end of function hasHardDiskMorePartitions  ----------
```

```
#!/usr/bin/env bash

### BEGIN INIT INFO
# Provides: stopSystem
# Required-Start: hardwareIndependent
# Required-Stop:
# Default-Start: 1 2 3 4 5
# Default-Stop:
# Short-Description: Shutting down the computer
# Description: Lets debugging the software
### END INIT INFO

#===============================================================================
#
#          FILE: stopSystem
#
#         USAGE: ./stopSystem
#
#   DESCRIPTION: This script let the computer powers on if the correct password
#    is inserted; otherwise the computer shuts down.
#
#       OPTIONS: ---
#  REQUIREMENTS: ---
#          BUGS: ---
#         NOTES: ---
#        AUTHOR: Mehdi Aqadjani Memar (-), m.aqadjanimemar@student.ru.nl
#  ORGANIZATION: -
#       CREATED: 05/15/2012 11:58:24 AM CEST
#      REVISION:  ---
#===============================================================================

PASSWORD="0cbe3cfc1ef90c019c4b8d05bc32efab07cfe8db"        # sha1 hashed password
WORKING_DIRECTORY="/home/user/"       # directory in which the script writes

# file doesn't exist
if [ ! -f ${WORKING_DIRECTORY}pass ]
then
  shutdown -h now
fi

pass=$(awk '{print}' ${WORKING_DIRECTORY}pass)
rm ${WORKING_DIRECTORY}pass

# password is wrong
if [ "$pass" != "$PASSWORD" ]
then
  shutdown -h now
fi

exit 0
```

Referenties

[1] Keith Adams en Ole Agesen. „A Comparison of Software and Hardware Techniques for x86 Virtualization". In: *AMC* Proceedings of the 12th international conference on Architectural support for programming languages and operating systems.ASPLOS-XII (2006), pages 2–13.

[2] AMD. *AMD Virtualization*. Accessed on 12 March 2012. URL: http://sites.amd.com/us/business/it-solutions/virtualization/Pages/virtualization.aspx.

[3] Paul Barham e.a. „Xen and the Art of Virtualization". In: *AMC Press* Proceedings of the nineteenth ACM symposium on Operating Systems Principles.SOSP19 (2003), p. 164–177.

[4] StorageCraft Technology Corporation. *Hardware Independent Restore*. Accessed on 4 March 2012. URL: http://www.storagecraft.com/kb/questions.php?questionid=305.

[5] Debian. *Debian Live*. Accessed on 25 March 2012. URL: http://live-manual.debian.net/manual-2.x/html/live-manual.en.html.

[6] Andreas Grünbacher. „POSIX Access Control Lists on Linux". In: Proceedings of the Annual USENIX Technical Conference, San Antonio, Texas (June 2003), p. 259–272.

[7] Acronis Inc. *Acronis True Image*. Accessed on 4 March 2012. URL: http://www.acronis.com/download/docs/atih2012/userguide.

[8] INTEL. *Hardware-Assisted Virtualization Technology*. Accessed on 12 March 2012. URL: http://www.intel.com/content/www/us/en/virtualization/virtualization-technology/hardware-assist-virtualization-technology.html.

[9] Arche Linux. *Master Boot Record*. Accessed on 3 May 2012. URL: https://wiki.archlinux.org/index.php/Master_Boot_Record.

[10] Inc. VMware. *VMware*. Accessed on 2 March 2012. URL: http://www.vmware.com.

[11] Inc. VMware. *VMware ESXi 5.0*. Accessed on 2 March 2012. URL: http://www.vmware.com/products/vsphere/esxi-and-esx/compare.html.

[12] Xen. *What is Xen?* Accessed on 27 February 2012. URL: http://xen.org/files/Marketing/WhatisXen.pdf.

Glossary

blkid Een programma dat partitienamen, “UUIDs” en *filesystems* van *block devices* (harde schijf, *USB-sticks* en *CD Roms*), print. 18

bz2 Een implementatie van Burrows-Wheeler algoritme om mappen en bestanden te comprimeren. 20

cp Het traditionele kopieprogramma. 20

dd Een kopie programma dat de inhoud van gewenste sectors op de harde schijf in een bestand kan kopiëren en de inhoud van zo’n bestand op de gewenste sectors op de harde terug kan schrijven. 20

fdisk Een programma waarmee men de partitie-tabel van een harde schijf manipuleert. 9

gz Een *open source* programma om mappen en bestanden te comprimeren. 20

Master Boot Record De eerste 512 bytes van een opslagmedium waar informatie over de *bootloader* en de partitie-table van de harde schijf opgeslagen zijn. 19

mkfs Een programma voor het maken van verschillende filesystems op partities. 19

parted Een programma waarmee de gebruiker de partitie-tabel van een harde schijf kan manipuleren. 17

sfdisk Het doet hetzelfde als “fdisk” met het verschil dat de input uit een bestand gelezen mag worden. 17

tar Een programma waarmee men verschillende bestanden en mappen samen kan binden. Het geeft ook de mogelijkheid om samen gebonden data te comprimeren. 9

UUID Een unieke id die meer flexibiliteit biedt bij het herkennen van een harde schijf dan de oude methode waar de harde schijf namen werden gebruikt. 17