



Red Hat Enterprise Linux 8

커널 모니터링 및 업데이트 관리

Red Hat Enterprise Linux 8에서 Linux 커널 관리에 대한 가이드

Red Hat Enterprise Linux 8 커널 모니터링 및 업데이트 관리

Red Hat Enterprise Linux 8에서 Linux 커널 관리에 대한 가이드

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

법적 공지

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Managing_monitoring_and Updating_the_kernel.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution-Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

초록

이 문서에서는 사용자와 관리자에게 Linux 커널 수준에서 워크스테이션을 구성하는 데 필요한 정보를 제공합니다. 이러한 조정으로 성능 향상, 더 쉬운 문제 해결 또는 시스템 최적화가 가능합니다.

차례

보다 포괄적 수용을 위한 오픈 소스 용어 교체	6
RED HAT 문서에 관한 피드백 제공	7
1장. LINUX 커널 RPM	8
1.1. RPM이란	8
RPM 패키지 유형	8
1.2. LINUX 커널 RPM 패키지 개요	8
1.3. KERNEL 패키지의 내용 표시	9
2장. YUM으로 커널 업데이트	11
2.1. 커널이란 무엇입니까?	11
2.2. YUM이란 무엇입니까?	11
2.3. 커널 업데이트	11
2.4. 커널 설치	11
3장. 커널 모듈 관리	13
3.1. 커널 모듈 소개	13
3.2. 부트 로더 사양 소개	13
3.3. 커널 모듈 종속성	14
3.4. 현재 로드된 커널 모듈 나열	14
3.5. 설치된 모든 커널 나열	15
3.6. 커널을 기본값으로 설정	15
3.7. 커널 모듈에 대한 정보 표시	16
3.8. 시스템 런타임 시 커널 모듈 로드	17
3.9. 시스템 런타임에서 커널 모듈 언로드	18
3.10. 부팅 프로세스의 초기 단계에서 커널 모듈 언로드 해제	19
3.11. 시스템 부팅 시 커널 모듈 자동 로드	20
3.12. 시스템 부팅 시 커널 모듈이 자동으로 로드되지 않도록 방지	21
3.13. 사용자 정의 커널 모듈 컴파일	23
4장. 보안 부팅을 위한 커널 모듈 서명	26
4.1. UEFI SECURE BOOT이란 무엇입니까?	26
4.2. UEFI SECURE BOOT 지원	27
4.3. X.509 키를 사용하여 커널 모듈을 인증하기 위한 요구 사항	28
4.4. 공개 키 소스	29
4.5. 공개 및 개인 키 쌍 생성	30
4.6. 시스템 인증 키의 출력 예	31
4.7. 공개 키를 MOK 목록에 추가하여 대상 시스템에 공개 키 등록	32
4.8. 개인 키로 커널 모듈 서명	33
4.9. 서명된 커널 모듈 로드	34
5장. 커널 명령줄 매개변수 구성	36
5.1. 커널 명령줄 매개변수 이해	36
5.2. GRUBBY의 정의	36
5.3. 어떤 부팅 항목	36
5.4. 모든 부팅 항목의 커널 명령줄 매개변수 변경	37
5.5. 단일 부팅 항목의 커널 명령줄 매개변수 변경	38
5.6. 부팅 시 일시적으로 커널 명령줄 매개변수 변경	39
5.7. 직렬 콘솔 연결을 사용하도록 GRUB 2 설정 설정	40
6장. 런타임 시 커널 매개변수 구성	41
6.1. 커널 매개변수란 무엇입니까?	41

6.2. SYSCTL을 사용하여 일시적으로 커널 매개변수 구성	42
6.3. SYSCTL을 사용하여 영구적으로 커널 매개변수 구성	42
6.4. /ETC/SYSCTL.D/의 구성 파일을 사용하여 커널 매개변수 조정	43
6.5. /PROC/SYS/를 통해 일시적으로 커널 매개변수 구성	44
7장. 가상화 환경에서 커널 패닉 매개변수를 비활성화 상태로 유지	45
7.1. 소프트 잠금이란 무엇입니까	45
7.2. 커널 패닉을 제어하는 매개변수	45
7.3. 가상 환경에서 끊기고 있는 소프트 잠금	46
8장. 데이터베이스 서버의 커널 매개변수 조정	47
8.1. 데이터베이스 서버 소개	47
8.2. 데이터베이스 애플리케이션 성능에 영향을 주는 매개변수	47
9장. 커널 로깅 시작하기	49
9.1. 커널 링 버퍼란 무엇입니까	49
9.2. 로그 수준 및 커널 로깅에 대한 PRINTK 역할	49
10장. KDUMP 설치	51
10.1. KDUMP란 무엇입니까?	51
10.2. ANACONDA를 사용하여 KDUMP 설치	51
10.3. 명령줄에 KDUMP 설치	52
11장. 명령줄에서 KDUMP 구성	53
11.1. KDUMP 크기 예상	53
11.2. KDUMP 메모리 사용량 구성	53
11.3. KDUMP 대상 구성	55
11.4. KDUMP 코어 수집기 구성	57
11.5. KDUMP 기본 오류 응답 구성	58
11.6. KDUMP의 설정 파일	59
11.7. KDUMP 설정 테스트	60
12장. KDUMP 활성화	63
12.1. 설치된 모든 커널에 KDUMP 활성화	63
12.2. 설치된 특정 커널의 KDUMP 활성화	63
12.3. KDUMP 서비스 비활성화	64
13장. 웹 콘솔에서 KDUMP 구성	67
13.1. 웹 콘솔에서 KDUMP 메모리 사용량 및 대상 위치 구성	67
13.2. 추가 리소스	69
14장. 지원되는 KDUMP 구성 및 대상	70
14.1. KDUMP의 메모리 요구 사항	70
14.2. 자동 메모리 예약의 최소 임계값	71
14.3. 지원되는 KDUMP 대상	72
14.4. 지원되는 KDUMP 필터링 수준	73
14.5. 지원되는 기본 오류 응답	74
14.6. FINAL_ACTION 매개변수 사용	74
15장. 펌웨어 지원 덤프 메커니즘	76
15.1. IBM POWERPC 하드웨어에서 펌웨어 지원 덤프	76
15.2. 펌웨어 지원 덤프 메커니즘 활성화	76
15.3. IBM Z 하드웨어에서 펌웨어 지원 덤프 메커니즘	77
15.4. FUJITSU PRIMEQUEST 시스템에서 IRONUMP 사용	78
16장. 코어 덤프 분석	80

16.1. CRASH 유틸리티 설치	80
16.2. 크래시 유틸리티 실행 및 종료	80
16.3. CRASH 유틸리티에 다양한 표시기 표시	82
16.4. 커널 OOPS 분석기 사용	85
16.5. KDUMP 도우미 도구	86
17장. EARLY KDUMP를 사용하여 부팅 시간 충돌 캡처	88
17.1. EARLY KDUMP이란 무엇입니까?	88
17.2. 초기 KDUMP 활성화	88
18장. 커널 라이브 패치로 패치 적용	90
18.1. KPATCH의 제한 사항	90
18.2. 타사 라이브 패칭 지원	90
18.3. 커널 라이브 패치에 액세스	91
18.4. 커널 라이브 패치 구성 요소	91
18.5. 커널 라이브 패치 작동 방법	92
18.6. 현재 설치된 커널을 라이브 패치 스트림에 가입	93
18.7. 라이브 패치 스트림에 향후 커널을 자동으로 등록	95
18.8. 라이브 패치 스트림에 대한 자동 서브스크립션 비활성화	97
18.9. 커널 패치 모듈 업데이트	98
18.10. 라이브 패치 패키지 제거	99
18.11. 커널 패치 모듈 설치 제거	100
18.12. KPATCH.SERVICE 비활성화	102
19장. ZSWAP을 사용하여 시스템 성능 개선	105
19.1. ZSWAP이란	105
19.2. 런타임 시 ZSWAP 활성화	105
19.3. ZSWAP을 영구적으로 활성화	106
20장. 애플리케이션에 대한 제한 설정	108
20.1. 제어 그룹 이해	108
20.2. 커널 리소스 컨트롤러란 무엇입니까?	109
20.3. 네임스페이스의 정의	111
20.4. CGROUPS-V1을 사용하여 CPU 제한 설정	112
21장. CGROUPS-V2를 사용하여 애플리케이션의 CPU 시간 분배 제어	118
21.1. CGROUPS-V2 마운트	118
21.2. CPU 시간 분배를 위해 CGROUP 준비	120
21.3. CPU 대역폭을 조정하여 애플리케이션 CPU 시간 분배 제어	124
21.4. CPU 가중치를 조정하여 애플리케이션의 CPU 시간 분배 제어	127
22장. SYSTEMD에서 제어 그룹 버전 1 사용	131
22.1. 제어 그룹 버전 1에서 SYSTEMD 역할	131
22.2. 일시적인 제어 그룹 생성	133
22.3. 영구 제어 그룹 생성	134
22.4. 명령줄에서 메모리 리소스 제어 설정 구성	135
22.5. 장치 파일로 메모리 리소스 제어 설정 구성	136
22.6. 일시적인 제어 그룹 제거	138
22.7. 영구 제어 그룹 제거	139
22.8. SYSTEMD 장치 나열	140
22.9. SYSTEMD 제어 그룹 계층 구조 보기	142
22.10. 리소스 컨트롤러 보기	143
22.11. 리소스 사용량 모니터링	144
23장. SYSTEMD와 함께 CGROUP 버전 2를 사용하여 리소스 관리 구성	146

23.1. 사전 요구 사항	146
23.2. 리소스 배포 모델 소개	146
23.3. SYSTEMD를 사용하여 CPU 리소스 할당	147
23.4. SYSTEMD의 CPU 시간 할당 정책 옵션	148
23.5. SYSTEMD를 사용하여 메모리 리소스 할당	149
23.6. SYSTEMD의 메모리 할당 구성 옵션	150
23.7. SYSTEMD를 사용하여 I/O 대역폭 구성	151
23.8. SYSTEMD에 대한 I/O 대역폭 구성 옵션	152
23.9. SYSTEMD를 사용하여 CPUSET 컨트롤러 구성	153
24장. SYSTEMD를 사용하여 CPU 유사성 및 NUMA 정책 구성	155
24.1. SYSTEMD를 사용하여 CPU 선호도 구성	155
24.2. SYSTEMD를 사용하여 NUMA 정책 구성	156
24.3. SYSTEMD의 NUMA 정책 구성 옵션	158
24.3.1. BPF Compiler Collection을 사용하여 시스템 성능 분석	159
24.3.1.1. BCC 소개	159
24.3.1.2. bcc-tools 패키지 설치	159
24.3.1.3. 선택한 bcc-tools를 성능 분석에 사용	160
execsnoop를 사용하여 시스템 프로세스 검사	160
opensnoop를 사용하여 명령이 여는 파일을 추적합니다.	161
biotop을 사용하여 디스크의 I/O 작업 검사	163
예기치 않게 느린 파일 시스템 작업 노출을 위해 xfsslower 사용	164
24.3.2. 커널 무결성 하위 시스템을 사용하여 보안 강화	166
24.3.2.1. 커널 무결성 하위 시스템	166
24.3.2.2. 무결성 측정 아키텍처	168
24.3.2.3. 확장 검증 모듈	168
24.3.2.4. 신뢰할 수 있는 암호화된 키	169
24.3.2.5. 신뢰할 수 있는 키 작업	170
24.3.2.6. 암호화된 키 작업	171
24.3.2.7. 무결성 측정 아키텍처 및 확장 검증 모듈 활성화	173
24.3.2.8. 무결성 측정 아키텍처를 사용하여 파일 해시 수집	177
24.3.3. Ansible 역할을 사용하여 커널 매개 변수 영구 구성	179
24.3.3.1. 커널 설정 역할 소개	179
24.3.3.2. 커널 설정 역할을 사용하여 선택한 커널 매개변수 적용	180

보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 [CTO Chris Wright의 메시지](#)를 참조하십시오.

RED HAT 문서에 관한 피드백 제공

문서 개선을 위한 의견을 보내 주십시오. 문서를 개선하는 방법을 알려주십시오.

- 특정 문구에 대한 간단한 의견 작성 방법은 다음과 같습니다.
 1. 문서가 *Multi-page HTML* 형식으로 표시되는지 확인합니다. 또한 문서 오른쪽 상단에 **피드백** 버튼이 있는지 확인합니다.
 2. 마우스 커서를 사용하여 주석 처리하려는 텍스트 부분을 강조 표시합니다.
 3. 강조 표시된 텍스트 아래에 표시되는 **피드백 추가** 팝업을 클릭합니다.
 4. 표시된 지침을 따릅니다.
- Bugzilla를 통해 피드백을 제출하려면 새 티켓을 생성합니다.
 1. [Bugzilla](#) 웹 사이트로 이동하십시오.
 2. Component로 **Documentation**을 선택하십시오.
 3. **Description** 필드에 문서 개선을 위한 제안 사항을 기입하십시오. 관련된 문서의 해당 부분 링크를 알려주십시오.
 4. **Submit Bug**를 클릭하십시오.

1장. LINUX 커널 RPM

다음 섹션에서는 Red Hat에서 제공 및 유지 관리하는 Linux 커널 RPM 패키지에 대해 설명합니다.

1.1. RPM이란

RPM 패키지는 다른 파일과 해당 메타데이터를 포함하는 파일입니다(시스템에 필요한 파일에 대한 정보).

특히 RPM 패키지는 **cpio** 아카이브로 구성됩니다.

The **cpio** 아카이브에는 다음이 포함됩니다.

- 파일
- RPM 헤더(패키지 메타데이터)
rpm 패키지 관리자는 이 메타데이터를 사용하여 종속성, 파일 설치 위치 및 기타 정보를 결정합니다.

RPM 패키지 유형

RPM 패키지는 다음 두 가지 유형이 있습니다. 두 유형 모두 파일 형식과 도구를 공유하지만 콘텐츠가 다르고 용도가 다릅니다.

- 소스 RPM(SRPM)
SRPM에는 소스 코드를 바이너리 RPM으로 빌드하는 방법을 설명하는 소스 코드와 SPEC 파일이 포함되어 있습니다. 소스 코드에 대한 패치도 포함되어 있습니다.
- 바이너리 RPM
바이너리 RPM에는 소스와 패치에서 빌드된 바이너리가 포함되어 있습니다.

1.2. LINUX 커널 RPM 패키지 개요

kernel RPM은 파일이 포함되지 않고 다음과 같은 필수 하위 패키지가 제대로 설치되었는지 확인하는 메타 패키지입니다.

- **kernel-core** - 커널의 바이너리 이미지, 시스템을 부트스트랩하기 위한 모든 initramfs 관련 개체 및 핵심 기능을 보장하는 최소 커널 모듈 수를 포함합니다. 이 하위 패키지만으로는 가상화 및 클라우드 환경에서 Red Hat Enterprise Linux 8 커널에 빠른 부팅 시간과 작은 디스크 크기 공간을 제공할 수 있습니다.
- **kernel-modules** - kernel **-core**에 없는 나머지 커널 모듈을 포함합니다.

위의 작은 커널 하위 패키지 집합은 특히 가상화 및 클라우드 환경에서 시스템 관리자에게 유지 관리 면적을 제공하기 위한 것입니다.

커널 패키지는 다음과 같습니다.

- **kernel-modules-extra** - 드물게 하드웨어를 위한 커널 모듈과 로드가 기본적으로 비활성화된 모듈이 포함되어 있습니다.
- **kernel-debug** - 성능 저하를 위해 커널 진단을 위해 수많은 디버깅 옵션이 활성화된 커널이 포함되어 있습니다.
- **kernel-tools** - Linux 커널 조작 및 지원 설명서를 위한 도구가 포함되어 있습니다.

- **kernel-devel** - 커널 헤더를 포함하고 커널 패키지에 대해 모듈을 빌드하기에 충분한 makefile을 포함합니다.
- **kernel-abi-stablelists** - RHEL 커널 ABI와 관련된 정보를 포함합니다. 여기에는 외부 Linux 커널 모듈에 필요한 커널 기호 목록과 적용을 지원하는 **yum** 플러그인의 목록이 포함되어 있습니다.
- **kernel-headers** - Linux 커널과 사용자 공간 라이브러리와 프로그램 간의 인터페이스를 지정하는 C 헤더 파일을 포함합니다. 헤더 파일은 대부분의 표준 프로그램 구축에 필요한 구조와 상수를 정의합니다.

추가 리소스

- [kernel-core, kernel-modules 및 kernel-modules-extras 패키지는 무엇입니까?](#)

1.3. KERNEL 패키지의 내용 표시

다음 절차에서는 **rpm** 명령을 사용하여 설치하지 않고 kernel 패키지 및 해당 하위 패키지의 내용을 보는 방법을 설명합니다.

사전 요구 사항

- CPU 아키텍처를 위한 **kernel**, **kernel-core**, **kernel-modules**, **kernel-modules-extra** RPM 패키지 획득

절차

- 커널의 모듈을 나열:

```
$ rpm -qlp <kernel_rpm>
(contains no files)
...
```

- 커널 코어의 모듈 나열:

```
$ rpm -qlp <kernel-core_rpm>
...
/lib/modules/4.18.0-80.el8.x86_64/kernel/fs/udf/udf.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/kernel/fs/xfs
/lib/modules/4.18.0-80.el8.x86_64/kernel/fs/xfs/xfs.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/kernel/kernel
/lib/modules/4.18.0-80.el8.x86_64/kernel/kernel/trace
/lib/modules/4.18.0-80.el8.x86_64/kernel/kernel/trace/ring_buffer_benchmark.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/kernel/lib
/lib/modules/4.18.0-80.el8.x86_64/kernel/lib/cordic.ko.xz
...
```

- **kernel-modules**의 모듈 나열:

```
$ rpm -qlp <kernel-modules_rpm>
...
/lib/modules/4.18.0-80.el8.x86_64/kernel/drivers/infiniband/hw/mlx4/mlx4_ib.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/kernel/drivers/infiniband/hw/mlx5/mlx5_ib.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/kernel/drivers/infiniband/hw/qedr/qedr.ko.xz
/lib/modules/4.18.0-80.el8.x86_64/kernel/drivers/infiniband/hw/usnic/usnic_verbs.ko.xz
```

```
/lib/modules/4.18.0-  
80.el8.x86_64/kernel/drivers/infiniband/hw/vmw_pvrDMA/vmw_pvrDMA.ko.xz  
...
```

- **kernel-modules-extra** 의 모듈 나열 :

```
$ rpm -qlp <kernel-modules-extra_rpm>  
...  
/lib/modules/4.18.0-80.el8.x86_64/extra/net/sched/sch_cbq.ko.xz  
/lib/modules/4.18.0-80.el8.x86_64/extra/net/sched/sch_choke.ko.xz  
/lib/modules/4.18.0-80.el8.x86_64/extra/net/sched/sch_drr.ko.xz  
/lib/modules/4.18.0-80.el8.x86_64/extra/net/sched/sch_dsMark.ko.xz  
/lib/modules/4.18.0-80.el8.x86_64/extra/net/sched/sch_gred.ko.xz  
...
```

추가 리소스

- **rpm(8)** 매뉴얼 페이지
- [RPM 패키지](#)

2장. YUM으로 커널 업데이트

다음 섹션에서는 Red Hat(Red Hat 커널)에서 제공 및 유지 관리하는 Linux 커널에 대한 정보와 Red Hat 커널을 계속 업데이트하는 방법에 대해 설명합니다. 결과적으로 운영 체제에는 모든 최신 버그 수정, 성능 향상 및 패치가 포함되어 새 하드웨어와의 호환성이 보장됩니다.

2.1. 커널이란 무엇입니까?

커널은 시스템 리소스를 관리하고 하드웨어와 소프트웨어 애플리케이션 간의 인터페이스를 제공하는 Linux 운영 체제의 핵심 부분입니다. Red Hat 커널은 Red Hat 엔지니어가 최신 기술 및 하드웨어와의 호환성과 안정성과 호환성을 중심으로 개발 및 강화하는 업스트림 Linux 메인라인 커널을 기반으로 하는 맞춤형 커널입니다.

Red Hat에서 새 커널 버전을 출시하기 전에 커널은 일련의 엄격한 품질 보증 테스트를 통과해야 합니다.

Red Hat 커널은 RPM 형식으로 패키징되어 **yum** 패키지 관리자에서 쉽게 업그레이드 및 확인할 수 있습니다.



주의

Red Hat에서 컴파일하지 않은 커널은 Red Hat에서 지원하지 않습니다.

2.2. YUM이란 무엇입니까?

이 섹션은 **yum 패키지 관리자**에 대한 설명입니다.

2.3. 커널 업데이트

다음 절차에서는 **yum** 패키지 관리자를 사용하여 커널을 업데이트하는 방법을 설명합니다.

절차

1. 커널을 업데이트하려면 다음을 사용합니다.

```
# yum update kernel
```

이 명령은 커널과 모든 종속성을 함께 사용 가능한 최신 버전으로 업데이트합니다.

2. 시스템을 재부팅하여 변경 사항을 적용합니다.



참고

RHEL 7에서 RHEL 8로 업그레이드하는 경우 RHEL 7에서 **RHEL 8로 업그레이드** 하는 관련 섹션을 따르십시오.

2.4. 커널 설치

다음 절차에서는 **yum** 패키지 관리자를 사용하여 새 커널을 설치하는 방법을 설명합니다.

절차

- 특정 커널 버전을 설치하려면 다음을 사용하십시오.

```
# yum install kernel-{version}
```

추가 리소스

- [Red Hat 코드 브라우저](#)
- [Red Hat Enterprise Linux Release Dates](#)

3장. 커널 모듈 관리

다음 섹션에서는 커널 모듈이 무엇인지, 정보를 표시하는 방법 및 커널 모듈을 사용하여 기본 관리 작업을 수행하는 방법에 대해 설명합니다.

3.1. 커널 모듈 소개

Red Hat Enterprise Linux 커널은 시스템을 재부팅하지 않고도 커널 모듈이라는 선택적 추가 기능을 사용하여 확장할 수 있습니다. Red Hat Enterprise Linux 8에서 커널 모듈은 압축된 **<KERNEL_MODULE_NAME>.ko.xz** 오브젝트 파일에 빌드된 추가 커널 코드입니다.

커널 모듈에서 활성화하는 가장 일반적인 기능은 다음과 같습니다.

- 새 하드웨어 지원을 추가하는 장치 드라이버
- **GFS2** 또는 **NFS**와 같은 파일 시스템 지원
- 시스템 호출

최신 시스템에서는 필요에 따라 커널 모듈이 자동으로 로드됩니다. 그러나 경우에 따라 모듈을 수동으로 로드하거나 언로드해야 합니다.

커널 자체와 마찬가지로 모듈은 필요한 경우 동작을 사용자 지정하는 매개변수를 사용할 수 있습니다.

현재 실행 중인 모듈, 커널에 로드할 수 있는 모듈, 모듈에서 허용하는 매개 변수를 검사하기 위한 툴링이 제공됩니다. 또한 툴링은 커널 모듈을 실행 중인 커널로 로드하고 언로드하는 메커니즘을 제공합니다.

3.2. 부트 로더 사양 소개

부트 로더 사양(BLS)은 부트 로더 구성 파일을 조작할 필요 없이 드롭인 디렉토리의 각 부트 옵션에 대한 부트로더 구성을 관리할 체계 및 파일 형식을 정의합니다. 이전 방법과 달리 각 부팅 항목은 이제 드롭인 디렉터리에 있는 별도의 구성 파일로 표시됩니다. 드롭인 디렉터리는 구성 파일을 편집하거나 다시 생성할 필요 없이 구성을 확장합니다. BLS는 부팅 메뉴 항목에 대해 이 개념을 확장합니다.

BLS를 사용하면 디렉터리에서 개별 부트 항목 파일을 추가, 제거 또는 편집하여 부트 로더 메뉴 옵션을 관리할 수 있습니다. 이렇게 하면 커널 설치 프로세스가 서로 다른 아키텍처에서 훨씬 더 간단하고 일관되게 수행됩니다.

grubby 도구는 BLS를 중심으로 하는 쉘 래퍼 스크립트이며 동일한 **grubby** 인수 및 옵션을 지원합니다. **dracut** 을 실행하여 초기 램디스크 이미지를 생성합니다. 이 설정을 사용하면 코어 부트 로더 구성 파일이 정적이며 커널 설치 후에는 수정되지 않습니다.

RHEL 8에서는 동일한 부트로더가 모든 아키텍처에서 사용되지 않으므로 이 전제는 특히 RHEL 8에서 관련이 있습니다. **GRUB2** 는 64비트 ARM과 같이 대부분의 경우에 사용되지만 Open Power Abstraction Layer(OPAL)가 있는 IBM Power Systems의 little-endian 변형은 **Petitboot** 를 사용하며 IBM Z 아키텍처에서는 **zipl** 을 사용합니다.

추가 리소스

- [5.2절. "grubby의 정의"](#)
- [5.3절. "어떤 부팅 항목"](#)
- **grubby(8)** 도움말 페이지

3.3. 커널 모듈 종속성

특정 커널 모듈은 하나 이상의 다른 커널 모듈에 의존하는 경우가 있습니다.

`/lib/modules/<KERNEL_VERSION>/modules.dep` 파일에는 해당 커널 버전에 대한 전체 커널 모듈 종속성 목록이 포함되어 있습니다.

종속성 파일은 **kmod** 패키지의 일부인 **depmod** 프로그램에 의해 생성됩니다. **kmod**에서 제공하는 대부분의 유틸리티는 작업을 수행할 때 모듈 종속성을 고려하여 수동 종속성 추적이 거의 필요하지 않습니다.



주의

커널 모듈의 코드는 무제한 모드의 커널 공간에서 실행됩니다. 이로 인해 로드 중인 모듈을 주의해야 합니다.

추가 리소스

- **modules.dep(5)** 매뉴얼 페이지
- **depmod(8)** 매뉴얼 페이지

3.4. 현재 로드된 커널 모듈 나열

다음 절차에서는 현재 로드된 커널 모듈을 확인하는 방법을 설명합니다.

사전 요구 사항

- **kmod** 패키지가 설치됩니다.

절차

- 현재 로드된 커널 모듈을 모두 나열하려면 다음을 실행합니다.

```
$ lsmod
```

Module	Size	Used by
fuse	126976	3
uinput	20480	1
xt_CHECKSUM	16384	1
ipt_MASQUERADE	16384	1
xt_conntrack	16384	1
ipt_REJECT	16384	1
nft_counter	16384	16
nf_nat_tftp	16384	0
nf_conntrack_tftp	16384	1 nf_nat_tftp
tun	49152	1
bridge	192512	0
stp	16384	1 bridge
llc	16384	2 bridge,stp

```
nf_tables_set      32768 5
nft_fib_inet       16384 1
...
```

위의 예에서 다음을 수행합니다.

- 첫 번째 열은 현재 로드된 모듈의 이름을 제공합니다.
- 두 번째 열에는 모듈당 메모리 크기가 킬로바이트 단위로 표시됩니다.
- 마지막 열에는 숫자 및 특정 모듈에 종속된 모듈의 이름이 선택적으로 표시됩니다.

추가 리소스

- `/usr/share/doc/kmod/README` 파일
- `lsmod(8)` 매뉴얼 페이지

3.5. 설치된 모든 커널 나열

다음 절차에서는 **grubby** 유틸리티를 사용하여 시스템에 설치된 커널을 모두 나열하는 방법을 설명합니다.

사전 요구 사항

- 루트 권한이 있습니다.

절차

- 설치된 커널을 모두 나열하려면 다음을 실행합니다.

```
# grubby --info=ALL | grep ^kernel

kernel="/boot/vmlinuz-4.18.0-305.10.2.el8_4.x86_64"
kernel="/boot/vmlinuz-4.18.0-240.el8.x86_64"
kernel="/boot/vmlinuz-0-rescue-41eb2e172d7244698abda79a51778f1b"
```

출력에 설치된 모든 커널의 경로가 표시되고 해당 버전도 표시됩니다.

추가 리소스

- [grubby의 정의](#)

3.6. 커널을 기본값으로 설정

다음 절차에서는 **grubby** 명령줄 툴 및 GRUB2를 사용하여 특정 커널을 기본값으로 설정하는 방법을 설명합니다.

절차

- **grubby** 툴을 사용하여 커널을 기본값으로 설정
 - 다음 명령을 실행하여 **grubby** 도구를 사용하여 커널을 기본값으로 설정합니다.

```
# grubby --set-default $kernel_path
```

명령은 **.conf** 접미사가 없는 시스템 ID를 인수로 사용합니다.



참고

시스템 ID는 **/boot/loader/entries/** 디렉토리에 있습니다.

- **id** 인수를 사용하여 커널을 기본값으로 설정
 - **id** 인수를 사용하여 부팅 항목을 나열한 다음 의도한 커널을 기본값으로 설정합니다.

```
# grubby --info ALL | grep id
# grubby --set-default /boot/vmlinuz-<version>.<architecture>
```



참고

제목 인수를 사용하여 부팅 항목을 나열하려면 **# grubby --info=ALL | grep title** 명령을 실행합니다.

- 다음 부팅에만 기본 커널 설정
 - 다음 명령을 실행하여 **grub2-reboot** 명령을 사용하여 다음 재부팅에만 기본 커널을 설정합니다.

```
# grub2-reboot <index|title|id>
```



주의

주의해서 다음 부팅에만 기본 커널을 설정합니다. 새 커널 RPM의 자체 빌드 커널을 설치하고 **/boot/loader/entries/** 디렉토리에 항목을 수동으로 추가하면 인덱스 값이 변경될 수 있습니다.

3.7. 커널 모듈에 대한 정보 표시

커널 모듈로 작업할 때 해당 모듈에 대한 자세한 정보를 확인할 수 있습니다. 다음 절차에서는 커널 모듈에 대한 추가 정보를 표시하는 방법을 설명합니다.

사전 요구 사항

- **kmod** 패키지가 설치됩니다.

절차

- 커널 모듈에 대한 정보를 표시하려면 다음을 실행합니다.

```
$ modinfo <KERNEL_MODULE_NAME>
```

예를 들면 다음과 같습니다.

```
$ modinfo virtio_net

filename:    /lib/modules/4.18.0-94.el8.x86_64/kernel/drivers/net/virtio_net.ko.xz
license:     GPL
description: Virtio network driver
rhelversion: 8.1
srcversion:  2E9345B281A898A91319773
alias:       virtio:d00000001v*
depends:      net_failover
intree:      Y
name:         virtio_net
vermagic:    4.18.0-94.el8.x86_64 SMP mod_unload modversions
...
parm:        napi_weight:int
parm:        csum:bool
parm:        gso:bool
parm:        napi_tx:bool
```

modinfo 명령은 지정된 커널 모듈에 대한 몇 가지 세부 정보를 표시합니다. 로드 여부에 관계없이 사용 가능한 모든 모듈에 대한 정보를 쿼리할 수 있습니다. **parm** 항목은 사용자가 모듈에 대해 설정할 수 있는 매개 변수와 예상 값 유형을 보여줍니다.



참고

커널 모듈 이름을 입력할 때 이름 끝에 **.ko.xz** 확장을 추가하지 마십시오. 커널 모듈 이름에는 확장 기능이 없으며 해당 파일은 다음과 같습니다.

추가 리소스

- **modinfo(8)** 매뉴얼 페이지

3.8. 시스템 런타임 시 커널 모듈 로드

Linux 커널의 기능을 확장하는 최적의 방법은 커널 모듈을 로드하는 것입니다. 다음 절차에서는 **modprobe** 명령을 사용하여 커널 모듈을 찾고 현재 실행 중인 커널로 로드하는 방법을 설명합니다.

사전 요구 사항

- 루트 권한
- **kmod** 패키지가 설치됩니다.
- 각 커널 모듈이 로드되지 않습니다. 이러한 경우가 맞는지 확인하려면 [로드된 커널 모듈을 나열합니다](#).

절차

1. 로드할 커널 모듈을 선택합니다.
모듈은 `/lib/modules/$(uname -r)/kernel/<SUBSYSTEM>/` 디렉터리에 있습니다.
2. 관련 커널 모듈을 로드합니다.

```
# modprobe <MODULE_NAME>
```



참고

커널 모듈 이름을 입력할 때 이름 끝에 **.ko.xz** 확장을 추가하지 마십시오. 커널 모듈 이름에는 확장 기능이 없으며 해당 파일은 다음과 같습니다.

3. 선택적으로 관련 모듈이 로드되었는지 확인합니다.

```
$ lsmod | grep <MODULE_NAME>
```

모듈이 올바르게 로드된 경우 이 명령은 관련 커널 모듈을 표시합니다. 예를 들면 다음과 같습니다.

```
$ lsmod | grep serio_raw
serio_raw          16384 0
```



중요

이 절차에 설명된 변경 사항은 시스템을 재부팅한 **후에도 유지되지 않습니다**. 시스템 재부팅 시 커널 모듈을 로드하는 방법에 대한 자세한 내용은 [시스템 부팅 시 커널 모듈 로드](#)를 참조하십시오.

추가 리소스

- **modprobe(8)** 매뉴얼 페이지

3.9. 시스템 런타임에서 커널 모듈 언로드

실행 중인 커널에서 특정 커널 모듈을 언로드해야 하는 경우도 있습니다. 다음 절차에서는 현재 로드된 커널에서 시스템 런타임 시 **modprobe** 명령을 사용하여 커널 모듈을 찾고 언로드하는 방법을 설명합니다.

사전 요구 사항

- 루트 권한
- **kmod** 패키지가 설치됩니다.

절차

1. **lsmod** 명령을 실행하고 언로드할 커널 모듈을 선택합니다.
커널 모듈에 종속성이 있는 경우 커널 모듈을 언로드하기 전에 해당 모듈을 언로드합니다. 종속성이 있는 모듈 식별에 대한 자세한 내용은 [현재 로드된 커널 모듈 및 커널 모듈 종속성](#) 목록을 참조하십시오.
2. 관련 커널 모듈을 언로드합니다.

```
# modprobe -r <MODULE_NAME>
```

커널 모듈 이름을 입력할 때 이름 끝에 **.ko.xz** 확장을 추가하지 마십시오. 커널 모듈 이름에는 확장 기능이 없으며 해당 파일은 다음과 같습니다.



주의

실행 중인 시스템에서 커널 모듈을 사용할 때 커널 모듈을 언로드하지 마십시오. 이렇게 하면 불안정하거나 작동하지 않는 시스템으로 이어질 수 있습니다.

3. 필요한 경우 관련 모듈이 언로드되었는지 확인합니다.

```
$ lsmod | grep <MODULE_NAME>
```

모듈을 성공적으로 언로드하면 이 명령은 출력을 표시하지 않습니다.



중요

이 절차를 완료한 후에도 부팅 시 자동으로 로드되도록 정의된 커널 모듈은 시스템을 재부팅한 **후에도 로드되지 않습니다**. 이러한 결과를 **방지하는 방법에 대한 자세한 내용은 시스템 부팅 시 커널 모듈 방지**를 참조하십시오.

추가 리소스

- **modprobe(8)** 매뉴얼 페이지

3.10. 부팅 프로세스의 초기 단계에서 커널 모듈 언로드 해제

특정 상황에서는 부팅 프로세스 초기에 커널 모듈을 언로드해야 합니다. 예를 들어, kernel 모듈에 시스템이 응답하지 않게 하는 코드가 포함되어 있고 사용자가 악성 커널 모듈을 영구적으로 비활성화하기 위해 단계에 도달할 수 없습니다. 이 경우 부트로더를 사용하여 커널 모듈의 로드를 임시로 차단할 수 있습니다.



중요

이 절차에 설명된 변경 사항은 다음 재부팅 **후에도 유지되지 않습니다**. 부팅 프로세스 중에 커널 모듈이 자동으로 로드되지 않도록 커널 모듈을 거부 목록에 추가하는 방법에 대한 자세한 내용은 **시스템 부팅 시 커널 모듈이 자동으로 로드되지 않도록** 참조하십시오.

사전 요구 사항

- 로드할 수 있는 커널 모듈이 있으며, 어떤 이유로든 로드하지 못하게 합니다.

절차

- 부팅 시퀀스가 계속하기 전에 원하는 커널 모듈을 언로드하도록 관련 부트로더 항목을 편집합니다.
 - 커서 키를 사용하여 관련 부트로더 항목을 강조 표시합니다.
 - **e** 키를 눌러 항목을 편집합니다.

그림 3.1. 커널 부팅 메뉴

```

Red Hat Enterprise Linux (4.18.0-305.10.2.el8_4.x86_64) 8.4 (Ootpa)
Red Hat Enterprise Linux (4.18.0-305.3.1.el8_4.x86_64) 8.4 (Ootpa)
Red Hat Enterprise Linux (4.18.0-240.22.1.el8_3.x86_64) 8.3 (Ootpa)
Red Hat Enterprise Linux (0-rescue-bfeb014063334b2da66b978e95804445) 8.3→

Use the ↑ and ↓ keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.

```

- 커서 키를 사용하여 **linux** 로 시작하는 행으로 이동합니다.
- Append **modprobe.blacklist=module_name** 을 행 끝에 추가합니다.

그림 3.2. 커널 부팅 항목

```

load_video
set gfx_payload=keep
insmod gzio
linux ($root)/vmlinuz-4.18.0-305.10.2.el8_4.x86_64 root=/dev/mapper/rhel-root \
ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv\
=rhel/swap rhgb quiet modprobe.blacklist=serio_raw
initrd ($root)/initramfs-4.18.0-305.10.2.el8_4.x86_64.img $tuned_initrd

Press Ctrl-x to start, Ctrl-c for a command prompt or Escape to
discard edits and return to the menu. Pressing Tab lists
possible completions.

```

serio_raw 커널 모듈은 부팅 프로세스 초기에 로드를 해제하는 악성 모듈을 보여줍니다.

- **CTRL+x** 키를 눌러 수정된 구성을 사용하여 부팅합니다.

검증

- 시스템이 완전히 부팅되면 관련 커널 모듈이 로드되지 않았는지 확인합니다.

```
# lsmod | grep serio_raw
```

추가 리소스

- [커널 모듈 관리](#)

3.11. 시스템 부팅 시 커널 모듈 자동 로드

다음 절차에서는 부팅 프로세스 중에 자동으로 로드되도록 커널 모듈을 구성하는 방법을 설명합니다.

사전 요구 사항

- 루트 권한
- **kmod** 패키지가 설치됩니다.

절차

1. 부팅 프로세스 중에 로드할 커널 모듈을 선택합니다.
모듈은 `/lib/modules/$(uname -r)/kernel/<SUBSYSTEM>/` 디렉터리에 있습니다.
2. 모듈에 대한 구성 파일을 생성합니다.

```
# echo <MODULE_NAME> > /etc/modules-load.d/<MODULE_NAME>.conf
```



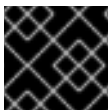
참고

커널 모듈 이름을 입력할 때 이름 끝에 **.ko.xz** 확장을 추가하지 마십시오. 커널 모듈 이름에는 확장 기능이 없으며 해당 파일은 다음과 같습니다.

3. 선택적으로 재부팅 후 관련 모듈이 로드되었는지 확인합니다.

```
$ lsmod | grep <MODULE_NAME>
```

위의 예제 명령은 성공하고 관련 커널 모듈을 표시해야 합니다.



중요

이 절차에 설명된 변경 사항은 시스템을 재부팅한 **후에도** 유지됩니다.

추가 리소스

- **modules-load.d(5)** 매뉴얼 페이지

3.12. 시스템 부팅 시 커널 모듈이 자동으로 로드되지 않도록 방지

다음 절차에서는 부팅 프로세스 중에 자동으로 로드되지 않도록 커널 모듈을 거부 목록에 추가하는 방법을 설명합니다.

사전 요구 사항

- 루트 권한
- **kmod** 패키지가 설치됩니다.
- 거부 목록의 커널 모듈이 현재 시스템 구성에 중요하지 않은지 확인합니다.

절차

1. 거부 목록에 넣을 커널 모듈을 선택합니다.

\$ lsmod

Module	Size	Used by
fuse	126976	3
xt_CHECKSUM	16384	1
ipt_MASQUERADE	16384	1
uinput	20480	1
xt_conntrack	16384	1
...		

lsmod 명령은 현재 실행 중인 커널에 로드된 모듈 목록을 표시합니다.

- 또는 잠재적으로 로드되지 않도록 하려는 로드되지 않은 커널 모듈을 식별합니다.
모든 커널 모듈은 `/lib/modules/<KERNEL_VERSION>/kernel/<SUBSYSTEM>/` 디렉터리에 있습니다.

2. 거부 목록에 대한 구성 파일을 생성합니다.

vim /etc/modprobe.d/blacklist.conf

```
# Blacklists <KERNEL_MODULE_1>
blacklist <MODULE_NAME_1>
install <MODULE_NAME_1> /bin/false

# Blacklists <KERNEL_MODULE_2>
blacklist <MODULE_NAME_2>
install <MODULE_NAME_2> /bin/false

# Blacklists <KERNEL_MODULE_n>
blacklist <MODULE_NAME_n>
install <MODULE_NAME_n> /bin/false
...
```

이 예제에서는 **vim** 편집기에서 편집한 **blacklist.conf** 파일의 내용을 보여줍니다. **blacklist** 행을 사용하면 부팅 프로세스 중에 관련 커널 모듈이 자동으로 로드되지 않습니다. 그러나 **blacklist** 명령은 거부 목록에 없는 다른 커널 모듈의 종속성으로 모듈이 로드되지 않도록 방지하지 않습니다. 따라서 **install** 행은 모듈을 설치하는 대신 **/bin/false**가 실행됩니다.

해시 기호로 시작하는 행은 파일을 읽기 쉽게 만드는 주석입니다.

**참고**

커널 모듈 이름을 입력할 때 이름 끝에 **.ko.xz** 확장을 추가하지 마십시오. 커널 모듈 이름에는 확장 기능이 없으며 해당 파일은 다음과 같습니다.

3. 다시 빌드하기 전에 현재 초기 ramdisk 이미지의 백업 사본을 생성합니다.

```
# cp /boot/initramfs-$(uname -r).img /boot/initramfs-$(uname -r).bak.$(date +%m-%d-%H%M%S).img
```

새 버전에 예기치 않은 문제가 발생하는 경우 위의 명령은 백업 **initramfs** 이미지를 생성합니다.

- 또는 커널 모듈을 거부 목록에 배치하려는 커널 버전에 해당하는 다른 초기 램디스크 이미지의 백업 사본을 생성합니다.

■

```
# cp /boot/initramfs-<SOME_VERSION>.img /boot/initramfs-
<SOME_VERSION>.img.bak.$(date +%m-%d-%H%M%S)
```

4. 변경 사항을 반영하려면 새 초기 램디스크 이미지를 생성합니다.

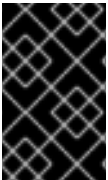
```
# dracut -f -v
```

- 현재 부팅된 것과 다른 커널 버전의 초기 램디스크 이미지를 빌드하는 경우 대상 **initramfs** 및 커널 버전을 둘 다 지정합니다.

```
# dracut -f -v /boot/initramfs-<TARGET_VERSION>.img
<CORRESPONDING_TARGET_KERNEL_VERSION>
```

5. 시스템을 재부팅합니다.

```
$ reboot
```



중요

이 절차에 설명된 변경 사항은 적용되며 시스템을 재부팅한 후에도 지속됩니다. 키 커널 모듈을 거부 목록에 잘못 배치하면 불안정하거나 작동하지 않는 시스템에 직면할 수 있습니다.

추가 리소스

- [커널 모듈이 자동으로 로드되지 않도록 하려면 어떻게 해야 하나요?](#)
- **dracut(8)** 매뉴얼 페이지

3.13. 사용자 정의 커널 모듈 컴파일

하드웨어 및 소프트웨어 수준에서 다양한 구성에서 요청한 대로 샘플링 커널 모듈을 빌드할 수 있습니다.

사전 요구 사항

- **kernel-devel**, **gcc** 및 **elfutils-libelf-devel** 패키지를 설치했습니다.
- 루트 권한이 있습니다.
- 사용자 지정 커널 모듈을 컴파일하는 **/root/testmodule/** 디렉터리를 생성하셨습니다.

절차

1. 다음 콘텐츠를 사용하여 **/root/testmodule/test.c** 파일을 만듭니다.

```
#include <linux/module.h>
#include <linux/kernel.h>

int init_module(void)
{ printk("Hello World\n This is a test\n"); return 0; }

void cleanup_module(void)
{ printk("Good Bye World"); }
```

test.c 파일은 커널 모듈에 기본 기능을 제공하는 소스 파일입니다. 파일은 조직 용도로 전용 **/root/testmodule/** 디렉토리에 생성되었습니다. 모듈 컴파일 후 **/root/testmodule/** 디렉토리에 여러 파일이 포함됩니다.

test.c 파일은 시스템 라이브러리에서 포함합니다.

- 예제 코드의 **printk()** 함수에는 **linux/kernel.h** 헤더 파일이 필요합니다.
- linux/module.h** 파일에는 C 프로그래밍 언어로 작성된 여러 소스 파일 간에 공유할 함수 선언과 매크로 정의가 포함되어 있습니다.
다음으로 **init_module()** 및 **cleanup_module()** 함수를 따라 텍스트를 인쇄하는 커널 로깅 함수 **printk()** 를 시작하고 종료합니다.

2. 다음 콘텐츠를 사용하여 **/root/testmodule/Makefile** 파일을 만듭니다.

```
obj-m := test.o
```

Makefile에는 컴파일러에서 특별히 **test.o** 라는 개체 파일을 생성해야 하는 명령이 포함되어 있습니다. **obj-m** 지시문은 결과 **test.ko** 파일이 로드 가능한 커널 모듈로 컴파일되도록 지정합니다. 또는 **obj-y** 지시문은 **test.ko** 를 기본 제공 커널 모듈로 빌드하도록 지시합니다.

3. 커널 모듈을 컴파일합니다.

```
# make -C /lib/modules/$(uname -r)/build M=/root/testmodule modules
make: Entering directory '/usr/src/kernels/4.18.0-305.el8.x86_64'
CC [M] /root/testmodule/test.o
Building modules, stage 2.
MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /root/testmodule/test.o
see include/linux/module.h for more information
CC /root/testmodule/test.mod.o
LD [M] /root/testmodule/test.ko
make: Leaving directory '/usr/src/kernels/4.18.0-305.el8.x86_64'
```

컴파일러는 최종 커널 모듈(**test.ko**)에 연결하기 전에 각 소스 파일(**test.c**)에 대한 오브젝트 파일(**test.o**)을 중간 단계로 생성합니다.

컴파일에 성공하면 **/root/testmodule/** 에 컴파일된 사용자 지정 커널 모듈과 관련된 추가 파일이 포함됩니다. 컴파일된 모듈 자체는 **test.ko** 파일로 표시됩니다.

검증

1. 선택 사항: **/root/testmodule/** 디렉토리의 내용을 확인합니다.

```
# ls -l /root/testmodule/
total 152
-rw-r--r--. 1 root root 16 Jul 26 08:19 Makefile
-rw-r--r--. 1 root root 25 Jul 26 08:20 modules.order
-rw-r--r--. 1 root root 0 Jul 26 08:20 Module.symvers
-rw-r--r--. 1 root root 224 Jul 26 08:18 test.c
-rw-r--r--. 1 root root 62176 Jul 26 08:20 test.ko
-rw-r--r--. 1 root root 25 Jul 26 08:20 test.mod
-rw-r--r--. 1 root root 849 Jul 26 08:20 test.mod.c
-rw-r--r--. 1 root root 50936 Jul 26 08:20 test.mod.o
-rw-r--r--. 1 root root 12912 Jul 26 08:20 test.o
```

2. kernel 모듈을 **/lib/modules/\$(uname -r)/** 디렉터리에 복사합니다.

```
# cp /root/testmodule/test.ko /lib/modules/$(uname -r)/
```

3. 모듈 종속성 목록을 업데이트합니다.

```
# depmod -a
```

4. 커널 모듈을 로드합니다.

```
# modprobe -v test  
insmod /lib/modules/4.18.0-305.el8.x86_64/test.ko
```

5. 커널 모듈이 성공적으로 로드되었는지 확인합니다.

```
# lsmod | grep test  
test                16384  0
```

6. 커널 링 버퍼에서 최신 메시지를 읽습니다.

```
# dmesg  
[74422.545004] Hello World  
                This is a test
```

추가 리소스

- [커널 모듈 관리](#)

4장. 보안 부팅을 위한 커널 모듈 서명

서명된 커널 모듈을 사용하여 시스템의 보안을 강화할 수 있습니다. 다음 섹션에서는 Secure Boot가 활성화된 UEFI 기반 빌드 시스템에서 RHEL 8과 함께 사용할 수 있도록 개인용으로 빌드된 커널 모듈을 자체 서명하는 방법에 대해 설명합니다. 또한 이 섹션에서는 커널 모듈을 배포하려는 타겟 시스템으로 공개 키를 가져오는 데 사용할 수 있는 옵션에 대한 개요를 제공합니다.

Secure Boot가 활성화된 경우 UEFI 운영 체제 부트 로더, Red Hat Enterprise Linux 커널 및 모든 커널 모듈에 개인 키로 서명하고 해당 공개 키로 인증해야 합니다. 서명 및 인증되지 않은 경우 시스템은 부팅 프로세스를 완료할 수 없습니다.

RHEL 8 배포판에는 다음이 포함됩니다.

- 서명된 부트 로더
- 서명된 커널
- 서명된 커널 모듈

또한 서명된 1단계 부트 로더와 서명된 커널에는 Red Hat 공개 키가 포함되어 있습니다. 서명된 실행 가능 바이너리 및 포함된 키를 사용하면 RHEL 8에서 UEFI 펌웨어가 UEFI Secure Boot를 지원하는 시스템에서 제공하는 Microsoft UEFI Secure Boot 인증 기관 키를 설치, 부팅 및 실행할 수 있습니다. 일부 UEFI 기반 시스템에 Secure Boot 지원이 포함된 것은 아닙니다.

사전 요구 사항

외부에서 빌드된 커널 모듈에 서명할 수 있으려면 빌드 시스템에 다음 테이블에 나열된 유틸리티를 설치합니다.

표 4.1. 필수 유틸리티

유틸리티	패키지에 의해 제공	사용됨	목적
openssl	openssl	빌드 시스템	공개 및 개인 X.509 키 쌍 생성
sign-file	kernel-devel	빌드 시스템	개인 키로 커널 모듈에 서명하는 데 사용되는 실행 가능 파일
mokutil	mokutil	대상 시스템	공개 키를 수동으로 등록하는 데 사용되는 선택적 유틸리티
keyctl	keyutils	대상 시스템	시스템 인증 키에 공개 키를 표시하는 데 사용되는 선택적 유틸리티



참고

커널 모듈을 빌드하고 서명하는 빌드 시스템에 UEFI Secure Boot를 활성화할 필요가 없으며 UEFI 기반 시스템일 필요도 없습니다.

4.1. UEFI SECURE BOOT이란 무엇입니까?

UEFI(*Unified Extensible Firmware Interface*) Secure Boot 기술을 사용하면 시스템 부트 로더가 암호화 키로 서명되었는지 확인할 수 있습니다. 펌웨어에 포함된 공개 키의 데이터베이스는 서명 키를 인증합니다.

다. 다음 단계 부트 로더 및 커널에서 서명을 확인할 수 있습니다. 결과적으로 신뢰할 수 있는 키로 서명되지 않은 커널 공간 코드를 실행하지 못하도록 할 수 있습니다.

UEFI Secure Boot는 다음과 같이 펌웨어에서 서명된 드라이버 및 커널 모듈로 신뢰 체인을 설정합니다.

- UEFI 개인 키 서명 및 공개 키는 **shim** 첫 단계 부팅 로더를 인증합니다. 인증 기관(CA)은 공개 키에 서명합니다. CA는 펌웨어 데이터베이스에 저장됩니다.
- **shim.efi**에는 GRUB 2 부트 로더와 커널을 인증하기 위한 **Red Hat 공개 키(CA 키 1)**가 포함되어 있습니다.
- 커널에 차례로 드라이버와 모듈을 인증하는 공개 키가 포함되어 있습니다.

Secure Boot는 UEFI 사양의 부팅 경로 검증 구성 요소입니다. 사양은 다음을 정의합니다.

- 비휘발성 스토리지에서 암호로 보호된 UEFI 변수를 위한 프로그래밍 인터페이스
- UEFI 변수에 신뢰할 수 있는 X.509 루트 인증서 저장
- 부트 로더 및 드라이버와 같은 UEFI 애플리케이션 검증
- 알려진 만료된 인증서 및 애플리케이션 해시를 취소하는 절차

UEFI Secure Boot는 두 번째 단계의 부트 로더의 설치 또는 제거를 방지하지 않습니다. 또한 이러한 변경 사항에 대한 명시적인 사용자 확인이 필요하지 않습니다. 부트 로더가 설치 또는 업데이트될 때 아닌 부팅 중에 서명을 확인합니다. 따라서 UEFI Secure Boot는 부팅 경로 조작을 중지하지 않습니다. 인증되지 않은 변경 사항을 감지하는 데 도움이 됩니다.



참고

부트 로더 또는 커널은 시스템에서 신뢰할 수 있는 키가 서명하는 한 작동합니다.

4.2. UEFI SECURE BOOT 지원

RHEL 8에는 UEFI(Unified Extensible Firmware Interface) Secure Boot 기능을 지원합니다. 즉, UEFI Secure Boot가 활성화된 시스템에서 RHEL 8을 설치하고 실행할 수 있습니다. 이러한 시스템에서는 로드된 모든 드라이버에 신뢰할 수 있는 키로 서명해야 합니다. 그렇지 않으면 시스템이 이를 허용하지 않습니다. Red Hat은 관련 Red Hat 키로 서명 및 인증된 드라이버를 제공합니다.

외부 빌드 드라이버를 로드하려면 해당 드라이버도 서명해야 합니다.

UEFI Secure Boot에서 설정한 제한 사항

- 시스템은 서명이 올바르게 인증된 후에만 커널 모드 코드를 실행합니다.
- GRUB 2 모듈 로드는 GRUB 2 모듈의 서명 및 검증에 대한 인프라가 없기 때문에 비활성화되어 있습니다. 이를 로드할 수 있도록 허용하면 Secure Boot에서 정의하는 보안 gateway 내부의 신뢰할 수 없는 코드를 실행하도록 구성됩니다.
- Red Hat은 RHEL 8에서 지원되는 모든 모듈이 포함된 서명된 GRUB 2 바이너리를 제공합니다.

추가 리소스

- [UEFI Secure Boot에서 설정한 제한 사항](#)

4.3. X.509 키를 사용하여 커널 모듈을 인증하기 위한 요구 사항

RHEL 8에서 커널 모듈이 로드되면 커널은 커널 시스템 인증 키(**.builtin_trusted_keys**) 및 커널 플랫폼 인증 키(**.platform**)에서 공개 **X.509** 키와 비교하여 모듈의 서명을 확인합니다. **.platform** 인증 키에는 타사 플랫폼 공급자 및 사용자 지정 공개 키가 포함되어 있습니다. 커널 system **.blacklist** 키의 키는 확인에서 제외됩니다. 다음 섹션에서는 시스템의 다양한 소스에서 로드된 키의 예제, 키, 키에 대한 개요를 제공합니다. 또한 커널 모듈을 인증하는 방법을 확인할 수 있습니다.

UEFI Secure Boot 기능이 활성화된 시스템에서 커널 모듈을 로드하려면 특정 조건을 충족해야 합니다.

UEFI Secure Boot가 활성화되어 있거나 **module.sig_enforce** 커널 매개변수가 지정된 경우 다음을 수행합니다.

- 시스템 인증 키(**.builtin_trusted_keys**) 및 플랫폼 인증 키(**.platform**)의 키와 서명된 커널 모듈만 로드할 수 있습니다.
- 공개 키는 시스템 해지 키 인증 키(**.blacklist**)에 없어야 합니다.

UEFI Secure Boot가 비활성화되어 있고 **module.sig_enforce** 커널 매개변수가 지정되지 않은 경우:

- 서명되지 않은 커널 모듈과 서명되지 않은 커널 모듈을 공개 키없이 로드할 수 있습니다.

시스템이 UEFI 기반이 아니거나 UEFI Secure Boot가 비활성화된 경우 다음을 수행합니다.

- 커널에 포함된 키만 **.builtin_trusted_keys** 및 **.platform**에 로드됩니다.
- 커널을 다시 빌드하지 않고 키 집합을 보강할 수 있는 기능이 없습니다.

표 4.2. 커널 모듈 인증 요구 사항 로드

모듈 서명됨	공개 키를 발견하고 서명이 유효한 경우	UEFI Secure Boot 상태	sig_enforce	모듈 로드	커널 테인트됨
서명되지 않음	-	활성화되지 않음	활성화되지 않음	성공	있음
		활성화되지 않음	활성화됨	실패	-
		활성화됨	-	실패	-
서명됨	없음	활성화되지 않음	활성화되지 않음	성공	있음
		활성화되지 않음	활성화됨	실패	-
		활성화됨	-	실패	-
서명됨	있음	활성화되지 않음	활성화되지 않음	성공	없음
		활성화됨	활성화됨	성공	없음

모듈 서명됨	공개 키를 발견하고 서명이 유효한 경우	UEFI Secure Boot 상태	sig_enforce	모듈 로드	커널 테인트됨
		활성화되지 않음	활성화됨	성공	없음
		활성화됨	-	성공	없음

4.4. 공개 키 소스

부팅 중에 커널은 일련의 영구 키 저장소에서 다음 인증 키에 X.509 키를 로드합니다.

- 시스템 인증 키(**.builtin_trusted_keys**)
- **.platform** 인증 키
- system **.blacklist** 인증 키

표 4.3. 시스템 인증 키 소스

X.509 키 소스	사용자는 키를 추가할 수 있습니다	UEFI Secure Boot 상태	부팅 중 로드된 키
커널에 포함	없음	-	.builtin_trusted_keys
UEFI Secure Boot "db"	제한됨	활성화되지 않음	없음
		활성화됨	.platform
shim.efi 부트 로더에 포함	없음	활성화되지 않음	없음
		활성화됨	.builtin_trusted_keys
MOK(Machine Owner Key) 목록	있음	활성화되지 않음	없음
		활성화됨	.platform

.builtin_trusted_keys:

- 부팅에 빌드된 인증 키
- 신뢰할 수 있는 공개 키 포함
- 키를 보려면 루트 권한이 필요합니다.

.platform:

- 부팅에 빌드된 인증 키
- 타사 플랫폼 공급자 및 사용자 정의 공개 키의 키 포함
- 키를 보려면 루트 권한이 필요합니다.

.blacklist

- 취소된 X.509 키를 사용한 인증 키
- 에서 키로 서명한 모듈 **.blacklist** 는 공개 키가. **builtin_trusted_keys**에 있는 경우에도 인증에 실패합니다.

UEFI Secure Boot db:

- 서명 데이터베이스
- UEFI 애플리케이션, UEFI 드라이버 및 부트로더의 키 (hashes) 저장
- 키를 시스템에 로드할 수 있습니다.

UEFI Secure Boot dbx:

- 취소된 서명 데이터베이스
- 키가 로드되지 않도록 방지
- 이 데이터베이스의 박탈된 키는 **.blacklist** 인증 키에 추가됩니다.

4.5. 공개 및 개인 키 쌍 생성

Secure Boot 지원 시스템에서 커널 모듈을 사용하기 위해서는 공용 및 개인 X.509 키 쌍을 생성해야 합니다. 나중에 개인 키를 사용하여 커널 모듈에 서명합니다. 서명된 모듈을 검증하려면 Secure Boot의 해당 공개 키를 MOK(Machine Owner Key)에 추가해야 합니다.

이 키 쌍 생성에 대한 일부 매개 변수는 구성 파일을 사용하여 가장 잘 지정됩니다.

절차

1. 키 쌍 생성에 대한 매개 변수를 사용하여 구성 파일을 생성합니다.

```
# cat << EOF > configuration_file.config
[ req ]
default_bits = 4096
distinguished_name = req_distinguished_name
prompt = no
string_mask = utf8only
x509_extensions = myexts

[ req_distinguished_name ]
O = Organization
CN = Organization signing key
emailAddress = E-mail address

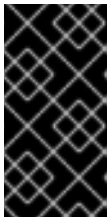
[ myexts ]
basicConstraints=critical,CA:FALSE
```

```
keyUsage=digitalSignature
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid
EOF
```

2. 다음 예와 같이 X.509 공용 및 개인 키 쌍을 만듭니다.

```
# openssl req -x509 -new -nodes -utf8 -sha256 -days 36500 \
  -batch -config configuration_file.config -outform DER \
  -out my_signing_key_pub.der \
  -keyout my_signing_key.priv
```

공개 키는 **my_signing_key_pub.der** 파일에 작성되고 개인 키는 **my_signing_key.priv** 파일에 작성됩니다.



중요

RHEL 8에서는 키 쌍의 유효 날짜가 중요하지 않습니다. 키는 만료되지 않지만 서명 키의 유효 기간 내에 커널 모듈에 서명하는 것이 좋습니다. 그러나 **sign-file** 유틸리티는 사용자에게 경고하지 않으며 유효 날짜에 관계없이 키를 사용할 수 있습니다.

3. 필요한 경우 아래 예와 같이 공개 키의 유효 날짜를 검토할 수 있습니다.

```
# openssl x509 -inform der -text -noout -in my_signing_key_pub.der
```

Validity

```
Not Before: Feb 14 16:34:37 2019 GMT
Not After : Feb 11 16:34:37 2029 GMT
```

4. 인증하고 커널 모듈을 로드하려는 모든 시스템에 공개 키를 등록합니다.



주의

강력한 보안 조치 및 액세스 정책을 적용하여 개인 키의 내용을 보호합니다. 잘못된 경우 키를 사용하여 해당 공개 키로 인증된 시스템을 손상시킬 수 있습니다.

추가 리소스

- **OpenSSL(1)** 매뉴얼 페이지
- [RHEL 보안 가이드](#)
- 공개 키를 MOK 목록에 추가하여 대상 시스템에 공개 키 등록

4.6. 시스템 인증 키의 출력 예

key **utils** 패키지의 **keyctl** 유틸리티를 사용하여 시스템 인증 키의 키에 대한 정보를 표시할 수 있습니다.

다음은 UEFI Secure Boot가 활성화된 RHEL 8 시스템의 **.builtin_trusted_keys**, **.platform** 및 **.blacklist** 인증 키의 축약된 출력 예입니다.

사전 요구 사항

- 루트 권한이 있습니다.
- key **utils** 패키지에서 **keyctl** 유틸리티를 설치하셨습니다.

```
# keyctl list %:.builtin_trusted_keys
6 keys in keyring:
...asymmetric: Red Hat Enterprise Linux Driver Update Program (key 3): bf57f3e87...
...asymmetric: Red Hat Secure Boot (CA key 1): 4016841644ce3a810408050766e8f8a29...
...asymmetric: Microsoft Corporation UEFI CA 2011: 13adbf4309bd82709c8cd54f316ed...
...asymmetric: Microsoft Windows Production PCA 2011: a92902398e16c49778cd90f99e...
...asymmetric: Red Hat Enterprise Linux kernel signing key: 4249689eefc77e95880b...
...asymmetric: Red Hat Enterprise Linux kpatch signing key: 4d38fd864ebe18c5f0b7...

# keyctl list %:.platform
4 keys in keyring:
...asymmetric: VMware, Inc.: 4ad8da0472073...
...asymmetric: Red Hat Secure Boot CA 5: cc6fafe72...
...asymmetric: Microsoft Windows Production PCA 2011: a929f298e1...
...asymmetric: Microsoft Corporation UEFI CA 2011: 13adbf4e0bd82...

# keyctl list %:.blacklist
4 keys in keyring:
...blacklist: bin:f5ff83a...
...blacklist: bin:0dfdbec...
...blacklist: bin:38f1d22...
...blacklist: bin:51f831f...
```

위의 **.builtin_trusted_keys** 키는 UEFI Secure Boot "db" 키와 **shim.efi** 부트 로더에 포함된 **Red Hat Secure Boot 1(CA 키 1)**에서 두 개의 키를 추가하는 것을 보여줍니다.

다음 예제에서는 커널 콘솔 출력을 보여줍니다. 이 메시지는 UEFI Secure Boot 관련 소스로 키를 식별합니다. 여기에는 UEFI Secure Boot db, 임베디드 shim 및 MOK 목록이 포함됩니다.

```
# dmesg | egrep 'integrity.*cert'
[1.512966] integrity: Loading X.509 certificate: UEFI:db
[1.513027] integrity: Loaded X.509 cert 'Microsoft Windows Production PCA 2011: a929023...
[1.513028] integrity: Loading X.509 certificate: UEFI:db
[1.513057] integrity: Loaded X.509 cert 'Microsoft Corporation UEFI CA 2011: 13adbf4309...
[1.513298] integrity: Loading X.509 certificate: UEFI:MokListRT (MOKvar table)
[1.513549] integrity: Loaded X.509 cert 'Red Hat Secure Boot CA 5: cc6fa5e72868ba494e93...
```

추가 리소스

- keyctl(1)**, **dmesg(1)** 매뉴얼 페이지

4.7. 공개 키를 MOK 목록에 추가하여 대상 시스템에 공개 키 등록

Secure Boot가 활성화된 UEFI 기반 시스템에서 RHEL 8을 부팅하면 커널이 시스템 인증 키 (**.builtin_trusted_keys**)로 로드되며 Secure Boot db 키 데이터베이스에 있는 모든 공개 키입니다. 동시에

커널은 박탈된 키의 dbx 데이터베이스에서 키를 제외합니다. 아래 섹션에서는 시스템 인증 키 (**.builtin_trusted_keys**)가 공개 키를 사용하여 커널 모듈을 인증할 수 있도록 타겟 시스템에서 공개 키를 가져오는 방법에 대해 설명합니다.

MOK(Machine Owner Key) 기능 기능을 사용하여 UEFI Secure Boot 키 데이터베이스를 확장할 수 있습니다. Secure Boot가 활성화된 UEFI 지원 시스템에서 RHEL 8을 부팅하면 키 데이터베이스의 키 외에도 MOK 목록의 키도 시스템 인증 키(**.builtin_trusted_keys**)에 추가됩니다. MOK 목록 키도 Secure Boot 데이터베이스 키와 동일한 방식으로 영구적으로 안전하게 저장되지만 두 가지 개별 기능입니다. MOK 기능은 **shim.efi**, **MokManager.efi**, **grubx64.efi** 및 **mokutil** 유틸리티에서 지원합니다.

MOK 키를 등록하려면 각 대상 시스템의 UEFI 시스템 콘솔에서 사용자가 수동으로 상호 작용해야 합니다. 그렇지만 MOK 기능은 새로 생성된 키 쌍을 테스트하고 이 모듈과 서명된 커널 모듈을 테스트할 수 있는 편리한 방법을 제공합니다.

절차

1. MOK 목록에 공개 키 추가를 요청합니다.

```
# mokutil --import my_signing_key_pub.der
```

MOK 등록 요청의 암호를 입력하고 확인하라는 메시지가 표시됩니다.

2. 시스템을 재부팅합니다.
보류 중인 MOK 키 등록 요청은 **shim.efi**에 의해 감지되고 UEFI 콘솔에서 등록을 완료할 수 있도록 **MokManager.efi**가 시작됩니다.
3. "Enroll MOK"를 선택하고 메시지가 표시되면 이전에 이 요청과 연결된 암호를 입력하고 등록을 확인합니다.
공개 키는 영구 MOK 목록에 추가됩니다.

MOK 목록에 키가 있으면 자동으로 이 인증 키에서 시스템 인증 키로 전파되고 UEFI Secure Boot가 활성화되면 이후 부팅됩니다.



참고

시스템에서 커널 모듈 인증을 용이하게 하려면 시스템 벤더에서 공개 키를 팩토리 펌웨어 이미지에 UEFI Secure Boot 키 데이터베이스에 통합하도록 요청하는 것을 고려하십시오.

4.8. 개인 키로 커널 모듈 서명

UEFI Secure Boot 메커니즘이 활성화된 경우 서명된 커널 모듈을 로드하여 시스템에서 향상된 보안 이점을 얻을 수 있습니다. 다음 섹션에서는 개인 키로 커널 모듈에 서명하는 방법을 설명합니다.

사전 요구 사항

- 공개 및 개인 키 쌍을 생성하고 공개 키의 유효 날짜를 알고 있습니다. 자세한 내용은 [공개 키 및 개인 키 쌍 생성](#)을 참조하십시오.
- 대상 시스템에 공개 키를 등록했습니다. 자세한 내용은 [MOK 목록에 공개 키를 추가하여 대상 시스템에서 공개 키 등록](#)을 참조하십시오.
- 서명에 사용할 수 있는 ELF 이미지 형식의 커널 모듈이 있습니다.

절차

- 아래 예와 같이 매개 변수를 사용하여 **Sign-file** 유틸리티를 실행합니다.

```
# /usr/src/kernels/$(uname -r)/scripts/sign-file sha256 my_signing_key.priv
my_signing_key_pub.der my_module.ko
```

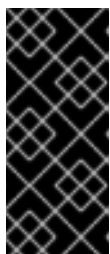
서명-파일 계산 및 커널 모듈 파일의 ELF 이미지에 서명을 직접 추가합니다. **modinfo** 유틸리티를 사용하여 커널 모듈 서명에 대한 정보를 표시할 수 있습니다(있는 경우).



참고

첨부된 서명은 ELF 이미지 섹션에 포함되지 않으며 ELF 이미지의 공식적인 일부가 아닙니다. 따라서 **readelf** 와 같은 유틸리티는 커널 모듈에 서명을 표시할 수 없습니다.

이제 커널 모듈을 로드할 준비가 되었습니다. 서명된 커널 모듈은 UEFI Secure Boot가 비활성화되거나 UEFI가 아닌 시스템에서도 로드할 수 있습니다. 즉, 서명된 커널 모듈과 서명되지 않은 버전의 커널 모듈을 모두 제공할 필요가 없습니다.



중요

RHEL 8에서 키 쌍의 유효 날짜는 중요합니다. 키는 만료되지 않지만 커널 모듈은 서명 키의 유효 기간 내에 서명해야 합니다. **sign-file** 유틸리티는 이에 대해 경고하지 않습니다. 예를 들어 2019년에만 유효한 키는 해당 키로 2019년에 서명된 커널 모듈을 인증하는 데 사용할 수 있습니다. 그러나 사용자는 2020년 커널 모듈에 서명하는 데 해당 키를 사용할 수 없습니다.

추가 리소스

- [커널 모듈에 대한 정보 표시](#)

4.9. 서명된 커널 모듈 로드

공개 키가 시스템 인증 키(**.builtin_trusted_keys**) 및 MOK 목록에 등록되고 개인 키를 사용하여 각 커널 모듈에 서명한 후 다음 섹션에 설명된 대로 **modprobe** 명령으로 서명된 커널 모듈을 마지막으로 로드할 수 있습니다.

사전 요구 사항

- 공용 및 개인 키 쌍을 생성했습니다. 자세한 내용은 [공개 키 및 개인 키 쌍 생성](#)을 참조하십시오.
- 공개 키를 시스템 인증 키에 등록했습니다. 자세한 내용은 [MOK 목록에 공개 키를 추가하여 대상 시스템에서 공개 키 등록](#)을 참조하십시오.
- 개인 키를 사용하여 커널 모듈에 서명했습니다. 자세한 내용은 [개인 키를 사용하여 커널 모듈 서명](#)을 참조하십시오.

절차

- 공개 키가 시스템 인증 키에 있는지 확인합니다.

```
# keyctl list %:.platform
```

2. `/lib/modules/$(uname -r)/extra/` 디렉토리를 생성하는 **kernel-modules-extra** 패키지를 설치합니다.

```
# yum -y install kernel-modules-extra
```

3. 커널 모듈을 원하는 커널의 `/extra/` 디렉토리에 복사합니다.

```
# cp my_module.ko /lib/modules/$(uname -r)/extra/
```

4. 모듈 종속성 목록을 업데이트합니다.

```
# depmod -a
```

5. 커널 모듈을 로드하고 성공적으로 로드되었는지 확인합니다.

```
# modprobe -v my_module  
# lsmod | grep my_module
```

- a. 선택적으로 부팅 시 모듈을 로드하려면 `/etc/modules-loaded.d/my_module.conf` 파일에 추가합니다.

```
# echo "my_module" > /etc/modules-load.d/my_module.conf
```

추가 리소스

- [커널 모듈 관리](#)

5장. 커널 명령줄 매개변수 구성

커널 명령줄 매개 변수는 부팅 시 Red Hat Enterprise Linux 커널의 특정 측면의 동작을 변경하는 방법입니다. 시스템 관리자는 부팅 시 설정할 옵션을 완전히 제어할 수 있습니다. 특정 커널 동작은 부팅 시에만 설정할 수 있으므로 이러한 변경을 수행하는 방법을 이해하는 것이 중요한 관리 기술입니다.



중요

커널 명령줄 매개 변수를 수정하여 시스템의 동작을 변경하는 것을 선택하면 시스템에 부정적인 영향을 미칠 수 있습니다. 따라서 프로덕션에 배포하기 전에 변경 사항을 테스트해야 합니다. 자세한 내용은 Red Hat 지원팀에 문의하십시오.

5.1. 커널 명령줄 매개변수 이해

커널 명령줄 매개변수는 다음과 같은 부팅 시간 설정에 사용됩니다.

- Red Hat Enterprise Linux 커널
- 초기 RAM 디스크
- 사용자 공간 기능

커널 부팅 시간 매개 변수는 기본값을 덮어쓰고 특정 하드웨어 설정을 설정하는 데 주로 사용됩니다.

기본적으로 GRUB2 부트로더를 사용하는 시스템의 커널 명령줄 매개 변수는 각 커널 부팅 항목에 대해 `/boot/grub2/grubenv` 파일의 **kernelopts** 변수에 정의됩니다.



참고

IBM Z의 경우 **zipl** 부트로더가 환경 변수를 지원하지 않기 때문에 커널 명령줄 매개 변수가 부팅 항목 구성 파일에 저장됩니다. 따라서 **kernelopts** 환경 변수를 사용할 수 없습니다.

추가 리소스

- **kernel-command-line(7)**, **bootparam(7)** 및 **dracut.cmdline(7)** 매뉴얼 페이지
- [Red Hat Enterprise Linux 8에서 사용자 정의 커널 설치 및 부팅 방법](#)

5.2. GRUBBY의 정의

grubby는 부트 로더별 구성 파일을 조작하는 유틸리티입니다.

grubby를 사용하여 기본 부팅 항목을 변경하고 GRUB2 메뉴 항목에서 인수를 추가/제거할 수도 있습니다.

자세한 내용은 **grubby(8)** 매뉴얼 페이지를 참조하십시오.

5.3. 어떤 부팅 항목

부팅 항목은 구성 파일에 저장되어 특정 커널 버전과 연결된 옵션 컬렉션입니다. 실제로는 시스템에 커널을 설치한 만큼의 부팅 항목이 있습니다. 부팅 항목 구성 파일은 `/boot/loader/entries/` 디렉토리에 있으며 다음과 같이 표시됩니다.

```
6f9cc9cb7d7845d49698c9537337cedc-4.18.0-5.el8.x86_64.conf
```

위의 파일 이름은 **/etc/machine-id** 파일에 저장된 시스템 ID와 커널 버전으로 구성됩니다.

부트 항목 구성 파일에는 커널 버전, 초기 ramdisk 이미지 및 커널이 **커널 명령줄 매개 변수가 포함된 환경 변수**에 대한 정보가 포함되어 있습니다. 부팅 항목 구성의 예제 내용은 다음과 같습니다.

```
title Red Hat Enterprise Linux (4.18.0-74.el8.x86_64) 8.0 (Ootpa)
version 4.18.0-74.el8.x86_64
linux /vmlinuz-4.18.0-74.el8.x86_64
initrd /initramfs-4.18.0-74.el8.x86_64.img $tuned_initrd
options $kernelopts $tuned_params
id rhel-20190227183418-4.18.0-74.el8.x86_64
grub_users $grub_users
grub_arg --unrestricted
grub_class kernel
```

kernelopts 환경 변수는 **/boot/grub2/grubenv** 파일에 정의되어 있습니다.

추가 리소스

- [Red Hat Enterprise Linux 8에서 사용자 정의 커널 설치 및 부팅 방법](#)

5.4. 모든 부팅 항목의 커널 명령줄 매개변수 변경

다음 절차에서는 시스템의 모든 부팅 항목에 대한 커널 명령줄 매개 변수를 변경하는 방법을 설명합니다.

사전 요구 사항

- **grubby** 유틸리티가 시스템에 설치되어 있는지 확인합니다.
- **zipl** 유틸리티가 IBM Z 시스템에 설치되어 있는지 확인합니다.

절차

- 매개변수를 추가하려면 다음을 수행합니다.

```
# grubby --update-kernel=ALL --args="<NEW_PARAMETER>"
```

GRUB2 부트로더를 사용하는 시스템의 경우 명령은 해당 파일의 kernel **opts** 변수에 새 커널 매개 변수를 추가하여 **/boot/grub2/grubenv** 파일을 업데이트합니다.

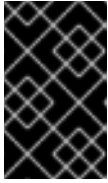
ziPL 부트로더를 사용하는 IBM Z에서 명령은 각 **/boot/loader/entries/<ENTRY>.conf** 파일에 새 커널 매개 변수를 추가합니다.

- IBM Z에서 옵션 없이 **zipl** 명령을 실행하여 부팅 메뉴를 업데이트합니다.
- 매개변수를 제거하려면 다음을 수행합니다.

```
# grubby --update-kernel=ALL --remove-args="<PARAMETER_TO_REMOVE>"
```

- IBM Z에서 옵션 없이 **zipl** 명령을 실행하여 부팅 메뉴를 업데이트합니다.
- 커널 패키지를 업데이트할 때마다 구성된 커널 옵션을 새 커널에 전파합니다.

```
# grub2-mkconfig -o /etc/grub2.cfg
```



중요

새로 설치된 커널은 이전에 구성된 커널에서 커널 명령줄 매개 변수를 상속하지 않습니다. 새로 설치된 커널에서 **grub2-mkconfig** 명령을 실행하여 필요한 매개 변수를 새 커널에 전파해야 합니다.

추가 리소스

- [커널 명령줄 매개 변수 이해](#)
- **grubby(8)** 및 **zipl(8)** 매뉴얼 페이지
- [grubby](#) 툴

5.5. 단일 부팅 항목의 커널 명령줄 매개 변수 변경

다음 절차에서는 시스템에서 단일 부팅 항목에 대한 커널 명령줄 매개 변수를 변경하는 방법을 설명합니다.

사전 요구 사항

- **grubby** 및 **zipl** 유틸리티가 시스템에 설치되어 있는지 확인합니다.

절차

- 매개 변수를 추가하려면 다음을 수행합니다.

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="<NEW_PARAMETER>"
```

- IBM Z에서 옵션 없이 **zipl** 명령을 실행하여 부팅 메뉴를 업데이트합니다.

- 매개 변수를 제거하려면 다음을 사용합니다.

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --remove-args="
<PARAMETER_TO_REMOVE>"
```

- IBM Z에서 옵션 없이 **zipl** 명령을 실행하여 부팅 메뉴를 업데이트합니다.



참고

grub.cfg 파일을 사용하는 시스템에는 기본적으로 각 커널 부팅 항목에 대한 **options** 매개 변수가 있으며, 이 매개 변수는 **kernelopts** 변수로 설정됩니다. 이 변수는 **/boot/grub2/grubenv** 구성 파일에 정의되어 있습니다.

중요

GRUB2 시스템에서:

- 커널 명령줄 매개 변수가 모든 부팅 항목에 대해 수정되면 **grubby** 유틸리티는 **/boot/grub2/grubenv** 파일에서 **kernelopts** 변수를 업데이트합니다.
- 커널 명령줄 매개 변수가 단일 부팅 항목으로 수정되면 **kernelopts** 변수가 확장되고 커널 매개 변수가 수정되고 결과 값은 해당 부팅 항목의 **/boot/loader/entries/<RELEVANT_KERNEL_BOOT_ENTRY.conf>** 파일에 저장됩니다.

ziPL 시스템에서:

- **grubby** 는 **/boot/loader/entries/<ENTRY>.conf** 파일에 개별 커널 부팅 항목의 커널 명령줄 매개 변수를 수정하고 저장합니다.

추가 리소스

- [커널 명령줄 매개변수 이해](#)
- **grubby(8)** 및 **zipl(8)** 매뉴얼 페이지
- [grubby](#) 툴

5.6. 부팅 시 일시적으로 커널 명령줄 매개변수 변경

다음 절차에 따라 단일 부팅 프로세스에서만 커널 매개 변수를 변경하여 커널 메뉴 항목을 일시적으로 변경할 수 있습니다.

절차

1. GRUB 2 부팅 메뉴가 나타나면 시작할 커널을 선택하고 **e** 키를 눌러 커널 매개 변수를 편집합니다.
2. 커서를 아래로 이동하여 커널 명령행을 찾습니다. 커널 명령줄은 64비트 IBM Power Series 및 x86-64 BIOS 기반 시스템에서 **linux** 로 시작하거나 UEFI 시스템에서 **linuxefi** 로 시작합니다.
3. 커서를 행 끝으로 이동합니다.



참고

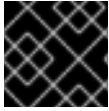
Ctrla를 눌러 행의 시작 부분으로 건너뛰고 **Ctrl**e를 눌러 행의 끝으로 건너뛰니다. 일부 시스템에서 **Home** 및 **End** 키도 작동할 수 있습니다.

4. 필요에 따라 커널 매개 변수를 편집합니다. 예를 들어 시스템을 긴급 모드로 실행하려면 **linux** 줄 끝에 **emergency** 매개변수를 추가합니다.

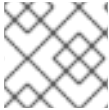
```
linux ($root)/vmlinuz-4.18.0-348.12.2.el8_5.x86_64 root=/dev/mapper/rhel-root ro
crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
rhgb quiet emergency
```

시스템 메시지를 사용하려면 **rhgb** 및 **quiet** 매개 변수를 제거합니다.

5. **Ctrl**x를 눌러 선택한 커널과 수정된 명령행 매개 변수를 사용하여 부팅합니다.

**중요**

Esc 키를 눌러 명령줄 편집을 그대로 두고 변경한 사용자가 모두 삭제됩니다.

**참고**

이 절차는 단일 부팅에만 적용되며 영구적으로 변경하지 않습니다.

5.7. 직렬 콘솔 연결을 사용하도록 GRUB 2 설정 설정

직렬 콘솔은 헤드리스 서버 또는 임베디드 시스템에 연결해야 하고 네트워크가 다운된 경우 유용합니다. 또는 보안 규칙을 방지하고 다른 시스템에서 로그인 액세스 권한을 얻어야 합니다.

직렬 콘솔 연결을 사용하도록 몇 가지 기본 GRUB 2 설정을 구성해야 합니다.

사전 요구 사항

- 루트 권한이 있습니다.

절차

1. **/etc/default/grub** 파일에 다음 두 행을 추가합니다.

```
GRUB_TERMINAL="serial"
GRUB_SERIAL_COMMAND="serial --speed=9600 --unit=0 --word=8 --parity=no --stop=1"
```

첫 번째 줄에서는 그래픽 터미널을 비활성화합니다. **GRUB_TERMINAL** 키는 **GRUB_TERMINAL_INPUT** 및 **GRUB_TERMINAL_OUTPUT** 키의 값을 재정의합니다.

두 번째 줄은 baud 속도(**--speed**), 패리티 및 기타 값을 사용자 환경과 하드웨어에 맞게 조정합니다. 예를 들어 115200과 같은 작업에는 훨씬 높은 세례 비율을 사용하는 것이 다음 로그 파일과 같은 작업에 적합합니다.

2. GRUB 2 설정 파일을 업데이트합니다.

- BIOS 기반 시스템에서 다음을 수행합니다.

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

- UEFI 기반 시스템에서 다음을 수행합니다.

```
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

3. 변경 사항을 적용하려면 시스템을 재부팅합니다.

6장. 런타임 시 커널 매개변수 구성

시스템 관리자는 런타임 시 Red Hat Enterprise Linux 커널 동작의 많은 부분을 수정할 수 있습니다. 이 섹션에서는 **sysctl** 명령을 사용하고 **/etc/sysctl.d/** 및 **/proc/sys/** 디렉터리의 구성 파일을 수정하여 런타임에 커널 매개 변수를 구성하는 방법을 설명합니다.

6.1. 커널 매개변수란 무엇입니까

커널 매개변수는 시스템이 실행되는 동안 조정할 수 있는 튜닝 가능한 값입니다. 변경 사항을 적용하려면 커널을 재부팅하거나 다시 컴파일할 필요가 없습니다.

다음은 통해 커널 매개변수를 처리할 수 있습니다.

- **sysctl** 명령
- **/proc/sys/** 디렉토리에 마운트된 가상 파일 시스템
- **/etc/sysctl.d/** 디렉토리에 있는 구성 파일

튜닝 가능 항목은 커널 하위 시스템에 의해 클래스로 나뉩니다. Red Hat Enterprise Linux에는 다음과 같은 조정 가능한 클래스가 있습니다.

표 6.1. sysctl 클래스 테이블

튜닝 가능 클래스	하위 시스템
abi	실행 도메인 및 개인 정보
암호화	암호화 인터페이스
debug	커널 디버깅 인터페이스
dev	장치 관련 정보
fs	전역 및 특정 파일 시스템 튜닝 가능 항목
kernel	글로벌 커널 튜닝 가능 항목
net	네트워크 튜닝 가능 항목
sunrpc	원격 프로시저 호출(NFS)
user	사용자 네임 스페이스 제한
vm	메모리, 버퍼 및 캐시 튜닝 및 관리



중요

프로덕션 시스템에서 커널 매개 변수를 구성하려면 신중하게 계획해야 합니다. 예기치 않은 변경으로 커널을 불안정하게 만들 수 있으므로 시스템을 재부팅해야 합니다. 커널 값을 변경하기 전에 유효한 옵션을 사용 중인지 확인합니다.

추가 리소스

- [sysctl\(8\)](#) 및 [sysctl.d\(5\)](#) 도움말 페이지

6.2. SYSCTL을 사용하여 일시적으로 커널 매개변수 구성

다음 절차에서는 **sysctl** 명령을 사용하여 런타임 시 커널 매개 변수를 일시적으로 설정하는 방법을 설명합니다. 명령은 튜닝 가능 항목을 나열하고 필터링하는 데도 유용합니다.

사전 요구 사항

- 루트 권한

절차

1. 모든 매개변수와 해당 값을 나열하려면 다음을 사용합니다.

```
# sysctl -a
```



참고

sysctl -a 명령은 런타임 및 부팅 시 조정할 수 있는 커널 매개 변수를 표시합니다.

2. 매개 변수를 일시적으로 구성하려면 다음 예와 같이 명령을 사용합니다.

```
# sysctl <TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE>
```

위의 샘플 명령은 시스템이 실행되는 동안 매개 변수 값을 변경합니다. 변경 사항은 재시작 없이 즉시 적용됩니다.



참고

시스템이 재부팅되면 변경 사항은 기본값으로 돌아갑니다.

추가 리소스

- [sysctl\(8\)](#) 도움말 페이지
- [sysctl을 사용하여 영구적으로 커널 매개변수 구성](#)
- [/etc/sysctl.d/의 구성 파일을 사용하여 커널 매개변수 조정](#)

6.3. SYSCTL을 사용하여 영구적으로 커널 매개변수 구성

다음 절차에서는 **sysctl** 명령을 사용하여 커널 매개 변수를 영구적으로 설정하는 방법을 설명합니다.

사전 요구 사항

- 루트 권한

절차

1. 모든 매개변수를 나열하려면 다음을 사용합니다.

```
# sysctl -a
```

명령은 런타임 시 구성할 수 있는 모든 커널 매개 변수를 표시합니다.

2. 매개변수를 영구적으로 구성하려면 다음을 수행합니다.

```
# sysctl -w <TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE> >> /etc/sysctl.conf
```

sample 명령은 튜닝 가능 항목을 변경하고 커널 매개 변수의 기본값을 재정의하는 **/etc/sysctl.conf** 파일에 씁니다. 변경 사항은 재시작 없이 즉시 영구적으로 적용됩니다.



참고

커널 매개 변수를 영구적으로 수정하려면 **/etc/sysctl.d/** 디렉터리의 구성 파일을 수동으로 변경할 수도 있습니다.

추가 리소스

- **sysctl(8)** 및 **sysctl.conf(5)** 도움말 페이지
- [/etc/sysctl.d/의 구성 파일을 사용하여 커널 매개변수 조정](#)

6.4. /ETC/SYSCTL.D/의 구성 파일을 사용하여 커널 매개변수 조정

다음 절차에서는 커널 매개 변수를 영구적으로 설정하기 위해 **/etc/sysctl.d/** 디렉터리에서 구성 파일을 수동으로 수정하는 방법을 설명합니다.

사전 요구 사항

- 루트 권한

절차

1. **/etc/sysctl.d/**에 새 구성 파일을 만듭니다.

```
# vim /etc/sysctl.d/<some_file.conf>
```

2. 다음과 같이 행당 하나씩 커널 매개변수를 포함합니다.

```
<TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE>
<TUNABLE_CLASS>.<PARAMETER>=<TARGET_VALUE>
```

3. 구성 파일을 저장합니다.
4. 시스템을 재부팅하여 변경 사항을 적용합니다.

- 또는 재부팅하지 않고 변경 사항을 적용하려면 다음을 실행합니다.

```
# sysctl -p /etc/sysctl.d/<some_file.conf>
```

명령을 사용하면 이전에 만든 구성 파일의 값을 읽을 수 있습니다.

추가 리소스

- [sysctl\(8\), sysctl.d\(5\) 도움말 페이지](#)

6.5. /PROC/SYS/를 통해 일시적으로 커널 매개변수 구성

다음 절차에서는 가상 파일 시스템 **/proc/sys/** 디렉터리의 파일을 통해 커널 매개 변수를 임시로 설정하는 방법을 설명합니다.

사전 요구 사항

- 루트 권한

절차

1. 설정할 커널 매개변수를 식별합니다.

```
# ls -l /proc/sys/<TUNABLE_CLASS>/
```

명령에서 반환한 쓰기 가능한 파일은 커널을 구성하는 데 사용할 수 있습니다. 읽기 전용 권한이 있는 파일은 현재 설정에 대한 피드백을 제공합니다.

2. 커널 매개변수에 target 값을 할당합니다.

```
# echo <TARGET_VALUE> > /proc/sys/<TUNABLE_CLASS>/<PARAMETER>
```

명령은 시스템을 다시 시작하면 구성 변경이 사라집니다.

3. 선택적으로 새로 설정된 커널 매개 변수의 값을 확인합니다.

```
# cat /proc/sys/<TUNABLE_CLASS>/<PARAMETER>
```

추가 리소스

- [sysctl을 사용하여 영구적으로 커널 매개변수 구성](#)
- [/etc/sysctl.d/의 구성 파일을 사용하여 커널 매개변수 조정](#)

7장. 가상화 환경에서 커널 패닉 매개변수를 비활성화 상태로 유지

RHEL 8에서 가상화 환경을 구성하는 경우, 가상화된 환경에서 시스템 패닉이 필요 없는 무분별한 소프트웨어 잠금을 트리거할 수 있으므로 **softlockup_panic** 및 **nmi_watchdog** 커널 매개 변수를 활성화해서는 안 됩니다.

다음 섹션에서는 다음 사항을 요약하여 이 조언 뒤에 있는 이유를 설명합니다.

- 소프트웨어 잠금의 원인은 무엇입니까.
- 소프트웨어 잠금에서 시스템의 동작을 제어하는 커널 매개 변수 설명.
- 가상화 환경에서 소프트웨어 잠금이 트리거되는 방법을 설명합니다.

7.1. 소프트웨어 잠금이란 무엇입니까

소프트 잠금은 작업이 일정 변경 없이 CPU의 커널 공간에서 실행될 때 일반적으로 버그로 인해 발생합니다. 이 작업은 또한 해당 특정 CPU에서 다른 작업을 실행하도록 허용하지 않습니다. 결과적으로 시스템 콘솔을 통해 사용자에게 경고가 표시됩니다. 이 문제를 소프트웨어 잠금 실행이라고도 합니다.

추가 리소스

- [CPU 소프트웨어 잠금이란 무엇입니까?](#)

7.2. 커널 패닉을 제어하는 매개변수

다음 커널 매개 변수를 설정하여 소프트웨어 잠금이 감지될 때 시스템의 동작을 제어할 수 있습니다.

softlockup_panic

소프트 잠금이 감지될 때 커널이 패닉될지 여부를 제어합니다.

유형	현재의	효과
정수	0	소프트 잠금 시 커널이 패닉되지 않음
정수	1	소프트 잠금 시 커널 패닉

기본적으로 RHEL8에서 이 값은 0입니다.

패닉을 해결하려면 시스템에서 하드 잠금을 먼저 감지해야 합니다. 탐지는 **nmi_watchdog** 매개 변수로 제어합니다.

nmi_watchdog

잠금 감지 메커니즘(워치독)이 활성화 상태인지 여부를 제어합니다. 이 매개변수는 정수 유형입니다.

현재의	효과
0	잠금 장치 비활성화
1	잠금 장치 사용

하드 잠금 변환기는 각 CPU에서 인터럽트에 응답하는 기능을 모니터링합니다.

watchdog_thresh

watchdog **hrtimer**, NMI 이벤트 및 소프트/하드 잠금 임계 값의 빈도를 제어합니다.

기본 임계값	소프트 잠금 임계값
10초	$2 * \text{watchdog_thresh}$

이 매개 변수를 0으로 설정 하면 완전히 잠금 탐지가 비활성화됩니다.

추가 리소스

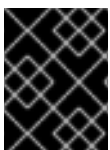
- [소프트 잠금 해제 및 하드 잠금업 사진](#)
- [커널 `sysctl`](#)

7.3. 가상 환경에서 끊기고 있는 소프트 잠금

[What is a soft lockup](#)에 설명된 대로 물리적 호스트에서 실행되는 소프트 잠금은 일반적으로 커널 또는 하드웨어 버그를 나타냅니다. 가상화 환경의 게스트 운영 체제에서도 발생하는 것과 동일한 현상은 잘못된 경고를 나타낼 수 있습니다.

호스트의 과부하가 많거나 메모리와 같은 특정 리소스에 대한 높은 경합으로 인해 일반적으로 가상의 소프트 잠금이 실행됩니다. 호스트가 20초 이상 게스트 CPU를 예약할 수 있기 때문입니다. 그런 다음 게스트 CPU를 다시 호스트에서 실행하도록 예약하면 *시간 건너뛰기가 발생하여 타이머로 인해 트리거됩니다*. 타이머에는 watchdog **hrtimer**도 포함되어 있으며 이는 결과적으로 게스트 CPU에 소프트 잠금을 보고할 수 있습니다.

가상화 환경의 소프트 잠금은 거부할 수 있으므로 소프트 잠금이 게스트 CPU에 보고될 때 시스템 패닉을 일으키는 커널 매개 변수를 활성화해서는 안 됩니다.



중요

게스트의 소프트 잠금을 이해하려면 호스트가 게스트를 작업으로 예약하고 게스트가 자체 작업을 예약한다는 것을 알아야 합니다.

추가 리소스

- [소프트 잠금이란 무엇입니까](#)
- [가상 머신 구성 요소 및 상호 작용](#)
- [가상 머신에서 "BUG: 소프트 잠금"을 보고합니다](#)

8장. 데이터베이스 서버의 커널 매개변수 조정

특정 데이터베이스 애플리케이션의 성능에 영향을 줄 수 있는 다양한 커널 매개 변수 세트가 있습니다. 다음 섹션에서는 데이터베이스 서버 및 데이터베이스의 효율적인 운영을 보호하기 위해 구성하는 커널 매개 변수에 대해 설명합니다.

8.1. 데이터베이스 서버 소개

데이터베이스 서버는 DBMS(데이터베이스 관리 시스템)의 기능을 제공하는 서비스입니다. DBMS는 데이터베이스 관리를 위한 유틸리티를 제공하며 최종 사용자, 애플리케이션 및 데이터베이스와 상호 작용합니다.

Red Hat Enterprise Linux 8은 다음과 같은 데이터베이스 관리 시스템을 제공합니다.

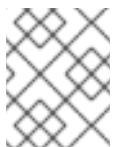
- MariaDB 10.3
- MariaDB 10.5 - RHEL 8.4 이후 사용 가능
- MySQL 8.0
- PostgreSQL 10
- PostgreSQL 9.6
- PostgreSQL 12 - RHEL 8.1.1 이후 사용 가능
- PostgreSQL 13 - RHEL 8.4 이후 사용 가능

8.2. 데이터베이스 애플리케이션 성능에 영향을 주는 매개변수

다음 커널 매개 변수는 데이터베이스 애플리케이션의 성능에 영향을 줍니다.

fs.aio-max-nr

시스템이 서버에서 처리할 수 있는 최대 비동기 I/O 작업 수를 정의합니다.



참고

fs.aio-max-nr 매개 변수를 늘리면 aio 제한을 늘리는 것 외에도 추가 변경이 발생하지 않습니다.

fs.file-max

시스템이 어떠한 인스턴스에서든 지원하는 최대 파일 프로세스(임시 파일 이름 또는 열린 파일에 할당된 ID) 수를 정의합니다.

커널은 애플리케이션에서 파일 핸들을 요청할 때마다 파일 핸들을 동적으로 할당합니다. 하지만 이 커널은 애플리케이션에서 릴리스될 때 이러한 파일 처리를 해제하지 않습니다. 커널은 이러한 파일을 대신 재사용합니다. 즉, 현재 사용된 파일 처리 수가 적을 수도 있지만 할당된 파일 핸들의 총 수가 증가할 수 있습니다.

kernel.shmall

시스템 전체에서 사용할 수 있는 총 공유 메모리 페이지 수를 정의합니다. 전체 주 메모리를 사용하려면 **kernel.shmall** 매개 변수의 값이 총 주 메모리 크기여야 합니다.

kernel.shmmax

Linux 프로세스가 가상 주소 공간에 할당할 수 있는 단일 공유 메모리 세그먼트의 최대 크기(바이트)를 정의합니다.

kernel.shmni

데이터베이스 서버가 처리할 수 있는 최대 공유 메모리 세그먼트 수를 정의합니다.

net.ipv4.ip_local_port_range

시스템이 특정 포트 번호 없이 데이터베이스 서버에 연결하려는 프로그램에 사용할 수 있는 포트 범위를 정의합니다.

net.core.rmem_default

TCP(Transmission Control Protocol)를 통해 기본 수신 소켓 메모리를 정의합니다.

net.core.rmem_max

TCP(Transmission Control Protocol)를 통해 최대 수신 소켓 메모리를 정의합니다.

net.core.wmem_default

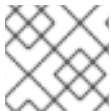
TCP(Transmission Control Protocol)를 통해 기본 전송 소켓 메모리를 정의합니다.

net.core.wmem_max

TCP(Transmission Control Protocol)를 통해 최대 전송 소켓 메모리를 정의합니다.

vm.dirty_bytes / vm.dirty_ratio

더티 데이터를 생성하는 프로세스가 **write()** 함수에서 시작되는 더티 가능 메모리의 백분율(바이트) 단위로 임계값을 정의합니다.



참고

vm.dirty_bytes 또는 **vm.dirty_ratio** 는 한 번에 지정할 수 있습니다.

vm.dirty_background_bytes / vm.dirty_background_ratio

커널이 하드 디스크에 더티 데이터를 적극적으로 쓰려고 하는 더티 가능 메모리의 백분율 / 바이트 단위의 임계값을 정의합니다.



참고

vm.dirty_background_bytes 또는 **vm.dirty_background_ratio** 는 한 번에 지정할 수 있습니다.

vm.dirty_writeback_centisecs

하드 디스크에 더티 데이터를 작성하는 커널 스레드의 주기적인 시작 간격 간 시간 간격을 정의합니다. 이 커널 매개 변수는 초당 100번째로 측정합니다.

vm.dirty_expire_centisecs

더티 데이터가 하드 디스크에 쓸 수 있을 만큼 오래된 시간을 정의합니다. 이 커널 매개 변수는 초당 100번째로 측정합니다.

추가 리소스

- [더티 페이지 캐시 나중 쓰기백 및 vm.dirty 매개변수](#)

9장. 커널 로깅 시작하기

로그 파일은 커널, 서비스 및 실행 중인 애플리케이션을 포함하여 시스템에 대한 메시지를 포함하는 파일입니다. Red Hat Enterprise Linux의 로깅 시스템은 기본 제공된 **syslog** 프로토콜을 기반으로 합니다. 다양한 유틸리티에서 이 시스템을 사용하여 이벤트를 기록하고 이를 로그 파일로 구성합니다. 이러한 파일은 운영 체제를 감사하거나 문제를 해결할 때 유용합니다.

9.1. 커널 링 버퍼란 무엇입니까

부팅 프로세스 중에 콘솔은 시스템 시작의 초기 단계에 대한 많은 중요한 정보를 제공합니다. 초기 메시지가 손실되는 것을 방지하기 위해 커널은 링 버퍼라는 것을 활용합니다. 이 버퍼는 커널 코드 내에서 **printk()** 함수에 의해 생성된 부팅 메시지를 포함하여 모든 메시지를 저장합니다. 그런 다음, 커널 링 버퍼의 메시지는 **syslog** 서비스에서 영구저장장치의 로그 파일에 읽고 저장됩니다.

위에서 언급한 버퍼는 크기가 고정되어 있으며 커널로 하드 코딩되는 재활용 데이터 구조입니다. 사용자는 **dmesg** 명령 또는 **/var/log/boot.log** 파일을 통해 커널 링 버퍼에 저장된 데이터를 표시할 수 있습니다. 링 버퍼가 가득 차면 새 데이터가 이전 데이터를 덮어씁니다.

추가 리소스

- **syslog(2)** 및 **dmesg(1)** 매뉴얼 페이지

9.2. 로그 수준 및 커널 로깅에 대한 PRINTK 역할

커널에서 보고하는 각 메시지에는 메시지의 중요성을 정의하는 로그 레벨이 연결되어 있습니다. [What is kernel ring buffer에 설명된 커널 링 버퍼](#)는 모든 로그 수준의 커널 메시지를 수집합니다. 버퍼에서 보내는 메시지를 콘솔에 인쇄하는 **kernel.printk** 매개 변수입니다.

로그 수준 값은 다음 순서로 구분됩니다.

- 0 - 커널 긴급. 시스템을 사용할 수 없습니다.
- 1 - 커널 경고. 즉시 조치를 취해야 합니다.
- 2 - 커널 조건이 중요한 것으로 간주됩니다.
- 3 - 일반적인 커널 오류 상태.
- 4 - 일반 커널 경고 조건.
- 5 - 정상이지만 심각한 상태의 커널 알림.
- 6 - 커널 정보 메시지.
- 7 - 커널 디버그 수준 메시지.

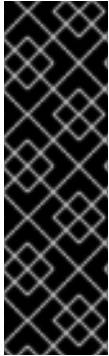
기본적으로 RHEL 8의 **kernel.printk**에는 다음 네 가지 값이 포함되어 있습니다.

```
# sysctl kernel.printk
kernel.printk = 7 4 1 7
```

4개의 값은 다음을 정의합니다.

1. 값. 콘솔 로그 수준은 콘솔에 인쇄된 메시지의 우선 순위를 가장 낮은 우선 순위를 정의합니다.

2. 값. 명시적인 로그 수준이 연결되지 않은 메시지의 기본 로그 수준입니다.
3. 값. 콘솔 로그 레벨에 가능한 최저 로그 수준 구성을 설정합니다.
4. 값. 부팅 시 콘솔 로그 수준의 기본값을 설정합니다.
위의 각 값은 오류 메시지를 처리하는 다른 규칙을 정의합니다.



중요

기본값 **7 4 1 7 printk** 값을 사용하면 커널 활동을 더 효과적으로 디버깅할 수 있습니다. 그러나 직렬 콘솔과 결합되면 이 인쇄기 **설정**은 RHEL 시스템이 일시적으로 응답하지 않을 수 있는 강력한 I/O 버스트를 유발할 수 있습니다. 이러한 상황을 방지하기 위해 **인쇄자** 값을 **4 4 1 7**로 설정하는 것이 일반적으로 작동하지만 추가 디버깅 정보를 손실하는 비용이 발생합니다.

또한 **quiet** 또는 **debug** 와 같은 특정 커널 명령행 매개 변수도 기본 **kernel.printk** 값을 변경합니다.

추가 리소스

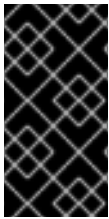
- **syslog(2)** 매뉴얼 페이지

10장. KDUMP 설치

kdump 서비스는 새로운 Red Hat Enterprise Linux 설치 시 기본적으로 설치 및 활성화됩니다. 다음 섹션에서는 **kdump** 가 무엇이며, 기본적으로 설정되지 않은 경우 **kdump** 를 설치하는 방법에 대해 설명합니다.

10.1. KDUMP란 무엇입니까?

kdump 는 크래시 덤프 메커니즘을 제공하는 서비스입니다. 이 서비스를 사용하면 분석을 위해 시스템 메모리의 내용을 저장할 수 있습니다. **kdump** 는 **kexec** 시스템 호출을 사용하여 재부팅하지 않고 두 번째 커널(커널 캡처 커널)로 부팅한 다음 충돌된 커널의 메모리 콘텐츠를 캡처하여 파일에 저장합니다. 두 번째 커널은 시스템 메모리의 예약된 부분에 상주합니다.



중요

커널 크래시 덤프는 시스템 장애 발생 시(심각한 버그)에서 사용할 수 있는 유일한 정보일 수 있습니다. 따라서 운영 **kdump** 는 미션 크리티컬한 환경에서 중요합니다. Red Hat은 시스템 관리자가 일반 커널 업데이트 주기에서 **kexec-tools** 를 정기적으로 업데이트하고 테스트할 것을 권장합니다. 이는 새로운 커널 기능을 구현할 때 특히 중요합니다.

머신의 설치된 모든 커널 또는 지정된 커널에서만 **kdump** 를 활성화할 수 있습니다. 이 기능은 시스템에 사용된 커널이 여러 개 있을 때 유용하며 그 중 일부는 충돌할 수 있다는 우려가 없을 정도로 안정적입니다.

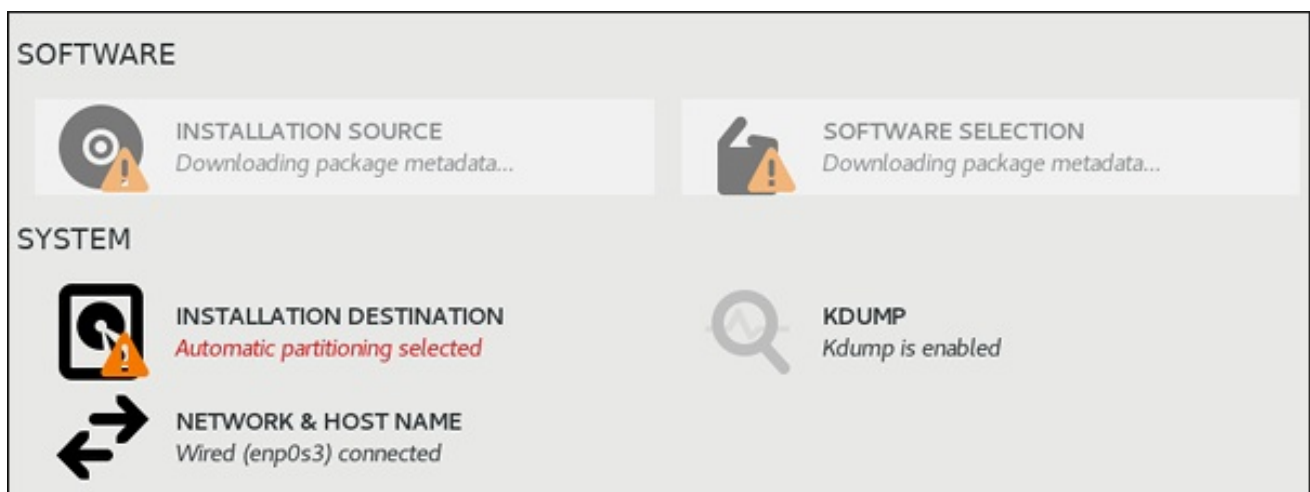
kdump 가 설치되면 기본 **/etc/kdump.conf** 파일이 생성됩니다. 파일에는 기본 최소 **kdump** 구성이 포함되어 있습니다. 이 파일을 편집하여 **kdump** 구성을 사용자 지정할 수 있지만 필수는 아닙니다.

10.2. ANACONDA 를 사용하여 KDUMP 설치

Anaconda 설치 프로그램은 대화식 설치 중에 **kdump** 구성을 위한 그래픽 인터페이스 화면을 제공합니다. 그러나 제한된 구성만 허용됩니다.

절차

1. **Kdump** 필드로 이동합니다.
2. **kdump** 를 활성화합니다.
3. **kdump** 용으로 예약해야 하는 메모리 양을 정의합니다.



10.3. 명령줄에 KDUMP 설치

사용자 지정 **Kickstart** 설치와 같은 일부 설치 옵션은 기본적으로 **kdump** 를 설치하거나 **활성화하지** 않는 경우도 있습니다. 이 경우 다음 절차를 따르십시오.

사전 요구 사항

- 활성 RHEL 서브스크립션
- **kexec-tools** 패키지
- **kdump** 구성 및 대상에 대한 요구 사항을 충족했습니다. 자세한 내용은 [지원되는 kdump 구성 및 대상](#)을 참조하십시오.

절차

1. **kdump** 가 시스템에 설치되어 있는지 확인하십시오.

```
$ rpm -q kexec-tools
```

패키지가 설치된 경우 출력합니다.

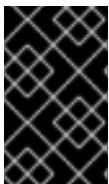
```
kexec-tools-2.0.17-11.el8.x86_64
```

패키지가 설치되지 않은 경우 출력합니다.

```
package kexec-tools is not installed
```

2. 다음을 수행하여 **kdump** 및 기타 필요한 패키지를 설치합니다.

```
# yum install kexec-tools
```



중요

kernel-3.10.0-693.el7부터 **kdump** 에서 **Intel IOMMU** 드라이버가 지원됩니다. 이전 버전의 경우 kernel-3.10.0-514[.XYZ].el7 및 이전 버전의 경우 **Intel IOMMU** 지원이 비활성화되어 있지 않으면 캡처 커널이 응답하지 않을 가능성이 높습니다.

11장. 명령줄에서 KDUMP 구성

다음 섹션에서는 **kdump** 환경을 계획 및 구축하는 방법을 설명합니다.

11.1. KDUMP 크기 예상

kdump 환경을 계획하고 빌드할 때 크래시 덤프 파일에 필요한 공간을 파악하는 것이 중요합니다.

makedumpfile --mem-usage 명령은 크래시 덤프 파일에 필요한 공간을 추정합니다. 메모리 사용량 보고서를 생성합니다. 이 보고서는 덤프 수준과 제외될 안전한 페이지를 결정하는 데 도움이 됩니다.

절차

- 다음 명령을 실행하여 메모리 사용량 보고서를 생성합니다.

```
# makedumpfile --mem-usage /proc/kcore
```

TYPE	PAGES	EXCLUDABLE	DESCRIPTION
ZERO	501635	yes	Pages filled with zero
CACHE	51657	yes	Cache pages
CACHE_PRIVATE	5442	yes	Cache pages + private
USER	16301	yes	User process pages
FREE	77738211	yes	Free pages
KERN_DATA	1333192	no	Dumpable kernel data

중요

makedumpfile --mem-usage 명령은 페이지에 필요한 메모리를 보고합니다. 즉, 커널 페이지 크기에 대해 사용 중인 메모리 크기를 계산해야 합니다.

기본적으로 RHEL 커널은 AMD64 및 Intel 64 CPU 아키텍처에서 4KB 크기의 페이지를 사용하고 IBM POWER 아키텍처에서 64KB 크기의 페이지를 사용합니다.

11.2. KDUMP 메모리 사용량 구성

kdump의 메모리는 시스템 부팅 중에 예약됩니다. 메모리 크기는 GRUB(Grand Unified Bootloader) 2 구성 파일에서 구성됩니다. 메모리 크기는 구성 파일에 지정된 **crashkernel=** 옵션 값과 시스템 실제 메모리 크기에 따라 달라집니다.

crashkernel= 옵션은 다양한 방법으로 정의할 수 있습니다. **crashkernel=** 값을 지정하거나 **auto** 옵션을 구성할 수 있습니다. **crashkernel=auto** 매개 변수는 시스템의 실제 메모리 크기에 따라 자동으로 메모리를 예약합니다. 구성된 경우 커널은 캡처 커널에 필요한 메모리 양을 자동으로 예약합니다. 이렇게 하면 OOM(Out-of-Memory) 오류가 발생하지 않도록 방지할 수 있습니다.



참고

kdump의 자동 메모리 할당은 시스템 하드웨어 아키텍처 및 사용 가능한 메모리 크기에 따라 다릅니다.

예를 들어 AMD64 및 Intel 64에서는 사용 가능한 메모리가 1GB 이상인 경우에만 **crashkernel=auto** 매개 변수가 작동합니다. 64비트 ARM 아키텍처 및 IBM Power Systems에는 2GB 이상의 사용 가능한 메모리가 필요합니다.

시스템에 자동 할당의 최소 메모리 임계값보다 적은 경우 예약된 메모리 양을 수동으로 구성할 수 있습니다.

사전 요구 사항

- 루트 권한.
- kdump** 구성 및 대상에 대한 요구 사항을 충족했습니다. 자세한 내용은 [지원되는 kdump 구성 및 대상을](#) 참조하십시오.

절차

1. **/etc/default/grub** 파일을 편집합니다.

2. **crashkernel=** 옵션을 설정합니다.

예를 들어 128MB의 메모리를 예약하려면 다음을 사용합니다.

```
crashkernel=128M
```

또는 설치된 메모리의 총 크기에 따라 예약된 메모리 양을 변수로 설정할 수 있습니다. 메모리 예약을 변수에 대한 구문은 **crashkernel=<range1> :< size1>,< range2>:<size2>** 입니다. 예를 들면 다음과 같습니다.

```
crashkernel=512M-2G:64M,2G-:128M
```

위 예제에서는 총 시스템 메모리가 512MB에서 2GB 사이인 경우 64MB의 메모리를 예약합니다. 총 메모리 양이 2GB를 초과하면 128MB가 예약됩니다.

- 예약된 메모리를 오프셋합니다.
일부 시스템은 **crashkernel** 예약이 매우 일찍 시작되므로 특정 고정 오프셋으로 메모리를 예약해야 하며, 특별한 용도로 일부 영역을 예약하려고 합니다. 오프셋이 설정되면 예약된 메모리가 여기서 시작됩니다. 예약된 메모리를 오프셋하려면 다음 구문을 사용하십시오.

```
crashkernel=128M@16M
```

위의 예에서 **kdump**는 16MB(실제 주소 0x01000000)부터 128MB의 메모리를 예약합니다. **offset** 매개 변수가 0으로 설정되거나 완전히 생략된 경우 **kdump**는 예약된 메모리를 자동으로 오프셋합니다. 변수 메모리 예약을 설정할 때 이 구문을 사용할 수도 있습니다. 이 경우 오프셋은 항상 마지막으로 지정됩니다(예: **crashkernel=512M-2G:64M,2G-:128M@16M**).

3. 다음 명령을 사용하여 GRUB2 설정 파일을 업데이트합니다.

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
```



참고

kdump 용 메모리를 구성하는 다른 방법은 모든 부팅 항목을 업데이트할 **grub2-editenv** 명령을 사용하여 **kernelopts** 변수에 **crashkernel=<SOME_VALUE>** 매개변수를 추가하는 것입니다. 또는 **grubby** 유틸리티를 사용하여 하나의 부팅 항목, 더 많은 부팅 항목 또는 모든 부팅 항목을 업데이트할 수 있습니다.

추가 리소스

- [kdump의 메모리 요구 사항](#)
- [커널 명령줄 매개 변수 구성](#)
- [grub2-mkconfig 스크립트는 GRUB_CMDLINE_LINUX의 옵션을 자동으로 무시합니다.](#)
- [시스템을 부팅하기 전에 grub에서 부팅 매개 변수를 수동으로 수정하는 방법](#)
- [Red Hat Enterprise Linux 8에서 사용자 정의 커널 설치 및 부팅 방법](#)
- [grubby\(8\) 도움말 페이지](#)

11.3. KDUMP 대상 구성

크래시 덤프는 일반적으로 로컬 파일 시스템에 파일로 저장되고 장치에 직접 작성됩니다. 또는 **NFS** 또는 **SSH** 프로토콜을 사용하여 네트워크를 통해 전송할 크래시 덤프를 설정할 수 있습니다. 크래시 덤프 파일을 유지하기 위한 옵션 중 하나만 한 번에 설정할 수 있습니다. 기본 동작은 로컬 파일 시스템의 **/var/crash/** 디렉터리에 저장하는 것입니다.

사전 요구 사항

- 루트 권한.
- **kdump** 구성 및 대상에 대한 요구 사항을 충족했습니다. 자세한 내용은 [지원되는 kdump 구성 및 대상을](#) 참조하십시오.

절차

- 크래시 덤프 파일을 로컬 파일 시스템의 **/var/crash/** 디렉터리에 저장하려면 **/etc/kdump.conf** 파일을 편집하고 경로를 지정합니다.

```
path /var/crash
```

옵션 경로 **/var/crash** 는 **kdump** 가 크래시 덤프 파일을 저장하는 파일 시스템의 경로를 나타냅니다. **/etc/kdump.conf** 파일에 dump 대상을 지정하면 **경로**가 지정된 덤프 대상과 상대적입니다.

/etc/kdump.conf 파일에 덤프 대상을 지정하지 않으면 **경로** 는 루트 디렉토리의 절대 경로를 나타냅니다. 현재 시스템에 마운트된 내용에 따라 덤프 대상과 조정된 덤프 경로가 자동으로 수행됩니다.

kdump 는 **/var/crash/var/crash** 디렉터리에 크래시 덤프 파일을 저장합니다. 덤프 대상이 **/var/crash** 에 마운트되고 옵션 경로 도 **/etc/kdump.conf** 파일에 **/var/crash** 로 설정됩니다. 예를 들어 다음 인스턴스에서 **ext4** 파일 시스템은 **/var/crash** 에 이미 마운트되어 있고 **경로**는 **/var/crash** 로 설정됩니다.

```
grep -v ^# etc/kdump.conf | grep -v ^$
ext4 /dev/mapper/vg00-varcrashvol
```

```
path /var/crash
core_collector makedumpfile -c --message-level 1 -d 31
```

그러면 **/var/crash/var/crash** 경로가 생성됩니다. 이 문제를 해결하려면 옵션 경로 **/var/crash** 대신 **/var/crash**를 사용하십시오.

- 크래시 덤프를 저장할 로컬 디렉토리를 변경하려면 루트로서 아래에 설명된 대로 **/etc/kdump.conf** 구성 파일을 편집합니다.

1. **#path /var/crash** 행의 시작 부분에서 해시 기호("#")를 제거합니다.
2. 값을 의도한 디렉터리 경로로 바꿉니다. 예를 들면 다음과 같습니다.

```
path /usr/local/cores
```



중요

RHEL 8에서는 kdump systemd 서비스를 시작할 때 **path** 지시문을 사용하여 **kdump** 대상으로 정의된 디렉터리가 있어야 합니다. 그렇지 않으면 서비스가 실패합니다. 이 동작은 서비스를 시작할 때 존재하지 않는 경우 디렉터리가 자동으로 생성된 RHEL의 이전 릴리스와 다릅니다.

- 파일을 다른 파티션에 루트로 작성하려면 아래 설명된 대로 **/etc/kdump.conf** 구성 파일을 편집합니다.
 1. 선택 사항에 따라 **#ext4** 행의 시작 부분에서 해시 기호("#")를 제거합니다.
 - 장치 이름(**#ext4 /dev/vg/lv_kdump** 행)
 - 파일 시스템 레이블(**#ext4 LABEL=/boot** 행)
 - UUID(**# ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937** 행)
 2. 파일 시스템 유형과 장치 이름, 레이블 또는 UUID를 원하는 값으로 변경합니다. 예를 들면 다음과 같습니다.

```
ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937
```



중요

LABEL= 또는 **UUID=** 를 사용하여 스토리지 장치를 지정하는 것이 좋습니다. **/dev/sda3** 과 같은 디스크 장치 이름은 재부팅 시 일관되게 보장되지 않습니다.



중요

IBM Z 하드웨어의 DASD(직접 액세스 스토리지 장치)에 덤프할 때 덤프 장치가 진행하기 전에 **/etc/dasd.conf** 에 올바르게 지정되어야 합니다.

- 장치에 크래시 덤프를 직접 작성하려면 **/etc/kdump.conf** 구성 파일을 편집합니다.
 1. **#raw /dev/vg/lv_kdump** 행의 시작 부분에서 해시 기호("#")를 제거합니다.
 2. 값을 원하는 장치 이름으로 바꿉니다. 예를 들면 다음과 같습니다.

```
raw /dev/sdb1
```

- **NFS** 프로토콜을 사용하여 크래시 덤프를 원격 머신에 저장하려면 **/etc/kdump.conf** 구성 파일을 편집합니다.
 1. **#nfs my.server.com:/export/tmp** 행의 시작 부분에서 해시 기호("#")를 제거합니다.
 2. 값을 유효한 호스트 이름 및 디렉토리 경로로 바꿉니다. 예를 들면 다음과 같습니다.

```
nfs penguin.example.com:/export/cores
```

- **SSH** 프로토콜을 사용하여 크래시 덤프를 원격 머신에 저장하려면 **/etc/kdump.conf** 구성 파일을 편집합니다.
 1. **#ssh user@my.server.com** 행의 시작 부분에서 해시 기호("#")를 제거합니다.
 2. 값을 유효한 사용자 이름과 호스트 이름으로 바꿉니다.
 3. 구성에 **SSH** 키를 포함합니다.
 - **#sshkey /root/.ssh/kdump_id_rsa** 행의 시작 부분에서 해시 기호를 제거합니다.
 - 값을 덤프하려는 서버에서 유효한 키 위치로 변경합니다. 예를 들면 다음과 같습니다.

```
ssh john@penguin.example.com
sshkey /root/.ssh/mykey
```

11.4. KDUMP 코어 수집기 구성

kdump 서비스는 **core_collector** 프로그램을 사용하여 크래시 덤프 이미지를 캡처합니다. RHEL에서 **makedumpfile** 유틸리티는 기본 코어 수집기입니다. 다음과 같이 덤프 파일을 줄이는 데 도움이 됩니다.

- 크래시 덤프 파일의 크기 압축 및 다양한 덤프 수준을 사용하는 필요한 페이지만 복사
- 불필요한 크래시 덤프 페이지 제외
- 크래시 덤프에 포함될 페이지 유형 필터링.

구문

```
core_collector makedumpfile -l --message-level 1 -d 31
```

옵션

- **-c, -l** 또는 **-p, -c** 옵션의 경우 **zlib**, **-l** 옵션의 경우 **lzo** 또는 **snappy**를 사용하여 각 페이지에 의해 압축 덤프 파일 형식을 지정합니다.
- **-d (dump_level)**: 덤프 파일에 복사되지 않도록 페이지를 제외합니다.
- **--message-level**: 메시지 유형을 지정합니다. 이 옵션으로 **message_level**을 지정하여 출력된 출력을 제한할 수 있습니다. 예를 들어 7을 **message_level**으로 지정하면 일반적인 메시지와 오류 메시지가 출력됩니다. **message_level**의 최대 값은 31입니다.

사전 요구 사항

- 루트 권한
- **kdump** 구성 및 대상에 대한 요구 사항을 충족했습니다. 자세한 내용은 [지원되는 kdump 구성 및 대상을](#) 참조하십시오.

절차

1. 루트 로서 **/etc/kdump.conf** 구성 파일을 편집하고 **#core_collector makedumpfile -l --message-level 1 -d 31** 부터 해시 기호("#")를 제거합니다.
2. 크래시 덤프 파일 압축을 활성화하려면 다음을 실행합니다.

```
core_collector makedumpfile -l --message-level 1 -d 31
```

l 옵션은 덤프 압축 파일 형식을 지정합니다. **d** 옵션은 덤프 수준을 **31**로 지정합니다. **--message-level** 옵션은 메시지 수준을 **1**로 지정합니다.

또한 **-c** 및 **-p** 옵션을 사용한 다음 예를 고려하십시오.

- **c**를 사용하여 크래시 덤프 파일을 압축하려면 다음을 실행합니다.

```
core_collector makedumpfile -c -d 31 --message-level 1
```

- **p**를 사용하여 크래시 덤프 파일을 압축하려면 다음을 실행합니다.

```
core_collector makedumpfile -p -d 31 --message-level 1
```

추가 리소스

- **makedumpfile(8) 매뉴얼 페이지**
- [kdump 설정 파일](#)

11.5. KDUMP 기본 오류 응답 구성

기본적으로 **kdump**가 구성된 대상 위치에서 크래시 덤프 파일을 생성하지 못하면 시스템 재부팅 및 덤프가 프로세스에서 손실됩니다. 이 동작을 변경하려면 아래 절차를 따르십시오.

사전 요구 사항

- 루트 권한.
- **kdump** 구성 및 대상에 대한 요구 사항을 충족했습니다. 자세한 내용은 [지원되는 kdump 구성 및 대상](#)을 참조하십시오.

절차

1. 루트로서 `/etc/kdump.conf` 구성 파일의 `#failure_action` 행의 시작 부분에서 해시 기호("#")를 제거합니다.
2. 값을 원하는 작업으로 바꿉니다.

```
failure_action poweroff
```

추가 리소스

- [kdump 대상 구성](#)

11.6. KDUMP의 설정 파일

kdump 커널의 구성 파일은 `/etc/sysconfig/kdump` 입니다. 이 파일은 **kdump** 커널 명령줄 매개 변수를 제어합니다.

대부분의 구성에서는 기본 옵션을 사용합니다. 그러나 일부 시나리오에서는 **kdump** 커널 동작을 제어하기 위해 특정 매개 변수를 수정해야 할 수 있습니다. 예를 들어 자세한 디버깅 출력을 가져오려면 **kdump** 커널 명령줄을 추가하도록 를 수정합니다.

이 섹션에서는 **kdump**의 `KDUMP_COMMANDLINE_REMOVE` 및 `KDUMP_COMMANDLINE_APPEND` 옵션을 수정하는 방법에 대해 설명합니다. 추가 구성 옵션에 대한 자세한 내용은 `Documentation/admin-guide/kernel-parameters.txt` 또는 `/etc/sysconfig/kdump` 파일을 참조하십시오.

- **KDUMP_COMMANDLINE_REMOVE**

이 옵션은 현재 **kdump** 명령줄에서 인수를 제거합니다. **kdump** 오류 또는 **kdump** 커널 부팅 실패를 유발할 수 있는 매개 변수를 제거합니다. 이러한 매개 변수는 이전 **KDUMP_COMMANDLINE** 프로세스에서 구문 분석되었거나 `/proc/cmdline` 파일에서 상속되었

을 수 있습니다. 이 변수가 구성되지 않은 경우 **/proc/cmdline** 파일의 모든 값을 상속합니다. 이 옵션을 구성하면 문제를 디버깅하는 데 유용한 정보도 제공됩니다.

예제

특정 인수를 제거하려면 다음과 같이 **KDUMP_COMMANDLINE_REMOVE** 에 추가합니다.

```
KDUMP_COMMANDLINE_REMOVE="hugepages hugepagesz slub_debug quiet
log_buf_len swiotlb"
```

-

KDUMP_COMMANDLINE_APPEND

이 옵션은 현재 명령줄에 인수를 추가합니다. 이러한 인수는 이전 **KDUMP_COMMANDLINE_REMOVE** 변수에서 구문 분석할 수 있습니다.

kdump 커널의 경우 **mce,cgroup,numa,hest_disable** 과 같은 특정 모듈을 비활성화하면 커널 오류를 방지할 수 있습니다. 이러한 모듈은 **kdump**용으로 예약된 커널 메모리의 상당 부분을 사용하거나 **kdump** 커널 부팅 실패를 일으킬 수 있습니다.

예제

kdump 커널 명령줄에서 메모리 **cgroup** 을 비활성화하려면 다음과 같이 명령을 실행합니다.

```
KDUMP_COMMANDLINE_APPEND="cgroup_disable=memory"
```

추가 리소스

-

documentation/admin-guide/kernel-parameters.txt 파일

-

/etc/sysconfig/kdump file

11.7. KDUMP 설정 테스트

크래시 덤프 프로세스가 작동하고 시스템이 프로덕션에 진입하기 전에 유효한지 테스트할 수 있습니다.



주의

아래 명령을 실행하면 커널이 충돌합니다. 다음 단계를 수행할 때는 주의해야 하며 활성 프로덕션 시스템에서는 번거로움 없이 사용하지 마십시오.

절차

1. **kdump** 가 활성화된 시스템을 재부팅합니다.

2. **kdump** 가 실행 중인지 확인합니다.

```
~]# systemctl is-active kdump
active
```

3. **Linux** 커널이 충돌하도록 강제 적용합니다.

```
echo 1 > /proc/sys/kernel/sysrq
echo c > /proc/sysrq-trigger
```



주의

위의 명령은 커널을 충돌하고 재부팅해야 합니다.

다시 부팅되면 주소- **octets-MM-DD-HH:MM:SS/vmcore** 파일은 **/etc/kdump.conf** 파일에 지정한 위치에 생성됩니다(기본적으로 **/var/crash/**).



참고

이 작업은 구성의 유효성을 확인합니다. 또한 이 작업을 사용하여 대표 작업 부하로 충돌 덤프를 완료하는 데 걸리는 시간을 기록할 수 있습니다.

추가 리소스

- [kdump 대상 구성](#)

12장. KDUMP 활성화

이 섹션에서는 설치된 모든 커널 또는 특정 커널의 **kdump** 서비스를 활성화하고 시작하는 데 필요한 정보 및 절차를 제공합니다.

12.1. 설치된 모든 커널에 KDUMP 활성화

머신에 설치된 모든 커널의 **kdump** 서비스를 활성화하고 시작할 수 있습니다.

사전 요구 사항

- 관리자 권한

절차

1. 설치된 모든 커널에 **crashkernel=auto** 명령줄 매개변수를 추가합니다.

```
# /grubby --update-kernel=ALL --args="crashkernel=auto"
```

2. **kdump** 서비스를 활성화합니다.

```
# systemctl enable --now kdump.service
```

검증

- **kdump** 서비스가 실행 중인지 확인합니다.

```
# systemctl status kdump.service
```

```
○ kdump.service - Crash recovery kernel arming
  Loaded: loaded (/usr/lib/systemd/system/kdump.service; enabled; vendor preset:
disabled)
  Active: active (live)
```

12.2. 설치된 특정 커널의 KDUMP 활성화

머신의 특정 커널에 대해 **kdump** 서비스를 활성화할 수 있습니다.

사전 요구 사항

- 관리자 권한

절차

1. 머신에 설치된 커널을 나열합니다.

```
# ls -a /boot/vmlinuz-*
/boot/vmlinuz-0-rescue-2930657cd0dc43c2b75db480e5e5b4a9 /boot/vmlinuz-4.18.0-330.el8.x86_64 /boot/vmlinuz-4.18.0-330.rt7.111.el8.x86_64
```

2. 시스템의 **GRUB(GRUB) 2** 설정 파일에 특정 **kdump** 커널을 추가합니다.

예를 들면 다음과 같습니다.

```
# grubby --update-kernel=vmlinuz-4.18.0-330.el8.x86_64 --args="crashkernel=auto"
```

3. **kdump** 서비스를 활성화합니다.

```
# systemctl enable --now kdump.service
```

검증

- **kdump** 서비스가 실행 중인지 확인합니다.

```
# systemctl status kdump.service

○ kdump.service - Crash recovery kernel arming
   Loaded: loaded (/usr/lib/systemd/system/kdump.service; enabled; vendor preset: disabled)
   Active: active (live)
```

12.3. KDUMP 서비스 비활성화

부팅 시 **kdump** 서비스를 비활성화하려면 아래 절차를 따르십시오.

사전 요구 사항

-

kdump 구성 및 대상에 대한 요구 사항을 충족했습니다. 자세한 내용은 [지원되는 kdump 구성 및 대상을](#) 참조하십시오.

- **kdump** 를 설치하기 위한 모든 구성은 요구 사항에 따라 설정됩니다. 자세한 내용은 [kdump 설치](#)를 참조하십시오.

절차

1. 현재 세션에서 **kdump** 서비스를 중지하려면 다음을 수행합니다.

```
# systemctl stop kdump.service
```

2. **kdump** 서비스를 비활성화하려면 다음을 수행합니다.

```
# systemctl disable kdump.service
```



주의

kptr_restrict=1 를 설정하는 것이 좋습니다. 이 경우 **kdumpctl** 서비스는 사용 중인 **KASLR**(커널 주소 공간 레이아웃)에 관계없이 크래시 커널을 로드합니다.

문제 해결 단계

kptr_restrict 가 (1)로 설정되지 않고 **KASLR**이 활성화된 경우 **/proc/kcore** 파일의 내용이 모든 0으로 생성됩니다. 결과적으로 **kdumpctl** 서비스가 **/proc/kcore** 에 액세스하지 못하고 **crash** 커널을 로드하지 못합니다.

이 문제를 해결하기 위해 **/usr/share/doc/kexec-tools/kexec-kdump-howto.txt** 파일에는 **kptr_restrict=1** 설정을 사용하는 경고 메시지가 표시됩니다.

kdumpctl 서비스가 크래시 커널을 로드하도록 하려면 **kernel.kptr_restrict = 1** 이 **sysctl.conf** 파일에 나열되어 있는지 확인합니다.

추가 리소스

- ***RHEL***에서 *기본 시스템 설정 구성*

13장. 웹 콘솔에서 KDUMP 구성

RHEL 8 웹 콘솔에서 kdump 구성을 설정하고 테스트합니다.

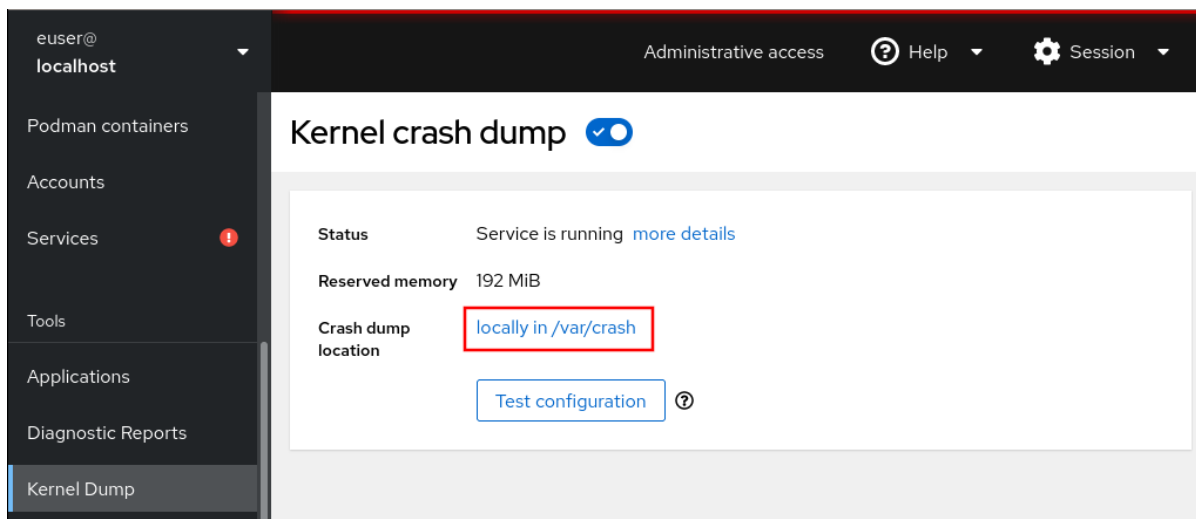
웹 콘솔은 **RHEL 8**의 기본 설치의 일부이며 부팅 시 **kdump** 서비스를 활성화 또는 비활성화합니다. 또한 웹 콘솔을 사용하면 **kdump**에 예약된 메모리를 설정하거나 압축되지 않았거나 압축된 형식으로 **vmcore saving** 위치를 선택할 수 있습니다.

13.1. 웹 콘솔에서 KDUMP 메모리 사용량 및 대상 위치 구성

아래 절차에서는 **RHEL** 웹 콘솔 인터페이스에서 **Kernel Dump** 탭을 사용하여 **kdump** 커널용으로 예약된 메모리 양을 구성하는 방법을 보여줍니다. 이 절차에서는 **vmcore** 덤프 파일의 대상 위치를 지정하는 방법과 구성을 테스트하는 방법을 설명합니다.

절차

1. **Kernel Dump** 탭을 열고 **kdump** 서비스를 시작합니다.
2. 명령줄을 사용하여 **kdump** 메모리 사용량을 구성합니다.
3. **Crash dump** 위치 옵션 옆에 있는 링크를 클릭합니다.



4. 드롭다운에서 **Local Filesystem** 옵션을 선택하고 덤프를 저장할 디렉터리를 지정합니다.

Crash dump location

Location: Local filesystem ▼

Directory: /var/crash

Compression: ☐ Compress crash dumps to save space

[Apply](#) [Cancel](#)

- 또는 드롭다운에서 **Remote over SSH (SSH를 통한 원격)** 옵션을 선택하여 **SSH** 프로토콜을 사용하여 **vmcore**를 원격 시스템으로 전송합니다.

Server, ssh 키 및 디렉터리 필드를 원격 시스템 주소, **ssh** 키 위치 및 대상 디렉토리로 채웁니다.

- 또 다른 선택 사항은 드롭다운에서 **NFS**를 통한 **Remote (원격)** 옵션을 선택하고 **NFS** 프로토콜을 사용하여 **vmcore**를 원격 시스템으로 보내도록 **Mount (마운트)** 필드를 채우는 것입니다.



참고

압축 확인란을 선택하여 **vmcore** 파일의 크기를 줄입니다.

5.

커널을 충돌하여 구성을 테스트합니다.

Status	Service is running more details
Reserved memory	192 MiB
Crash dump location	locally in /var/crash
	Test configuration ?

- a. **테스트 구성** 을 클릭합니다.
- b. **Test kdump** 설정 필드에서 **Crash** 시스템을 클릭합니다.



주의

이 단계에서는 커널 실행을 중단하고 시스템이 충돌하고 데이터가 손실됩니다.

추가 리소스

- [지원되는 kdump 대상](#)
- [OpenSSH로 두 시스템 간의 보안 통신 사용](#)

13.2. 추가 리소스

- [RHEL 웹 콘솔을 사용하여 시작하기](#)

14장. 지원되는 KDUMP 구성 및 대상

14.1. KDUMP의 메모리 요구 사항

kdump가 커널 크래시 덤프를 캡처하여 추가 분석을 위해 저장할 수 있으려면 시스템 메모리의 일부를 캡처 커널에 맞게 영구적으로 예약해야 합니다. 예약 시 기본 커널에서 시스템 메모리의 이 부분을 사용할 수 없습니다.

메모리 요구 사항은 특정 시스템 매개 변수에 따라 다릅니다. 주요 요인 중 하나는 시스템의 하드웨어 아키텍처입니다. 정확한 머신 아키텍처(예: **Intel 64** 및 **AMD64, x86_64**라고도 함)를 찾아서 표준 출력에 출력하려면 다음 명령을 사용합니다.

```
$ uname -m
```

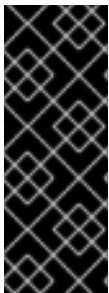
다음 표에는 사용 가능한 최신 버전에서 **kdump**에 대한 메모리 크기를 자동으로 예약하는 최소 메모리 요구 사항이 나열되어 있습니다. 시스템의 아키텍처 및 사용 가능한 총 물리적 메모리에 따라 크기가 변경됩니다.

표 14.1. **kdump**에 필요한 최소 예약 메모리 양

아키텍처	사용 가능한 메모리	최소 예약 메모리
AMD64 and Intel 64 (x86_64)	1GB ~ 4GB	160MB의 RAM.
	4GB ~ 64GB	192MB의 RAM.
	64GB ~ 1TB	256MB의 RAM.
	1TB 이상	512MB의 RAM.
64비트 ARM 아키텍처(arm64)	2GB 이상	448MB의 RAM.
IBM Power Systems(ppc64le)	2GB ~ 4GB	384MB의 RAM.
	4GB ~ 16GB	512MB의 RAM.
	16GB ~ 64GB	1GB RAM.
	64GB ~ 128GB	2GB RAM.
	128GB 이상	4GB RAM.
IBM Z (s390x)	1GB ~ 4GB	160MB의 RAM.

아키텍처	사용 가능한 메모리	최소 예약 메모리
	4GB ~ 64GB	192MB의 RAM.
	64GB ~ 1TB	256MB의 RAM.
	1TB 이상	512MB의 RAM.

많은 시스템에서 **kdump**는 필요한 메모리의 양을 추정하여 자동으로 예약할 수 있습니다. 이 동작은 기본적으로 활성화되어 있지만 시스템 아키텍처에 따라 달라지는 특정 양의 사용 가능한 메모리가 있는 시스템에서만 작동합니다.



중요

시스템의 총 메모리 크기에 따라 예약된 메모리의 자동 구성은 최상의 노력 추정입니다. 실제 필요한 메모리는 I/O 장치와 같은 다른 요인에 따라 다를 수 있습니다. 메모리가 충분하지 않으면 커널 패닉이 발생하는 경우 디버그 커널이 캡처 커널로 부팅되지 않을 수 있습니다. 이 문제를 방지하려면 크래시 커널 메모리를 충분히 늘립니다.

추가 리소스

- [RHEL8 마이너 릴리스 사이에 **crashkernel** 매개변수는 어떻게 변경되었습니까?](#)
- [기술 사양 및 제한 테이블](#)
- [자동 메모리 예약의 최소 임계값](#)

14.2. 자동 메모리 예약의 최소 임계값

일부 시스템에서는 부트 로더 구성 파일에서 **crashkernel=auto** 매개 변수를 사용하거나 그래픽 구성 유틸리티에서 이 옵션을 활성화하여 **kdump**에 대한 메모리를 자동으로 할당할 수 있습니다. 이 자동 예약이 작동하려면 시스템에서 특정 양의 총 메모리를 사용할 수 있어야 합니다. 양은 시스템 아키텍처에 따라 다릅니다.

아래 표에는 자동 메모리 할당에 대한 임계값이 나열되어 있습니다. 시스템에 지정된 임계값보다 작은 메모리가 있는 경우 메모리를 수동으로 구성해야 합니다.

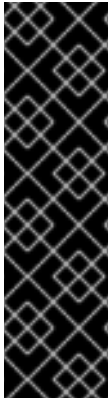
표 14.2. 자동 메모리 예약에 필요한 최소 메모리 양

아키텍처	필요한 메모리
AMD64 and Intel 64 (x86_64)	2GB
IBM Power Systems(ppc64le)	2GB
IBM Z (s390x)	4GB

14.3. 지원되는 KDUMP 대상

커널 크래시가 캡처되면 **vmcore** 덤프 파일을 장치에 직접 기록하거나 로컬 파일 시스템에 파일로 저장하거나 네트워크를 통해 전송할 수 있습니다. 아래 표에는 현재 **kdump** 에서 지원되거나 명시적으로 지원되지 않는 덤프 대상의 전체 목록이 포함되어 있습니다.

유형	지원되는 대상	지원되지 않는 대상
원시 장치	로컬로 연결된 원시 디스크 및 파티션 모두.	
로컬 파일 시스템	직접 연결된 디스크 드라이브, 하드웨어 RAID 논리 드라이브, LVM 장치 및 mdraid 어레이의 ext2 , ext3 , ext4 및 xfs 파일 시스템.	자동 유형(자동 파일 시스템 감지)을 포함하여 이 표에서 지원되는 것으로 명시적으로 나열되지 않은 로컬 파일 시스템.
원격 디렉토리	IPv4 를 통해 NFS 또는 SSH 프로토콜을 사용하여 액세스하는 원격 디렉토리.	NFS 프로토콜을 사용하여 액세스한 rootfs 파일 시스템의 원격 디렉토리입니다.
하드웨어 및 소프트웨어 이니시에이터를 통해 iSCSI 프로토콜을 사용하여 액세스하는 원격 디렉토리입니다.	be2iscsi 하드웨어에서 iSCSI 프로토콜을 사용하여 액세스하는 원격 디렉토리입니다.	다중 경로 기반 스토리지.
		IPv6 를 통해 액세스하는 원격 디렉토리.
		SMB 또는 CIFS 프로토콜을 사용하여 액세스하는 원격 디렉토리.
		FCoE (<i>Fibre Channel over Ethernet</i>) 프로토콜을 사용하여 액세스하는 원격 디렉토리.
		무선 네트워크 인터페이스를 사용하여 액세스되는 원격 디렉토리.



중요

펌웨어 지원 덤프(**fadump**)를 사용하여 **vmcore**를 캡처하고 **SSH** 또는 **NFS** 프로토콜을 사용하여 원격 머신에 저장하면 네트워크 인터페이스의 이름이 **kdump-<interface-name>**으로 변경됩니다. **<interface-name>**이 일반인 경우 이름 변경이 발생합니다(예: ***eth#**, **net#** 등). 이 문제는 초기 **RAM** 디스크(**initrd**)의 **vmcore** 캡처 스크립트를 네트워크 인터페이스 이름에 **kdump-** 접두사를 추가하여 영구적인 명명을 보안하기 때문에 발생합니다. 동일한 **initrd**도 일반 부팅에도 사용되므로 프로덕션 커널의 인터페이스 이름도 변경됩니다.

추가 리소스



kdump 대상 구성

14.4. 지원되는 KDUMP 필터링 수준

덤프 파일의 크기를 줄이기 위해 **kdump**는 **makedumpfile** 코어 수집기를 사용하여 데이터를 압축하고 선택적으로 원하지 않는 정보를 생략합니다. 아래 표에는 현재 **makedumpfile** 유틸리티에서 지원하는 전체 필터링 수준 목록이 포함되어 있습니다.

옵션	설명
1	0 페이지
2	캐시 페이지
4	캐시 개인
8	사용자 페이지
16	사용 가능한 페이지



참고

makedumpfile 명령은 투명한 대규모 페이지 및 **hugetlbfs** 페이지 제거를 지원합니다. 이러한 **hugepages** 사용자 페이지를 모두 고려하고 **-8** 수준을 사용하여 제거합니다.

추가 리소스



코어 수집기 구성

14.5. 지원되는 기본 오류 응답

기본적으로 **kdump** 에서 코어 덤프를 생성하지 못하면 운영 체제가 재부팅됩니다. 그러나 코어 덤프를 기본 대상에 저장하지 못하는 경우 다른 작업을 수행하도록 **kdump** 를 구성할 수 있습니다. 아래 표에는 현재 지원되는 모든 기본 작업이 나열되어 있습니다.

옵션	설명
dump_to_rootfs	코어 덤프를 루트 파일 시스템에 저장해 봅니다. 이 옵션은 네트워크 대상과 함께 특히 유용합니다. 네트워크 대상에 연결할 수 없는 경우 이 옵션은 코어 덤프를 로컬로 저장하도록 kdump 를 구성합니다. 나중에 시스템이 재부팅됩니다.
reboot	시스템을 재부팅하여 프로세스의 코어 덤프를 손실합니다.
Halted	시스템을 중지하고 프로세스의 코어 덤프를 손실합니다.
poweroff	시스템의 전원을 끄고 프로세스에서 코어 덤프를 손실합니다.
셸	initramfs 내에서 셸 세션을 실행하여 사용자가 코어 덤프를 수동으로 기록할 수 있습니다.
final_action	kdump 가 성공한 후 또는 shell 또는 dump_to_rootfs 오류 작업이 완료되면 reboot, halt, poweroff 작업 등의 추가 작업을 활성화합니다. 기본 final_action 옵션은 reboot 입니다.

추가 리소스

- [kdump 기본 오류 응답 구성](#)

14.6. FINAL_ACTION 매개변수 사용

final_action 매개변수를 사용하면 **kdump** 가 성공한 후 **reboot, halt, poweroff** 작업 또는 **shell** 또는 **dump_to_rootfs** 를 사용하여 호출된 **failure_response** 메커니즘과 같은 특정 추가 작업을 사용할 수 있습니다. **final_action** 옵션을 지정하지 않으면 기본적으로 재부팅 됩니다.

절차

1.

'etc/kdump.conf 파일을 편집하고 **final_action** 매개 변수를 추가합니다.

```
final_action <reboot | halt | poweroff>
```

2.

kdump 서비스를 다시 시작하십시오.

```
kdumpctl restart
```

15장. 펌웨어 지원 덤프 메커니즘

펌웨어 지원 덤프(**fadump**)는 **IBM POWER** 시스템의 **kdump** 메커니즘 대신 제공되는 덤프 캡처 메커니즘입니다. **kexec** 및 **kdump** 메커니즘은 **AMD64** 및 **Intel 64** 시스템에서 코어 덤프를 캡처하는 데 유용합니다. 그러나 미니 시스템 및 메인프레임 컴퓨터와 같은 일부 하드웨어는 온보드 펌웨어를 활용하여 메모리 영역을 격리하고 충돌 분석에 중요한 데이터의 실수로 덮어쓰는 것을 방지합니다. 이 섹션에서는 **fadump** 메커니즘 및 **RHEL**과의 통합 방법에 대해 설명합니다. **fadump** 유틸리티는 **IBM POWER** 시스템에서 이러한 확장된 덤프 기능에 최적화되어 있습니다.

15.1. IBM POWERPC 하드웨어에서 펌웨어 지원 덤프

fadump 유틸리티는 **PCI** 및 **I/O** 장치를 사용하여 완전히 재설정된 시스템에서 **vmcore** 파일을 캡처합니다. 이 메커니즘은 펌웨어를 사용하여 충돌 중에 메모리 영역을 보존한 다음 **kdump** 사용자 공간 스크립트를 다시 사용하여 **vmcore** 파일을 저장합니다. 메모리 영역은 부팅 메모리, 시스템 레지스터 및 **PTE**(하드웨어 페이지 테이블 항목)를 제외한 모든 시스템 메모리 콘텐츠로 구성됩니다.

fadump 메커니즘은 파티션을 재부팅하고 새 커널을 사용하여 이전 커널 충돌의 데이터를 덤프하여 기존 덤프 유형에 비해 안정성이 향상됩니다. **fadump**에는 **IBM POWER6** 프로세서 기반 또는 이후 버전 하드웨어 플랫폼이 필요합니다.

PowerPC 하드웨어 재설정 방법을 비롯한 **fadump** 메커니즘에 대한 자세한 내용은 `/usr/share/doc/kexec-tools/fadump-howto.txt` 파일을 참조하십시오.



참고

부트 메모리라고 알려져 있는 메모리 영역은 충돌 이벤트 후 커널을 부팅하는 데 필요한 **RAM**의 양입니다. 기본적으로 부팅 메모리 크기는 총 시스템 **RAM** 중 **256MB** 또는 **5%**입니다.

kexec 시작 이벤트와 달리 **fadump** 메커니즘은 **production** 커널을 사용하여 크래시 덤프를 복구합니다. 충돌 후 부팅 시 **PowerPC** 하드웨어를 사용하면 장치 노드 `/proc/device-tree/rtas/ibm.kernel-dump`를 **proc** 파일 시스템(**proc fs**)에서 사용할 수 있습니다. **fadump** 인식 **kdump** 스크립트에서 저장된 **vmcore**가 있는지 확인한 다음 시스템 재부팅을 정상적으로 완료합니다.

15.2. 펌웨어 지원 덤프 메커니즘 활성화

IBM POWER 시스템의 크래시 덤프 기능은 펌웨어 지원 덤프(**fadump**) 메커니즘을 활성화하여 향상할 수 있습니다.

열거

1. **kdump 설치 및 구성.**
2. **/etc/default/grub 파일의 GRUB_CMDLINE_LINUX 행에 fadump=on 을 추가합니다.**

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=rhel/swap crashkernel=auto
rd.lvm.lv=rhel/root rhgb quiet fadump=on"
```

3. (선택 사항) 기본값을 사용하는 대신 예약된 부팅 메모리를 지정하려면 **/etc/default/grub** 에서 **crashkernel=xxM** 을 **GRUB_CMDLINE_LINUX** 로 구성합니다. 여기서 **xx** 는 메가바이트 단위로 필요한 메모리 양입니다.

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=rhel/swap crashkernel=xxM rd.lvm.lv=rhel/root rhgb
quiet fadump=on"
```

중요

Red Hat은 부팅 구성 옵션을 실행하기 전에 모든 부팅 구성 옵션을 테스트하는 것이 좋습니다. 크래시 커널에서 부팅할 때 **OOM**(메모리 부족) 오류를 관찰하는 경우 크래시 커널이 정상적으로 부팅될 때까지 **crashkernel=** 인수에 지정된 값을 늘립니다. 이 경우 시험 및 오류가 필요할 수 있습니다.

15.3. IBM Z 하드웨어에서 펌웨어 지원 덤프 메커니즘

IBM Z 시스템은 다음과 같은 펌웨어 지원 덤프 메커니즘을 지원합니다.

- 독립 실행형 덤프 (**sadump**)
- **VMDUMP**

kdump 인프라는 **IBM Z** 시스템에서 지원 및 활용됩니다. 그러나 **IBM Z**에 대한 펌웨어 지원 덤프 (**fadump**) 방법 중 하나를 사용하면 다음과 같은 다양한 이점을 얻을 수 있습니다.

- 애 석한ump 메커니즘은 시스템 콘솔에서 시작되고 제어되며 **IPL** 부팅 가능한 장치에 저장됩니다.

- **VMDUMP** 메커니즘은 애석한 덤프와 유사합니다. 이 도구는 시스템 콘솔에서도 시작되지만 하드웨어에서 결과 덤프를 검색하여 분석을 위해 시스템에 복사합니다.
- 이러한 방법(다른 하드웨어 기반 덤프 메커니즘과 유사하게)은 **kdump** 서비스가 시작되기 전에 초기 부팅 단계에서 시스템의 상태를 캡처할 수 있습니다.
- **VMDUMP**에는 **Red Hat Enterprise Linux** 시스템으로 덤프 파일을 수신하는 메커니즘이 포함되어 있지만 **VMDUMP**의 구성 및 제어는 **IBM Z** 하드웨어 콘솔에서 관리됩니다.

IBM은 독립 실행형 덤프 프로그램 기사와 **VMDUMP**를 사용한 **z/VM**에 덤프를 생성하는 **VMDUMP**에 대해 자세히 설명합니다.

IBM에는 **Red Hat Enterprise Linux 7.4**에서 덤프 툴 사용 설명서도 **Red Hat Enterprise Linux 7**에서 사용할 수 있습니다.

추가 리소스

- [독립형 덤프 프로그램](#)
- [VMDUMP를 사용하여 z/VM에서 덤프 생성](#)
- [Red Hat Enterprise Linux에서 덤프 툴 사용 7.4](#)

15.4. FUJITSU PRIMEQUEST 시스템에서 IRONUMP 사용

Fujitsu unfortunately ump 메커니즘은 **kdump**를 성공적으로 완료할 수 없는 경우 폴백 덤프 캡처를 제공하도록 설계되었습니다. **MMB**(시스템 관리 보드) 인터페이스에서 수동으로 호출됩니다. **MMB**를 사용하여 **Intel 64** 또는 **AMD 64** 서버에 대해 **kdump**를 구성한 다음 다음 추가 단계를 수행하여 안도프를 활성화합니다.

절차

1. **/etc/sysctl.conf** 파일에서 다음 행을 추가하거나 편집하여 **kdump**가 **expect ump**에서 예상대로 시작되도록 합니다.

```
kernel.panic=0
kernel.unknown_nmi_panic=1
```



주의

특히 **kdump** 후에 시스템이 재부팅되지 않도록 하십시오. **kdump** 가 **vmcore** 파일을 저장하지 못한 후 시스템이 재부팅되면 안 됩니다.

2.

/etc/kdump.conf 에서 **halt** 또는 **shell** 으로 적절하게 **failure_action** 매개 변수를 설정합니다.

```
failure_action shell
```

추가 리소스

-

FujiJITSU Server PRIMEQUEST 2000 시리즈 설치 설명서

16장. 코어 덤프 분석

시스템 충돌의 원인을 확인하기 위해 **crash** 유틸리티를 사용하면 **GDB(GNU Debugger)**와 매우 유사한 대화형 프롬프트를 제공합니다. 이 유틸리티를 사용하면 실행 중인 **Linux** 시스템뿐만 아니라 **kdump, netdump, diskdump** 또는 **xendump** 에서 생성된 코어 덤프를 대화형으로 분석할 수 있습니다. 또는 **Kernel Oops Analyzer** 또는 **Kdump Helper** 도구를 사용하는 옵션이 있습니다.

16.1. CRASH 유틸리티 설치

다음 절차에서는 충돌 분석 툴을 설치하는 방법을 설명합니다.

절차

1. 관련 리포지토리를 활성화합니다.

```
# subscription-manager repos --enable baseos repository
```

```
# subscription-manager repos --enable appstream repository
```

```
# subscription-manager repos --enable rhel-8-for-x86_64-baseos-debug-rpms
```

2. **crash** 패키지를 설치합니다.

```
# yum install crash
```

3. **kernel-debuginfo** 패키지를 설치합니다.

```
# yum install kernel-debuginfo
```

패키지는 실행 중인 커널에 해당하며 덤프 분석에 필요한 데이터를 제공합니다.

추가 리소스

- [기본 시스템 설정 구성](#)

16.2. 크래시 유틸리티 실행 및 종료

다음 절차에서는 시스템 충돌 원인을 분석하기 위해 **crash** 유틸리티를 시작하는 방법을 설명합니다.

사전 요구 사항

- 현재 실행 중인 커널을 식별합니다(예: **4.18.0-5.el8.x86_64**).

절차

1.

crash 유틸리티를 시작하려면 명령에 두 개의 매개 변수를 전달해야 합니다.

- 특정 **kernel-debuginfo** 패키지를 통해 제공되는 **debug-info**(예: **/usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinux**)
- 실제 **vmcore** 파일(예: **/var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore**)

결과 **crash** 명령은 다음과 같습니다.

```
# crash /usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinux /var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore
```

kdump 에서 캡처한 동일한 **<kernel>** 버전을 사용합니다.

예 16.1. crash 유틸리티 실행

다음 예제에서는 **4.18.0-5.el8.x86_64** 커널을 사용하여 **14:05 PM**에 생성된 코어 덤프를 분석하는 방법을 보여줍니다.

```
...
WARNING: kernel relocated [202MB]: patching 90160 gdb minimal_symbol values

KERNEL: /usr/lib/debug/lib/modules/4.18.0-5.el8.x86_64/vmlinux
DUMPFILE: /var/crash/127.0.0.1-2018-10-06-14:05:33/vmcore [PARTIAL DUMP]
CPUS: 2
DATE: Sat Oct 6 14:05:16 2018
UPTIME: 01:03:57
LOAD AVERAGE: 0.00, 0.00, 0.00
TASKS: 586
NODENAME: localhost.localdomain
RELEASE: 4.18.0-5.el8.x86_64
VERSION: #1 SMP Wed Aug 29 11:51:55 UTC 2018
```

```

MACHINE: x86_64 (2904 Mhz)
MEMORY: 2.9 GB
PANIC: "sysrq: SysRq : Trigger a crash"
PID: 10635
COMMAND: "bash"
TASK: ffff8d6c84271800 [THREAD_INFO: ffff8d6c84271800]
CPU: 1
STATE: TASK_RUNNING (SYSRQ)

```

```
crash>
```

2.

대화형 프롬프트를 종료하고 크래시 를 종료하려면 **exit** 또는 **q** 를 입력합니다.

예 16.2. crash 유틸리티 종료

```
crash> exit
~]#
```

참고

crash 명령을 라이브 시스템을 디버깅하기 위한 강력한 도구로 사용할 수도 있습니다. 그러나 시스템을 손상시키지 않도록 주의해서 사용하십시오.

추가 리소스

- [예상치 못한 시스템 재시작 가이드](#)

16.3. CRASH 유틸리티에 다양한 표시기 표시

다음 절차에서는 **crash** 유틸리티를 사용하고 커널 메시지 버퍼, 역추적, 프로세스 상태, 가상 메모리 정보 및 열기 파일과 같은 다양한 표시기를 표시하는 방법을 설명합니다.

메시지 버퍼 표시

- 커널 메시지 버퍼를 표시하려면 아래 예제에 표시된 대로 대화형 프롬프트에서 **log** 명령을 입력합니다.

```

crash> log
... several lines omitted ...
EIP: 0060:[<c068124f>] EFLAGS: 00010096 CPU: 2
EIP is at sysrq_handle_crash+0xf/0x20
EAX: 00000063 EBX: 00000063 ECX: c09e1c8c EDX: 00000000

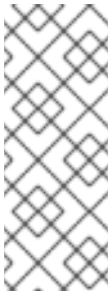
```

```

ESI: c0a09ca0 EDI: 00000286 EBP: 00000000 ESP: ef4dbf24
DS: 007b ES: 007b FS: 00d8 GS: 00e0 SS: 0068
Process bash (pid: 5591, ti=ef4da000 task=f196d560 task.ti=ef4da000)
Stack:
c068146b c0960891 c0968653 00000003 00000000 00000002 efade5c0 c06814d0
<0> ffffffff c068150f b7776000 f2600c40 c0569ec4 ef4dbf9c 00000002 b7776000
<0> efade5c0 00000002 b7776000 c0569e60 c051de50 ef4dbf9c f196d560 ef4dbfb4
Call Trace:
[<c068146b>] ? __handle_sysrq+0xfb/0x160
[<c06814d0>] ? write_sysrq_trigger+0x0/0x50
[<c068150f>] ? write_sysrq_trigger+0x3f/0x50
[<c0569ec4>] ? proc_reg_write+0x64/0xa0
[<c0569e60>] ? proc_reg_write+0x0/0xa0
[<c051de50>] ? vfs_write+0xa0/0x190
[<c051e8d1>] ? sys_write+0x41/0x70
[<c0409adc>] ? syscall_call+0x7/0xb
Code: a0 c0 01 0f b6 41 03 19 d2 f7 d2 83 e2 03 83 e0 cf c1 e2 04 09 d0 88 41 03 f3 c3 90 c7 05
c8 1b 9e c0 01 00 00 00 0f ae f8 89 f6 <c6> 05 00 00 00 00 01 c3 89 f6 8d bc 27 00 00 00 00 8d 50
d0 83
EIP: [<c068124f>] sysrq_handle_crash+0xf/0x20 SS:ESP 0068:ef4dbf24
CR2: 0000000000000000

```

명령 사용에 대한 자세한 내용은 **help** 로그를 입력합니다.



참고

커널 메시지 버퍼에는 시스템 충돌에 대한 가장 필수적인 정보가 포함되어 있으므로 **vmcore-dmesg.txt** 파일에 항상 먼저 덤프됩니다. 이 기능은 타겟 위치에 공간이 부족하여 전체 **vmcore** 파일을 가져오려고 할 때 유용합니다. 기본적으로 **vmcore-dmesg.txt** 는 **/var/crash/** 디렉터리에 있습니다.

역추적 표시

•

커널 스택 추적을 표시하려면 **bt** 명령을 사용합니다.

```

crash> bt
PID: 5591 TASK: f196d560 CPU: 2 COMMAND: "bash"
#0 [ef4dbdcc] crash_kexec at c0494922
#1 [ef4dbe20] oops_end at c080e402
#2 [ef4dbe34] no_context at c043089d
#3 [ef4dbe58] bad_area at c0430b26
#4 [ef4dbe6c] do_page_fault at c080fb9b
#5 [ef4dbee4] error_code (via page_fault) at c080d809
EAX: 00000063 EBX: 00000063 ECX: c09e1c8c EDX: 00000000 EBP: 00000000
DS: 007b ESI: c0a09ca0 ES: 007b EDI: 00000286 GS: 00e0
CS: 0060 EIP: c068124f ERR: ffffffff EFLAGS: 00010096
#6 [ef4dbf18] sysrq_handle_crash at c068124f
#7 [ef4dbf24] __handle_sysrq at c0681469
#8 [ef4dbf48] write_sysrq_trigger at c068150a

```

```
#9 [ef4dbf54] proc_reg_write at c0569ec2
#10 [ef4dbf74] vfs_write at c051de4e
#11 [ef4dbf94] sys_write at c051e8cc
#12 [ef4dbfb0] system_call at c0409ad5
    EAX: ffffffff EBX: 00000001 ECX: b7776000 EDX: 00000002
    DS: 007b   ESI: 00000002 ES: 007b   EDI: b7776000
    SS: 007b   ESP: bfc2088 EBP: bfc20b4 GS: 0033
    CS: 0073   EIP: 00edc416 ERR: 00000004 EFLAGS: 00000246
```

bt <pid> 를 입력하여 특정 프로세스의 역추적을 표시하거나, **bt** 사용법에 대한 자세한 내용을 보려면 **help bt** 를 입력합니다.

프로세스 상태 표시

-

시스템의 프로세스 상태를 표시하려면 **ps** 명령을 사용합니다.

```
crash> ps
  PID PPID CPU TASK ST %MEM VSZ RSS COMM
>  0   0  0 c09dc560 RU 0.0  0  0 [swapper]
>  0   0  1 f7072030 RU 0.0  0  0 [swapper]
    0   0  2 f70a3a90 RU 0.0  0  0 [swapper]
>  0   0  3 f70ac560 RU 0.0  0  0 [swapper]
    1   0  1 f705ba90 IN 0.0 2828 1424 init
... several lines omitted ...
5566   1  1 f2592560 IN 0.0 12876 784 auditd
5567   1  2 ef427560 IN 0.0 12876 784 auditd
5587 5132  0 f196d030 IN 0.0 11064 3184 sshd
> 5591 5587  2 f196d560 RU 0.0 5084 1648 bash
```

ps <pid> 를 사용하여 특정 단일 프로세스의 상태를 표시합니다. **ps** 사용법에 대한 자세한 내용은 **help ps** 를 사용합니다.

가상 메모리 정보 표시

-

기본 가상 메모리 정보를 표시하려면 대화형 프롬프트에서 **vm** 명령을 입력합니다.

```
crash> vm
PID: 5591 TASK: f196d560 CPU: 2 COMMAND: "bash"
  MM   PGD   RSS  TOTAL_VM
f19b5900 ef9c6000 1648k 5084k
  VMA   START   END  FLAGS FILE
f1bb0310 242000 260000 8000875 /lib/ld-2.12.so
f26af0b8 260000 261000 8100871 /lib/ld-2.12.so
efbc275c 261000 262000 8100873 /lib/ld-2.12.so
efbc2a18 268000 3ed000 8000075 /lib/libc-2.12.so
efbc23d8 3ed000 3ee000 8000070 /lib/libc-2.12.so
efbc2888 3ee000 3f0000 8100071 /lib/libc-2.12.so
```

```
efbc2cd4 3f0000 3f1000 8100073 /lib/libc-2.12.so
efbc243c 3f1000 3f4000 100073
efbc28ec 3f6000 3f9000 8000075 /lib/libdl-2.12.so
efbc2568 3f9000 3fa000 8100071 /lib/libdl-2.12.so
efbc2f2c 3fa000 3fb000 8100073 /lib/libdl-2.12.so
f26af888 7e6000 7fc000 8000075 /lib/libtinfo.so.5.7
f26aff2c 7fc000 7ff000 8100073 /lib/libtinfo.so.5.7
efbc211c d83000 d8f000 8000075 /lib/libnss_files-2.12.so
efbc2504 d8f000 d90000 8100071 /lib/libnss_files-2.12.so
efbc2950 d90000 d91000 8100073 /lib/libnss_files-2.12.so
f26afe00 edc000 edd000 4040075
f1bb0a18 8047000 8118000 8001875 /bin/bash
f1bb01e4 8118000 811d000 8101873 /bin/bash
f1bb0c70 811d000 8122000 100073
f26afae0 9fd9000 9ffa000 100073
... several lines omitted ...
```

vm <pid> 를 사용하여 단일 특정 프로세스에 대한 정보를 표시하거나 **vm** 사용법에 대한 자세한 내용은 **help vm** 을 사용합니다.

열려 있는 파일 표시

-

열려 있는 파일에 대한 정보를 표시하려면 **files** 명령을 사용합니다.

```
crash> files
PID: 5591 TASK: f196d560 CPU: 2 COMMAND: "bash"
ROOT: / CWD: /root
FD  FILE  DENTRY  INODE  TYPE  PATH
0  f734f640 eedc2c6c eecd6048 CHR  /pts/0
1  efade5c0 eee14090 f00431d4 REG  /proc/sysrq-trigger
2  f734f640 eedc2c6c eecd6048 CHR  /pts/0
10 f734f640 eedc2c6c eecd6048 CHR  /pts/0
255 f734f640 eedc2c6c eecd6048 CHR  /pts/0
```

<pid> 파일을 사용하여 선택한 프로세스만 연 파일을 표시하거나 파일 사용에 대한 자세한 내용은 도움말 파일을 사용합니다.

16.4. 커널 OOPS 분석기 사용

Kernel Oops Analyzer 틀은 지식 베이스의 알려진 문제와 **oops** 메시지를 비교하여 크래시 덤프를 분석합니다.

사전 요구 사항

- 커널 **Oops** 분석기에 공급되도록 **oops** 메시지를 보호합니다.

절차

1. **Kernel Oops Analyzer** 툴 액세스.
2. 커널 충돌 문제를 진단하려면 **vmcore**에 생성된 커널 **oops** 로그를 업로드합니다.
- 또는 텍스트 메시지 또는 **vmcore-dmesg.txt**를 입력으로 제공하여 커널 크래시 문제를 진단할 수도 있습니다.

The screenshot displays the 'Kernel Oops Analyzer' web interface. It is divided into two main sections: 'Option 1: File Input' and 'Option 2: Text Input'.

Option 1: File Input includes a file selection button labeled 'Choose File' with the text 'No file chosen' next to it. Below this, a note states: 'Choose and upload the [kernel oops log](#) generated from a vmcore. Maximum file size for uploaded kernel oops log is 10 MB.' At the bottom of this section is a blue button labeled 'Detect'.

Option 2: Text Input features a large text area for pasting content. At the bottom of this section are two buttons: a blue 'Detect' button and a white 'Clear' button with a circular arrow icon.

3. **DEœT** (삭제)를 클릭하여 **makedumpfile**의 정보를 알려진 솔루션과 기준으로 **oops** 메시지를 비교합니다.

추가 리소스

- [커널 Oops 분석기 문서](#)
- [예상치 못한 시스템 재시작 가이드](#)

16.5. KDUMP 도우미 도구

Kdump 도우미 도구는 제공된 정보를 사용하여 **kdump**를 설정하는 데 도움이 됩니다. **Kdump Helper**는 기본 설정에 따라 구성 스크립트를 생성합니다. 서버에서 스크립트를 시작하고 실행하면 **kdump** 서비스가 설정됩니다.

추가 리소스

-

kdump 도우미

17장. EARLY KDUMP를 사용하여 부팅 시간 충돌 캡처

시스템 관리자는 **kdump** 서비스의 초기 **kdump** 지원을 활용하여 부팅 프로세스 초기 단계에서 충돌하는 커널의 **vmcore** 파일을 캡처할 수 있습니다. 이 섹션에서는 초기 **kdump**의 개념, 구성 방법 및 이 메커니즘의 상태를 확인하는 방법에 대해 설명합니다.

17.1. EARLY KDUMP이란 무엇입니까?

부팅 단계에서 커널 충돌은 **kdump** 서비스가 아직 시작되지 않을 때 발생하며 충돌한 커널 메모리의 콘텐츠를 쉽게 캡처하고 저장할 수 없습니다. 따라서 문제 해결에 중요한 정보가 손실됩니다.

이 문제를 해결하기 위해 **RHEL 8**은 **kdump** 서비스의 일부로 초기 **kdump** 기능을 도입했습니다.

17.2. 초기 KDUMP 활성화

이 섹션에서는 초기 **kdump** 기능을 활성화하여 초기 부팅 커널 충돌에 대한 정보가 손실될 위험을 제거하는 방법에 대해 설명합니다.

사전 요구 사항

- 활성화 **RHEL** 서브스크립션.
- 시스템 **CPU** 아키텍처용 **kexec-tools** 패키지가 포함된 리포지토리
- **kdump** 구성 및 대상 요구 사항 충족.

절차

1. **kdump** 서비스가 활성화되어 활성 상태인지 확인합니다.

```
# systemctl is-enabled kdump.service && systemctl is-active kdump.service enabled active
```

kdump가 활성화되어 실행되고 있지 않으면 필요한 모든 구성을 설정하고 **kdump** 서비스가 활성화되었는지 확인합니다.

2.

초기 **kdump** 기능을 사용하여 부팅 커널의 **initramfs** 이미지를 다시 빌드합니다.

```
dracut -f --add earlykdump
```

3.

rd.earlykdump 커널 명령줄 매개변수를 추가합니다.

```
grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="rd.earlykdump"
```

4.

시스템을 재부팅하여 변경 사항을 반영합니다.

```
reboot
```

5.

선택적으로 **that rd.earlykdump** 가 성공적으로 추가되었으며 초기 **kdump** 기능이 활성화되었는지 확인합니다.

```
# cat /proc/cmdline  
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-187.el8.x86_64 root=/dev/mapper/rhel-root  
ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root  
rd.lvm.lv=rhel/swap rhgb quiet rd.earlykdump
```

```
# journalctl -x | grep early-kdump  
Mar 20 15:44:41 redhat dracut-cmdline[304]: early-kdump is enabled.  
Mar 20 15:44:42 redhat dracut-cmdline[304]: kexec: loaded early-kdump kernel
```

추가 리소스

•

/usr/share/doc/kexec-tools/early-kdump-howto.txt file

•

[초기 kdump 지원이란 무엇이며 어떻게 구성합니까?](#)

•

[kdump 활성화](#)

18장. 커널 라이브 패치로 패치 적용

Red Hat Enterprise Linux 커널 라이브 패치 솔루션을 사용하여 프로세스를 재부팅하거나 다시 시작하지 않고도 실행 중인 커널을 패치할 수 있습니다.

이 솔루션을 사용하면 시스템 관리자가 다음을 수행합니다.

- 커널에 중요한 보안 패치를 즉시 적용할 수 있음.
- 사용자가 로그아웃하거나 예약된 다운 타임을 위해 장기 실행 작업이 완료될 때까지 기다릴 필요는 없습니다.
- 시스템의 가동 시간을 더 많이 제어하고 보안이나 안정성을 포기하지 않습니다.

커널 라이브 패치 솔루션을 사용하여 모든 심각 또는 중요한 **CVE**가 해결되지는 않습니다. **Red Hat**의 목표는 보안 관련 패치에 필요한 재부팅을 줄이는 것입니다. 이러한 패치를 완전히 제거하는 것이 아니라. 라이브 패치 범위에 대한 자세한 내용은 [고객 포털 솔루션 문서](#)를 참조하십시오.



주의

일부 비호환성은 커널 라이브 패치와 기타 커널 하위 구성 요소 사이에 있습니다. 커널 라이브 패치를 사용하기 전에 **kpatch**의 **제한 사항**을 주의 깊게 읽습니다.

18.1. KPATCH의 제한 사항

- **kpatch** 기능은 범용 커널 업그레이드 메커니즘이 아닙니다. 시스템을 재부팅할 때 간단한 보안 및 버그 수정 업데이트를 적용하는 데 사용됩니다.
- 패치 중 또는 로드 후 **SystemTap** 또는 **kprobe** 툴을 사용하지 마십시오. 이러한 프로브가 제거될 때까지 패치가 적용되지 않을 수 있습니다.

18.2. 타사 라이브 패칭 지원

kpatch 유틸리티는 **Red Hat** 리포지토리에서 제공하는 **RPM** 모듈을 통해 **Red Hat**에서 지원하는 유일한 커널 라이브 패치 유틸리티입니다. **Red Hat**은 **Red Hat** 자체에서 제공하지 않은 라이브 패치를 지원하지 않습니다.

타사 라이브 패치와 함께 발생하는 문제에 대한 지원이 필요한 경우, 근본 원인 판단이 필요한 조사 시작 시점에 라이브 패치 벤더와 함께 사례를 여는 것이 좋습니다. 따라서 벤더가 허용하는 경우 소스 코드를 제공하고, **Red Hat** 지원으로 조사하기 전에 근본 원인 결정에 대한 지원을 제공할 수 있습니다.

타사 라이브 패치로 실행되는 모든 시스템에 대해 **Red Hat**은 **Red Hat**이 내장 및 지원되는 소프트웨어를 사용하여 해고할 권리가 있습니다. 이 작업을 수행할 수 없는 경우 동일한 동작을 관찰하기 위해 유사한 시스템 및 워크로드를 실시간 패치를 적용하지 않고 테스트 환경에 배포해야 합니다.

타사 소프트웨어 지원 정책에 대한 자세한 내용은 **Red Hat** 글로벌 지원 서비스에서 타사 소프트웨어, 드라이버 및/또는 인증되지 않은 하드웨어/하이퍼바이저 또는 게스트 운영 체제를 어떻게 처리합니까?를 참조하십시오.

18.3. 커널 라이브 패치에 액세스

커널 라이브 패치 기능은 **RPM** 패키지로 제공되는 커널 모듈(**kmod**)으로 구현됩니다.

모든 고객은 일반적인 채널을 통해 제공되는 커널 라이브 패치에 액세스할 수 있습니다. 그러나 연장된 지원 오퍼링에 가입하지 않은 고객은 다음 마이너 릴리스가 제공되면 현재 마이너 릴리스의 새 패치에 대한 액세스가 손실됩니다. 예를 들어, 표준 서브스크립션을 보유한 고객은 **RHEL 8.3** 커널이 릴리스될 때까지 **RHEL 8.2** 커널을 라이브 패치할 수 있습니다.

18.4. 커널 라이브 패치 구성 요소

커널 라이브 패치의 구성 요소는 다음과 같습니다.

커널 패치 모듈

-

커널 라이브 패치를 위한 제공 메커니즘.

- 패치되는 커널을 위해 특별히 빌드되는 커널 모듈입니다.
- **patch** 모듈에는 커널에 필요한 수정 사항이 포함되어 있습니다.
- **patch** 모듈은 **livepatch** 커널 하위 시스템에 등록하고 대체 기능에 대한 해당 포인터와 함께 교체할 원래 기능에 대한 정보를 제공합니다. 커널 패치 모듈은 **RPM**으로 제공됩니다.
- 명령 규칙은 **kpatch_<kernel version>_<kpatch version>_<kpatch release>** 입니다. 이름의 "커널 버전" 부분에는 점들이 밑줄로 바뀝니다.

kpatch 유틸리티

패치 모듈을 관리하는 명령줄 유틸리티입니다.

kpatch 서비스

multiuser.target에 필요한 **systemd** 서비스. 이 대상은 부팅 시 커널 패치 모듈을 로드합니다.

kpatch-dnf 패키지

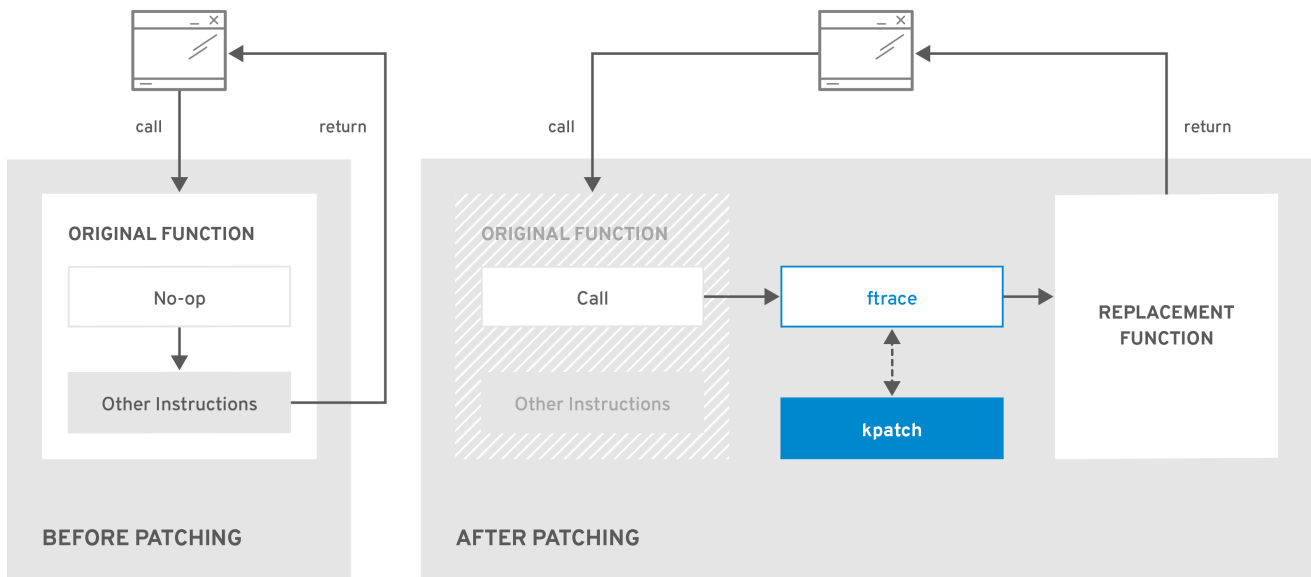
RPM 패키지 형태로 제공되는 **DNF** 플러그인. 이 플러그인은 커널 라이브 패치에 대한 자동 서브스크립션을 관리합니다.

18.5. 커널 라이브 패치 작동 방법

kpatch 커널 패치 솔루션은 **livepatch** 커널 하위 시스템을 사용하여 이전 기능을 새 함수로 리디렉션합니다. 시스템에 라이브 커널 패치를 적용하면 다음과 같은 문제가 발생합니다.

1. 커널 패치 모듈은 **/var/lib/kpatch/** 디렉토리에 복사되고 다음 부팅 시 **systemd**를 통해 커널에 재애플리케이션을 등록합니다.
2. **kpatch** 모듈은 실행 중인 커널에 로드되고 새 함수는 새 코드 메모리에 있는 위치에 대한 포인터를 사용하여 **ftrace** 메커니즘에 등록됩니다.
3. 커널이 패치된 기능에 액세스하면 원래 기능을 무시하고 커널을 패치된 함수 버전으로 리디렉션하는 **ftrace** 메커니즘에 의해 리디렉션됩니다.

그림 18.1. 커널 라이브 패치 작동 방법



RHEL_424549_0119

18.6. 현재 설치된 커널을 라이브 패치 스트림에 가입

커널 패치 모듈은 패치되는 커널 버전과 관련된 **RPM** 패키지로 제공됩니다. 각 **RPM** 패키지는 시간에 따라 누적 업데이트됩니다.

다음 절차에서는 지정된 커널에 대한 향후 누적 라이브 패치 업데이트에 서브스크립션하는 방법을 설명합니다. 라이브 패치는 누적되므로 지정된 커널에 대해 배포되는 개별 패치를 선택할 수 없습니다.



주의

Red Hat은 **Red Hat** 지원 시스템에 적용되는 타사 라이브 패치를 지원하지 않습니다.

사전 요구 사항

- 루트 권한

절차

1.

선택적으로 커널 버전을 확인합니다.

```
# uname -r
4.18.0-94.el8.x86_64
```

2.

커널 버전에 해당하는 라이브 패치 패키지를 검색합니다.

```
# yum search $(uname -r)
```

3.

라이브 패치 패키지를 설치합니다.

```
# yum install "kpatch-patch = $(uname -r)"
```

위의 명령은 특정 커널에 대해서만 최신 누적 라이브 패치를 설치하고 적용합니다.

라이브 패치 패키지 버전이 1-1 이상인 경우 패키지에 **patch** 모듈이 포함됩니다. 이 경우 라이브 패치 패키지를 설치하는 동안 커널이 자동으로 패치됩니다.

커널 패치 모듈은 향후 재부팅 중에 **systemd** 시스템 및 서비스 관리자가 로드할 **/var/lib/kpatch/** 디렉토리에 설치됩니다.



참고

지정된 커널에 사용할 수 있는 라이브 패치가 없는 경우 빈 라이브 패치 패키지가 설치됩니다. 비어 있는 라이브 패치 패키지에는 **kpatch_version-kpatch_release**가 0-0입니다(예: **kpatch-patch-4_18_0-94-0.el8.x86_64.rpm**). 빈 **RPM** 설치하는 지정된 커널의 향후 모든 라이브 패치에 대해 시스템을 서브스크립션합니다.

4.

선택적으로 커널이 패치되었는지 확인합니다.

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_94_1_1 [enabled]

Installed patch modules:
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
...
```

출력은 커널 패치 모듈이 커널에 로드되었음을 보여주므로 이제 **kpatch-patch-4_18_0-94-1-1.el8.x86_64.rpm** 패키지의 최신 수정 사항과 패치됩니다.

추가 리소스

- **kpatch(1) 매뉴얼 페이지**
- **RHEL** 에서 기본 시스템 설정 구성

18.7. 라이브 패치 스트림에 향후 커널을 자동으로 등록

kpatch-dnf YUM 플러그인을 사용하여 시스템을 서브스크립션하여 커널 라이브 패치라고도 하는 커널 패치 모듈에서 제공하는 수정 사항을 적용할 수 있습니다. 플러그인은 시스템이 현재 사용하는 모든 커널에 대해 자동 서브스크립션을 활성화하고 나중에 커널에 설치할 수 있습니다.

사전 요구 사항

- 루트 권한이 있습니다.

절차

1. 선택적으로 설치된 모든 커널과 현재 실행 중인 커널을 확인합니다.

```
# yum list installed | grep kernel
Updating Subscription Management repositories.
Installed Packages
...
kernel-core.x86_64      4.18.0-240.10.1.el8_3    @rhel-8-for-x86_64-baseos-rpms
kernel-core.x86_64      4.18.0-240.15.1.el8_3    @rhel-8-for-x86_64-baseos-rpms
...

# uname -r
4.18.0-240.10.1.el8_3.x86_64
```

2. **kpatch-dnf** 플러그인을 설치합니다.

```
# yum install kpatch-dnf
```

3.

커널 라이브 패치에 대한 자동 서브스크립션을 활성화합니다.

```
# yum kpatch auto
Updating Subscription Management repositories.
Last metadata expiration check: 19:10:26 ago on Wed 10 Mar 2021 04:08:06 PM CET.
Dependencies resolved.

=====
Package                                Architecture
=====
Installing:
kpatch-patch-4_18_0-240_10_1          x86_64
kpatch-patch-4_18_0-240_15_1          x86_64

Transaction Summary
=====
Install 2 Packages
...
```

이 명령은 현재 설치된 커널을 모두 서브스크립션하여 커널 라이브 패치를 수신합니다. 또한 명령은 설치된 모든 커널에 대해 최신 누적 라이브 패치를 설치하고 적용합니다.

나중에 커널을 업데이트하면 새 커널 설치 프로세스 중에 라이브 패치가 자동으로 설치됩니다.

커널 패치 모듈은 향후 재부팅 중에 **systemd** 시스템 및 서비스 관리자가 로드할 **/var/lib/kpatch/** 디렉토리에 설치됩니다.



참고

지정된 커널에 사용할 수 있는 라이브 패치가 없는 경우 빈 라이브 패치 패키지가 설치됩니다. 빈 라이브 패치 패키지에는 **kpatch_version-kpatch_release** 가 0-0입니다(예: **kpatch-patch-4_18_0-240-0.el8.x86_64.rpm**). 빈 RPM 설치되는 지정된 커널의 모든 향후 라이브 패치에 시스템을 등록합니다.

검증 단계

•

설치된 모든 커널이 패치되었는지 확인합니다.

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_240_10_1_0_1 [enabled]
```

Installed patch modules:**kpatch_4_18_0_240_10_1_0_1 (4.18.0-240.10.1.el8_3.x86_64)****kpatch_4_18_0_240_15_1_0_2 (4.18.0-240.15.1.el8_3.x86_64)**

출력에는 커널이 모두 실행되고 있으며 설치된 기타 커널이 **kpatch-patch-4_18_0-240_10_1-0-1.rpm** 및 **kpatch-patch-4_18_0-240_15_1-0-1.rpm** 패키지의 수정 사항으로 패치되었습니다.

추가 리소스

- **kpatch(1)** 및 **dnf-kpatch(8)** 매뉴얼 페이지
- [RHEL 에서 기본 시스템 설정 구성](#)

18.8. 라이브 패치 스트림에 대한 자동 서브스크립션 비활성화

커널 패치 모듈에서 제공하는 수정 사항에 맞게 시스템을 서브스크립션하면 서브스크립션이 자동으로 제공됩니다. 이 기능을 비활성화하여 **kpatch-patch** 패키지의 자동 설치를 비활성화할 수 있습니다.

사전 요구 사항

- 루트 권한이 있습니다.

절차

1.

선택적으로 설치된 모든 커널과 현재 실행 중인 커널을 확인합니다.

```
# yum list installed | grep kernel
Updating Subscription Management repositories.
Installed Packages
...
kernel-core.x86_64      4.18.0-240.10.1.el8_3      @rhel-8-for-x86_64-baseos-rpms
kernel-core.x86_64      4.18.0-240.15.1.el8_3      @rhel-8-for-x86_64-baseos-rpms
...

# uname -r
4.18.0-240.10.1.el8_3.x86_64
```

2.

커널 라이브 패치에 대한 자동 서브스크립션을 비활성화합니다.

```
# yum kpatch manual
Updating Subscription Management repositories.
```

검증 단계

- 성공적인 결과를 확인할 수 있습니다:

```
# yum kpatch status
...
Updating Subscription Management repositories.
Last metadata expiration check: 0:30:41 ago on Tue Jun 14 15:59:26 2022.
Kpatch update setting: manual
```

추가 리소스

- [kpatch\(1\) 및 dnf-kpatch\(8\) 매뉴얼 페이지](#)

18.9. 커널 패치 모듈 업데이트

커널 패치 모듈은 **RPM** 패키지를 통해 전달 및 적용되므로 누적 커널 패치 모듈을 업데이트하는 것은 다른 **RPM** 패키지를 업데이트하는 것과 같습니다.

사전 요구 사항

- 현재 설치된 커널을 라이브 패치 스트림에 서브스크립션하는 방법에 설명된 대로 시스템이 라이브 패치 스트림에 서브스크립션됩니다.

절차

- 현재 커널의 새 누적 버전으로 업데이트합니다.

```
# yum update "kpatch-patch = $(uname -r)"
```

위의 명령은 현재 실행 중인 커널에 사용할 수 있는 모든 업데이트를 자동으로 설치하고 적용합니다. 향후 출시되는 누적 라이브 패치 포함.

- 또는 설치된 모든 커널 패치 모듈을 업데이트합니다.

```
# yum update "kpatch-patch"
```



참고

시스템이 동일한 커널로 재부팅되면 **kpatch.service systemd** 서비스에 의해 커널이 자동으로 다시 패치됩니다.

추가 리소스

- [RHEL 에서 기본 시스템 설정 구성](#)

18.10. 라이브 패치 패키지 제거

다음 절차에서는 라이브 패치 패키지를 제거하여 **Red Hat Enterprise Linux** 커널 라이브 패치 솔루션을 비활성화하는 방법을 설명합니다.

사전 요구 사항

- 루트 권한
- 라이브 패치 패키지가 설치되어 있습니다.

절차

1. 라이브 패치 패키지를 선택합니다.

```
# yum list installed | grep kpatch-patch
kpatch-patch-4_18_0-94.x86_64      1-1.el8      @@commandline
...
```

위의 예제 출력에는 설치된 라이브 패치 패키지가 나열되어 있습니다.

2. 라이브 패치 패키지를 제거합니다.

```
# yum remove kpatch-patch-4_18_0-94.x86_64
```

라이브 패치 패키지가 제거되면 다음 번 재부팅할 때까지 커널이 패치되지만 커널 패치 모듈은 디스크에서 제거됩니다. 나중에 재부팅할 때 해당 커널은 더 이상 패치되지 않습니다.

3.

시스템을 재부팅합니다.

4.

라이브 패치 패키지가 제거되었는지 확인합니다.

```
# yum list installed | grep kpatch-patch
```

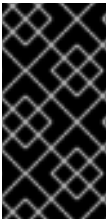
패키지가 성공적으로 제거된 경우 명령은 출력을 표시하지 않습니다.

5.

선택적으로 커널 라이브 패치 솔루션이 비활성화되었는지 확인합니다.

```
# kpatch list
Loaded patch modules:
```

예제 출력에서는 현재 로드된 패치 모듈이 없으므로 커널이 패치되지 않았음을 보여줍니다.



중요

현재 **Red Hat**은 시스템을 재부팅하지 않고도 실시간 패치 되돌리기를 지원하지 않습니다. 문제가 발생할 경우 지원 팀에 문의하십시오.

추가 리소스

- [kpatch\(1\) 매뉴얼 페이지](#)
- [RHEL에서 기본 시스템 설정 구성](#)

18.11. 커널 패치 모듈 설치 제거

다음 절차에서는 **Red Hat Enterprise Linux** 커널 라이브 패칭 솔루션이 후속 부팅 시 커널 패치 모듈을 적용하지 못하도록 하는 방법을 설명합니다.

사전 요구 사항

- 루트 권한

- 라이브 패치 패키지가 설치되어 있습니다.
- 커널 패치 모듈이 설치 및 로드되었습니다.

절차

1. 커널 패치 모듈을 선택합니다.

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_94_1_1 [enabled]

Installed patch modules:
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
...
```

2. 선택한 커널 패치 모듈을 제거하십시오.

```
# kpatch uninstall kpatch_4_18_0_94_1_1
uninstalling kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
```

- 설치 제거된 커널 패치 모듈이 계속 로드됩니다.

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_94_1_1 [enabled]

Installed patch modules:
<NO_RESULT>
```

선택한 모듈이 제거되면 다음 번 재부팅할 때까지 커널이 패치되지만 커널 패치 모듈은 디스크에서 제거됩니다.

3. 시스템을 재부팅합니다.
4. 선택적으로 커널 패치 모듈이 제거되었는지 확인합니다.

```
# kpatch list
Loaded patch modules:
...
```

위의 예제 출력에서는 로드되거나 설치된 커널 패치 모듈이 표시되지 않으므로 커널이 패치되지 않고 커널 라이브 패치 솔루션이 활성화되어 있지 않습니다.



중요

현재 **Red Hat**은 시스템을 재부팅하지 않고도 실시간 패치 되돌리기를 지원하지 않습니다. 문제가 발생할 경우 지원 팀에 문의하십시오.

추가 리소스

- [kpatch\(1\) 매뉴얼 페이지](#)

18.12. KPATCH.SERVICE 비활성화

다음 절차에서는 **Red Hat Enterprise Linux** 커널 라이브 패칭 솔루션이 후속 부팅 시 모든 커널 패치 모듈을 전역적으로 적용하지 못하도록 하는 방법을 설명합니다.

사전 요구 사항

- 루트 권한
- 라이브 패치 패키지가 설치되어 있습니다.
- 커널 패치 모듈이 설치 및 로드되었습니다.

절차

1. **kpatch.service** 가 활성화되었는지 확인합니다.

```
# systemctl is-enabled kpatch.service
enabled
```

2.

kpatch.service 를 비활성화합니다 :

```
# systemctl disable kpatch.service
Removed /etc/systemd/system/multi-user.target.wants/kpatch.service.
```

•

적용된 커널 패치 모듈이 계속 로드됩니다.

```
# kpatch list
Loaded patch modules:
kpatch_4_18_0_94_1_1 [enabled]

Installed patch modules:
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
```

3.

시스템을 재부팅합니다.

4.

필요한 경우 **kpatch.service** 의 상태를 확인합니다 :

```
# systemctl status kpatch.service
● kpatch.service - "Apply kpatch kernel patches"
   Loaded: loaded (/usr/lib/systemd/system/kpatch.service; disabled; vendor preset: disabled)
   Active: inactive (dead)
```

예제 출력은 **kpatch.service** 가 비활성화되었으며 실행되고 있지 않음을 확인합니다. 따라서 커널 라이브 패치 솔루션이 활성화되어 있지 않습니다.

5.

커널 패치 모듈이 로드 해제되었는지 확인합니다.

```
# kpatch list
Loaded patch modules:
<NO_RESULT>

Installed patch modules:
kpatch_4_18_0_94_1_1 (4.18.0-94.el8.x86_64)
```

위의 예제 출력은 커널 패치 모듈이 여전히 설치되어 있지만 커널이 패치되지 않았음을 보여줍니다.



중요

현재 **Red Hat**은 시스템을 재부팅하지 않고도 실시간 패치 되돌리기를 지원하지 않습니다. 문제가 발생할 경우 지원 팀에 문의하십시오.

추가 리소스

- [kpatch\(1\) 매뉴얼 페이지](#)
- [RHEL 에서 기본 시스템 설정 구성](#)

19장. ZSWAP을 사용하여 시스템 성능 개선

zswap 커널 기능을 활성화하여 시스템 성능을 향상시킬 수 있습니다.

19.1. ZSWAP이란

이 섹션에서는 **zswap** 이 무엇이며 어떻게 시스템 성능을 향상시킬 수 있는지 설명합니다.

zswap 은 스왑 페이지에 대해 압축된 **RAM** 캐시를 제공하는 커널 기능입니다. 이 메커니즘은 다음과 같이 작동합니다. **zswap** 은 스왑 아웃 중인 페이지를 가져와서 동적으로 할당된 **RAM** 기반 메모리 풀로 압축하려고 합니다. 풀이 가득 차거나 **RAM**이 소진되면 **zswap** *evicts pages from the compressed cache on an LRU basis (recently used recently used) to the backing swap device*. 페이지 압축이 스왑 캐시로 압축 해제되면 **zswap** 은 풀에서 압축 버전을 해제합니다.

zswap의 이점

- 상당한 I/O 감소
- 워크로드 성능 향상

Red Hat Enterprise Linux 8에서는 기본적으로 **zswap** 이 활성화됩니다.

추가 리소스

- [Zswap이란 무엇입니까?](#)

19.2. 런타임 시 ZSWAP 활성화

sysfs 인터페이스를 사용하여 시스템 런타임에서 **zswap** 기능을 활성화할 수 있습니다.

사전 요구 사항

- 루트 권한이 있습니다.

절차

- **zswap** 을 활성화합니다.

```
# echo 1 > /sys/module/zswap/parameters/enabled
```

검증 단계

- **zswap** 이 활성화되어 있는지 확인합니다.

```
# grep -r . /sys/kernel/debug/zswap

duplicate_entry:0
pool_limit_hit:13422200
pool_total_size:6184960 (pool size in total in pages)
reject_alloc_fail:5
reject_compress_poor:0
reject_kmemcache_fail:0
reject_reclaim_fail:13422200
stored_pages:4251 (pool size after compression)
written_back_pages:0
```

추가 리소스

- [Zswap 기능을 활성화하는 방법은 무엇입니까?](#)

19.3. ZSWAP을 영구적으로 활성화

zswap.enabled=1 커널 명령줄 매개 변수를 제공하여 **zswap** 기능을 영구적으로 활성화할 수 있습니다.

사전 요구 사항

- 루트 권한이 있습니다.
- **grubby** 또는 **zipl** 유틸리티가 시스템에 설치되어 있습니다.

절차

1. **zswap** 을 영구적으로 활성화합니다.

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="zswap.enabled=1"
```

2.

변경 사항을 적용하려면 시스템을 재부팅합니다.

검증 단계

- **zswap** 이 활성화되어 있는지 확인합니다.

```
# cat /proc/cmdline
```

```
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-372.9.1.el8.x86_64
root=/dev/mapper/rhel-root ro crashkernel=auto
resume=/dev/mapper/rhel-swap
rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet
zswap.enabled=1
```

추가 리소스

- [Zswap 기능을 활성화하는 방법은 무엇입니까?](#)
- [커널 명령줄 매개변수 구성](#)

20장. 애플리케이션에 대한 제한 설정

cgroup(제어 그룹) 커널 기능을 사용하여 제한을 설정하고 프로세스의 하드웨어 리소스에 우선 순위를 지정하거나 격리할 수 있습니다. 이를 통해 애플리케이션의 리소스 사용을 보다 효율적으로 활용하도록 세부적으로 제어할 수 있습니다.

20.1. 제어 그룹 이해

제어 그룹은 계층적으로 정렬된 그룹인 **cgroup** 으로 프로세스를 구성할 수 있는 **Linux** 커널 기능입니다. 계층 구조(제어 그룹 트리)는 기본적으로 **/sys/fs/cgroup/** 디렉터리에 마운팅된 **cgroup** 가상 파일 시스템에 구조를 제공하여 정의됩니다. **systemd** 시스템 및 서비스 관리자는 **cgroup** 을 활용하여 관리하는 모든 장치 및 서비스를 구성합니다. 또는 **/sys/fs/cgroup/** 디렉토리에서 하위 디렉터리를 생성 및 제거하여 **cgroup** 계층 구조를 수동으로 관리할 수 있습니다.

그런 다음 리소스 컨트롤러(커널 구성 요소)는 해당 프로세스의 **CPU** 시간, 메모리, 네트워크 대역폭 또는 다양한 조합과 같은 시스템 리소스(예: **CPU** 시간, 메모리, 네트워크 대역폭 또는 다양한 조합)를 제한, 우선 순위 지정 또는 할당하여 **cgroup** 의 프로세스 동작을 수정합니다.

cgroup 의 부가 가치는 애플리케이션 및 사용자 간에 하드웨어 리소스를 구분할 수 있는 프로세스 집계입니다. 따라서 사용자 환경의 효율성, 안정성 및 보안을 전반적으로 높일 수 있습니다.

컨트롤 그룹 버전 1

제어 그룹 버전 1 (**cgroups-v1**)은 리소스별 컨트롤러 계층 구조를 제공합니다. 즉, **CPU**, 메모리, **I/O** 등의 각 리소스에 고유한 제어 그룹 계층 구조가 있습니다. 하나의 컨트롤러가 각각의 리소스를 관리하는 다른 컨트롤러와 조정하는 방식으로 다양한 제어 그룹 계층 구조를 결합할 수 있습니다. 그러나 두 컨트롤러는 적절한 조정을 허용하지 않는 다른 프로세스 계층 구조에 속할 수 있습니다.

cgroups-v1 컨트롤러는 많은 기간 동안 개발되었기 때문에 제어 파일의 동작과 이름이 동일하지 않습니다.

컨트롤 그룹 버전 2

계층적 유연성에서 비롯된 컨트롤러 조정 문제로 인해 제어 그룹 버전 2 가 개발되었습니다.

컨트롤 그룹 버전 2 (**cgroups-v2**)는 모든 리소스 컨트롤러가 마운트되는 단일 제어 그룹 계층 구조를 제공합니다.

제어 파일 동작 및 명령은 다양한 컨트롤러에서 일관되게 유지됩니다.



참고

cgroups-v2는 **RHEL 8.2** 이상 버전에서 완전히 지원됩니다. 자세한 내용은 **RHEL 8**에서 **Control Group v2**가 완전히 지원됩니다.

이 하위 섹션은 **Devconf.cz 2019** 프레젠테이션을 기반으로 합니다.^[1]

추가 리소스

- [커널 리소스 컨트롤러란](#)
- [cgroups\(7\) 도움말 페이지](#)
- [제어 그룹에서 systemd 역할](#)

20.2. 커널 리소스 컨트롤러란 무엇입니까?

제어 그룹의 기능은 커널 리소스 컨트롤러에서 활성화합니다. **RHEL 8**에서는 제어 그룹 버전 1 (**cgroups-v1**) 및 제어 그룹 버전 2 (**cgroups-v2**)에 대한 다양한 컨트롤러를 지원합니다.

리소스 컨트롤러(제어 그룹 하위 시스템이라고도 함)는 **CPU** 시간, 메모리, 네트워크 대역폭 또는 디스크 **I/O**와 같은 단일 리소스를 나타내는 커널 하위 시스템입니다. **Linux** 커널은 **systemd** 시스템 및 서비스 관리자가 자동으로 마운트하는 리소스 컨트롤러를 제공합니다. **/proc/cgroups** 파일에서 현재 마운트된 리소스 컨트롤러 목록을 찾습니다.

다음 컨트롤러는 **cgroups-v1**에 사용할 수 있습니다.

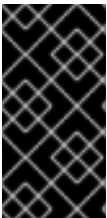
- **blkio** - 블록 장치로 들어오고 나가는 입력/출력 액세스에 대한 제한을 설정할 수 있습니다.
- **CPU** - 제어 그룹의 작업에 대해 **CFS(Completely Fair Scheduler)** 스케줄러의 매개변수를 조정할 수 있습니다. 동일한 마운트에서 **cpuacct** 컨트롤러와 함께 마운트됩니다.
- **cpuacct** - 제어 그룹의 작업에서 사용하는 **CPU** 리소스에 대한 자동 보고서를 생성합니다. 동

일한 마운트의 **cpu** 컨트롤러와 함께 마운트됩니다.

- **cpuset** - CPU의 지정된 하위 집합에서만 실행되도록 제어 그룹 작업을 제한하고 지정된 메모리 노드에서만 메모리를 사용하도록 작업을 지시하는 데 사용할 수 있습니다.
- **장치** - 제어 그룹의 작업에 대한 장치에 대한 액세스를 제어할 수 있습니다.
- **카나리아** - 제어 그룹에서 작업을 일시 중단하거나 재개하는 데 사용할 수 있습니다.
- **memory** - 제어 그룹의 작업에서 메모리 사용 제한을 설정하고 해당 작업에서 사용하는 메모리 리소스에 대한 자동 보고서를 생성하는 데 사용할 수 있습니다.
- **net_cls** - Linux 트래픽 컨트롤러(thetc 명령)를 통해 특정 제어 그룹 작업에서 시작된 패킷을 식별할 수 있는 클래스 식별자(classid)로 네트워크 패킷을 태그합니다. **net_cls**의 하위 시스템인 **net_filter** (iptables)는 이 태그를 사용하여 이러한 패킷에 대한 작업을 수행할 수도 있습니다. **net_filter**는 Linux 방화벽(iptables 명령을 통해)이 특정 제어 그룹 작업에서 시작되는 패킷을 식별할 수 있는 방화벽 식별자(fwid)로 네트워크 소켓에 태그를 지정합니다.
- **net_prio** - 네트워크 트래픽의 우선 순위를 설정합니다.
- **PID** - 제어 그룹의 여러 프로세스 및 하위 항목에 대한 제한을 설정할 수 있습니다.
- **perf_event** - perf 성능 모니터링 및 보고 유틸리티를 통해 모니터링을 위한 작업을 그룹화할 수 있습니다.
- **RDMA** - 제어 그룹의 Remote Direct Memory Access/InfiniBand 특정 리소스에 제한을 설정할 수 있습니다.
- **hugetlb** - 제어 그룹의 작업에 따라 대규모 가상 메모리 페이지 사용을 제한하는 데 사용할 수 있습니다.

다음 컨트롤러는 **cgroups-v2**에 사용할 수 있습니다.

- **io - cgroups-v1** 의 **blkio** 에 후속 조치.
- **메모리 - cgroups-v1** 의 메모리 후속 작업.
- **PID - cgroups-v1** 에서 **pid** 와 동일합니다.
- **RDMA - cgroups-v1** 의 것과 같습니다.
- **cpu - cgroups-v1** 의 **cpu** 및 **cpuacct** 에 대한 후속 조치.
- **cpuset** - 새 파티션 기능으로 핵심 기능(**cpus{,.effective}**, **mems{,.effective}**)만 지원합니다.
- **perf_event** - 지원은 고유하며 명시적인 제어 파일이 아닙니다. 해당 **cgroup** 내의 모든 작업을 프로파일링하는 **perf** 명령에 **v2 cgroup** 을 매개 변수로 지정할 수 있습니다.



중요

리소스 컨트롤러는 동시에 **cgroups-v1** 계층 구조 또는 **cgroups-v2** 계층 구조에서 사용할 수 있습니다.

추가 리소스

- **cgroups(7)** 도움말 페이지
- **/usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups-v1/** 디렉터리에 있는 문서(**kernel-doc** 패키지 설치 후).

20.3. 네임스페이스의 정의

네임스페이스는 소프트웨어 개체를 구성하고 식별하는 가장 중요한 방법 중 하나입니다.

네임스페이스는 글로벌 시스템 리소스(예: 마운트 지점, 네트워크 장치 또는 호스트 이름)를 추상화하여 글로벌 리소스의 자체 격리된 인스턴스가 있는 네임스페이스 내에서 처리합니다. 네임스페이스를 사용

하는 가장 일반적인 기술 중 하나는 컨테이너입니다.

특정 글로벌 리소스에 대한 변경 사항은 해당 네임스페이스의 프로세스에만 표시되며 나머지 시스템 또는 기타 네임스페이스에는 영향을 주지 않습니다.

프로세스가 멤버인 네임스페이스를 검사하려면 `/proc/<PID>/ns/` 디렉터리에서 심볼릭 링크를 확인할 수 있습니다.

다음 표는 격리하는 지원되는 네임스페이스 및 리소스를 보여줍니다.

네임스페이스	격리
마운트	마운트 지점
UTS	호스트 이름 및 NIS 도메인 이름
IPC	System V IPC, POSIX 메시지 대기열
PID	프로세스 ID
네트워크	네트워크 장치, 스택, 포트 등
사용자	사용자 및 그룹 ID
컨트롤 그룹	그룹 root 디렉토리 제어

추가 리소스

- [namespaces\(7\) 및 cgroup_namespaces\(7\) 도움말 페이지](#)
- [제어 그룹 이해](#)

20.4. CGROUPS-V1을 사용하여 CPU 제한 설정

경우에 따라 애플리케이션이 많은 CPU 시간을 소비하는 경우가 있어 환경의 전반적인 상태에 부정적인 영향을 줄 수 있습니다. 제어 그룹 버전 1 (cgroups-v1)을 사용하여 애플리케이션에 대한 CPU 제한을 구성하려면 `/sys/fs/` 가상 파일 시스템을 사용합니다.

사전 요구 사항

- 루트 권한이 있습니다.
- 제한하려는 **CPU** 사용량이 있는 애플리케이션이 있습니다.
- **cgroups-v1** 컨트롤러가 마운트되었는지 확인합니다.

```
# mount -l | grep cgroup
tmpfs on /sys/fs/cgroup type tmpfs (ro,nosuid,nodev,noexec,seclabel,mode=755)
cgroup on /sys/fs/cgroup/systemd type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,xattr,release_agent=/usr/lib/systemd/syste
md-cgroups-agent,name=systemd)
cgroup on /sys/fs/cgroup/cpu,cpuacct type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,cpu,cpuacct)
cgroup on /sys/fs/cgroup/perf_event type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,perf_event)
cgroup on /sys/fs/cgroup/pids type cgroup
(rw,nosuid,nodev,noexec,relatime,seclabel,pids)
...
```

절차

1.

CPU 사용량에서 제한할 애플리케이션의 **PID**(프로세스 ID)를 식별합니다.

```
# top
top - 11:34:09 up 11 min, 1 user, load average: 0.51, 0.27, 0.22
Tasks: 267 total, 3 running, 264 sleeping, 0 stopped, 0 zombie
%Cpu(s): 49.0 us, 3.3 sy, 0.0 ni, 47.5 id, 0.0 wa, 0.2 hi, 0.0 si, 0.0 st
MiB Mem : 1826.8 total, 303.4 free, 1046.8 used, 476.5 buff/cache
MiB Swap: 1536.0 total, 1396.0 free, 140.0 used. 616.4 avail Mem

  PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6955 root    20  0 228440 1752 1472 R 99.3  0.1   0:32.71 sha1sum
 5760 jdoe    20  0 3603868 205188 64196 S  3.7 11.0   0:17.19 gnome-shell
 6448 jdoe    20  0 743648 30640 19488 S  0.7  1.6   0:02.73 gnome-terminal-
    1 root    20  0 245300 6568 4116 S  0.3  0.4   0:01.87 systemd
 505 root    20  0  0  0  0 I 0.3  0.0   0:00.75 kworker/u4:4-events_unbound
...
```

top 프로그램의 예제 출력에서는 **PID 6955**(유연 적 애플리케이션 **sha1sum**)가 많은 **CPU** 리소스를 사용함을 보여줍니다.

2.

cpu 리소스 컨트롤러 디렉터리에 하위 디렉터를 생성합니다.

```
# mkdir /sys/fs/cgroup/cpu/Example/
```

위의 디렉터리는 특정 프로세스를 배치하고 프로세스에 특정 **CPU** 제한을 적용할 수 있는 제어 그룹을 나타냅니다. 동시에 일부 **cgroups-v1** 인터페이스 파일과 **cpu** 컨트롤러별 파일이 디렉터리에 생성됩니다.

3.

선택적으로 새로 생성된 제어 그룹을 검사합니다.

```
# ll /sys/fs/cgroup/cpu/Example/
-rw-r--r--. 1 root root 0 Mar 11 11:42 cgroup.clone_children
-rw-r--r--. 1 root root 0 Mar 11 11:42 cgroup.procs
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.stat
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_all
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_sys
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_percpu_user
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_sys
-r--r--r--. 1 root root 0 Mar 11 11:42 cpuacct.usage_user
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.cfs_period_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.cfs_quota_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.rt_period_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.rt_runtime_us
-rw-r--r--. 1 root root 0 Mar 11 11:42 cpu.shares
-r--r--r--. 1 root root 0 Mar 11 11:42 cpu.stat
-rw-r--r--. 1 root root 0 Mar 11 11:42 notify_on_release
-rw-r--r--. 1 root root 0 Mar 11 11:42 tasks
```

예제 출력에서는 **Example** 제어 그룹의 프로세스에 대해 설정할 수 있는 특징 구성 및/또는 제한을 나타내는 **cpuacct.us**, **cpu.cfs_period_us** 와 같은 파일을 보여줍니다. 각 파일 이름 앞에 속해 있는 제어 그룹 컨트롤러의 이름이 붙습니다.

기본적으로 새로 생성된 제어 그룹은 제한 없이 시스템의 전체 **CPU** 리소스에 대한 액세스를 상속합니다.

4.

제어 그룹에 대한 **CPU** 제한을 구성합니다.

```
# echo "1000000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
# echo "200000" > /sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
```

cpu.cfs_period_us 파일은 제어 그룹의 **CPU** 리소스에 대한 액세스를 제한당해야 하는 빈도

에 대한 마이크로초(여기에서 "us"로 표시되는 시간)를 나타냅니다. 상한값은 1초이고 하한값은 1000마이크로초입니다.

cpu.cfs_quota_us 파일은 제어 그룹의 모든 프로세스가 하나의 기간(**cpu.cfs_period_us**에 정의한 대로) 동안 실행되는 총 시간(마이크로초)을 나타냅니다. 제어 그룹의 프로세스가 한 기간 동안 할당량에 지정된 모든 시간을 사용하는 즉시 나머지 기간 동안 제한되며 다음 기간까지 실행할 수 없습니다. 하한값은 1000마이크로초입니다.

위의 예제 명령은 CPU 시간 제한을 설정하여 **Example** 제어 그룹에서 집계된 모든 프로세스가 1초(**cpu.cfs_quota_us**로 정의됨)마다 0.2초(**cpu.cfs_period_us**로 정의됨) 동안만 실행할 수 있습니다.

5.

선택적으로 제한을 확인합니다.

```
# cat /sys/fs/cgroup/cpu/Example/cpu.cfs_period_us
/sys/fs/cgroup/cpu/Example/cpu.cfs_quota_us
1000000
200000
```

6.

Example 제어 그룹에 애플리케이션의 PID를 추가합니다.

```
# echo "6955" > /sys/fs/cgroup/cpu/Example/cgroup.procs

or

# echo "6955" > /sys/fs/cgroup/cpu/Example/tasks
```

이전 명령을 사용하면 원하는 애플리케이션이 **Example** 제어 그룹의 구성원이 되므로 **Example** 제어 그룹에 대해 구성된 CPU 제한을 초과하지 않습니다. PID는 시스템의 기존 프로세스를 나타냅니다. 여기에서 PID 6955는 CPU 컨트롤러의 사용 사례를 설명하는 데 사용되는 **sha1sum /dev/zero &** 프로세스에 할당되었습니다.

7.

애플리케이션이 지정된 제어 그룹에서 실행되는지 확인합니다.

```
# cat /proc/6955/cgroup
12:cpuset:/
11:hugetlb:/
10:net_cls,net_prio:/
9:memory:/user.slice/user-1000.slice/user@1000.service
8:devices:/user.slice
7:blkio:/
6:freezer:/
5:rdma:/
```

```
4:pids:/user.slice/user-1000.slice/user@1000.service
3:perf_event:/
2:cpu,cpuacct:/Example
1:name=systemd:/user.slice/user-1000.slice/user@1000.service/gnome-terminal-
server.service
```

위의 예제 출력은 원하는 애플리케이션의 프로세스가 **Example** 제어 그룹에서 실행되며 애플리케이션의 프로세스에 **CPU** 제한을 적용하는 것을 보여줍니다.

8.

제한된 애플리케이션의 현재 **CPU** 사용량을 식별합니다.

```
# top
top - 12:28:42 up 1:06, 1 user, load average: 1.02, 1.02, 1.00
Tasks: 266 total, 6 running, 260 sleeping, 0 stopped, 0 zombie
%Cpu(s): 11.0 us, 1.2 sy, 0.0 ni, 87.5 id, 0.0 wa, 0.2 hi, 0.0 si, 0.2 st
MiB Mem : 1826.8 total, 287.1 free, 1054.4 used, 485.3 buff/cache
MiB Swap: 1536.0 total, 1396.7 free, 139.2 used. 608.3 avail Mem

  PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6955 root    20  0 228440 1752 1472 R 20.6  0.1 47:11.43 sha1sum
 5760 jdoe    20  0 3604956 208832 65316 R 2.3 11.2 0:43.50 gnome-shell
 6448 jdoe    20  0 743836 31736 19488 S 0.7  1.7 0:08.25 gnome-terminal-
 505 root    20  0  0  0  0 I 0.3  0.0 0:03.39 kworker/u4:4-events_unbound
 4217 root    20  0 74192 1612 1320 S 0.3  0.1 0:01.19 spice-vdagentd
...
```

PID 6955의 **CPU** 사용량이 **99%**에서 **20%**로 감소했습니다.



중요

`cpu.cfs_period_us` 및 `cpu.cfs_quota_us`에 해당하는 `cgroups-v2` 파일은 `cpu.max` 파일입니다. `cpu.max` 파일은 `cpu` 컨트롤러를 통해 사용할 수 있습니다.

추가 리소스

- [제어 그룹 이해](#)
- [커널 리소스 컨트롤러는 무엇입니까.](#)
- [cgroups\(7\), sysfs\(5\) 도움말 페이지](#)

[1]

Linux Control Group v2 - Waiman Long의 Devconf.cz 2019 프레젠테이션 소개

21장. CGROUPS-V2를 사용하여 애플리케이션의 CPU 시간 분배 제어

일부 애플리케이션은 너무 많은 CPU 시간을 사용하므로 환경의 전체 상태에 부정적인 영향을 미칠 수 있습니다. 애플리케이션을 제어 그룹 버전 2 (cgroups-v2)에 배치하고 해당 제어 그룹의 CPU 제한을 구성할 수 있습니다. 따라서 CPU 소비에서 애플리케이션을 규제할 수 있습니다.

사용자는 제어 그룹에 할당된 CPU 시간 분배를 제어하는 두 가지 방법을 제공합니다.

- **CPU 대역폭 설정 (cpu.max 컨트롤러 파일 편집)**
- **CPU 가중치 설정 (cpu.weight 컨트롤러 파일 편집)**

21.1. CGROUPS-V2 마운트

부팅 프로세스 중에 RHEL 8은 기본적으로 cgroup-v1 가상 파일 시스템을 마운트합니다. 애플리케이션에 대한 리소스를 제한하는 cgroup-v2 기능을 활용하려면 시스템을 수동으로 구성합니다.

사전 요구 사항

- 루트 권한이 있습니다.

절차

1.

systemd 시스템 및 서비스 관리자가 시스템 부팅 중에 **cgroups-v2**를 기본적으로 마운트하도록 시스템을 구성합니다.

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --
args="systemd.unified_cgroup_hierarchy=1"
```

그러면 현재 부팅 항목에 필요한 커널 명령줄 매개 변수가 추가됩니다.

모든 커널 부팅 항목에 **systemd.unified_cgroup_hierarchy=1** 매개 변수를 추가하려면 다음을 수행합니다.

```
# grubby --update-kernel=ALL --args="systemd.unified_cgroup_hierarchy=1"
```

2.

변경 사항을 적용하려면 시스템을 재부팅합니다.

검증 단계

1.

선택적으로 **cgroups-v2** 파일 시스템이 마운트되었는지 확인합니다.

```
# mount -l | grep cgroup
cgroup2 on /sys/fs/cgroup type cgroup2
(rw,nosuid,nodev,noexec,relatime,seclabel,nsdelegate)
```

cgroups-v2 파일 시스템이 **/sys/fs/cgroup/** 디렉터리에 성공적으로 마운트되었습니다.

2.

선택적으로 **/sys/fs/cgroup/** 디렉터리의 콘텐츠를 검사합니다.

```
# ll /sys/fs/cgroup/
-r--r--r--. 1 root root 0 Apr 29 12:03 cgroup.controllers
-rw-r--r--. 1 root root 0 Apr 29 12:03 cgroup.max.depth
-rw-r--r--. 1 root root 0 Apr 29 12:03 cgroup.max.descendants
-rw-r--r--. 1 root root 0 Apr 29 12:03 cgroup.procs
-r--r--r--. 1 root root 0 Apr 29 12:03 cgroup.stat
-rw-r--r--. 1 root root 0 Apr 29 12:18 cgroup.subtree_control
-rw-r--r--. 1 root root 0 Apr 29 12:03 cgroup.threads
-rw-r--r--. 1 root root 0 Apr 29 12:03 cpu.pressure
-r--r--r--. 1 root root 0 Apr 29 12:03 cpuset.cpus.effective
-r--r--r--. 1 root root 0 Apr 29 12:03 cpuset.mems.effective
-r--r--r--. 1 root root 0 Apr 29 12:03 cpu.stat
drwxr-xr-x. 2 root root 0 Apr 29 12:03 init.scope
-rw-r--r--. 1 root root 0 Apr 29 12:03 io.pressure
-r--r--r--. 1 root root 0 Apr 29 12:03 io.stat
-rw-r--r--. 1 root root 0 Apr 29 12:03 memory.pressure
-r--r--r--. 1 root root 0 Apr 29 12:03 memory.stat
drwxr-xr-x. 69 root root 0 Apr 29 12:03 system.slice
drwxr-xr-x. 3 root root 0 Apr 29 12:18 user.slice
```

기본적으로 루트 제어 그룹 이라고도 하는 **/sys/fs/cgroup/** 디렉터리에는 인터페이스 파일 (**CGroup** 시작) 및 **cpuset.cpus.effective** 와 같은 컨트롤러별 파일이 포함되어 있습니다. 또한 **systemd** 와 관련된 일부 디렉터리(예: **/sys/fs/cgroup/init.scope**, **/sys/fs/cgroup/system.slice**, **/sys/fs/cgroup/user.slice**).

추가 리소스

•

[제어 그룹 이해](#)

- [커널 리소스 컨트롤러는 무엇입니까.](#)
- [cgroups\(7\), sysfs\(5\) 도움말 페이지](#)

21.2. CPU 시간 분배를 위해 CGROUP 준비

애플리케이션의 CPU 사용을 제어하려면 특정 CPU 컨트롤러를 활성화하고 전용 제어 그룹을 생성해야 합니다. **cgroup** 파일의 조직의 명확성을 유지하기 위해 **/sys/fs/cgroup/** root 제어 그룹 내에 두 개 이상의 하위 제어 그룹을 생성하는 것이 좋습니다.

사전 요구 사항

- 제어하려는 프로세스의 **PID**를 확인했습니다.
- 루트 권한이 있습니다.
- **cgroups-v2** 파일 시스템을 마운트했습니다.

절차

1.

제한하려는 CPU 소비 애플리케이션의 **PID**(프로세스 ID)를 식별합니다.

```
# top
Tasks: 104 total, 3 running, 101 sleeping, 0 stopped, 0 zombie
%Cpu(s): 17.6 us, 81.6 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.8 hi, 0.0 si, 0.0 st
MiB Mem : 3737.4 total, 3312.7 free, 133.3 used, 291.4 buff/cache
MiB Swap: 4060.0 total, 4060.0 free, 0.0 used. 3376.1 avail Mem

  PID USER   PR NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 34578 root    20  0 18720 1756 1468 R 99.0  0.0   0:31.09 sha1sum
 34579 root    20  0 18720 1772 1480 R 99.0  0.0   0:30.54 sha1sum
    1 root    20  0 186192 13940 9500 S  0.0  0.4   0:01.60 systemd
    2 root    20  0     0     0     0 S  0.0  0.0   0:00.01 kthreadd
    3 root     0 -20     0     0     0 I  0.0  0.0   0:00.00 rcu_gp
    4 root     0 -20     0     0     0 I  0.0  0.0   0:00.00 rcu_par_gp
...
```

예제 출력은 **PID 34578** 및 **34 579** (**sha1sum**의 설명 애플리케이션 2개)가 CPU라는 많은 리소스를 사용한다는 것을 알 수 있습니다. 두 애플리케이션 모두 **cgroups-v2** 기능 관리를 시연하

는 데 사용되는 예제 애플리케이션입니다.

2.

cpu 및 **cpu set** 컨트롤러를 **/sys/fs/cgroup/cgroup.controllers** 파일에서 사용할 수 있는지 확인합니다.

```
# cat /sys/fs/cgroup/cgroup.controllers
cpuset cpu io memory hugetlb pids rdma
```

3.

CPU 관련 컨트롤러를 활성화합니다.

```
# echo "+cpu" >> /sys/fs/cgroup/cgroup.subtree_control
# echo "+cpuset" >> /sys/fs/cgroup/cgroup.subtree_control
```

이러한 명령을 사용하면 **/sys/fs/cgroup/** 루트 제어 그룹의 즉시 하위 그룹에 대해 **cpu** 및 **cpuset** 컨트롤러를 사용할 수 있습니다. 하위 그룹은 기준에 따라 프로세스를 지정하고 각 프로세스에 제어 검사를 적용할 수 있는 위치입니다.

사용자는 모든 수준에서 **cgroup.subtree_control** 파일의 내용을 읽고 즉시 하위 그룹에서 활성화에 사용할 수 있는 컨트롤러를 파악할 수 있습니다.



참고

기본적으로 **root** 제어 그룹의 **/sys/fs/cgroup/cgroup.subtree_control** 파일에는 메모리 및 **Pids** 컨트롤러가 포함되어 있습니다.

4.

/sys/fs/cgroup/Example/ 디렉토리를 생성합니다.

```
# mkdir /sys/fs/cgroup/Example/
```

/sys/fs/cgroup/Example/ 디렉토리는 하위 그룹을 정의합니다. 또한 이전 단계에서 이 하위 그룹에 대해 **cpu** 및 **cpuset** 컨트롤러를 활성화했습니다.

/sys/fs/cgroup/Example/ 디렉토리를 생성하면 일부 **cgroups-v2** 인터페이스 파일과 **cpu** 및 **cpuset** 컨트롤러별 파일이 디렉토리에 자동으로 생성됩니다. **/sys/fs/cgroup/Example/** 디렉토리에는 메모리 및 **pids** 컨트롤러에 대한 컨트롤러별 파일도 포함되어 있습니다.

5.

선택적으로 새로 생성된 하위 제어 그룹을 검사합니다.

```
# ll /sys/fs/cgroup/Example/
-r--r--r--. 1 root root 0 Jun  1 10:33 cgroup.controllers
-r--r--r--. 1 root root 0 Jun  1 10:33 cgroup.events
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.freeze
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.max.depth
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.max.descendants
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.procs
-r--r--r--. 1 root root 0 Jun  1 10:33 cgroup.stat
-rw-r--r--. 1 root root 0 Jun  1 10:33 cgroup.subtree_control
...
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpuset.cpus
-r--r--r--. 1 root root 0 Jun  1 10:33 cpuset.cpus.effective
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpuset.cpus.partition
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpuset.mems
-r--r--r--. 1 root root 0 Jun  1 10:33 cpuset.mems.effective
-r--r--r--. 1 root root 0 Jun  1 10:33 cpu.stat
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpu.weight
-rw-r--r--. 1 root root 0 Jun  1 10:33 cpu.weight.nice
...
-r--r--r--. 1 root root 0 Jun  1 10:33 memory.events.local
-rw-r--r--. 1 root root 0 Jun  1 10:33 memory.high
-rw-r--r--. 1 root root 0 Jun  1 10:33 memory.low
...
-r--r--r--. 1 root root 0 Jun  1 10:33 pids.current
-r--r--r--. 1 root root 0 Jun  1 10:33 pids.events
-rw-r--r--. 1 root root 0 Jun  1 10:33 pids.max
```

예제 출력에는 **cpuset.cpus** 및 **cpu. max** 와 같은 파일이 표시됩니다. 이러한 파일은 **cpuset** 및 **cpu** 컨트롤러에 따라 다릅니다. **cpuset** 및 **cpu** 컨트롤러는 **/sys/fs/cgroup/cgroup_control** 파일을 사용하여 루트의(**/sys/fs/cgroup/cgroup_control**) 직접 하위 제어 그룹에 대해 수동으로 활성화됩니다.

디렉터리에는 활성화된 컨트롤러에 관계없이 모든 제어 그룹에 공통된 **cgroup.procs** 또는 **cgroup.controllers** 와 같은 일반적인 **cgroup** 제어 인터페이스 파일도 포함되어 있습니다.

memory.high 및 **pids. max** 와 같은 파일은 루트 제어 그룹(**/sys/fs/cgroup/**)에 있으며 항상 기본적으로 활성화되는 메모리 및 **pids** 컨트롤러와 관련이 있습니다.

기본적으로 새로 생성된 하위 그룹은 제한 없이 모든 시스템의 **CPU** 및 메모리 리소스에 대한 액세스를 상속합니다.

6.

/sys/fs/cgroup/Example/ 에서 **CPU** 관련 컨트롤러를 활성화하여 **CPU**와만 관련된 컨트롤러를 가져옵니다.

```
# echo "+cpu" >> /sys/fs/cgroup/Example/cgroup.subtree_control
# echo "+cpuset" >> /sys/fs/cgroup/Example/cgroup.subtree_control
```

이러한 명령을 사용하면 즉시 하위 제어 그룹에는 메모리 또는 피드 컨트롤러가 아닌 **CPU** 시간 배분을 규제하는 것과 관련된 컨트롤러만 사용할 수 있습니다.

7.

`/sys/fs/cgroup/Example/tasks/` 디렉토리를 생성합니다.

```
# mkdir /sys/fs/cgroup/Example/tasks/
```

`/sys/fs/cgroup/Example/tasks/` 디렉토리는 `cpu` 및 `cpu set` 컨트롤러에만 관련된 파일이 있는 하위 그룹을 정의합니다.

8.

선택적으로 다른 하위 제어 그룹을 검사합니다.

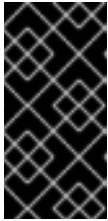
```
# ll /sys/fs/cgroup/Example/tasks
-r--r--r--. 1 root root 0 Jun  1 11:45 cgroup.controllers
-r--r--r--. 1 root root 0 Jun  1 11:45 cgroup.events
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.freeze
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.max.depth
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.max.descendants
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.procs
-r--r--r--. 1 root root 0 Jun  1 11:45 cgroup.stat
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.subtree_control
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.threads
-rw-r--r--. 1 root root 0 Jun  1 11:45 cgroup.type
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpu.max
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpu.pressure
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpuset.cpus
-r--r--r--. 1 root root 0 Jun  1 11:45 cpuset.cpus.effective
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpuset.cpus.partition
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpuset.mems
-r--r--r--. 1 root root 0 Jun  1 11:45 cpuset.mems.effective
-r--r--r--. 1 root root 0 Jun  1 11:45 cpu.stat
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpu.weight
-rw-r--r--. 1 root root 0 Jun  1 11:45 cpu.weight.nice
-rw-r--r--. 1 root root 0 Jun  1 11:45 io.pressure
-rw-r--r--. 1 root root 0 Jun  1 11:45 memory.pressure
```

9.

CPU 시간을 제어할 프로세스가 동일한 **CPU**에서 경쟁하는지 확인합니다.

```
# echo "1" > /sys/fs/cgroup/Example/tasks/cpuset.cpus
```

이전 명령을 실행하면 **Example/tasks** 하위 제어 그룹에 배치할 프로세스가 동일한 **CPU**에서 경쟁합니다. 이 설정은 **cpu** 컨트롤러가 활성화하는 데 중요합니다.



중요

cpu 컨트롤러는 관련 하위 제어 그룹에 단일 **CPU**에서 시간을 위해 경쟁하는 최소 2개의 프로세스가 있는 경우에만 활성화됩니다.

검증 단계

1.

선택사항: 즉시 하위 **cgroups**에 대해 **CPU** 관련 컨트롤러가 활성화되었는지 확인합니다.

```
# cat /sys/fs/cgroup/cgroup.subtree_control
/sys/fs/cgroup/Example/cgroup.subtree_control
cpuset cpu memory pids
cpuset cpu
```

2.

선택사항: **CPU** 시간을 제어할 프로세스가 동일한 **CPU**에서 경쟁할지 확인합니다.

```
# cat /sys/fs/cgroup/Example/tasks/cpuset.cpus
1
```

추가 리소스

- [제어 그룹 이해](#)
- [커널 리소스 컨트롤러는 무엇입니까.](#)
- [cgroups-v2 마운트](#)
- [cgroups\(7\), sysfs\(5\) 도움말 페이지](#)

21.3. CPU 대역폭을 조정하여 애플리케이션 CPU 시간 분배 제어

CPU 시간을 특정 **cgroup** 트리의 애플리케이션에 분산하기 위해 **CPU** 컨트롤러의 관련 파일에 값을 할당해야 합니다.

사전 요구 사항

- 루트 권한이 있습니다.
- CPU 시간 배분을 제어하려는 애플리케이션이 두 개 이상 있습니다.
- CPU 시간 배분을 위해 **cgroup** 준비에 설명된 것과 동일한 CPU에서 관련 애플리케이션을 경쟁하게 되었습니다.
- **cgroups-v2** 마운트에 설명된 대로 **cgroups-v2** 파일 시스템을 마운트 했습니다.
- CPU 시간 분산을 위해 **cgroup** 준비에 설명된 대로 상위 제어 그룹과 하위 제어 그룹에서 모두 **cpu** 및 **cpuset** 컨트롤러를 활성화했습니다.
- 아래 예제와 같이 **/sys/fs/cgroup/** root 제어 그룹 내부에 두 단계의 하위 제어 그룹을 생성했습니다.

```

...
|— Example
|   |— tasks
...

```

절차

1.

제어 그룹 내에서 리소스 제한을 수행하도록 CPU 대역폭을 구성합니다.

```
# echo "200000 1000000" > /sys/fs/cgroup/Example/tasks/cpu.max
```

첫 번째 값은 마이크로초(마이크로)에서 허용된 시간 할당량으로, 하위 그룹의 모든 프로세스를 한 기간 동안 전체적으로 실행할 수 있습니다. 두 번째 값은 기간 길이를 지정합니다.

단일 기간 동안 제어 그룹의 프로세스가 이 할당량에서 지정한 시간을 집합적으로 고갈하면 나머지 기간 동안 제한되고 다음 기간 동안 실행되지 않습니다.

이 명령은 CPU 시간 분배 제어를 설정하여 **/sys/fs/cgroup/Example/tasks** 하위 그룹의 모든 프로세스가 1초마다 0.2초 동안만 CPU에서 실행될 수 있습니다. 즉, 1초의 5분의 1이 됩니다.

2.

선택적으로 시간 할당량을 확인합니다.

```
# cat /sys/fs/cgroup/Example/tasks/cpu.max
200000 1000000
```

3.

애플리케이션의 **PID**를 **Example/tasks** 하위 그룹에 추가합니다.

```
# echo "34578" > /sys/fs/cgroup/Example/tasks/cgroup.procs
# echo "34579" > /sys/fs/cgroup/Example/tasks/cgroup.procs
```

예제 명령은 원하는 애플리케이션이 **Example/tasks** 하위 그룹의 구성원이 되어 이 하위 그룹에 대해 구성된 **CPU** 시간 배분을 초과하지 않도록 합니다.

검증 단계

1.

애플리케이션이 지정된 제어 그룹에서 실행되는지 확인합니다.

```
# cat /proc/34578/cgroup /proc/34579/cgroup
0::/Example/tasks
0::/Example/tasks
```

위의 출력은 **Example/tasks** 하위 그룹에서 실행되는 지정된 애플리케이션의 프로세스를 보여줍니다.

2.

제한된 애플리케이션의 현재 **CPU** 사용량을 검사합니다.

```
# top
top - 11:13:53 up 23:10, 1 user, load average: 0.26, 1.33, 1.66
Tasks: 104 total, 3 running, 101 sleeping, 0 stopped, 0 zombie
%Cpu(s): 3.0 us, 7.0 sy, 0.0 ni, 89.5 id, 0.0 wa, 0.2 hi, 0.2 si, 0.2 st
MiB Mem : 3737.4 total, 3312.6 free, 133.4 used, 291.4 buff/cache
MiB Swap: 4060.0 total, 4060.0 free, 0.0 used. 3376.0 avail Mem

  PID USER  PR  NI  VIRT  RES  SHR S %CPU %MEM  TIME+ COMMAND
34578 root   20   0 18720 1756 1468 R 10.0  0.0 37:36.13 sha1sum
34579 root   20   0 18720 1772 1480 R 10.0  0.0 37:41.22 sha1sum
   1 root   20   0 186192 13940 9500 S  0.0  0.4  0:01.60 systemd
   2 root   20   0    0    0    0 S  0.0  0.0  0:00.01 kthreadd
   3 root    0 -20    0    0    0 I  0.0  0.0  0:00.00 rcu_gp
   4 root    0 -20    0    0    0 I  0.0  0.0  0:00.00 rcu_par_gp
...
```

PID 34578 및 **PID 34 579**의 CPU 사용량이 10%로 감소했습니다. **Example/tasks** 하위 그룹은 프로세스를 전체적으로 CPU 시간의 20%로 조정합니다. 제어 그룹에는 두 개의 프로세스가 있으므로 각각 CPU 시간의 10%를 활용할 수 있습니다.

추가 리소스

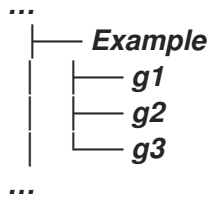
- [제어 그룹 이해](#)
- [커널 리소스 컨트롤러는 무엇입니까.](#)
- [cgroups-v2 마운트](#)
- [CPU 시간 분배를 위해 cgroup 준비](#)
- [cgroups\(7\), sysfs\(5\) 도움말 페이지](#)

21.4. CPU 가중치를 조정하여 애플리케이션의 CPU 시간 분배 제어

CPU 시간을 특정 **cgroup** 트리의 애플리케이션에 분산하기 위해 **CPU** 컨트롤러의 관련 파일에 값을 할당해야 합니다.

사전 요구 사항

- 루트 권한이 있습니다.
- CPU 시간 배분을 제어하려는 애플리케이션이 있습니다.
- CPU 시간 배분을 위해 **cgroup** 준비에 설명된 것과 동일한 CPU에서 관련 애플리케이션을 경쟁하게 되었습니다.
- **cgroups-v2** 마운트에 설명된 대로 **cgroups-v2** 파일 시스템을 마운트 했습니다.
- 다음 예와 같이 **/sys/fs/cgroup/ root** 제어 그룹 내에 하위 제어 그룹의 두 가지 수준 계층 구조를 생성했습니다.



- **CPU 시간 분산을 위해 cgroup** 준비에 설명된 대로 상위 제어 그룹 및 하위 제어 그룹에서 **cpu** 및 **cpu set** 컨트롤러를 활성화했습니다.

절차

1.

제어 그룹 내에서 리소스 제한을 달성하도록 원하는 **CPU** 가중치를 구성합니다.

```
# echo "150" > /sys/fs/cgroup/Example/g1/cpu.weight
# echo "100" > /sys/fs/cgroup/Example/g2/cpu.weight
# echo "50" > /sys/fs/cgroup/Example/g3/cpu.weight
```

2.

애플리케이션의 **PID**를 **g1**, **g2** 및 **g3** 하위 그룹에 추가합니다.

```
# echo "33373" > /sys/fs/cgroup/Example/g1/cgroup.procs
# echo "33374" > /sys/fs/cgroup/Example/g2/cgroup.procs
# echo "33377" > /sys/fs/cgroup/Example/g3/cgroup.procs
```

예제 명령을 사용하면 원하는 애플리케이션이 예제/**g*** 하위 **cgroups**의 멤버가 되고 해당 **cgroups**의 구성에 따라 **CPU** 시간이 배포됩니다.

프로세스가 실행 중인 하위 **cgroup(g1,g2, g3)**의 가중치는 상위 **cgroup(Example)** 수준에서 요약됩니다. 그런 다음 **CPU** 리소스는 해당 가중치에 비례하여 배포됩니다.

결과적으로 모든 프로세스가 동시에 실행되면 커널은 해당 **cgroup**의 **cpu.weight** 파일에 따라 각 **CPU** 시간을 비례합니다.

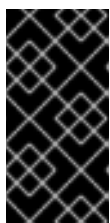
하위 cgroup	cpu.weight file	CPU 시간 할당
g1	150	~50% (150/300)
g2	100	~33% (100/300)

하위 cgroup	cpu.weight file	CPU 시간 할당
g3	50	~16% (50/300)

cpu.weight 컨트롤러 파일의 값은 백분율이 아닙니다.

한 프로세스가 중단되어 **cgroup g2**에 실행 중인 프로세스가 없으면 계산에서는 **cgroup g2**가 생략되고 **cgroup g 1** 및 **g 3**의 가중치만 줄어듭니다:

하위 cgroup	cpu.weight file	CPU 시간 할당
g1	150	~75% (150/200)
g3	50	~25% (50/200)



중요

하위 **cgroup**에 여러 개의 실행 중인 프로세스가 있는 경우 해당 **cgroup**에 할당된 **CPU** 시간이 해당 **cgroup**의 멤버 프로세스에 동일하게 배포됩니다.

검증

1.

애플리케이션이 지정된 제어 그룹에서 실행되는지 확인합니다.

```
# cat /proc/33373/cgroup /proc/33374/cgroup /proc/33377/cgroup
0::/Example/g1
0::/Example/g2
0::/Example/g3
```

명령 출력에는 **Example/g*** 하위 **cgroup**에서 실행되는 지정된 애플리케이션의 프로세스가 표시됩니다.

2.

제한된 애플리케이션의 현재 **CPU** 사용량을 검사합니다.

```
# top
top - 05:17:18 up 1 day, 18:25, 1 user, load average: 3.03, 3.03, 3.00
Tasks: 95 total, 4 running, 91 sleeping, 0 stopped, 0 zombie
%Cpu(s): 18.1 us, 81.6 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.3 hi, 0.0 si, 0.0 st
MiB Mem : 3737.0 total, 3233.7 free, 132.8 used, 370.5 buff/cache
```

MiB Swap: 4060.0 total, 4060.0 free, 0.0 used. 3373.1 avail Mem

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
33373	root	20	0	18720	1748	1460	R	49.5	0.0	415:05.87	sha1sum
33374	root	20	0	18720	1756	1464	R	32.9	0.0	412:58.33	sha1sum
33377	root	20	0	18720	1860	1568	R	16.3	0.0	411:03.12	sha1sum
760	root	20	0	416620	28540	15296	S	0.3	0.7	0:10.23	tuned
1	root	20	0	186328	14108	9484	S	0.0	0.4	0:02.00	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthread

...



참고

명확한 설명을 위해 모든 예제 프로세스를 단일 **CPU**에서 실행하도록 했습니다. **CPU** 가중치는 여러 **CPU**에서 사용되는 경우에도 동일한 원칙을 적용합니다.

PID 33373, PID 33374 및 PID 33 377의 **CPU** 리소스가 해당 하위 **cgroup**에 할당된 가중치 **150, 100, 50**에 따라 할당되었습니다. 가중치는 각 애플리케이션의 **CPU** 시간을 약 **50%, 33%, 16%**에 해당합니다.

추가 리소스

- [제어 그룹 이해](#)
- [커널 리소스 컨트롤러란](#)
- [cgroups-v2 마운트](#)
- [CPU 시간 분배를 위해 cgroup 준비](#)
- [리소스 배포 모델](#)
- [cgroups\(7\), sysfs\(5\) 도움말 페이지](#)

22장. SYSTEMD에서 제어 그룹 버전 1 사용

다음 섹션에서는 **cgroup**(제어 그룹) 생성, 수정 및 제거와 관련된 작업에 대한 개요를 제공합니다. **systemd** 시스템 및 서비스 관리자가 제공하는 유틸리티는 **cgroup** 관리의 기본 방식이며 향후 지원됩니다.

22.1. 제어 그룹 버전 1에서 SYSTEMD 역할

RHEL 8은 **cgroup** 계층 구조의 시스템에 **systemd** 장치 트리를 바인딩하여 리소스 관리 설정을 프로세스 수준에서 애플리케이션 수준으로 이동합니다. 따라서 **systemctl** 명령을 사용하거나 **systemd** 유닛 파일을 수정하여 시스템 리소스를 관리할 수 있습니다.

기본적으로 **systemd** 시스템 및 서비스 관리자는 슬라이스, 범위 및 서비스 유닛을 사용하여 제어 그룹에서 프로세스를 구성하고 구조화합니다. **systemctl** 명령을 사용하면 사용자 지정 슬라이스를 생성하여 이 구조를 추가로 수정할 수 있습니다. 또한 **systemd** 는 중요한 커널 리소스 컨트롤러의 계층 구조를 **/sys/fs/cgroup/** 디렉터리에 자동으로 마운트합니다.

리소스 제어에는 세 가지 **systemd** 장치 유형이 사용됩니다.

- service** - 장치 구성 파일에 따라 **systemd** 가 시작된 프로세스 또는 프로세스 그룹입니다. 서비스는 지정된 프로세스를 캡슐화하여 하나의 집합으로 시작하고 중지할 수 있도록 합니다. 서비스의 이름은 다음과 같은 방식으로 지정됩니다.

<name>.service

- 범위** - 외부에서 생성된 프로세스 그룹입니다. 범위는 **fork()** 함수를 통해 임의의 프로세스에 의해 시작 및 중지된 다음 런타임 시 **systemd** 에 의해 등록된 프로세스를 캡슐화합니다. 예를 들어 사용자 세션, 컨테이너 및 가상 시스템은 범위로 처리됩니다. 범위는 다음과 같이 이름이 지정됩니다.

<name>.scope

- 슬라이스** - 계층적으로 구성된 유닛 그룹입니다. 슬라이스에서는 범위 및 서비스가 배치되는 계층 구조를 구성합니다. 실제 프로세스는 범위 또는 서비스에 포함되어 있습니다. 슬라이스 유닛의 모든 이름은 계층 구조의 위치에 대한 경로에 해당합니다. 대시("-") 문자는 **-.slice** 루트 슬라이스에서 슬라이스에 대한 경로 구성 요소의 구분 기호 역할을 합니다. 다음 예에서 다음을 수행합니다.

<parent-name>.slice

parent-name.slice 는 **parent.slice** 의 하위 슬라이스로, **-.slice** 루트 슬라이스의 하위 슬라이스입니다. **parent-name.slice** 는 **parent-name-name2.slice** 라는 고유한 하위 슬라이스를 가질 수 있습니다.

서비스, 범위 및 슬라이스 유닛은 제어 그룹 계층 구조의 오브젝트에 직접 매핑됩니다. 이러한 유닛이 활성화되면 유닛 이름에서 빌드된 그룹 경로를 제어하기 위해 직접 매핑됩니다.

다음은 제어 그룹 계층 구조의 축약된 예입니다.

```
Control group /:
-.slice
├─user.slice
│   └─user-42.slice
│       └─session-c1.scope
│           └─967 gdm-session-worker [pam/gdm-launch-environment]
│               └─1035 /usr/libexec/gdm-x-session gnome-session --autostart
│                   /usr/share/gdm/greeter/autostart
│                       └─1054 /usr/libexec/Xorg vt1 -displayfd 3 -auth /run/user/42/gdm/Xauthority -
background none -noreset -keeppty -verbose 3
│                       └─1212 /usr/libexec/gnome-session-binary --autostart /usr/share/gdm/greeter/autostart
│                       └─1369 /usr/bin/gnome-shell
│                       └─1732 ibus-daemon --xim --panel disable
│                       └─1752 /usr/libexec/ibus-dconf
│                       └─1762 /usr/libexec/ibus-x11 --kill-daemon
│                       └─1912 /usr/libexec/gsd-xsettings
│                       └─1917 /usr/libexec/gsd-a11y-settings
│                       └─1920 /usr/libexec/gsd-clipboard
...
├─init.scope
│   └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
├─system.slice
│   └─rngd.service
│       └─800 /sbin/rngd -f
│   └─systemd-udev.service
│       └─659 /usr/lib/systemd/systemd-udev
│   └─chronyd.service
│       └─823 /usr/sbin/chronyd
│   └─auditd.service
│       └─761 /sbin/auditd
│       └─763 /usr/sbin/sedispatch
│   └─accounts-daemon.service
│       └─876 /usr/libexec/accounts-daemon
│   └─example.service
│       └─929 /bin/bash /home/jdoe/example.sh
│       └─4902 sleep 1
...
```

위의 예제에서는 서비스와 범위에 프로세스가 포함되어 있으며 자체 프로세스가 포함되지 않은 슬라이스에 배치됩니다.

추가 리소스

- [Red Hat Enterprise Linux에서 기본 시스템 설정 구성](#)
- [커널 리소스 컨트롤러란](#)
- [systemd.resource-control\(5\), cgroups\(7\), fork \(\), fork\(2\) 매뉴얼 페이지](#)

22.2. 일시적인 제어 그룹 생성

Transient cgroup 은 런타임 중에 단위(서비스 또는 범위)에서 사용하는 리소스에 대한 제한을 설정합니다.

절차

- 임시 제어 그룹을 생성하려면 다음 형식으로 **systemd-run** 명령을 사용합니다.

```
# systemd-run --unit=<name> --slice=<name>.slice <command>
```

이 명령은 일시적인 서비스 또는 범위 유닛을 생성 및 시작하며 이러한 유닛에서 사용자 지정 명령을 실행합니다.

- **unit =<name>** 옵션은 유닛에 이름을 제공합니다. **--unit** 을 지정하지 않으면 이름이 자동으로 생성됩니다.
- **--slice=<name>.slice** 옵션은 서비스 또는 범위 장치를 지정된 슬라이스의 멤버로 만듭니다. **<name>.slice** 를 **systemctl -t** 슬라이스의 출력에 표시된 것처럼 기존 슬라이스의 이름으로 교체하거나 고유한 이름을 전달하여 새 슬라이스를 생성합니다. 기본적으로 서비스 및 범위는 **system.slice** 의 구성원으로 생성됩니다.
- **<command>** 를 서비스 또는 범위 유닛에서 실행하려는 명령으로 바꿉니다.

서비스 또는 범위가 성공적으로 생성 및 시작되었는지 확인하려면 다음 메시지가 표시됩니다.

Running as unit <name>.service

- 선택적으로 런타임 정보를 수집하기 위해 프로세스가 완료된 후 장치를 계속 실행합니다.

systemd-run --unit=<name> --slice=<name>.slice --remain-after-exit <command>

명령은 일시적인 서비스 유닛을 생성 및 시작하고 이러한 유닛에서 사용자 지정 명령을 실행합니다. **remain -after-exit** 옵션을 사용하면 프로세스가 완료된 후에도 서비스가 계속 실행됩니다.

추가 리소스

- [제어 그룹 이해](#)
- [제어 그룹에서 **systemd** 역할](#)
- [RHEL에서 기본 시스템 설정 구성](#)
- [systemd-run\(1\) 매뉴얼 페이지](#)

22.3. 영구 제어 그룹 생성

서비스에 영구 제어 그룹을 할당하려면 유닛 구성 파일을 편집해야 합니다. 구성은 시스템 재부팅 후에 유지되므로 자동으로 시작되는 서비스를 관리하는 데 사용할 수 있습니다.

절차

- 영구 제어 그룹을 생성하려면 다음을 실행합니다.

systemctl enable <name>.service

위의 명령은 `/usr/lib/systemd/system/` 디렉토리에 자동으로 유닛 구성 파일을 생성하고 기본적으로 **<name>.service** 를 **system.slice** 유닛에 할당합니다.

추가 리소스

- 제어 그룹 이해
- 제어 그룹에서 **systemd** 역할
- **RHEL**에서 기본 시스템 설정 구성
- **systemd-run(1)** 매뉴얼 페이지

22.4. 명령줄에서 메모리 리소스 제어 설정 구성

명령줄 인터페이스에서 명령 실행은 프로세스 그룹의 하드웨어 리소스에 대한 제한을 설정, 우선 순위화 또는 제어하는 방법 중 하나입니다.

절차

- 서비스의 메모리 사용량을 제한하려면 다음을 실행합니다.

```
# systemctl set-property example.service MemoryMax=1500K
```

명령은 **1500KB**의 메모리 제한을 **example.service** 서비스가 속하는 제어 그룹에서 실행되는 프로세스에 즉시 할당합니다. 이 구성 변형에서 **MemoryMax** 매개변수는 **/etc/systemd/system.control/example.service.service.d/50-MemoryMax.conf** 파일에 정의되어 있으며 **/sys/fs/cgroup/memory/system.slice/example.service/memory.limit_in_bytes** 파일의 값을 제어합니다.

- 필요한 경우 서비스의 메모리 사용량을 일시적으로 제한하려면 다음을 실행합니다.

```
# systemctl set-property --runtime example.service MemoryMax=1500K
```

명령은 메모리 제한을 **example.service** 서비스에 즉시 할당합니다. **MemoryMax** 매개변수는 **/run/systemd/system.control/example.service.d/50-MemoryMax.conf** 파일을 다음 재부팅할 때까지 정의됩니다. 재부팅하면 전체 **/run/systemd/system.control/** 디렉토리 및 **MemoryMax**가 제거됩니다.



참고

50-MemoryMax.conf 파일은 메모리 제한을 4096바이트(AMD64 및 Intel 64에 고유한 하나의 커널 페이지 크기)로 저장합니다. 실제 바이트 수는 CPU 아키텍처에 따라 다릅니다.

추가 리소스

- [제어 그룹 이해](#)
- [커널 리소스 컨트롤러는 무엇입니까.](#)
- [systemd.resource-control\(5\) 및 cgroups\(7\) 도움말 페이지](#)
- [제어 그룹에서 systemd 역할](#)

22.5. 장치 파일로 메모리 리소스 제어 설정 구성

각 영구 단위는 **systemd** 시스템 및 서비스 관리자가 감독하며 **/usr/lib/systemd/system/** 디렉터리에 유닛 구성 파일이 있습니다. 영구 유닛의 리소스 제어 설정을 변경하려면 텍스트 편집기 또는 명령줄 인터페이스에서 해당 유닛 구성 파일을 수동으로 수정합니다.

장치 파일을 수동으로 수정하는 것은 프로세스 그룹의 하드웨어 리소스에 대한 제한을 설정, 우선 순위화 또는 제어하는 방법 중 하나입니다.

절차

1. 서비스의 메모리 사용량을 제한하려면 다음과 같이 **/usr/lib/systemd/system/example.service** 파일을 수정합니다.

```
...
[Service]
MemoryMax=1500K
...
```

위의 구성에서는 제어 그룹에서 실행되는 프로세스의 최대 메모리 사용 제한(예. **.service**)을 배치합니다.



참고

K, M, G 또는 **T** 접미사를 사용하여 킬로바이트, 메가바이트, 기가바이트 또는 테라바이트를 측정 단위로 식별합니다.

2.

모든 장치 구성 파일을 다시 로드합니다.

```
# systemctl daemon-reload
```

3.

서비스를 다시 시작하십시오.

```
# systemctl restart example.service
```

4.

시스템을 재부팅합니다.

5.

선택적으로 변경 사항이 적용되었는지 확인합니다.

```
# cat /sys/fs/cgroup/memory/system.slice/example.service/memory.limit_in_bytes
1536000
```

예제 출력에서는 메모리 사용량이 약 **1,500KB**로 제한되었음을 보여줍니다.



참고

memory.limit_in_bytes 파일은 메모리 제한을 **AMD64** 및 **Intel 64**에 고유한 **4096**바이트의 배수로 저장합니다. 실제 바이트 수는 **CPU** 아키텍처에 따라 다릅니다.

추가 리소스

•

[제어 그룹 이해](#)

•

[커널 리소스 컨트롤러는 무엇입니까.](#)

- [systemd.resource-control\(5\), cgroups\(7\) 도움말 페이지](#)
- [RHEL에서 기본 시스템 설정 구성](#)
- [제어 그룹에서 systemd 역할](#)

22.6. 일시적인 제어 그룹 제거

더 이상 프로세스 그룹의 하드웨어 리소스에 대한 액세스를 제한, 우선 순위를 설정하거나 제어할 필요가 없는 경우 **systemd** 시스템 및 서비스 관리자를 사용하여 **cgroup**(임시 제어 그룹)을 제거할 수 있습니다.

Transient cgroup 은 서비스 또는 범위 유닛에 포함된 모든 프로세스가 완료되고 나면 자동으로 릴리스됩니다.

절차

- 모든 프로세스와 함께 서비스 유닛을 중지하려면 다음을 실행합니다.

```
# systemctl stop name.service
```

- 하나 이상의 유닛 프로세스를 종료하려면 다음을 실행합니다.

```
# systemctl kill name.service --kill-who=PID,... --signal=<signal>
```

위의 명령은 **--kill-who** 옵션을 사용하여 종료할 제어 그룹에서 프로세스를 선택합니다. 동시에 여러 프로세스를 종료하려면 쉼표로 구분된 **PID** 목록을 전달합니다. **signal** 옵션은 지정된 프로세스에 전송할 **POSIX** 신호의 유형을 결정합니다. 기본 신호는 **SIGTERM** 입니다.

추가 리소스

- [제어 그룹 이해](#)
- [커널 리소스 컨트롤러란](#)

- [systemd.resource-control\(5\), cgroups\(7\) 도움말 페이지](#)
- [제어 그룹에서 systemd 역할](#)
- [RHEL에서 기본 시스템 설정 구성](#)

22.7. 영구 제어 그룹 제거

더 이상 프로세스 그룹의 하드웨어 리소스에 대한 액세스를 제한, 우선 순위를 설정하거나 제어할 필요가 없는 경우 **systemd** 시스템 및 서비스 관리자를 사용하여 **cgroup**(영구 제어 그룹)을 제거할 수 있습니다.

영구 **cgroup** 은 서비스 또는 범위 유닛이 중지되거나 비활성화되고 해당 구성 파일이 삭제되면 릴리스됩니다.

절차

1. 서비스 유닛을 중지합니다.

```
# systemctl stop <name>.service
```

2. 서비스 유닛을 비활성화합니다.

```
# systemctl disable <name>.service
```

3. 관련 장치 구성 파일을 제거합니다.

```
# rm /usr/lib/systemd/system/<name>.service
```

4. 변경 사항이 적용되도록 모든 장치 구성 파일을 다시 로드합니다.

```
# systemctl daemon-reload
```

추가 리소스

- [제어 그룹 이해](#)
- [커널 리소스 컨트롤러는 무엇입니까.](#)
- [systemd.resource-control\(5\), cgroups\(7\) 및 systemd.kill\(5\) 도움말 페이지](#)
- [제어 그룹에서 systemd 역할](#)
- [RHEL에서 기본 시스템 설정 구성](#)

22.8. SYSTEMD 장치 나열

다음 절차에서는 **systemd** 시스템 및 서비스 관리자를 사용하여 유닛을 나열하는 방법을 설명합니다.

절차

- 시스템의 활성 유닛을 모두 나열하려면 **# systemctl** 명령을 실행하고 터미널에서 다음 예와 유사한 출력을 반환합니다.

```
# systemctl
UNIT                                LOAD    ACTIVE SUB    DESCRIPTION
...
init.scope                         loaded active running  System and Service Manager
session-2.scope                    loaded active running  Session 2 of user jdoe
abrt-ccpp.service                  loaded active exited   Install ABRT coredump
hook
abrt-oops.service                  loaded active running  ABRT kernel log watcher
abrt-vmcore.service                loaded active exited   Harvest vmcores for
ABRT
abrt-xorg.service                  loaded active running  ABRT Xorg log watcher
...
-.slice                             loaded active active   Root Slice
machine.slice                      loaded active active   Virtual Machine and
Container Slice system-getty.slice loaded
active active system-getty.slice
system-lvm2\x2dpvscan.slice        loaded active active   system-
lvm2\x2dpvscan.slice
system-sshd\x2dkeygen.slice         loaded active active   system-
sshd\x2dkeygen.slice
system-systemd\x2dhibernate\x2dresume.slice loaded active active   system-
systemd\x2dhibernate\x2dresume>
```

```

system-user\x2druntime\x2ddir.slice      loaded active active  system-
user\x2druntime\x2ddir.slice
system.slice                             loaded active active  System Slice
user-1000.slice                          loaded active active  User Slice of UID 1000
user-42.slice                            loaded active active  User Slice of UID 42
user.slice                               loaded active active  User and Session Slice
...

```

- **UNIT** - 제어 그룹 계층 구조에서 유닛 위치를 반영하는 유닛의 이름입니다. 리소스 제어와 관련된 단위는 슬라이스, 범위 및 서비스입니다.
- **LOAD** - 유닛 구성 파일이 올바르게 로드되었는지를 나타냅니다. 유닛 파일을 로드하지 못한 경우 필드에 로드된 대신 상태 오류가 포함됩니다. 기타 단위 로드 상태는 **stub**, **merged** 및 **masked** 입니다.
- **ACTIVE - SUB**의 일반화인 고급 유닛 활성화 상태입니다.
- **SUB** - 하위 수준 유닛 활성화 상태입니다. 가능한 값은 유닛 유형에 따라 다릅니다.
- **DESCRIPTION** - 장치 내용 및 기능에 대한 설명입니다.

- 비활성 유닛을 나열하려면 다음을 실행합니다.

```
# systemctl --all
```

- 출력의 정보 양을 제한하려면 다음을 실행합니다.

```
# systemctl --type service,masked
```

type 옵션에는 서비스 및 슬라이스 와 같은 쉼표로 구분된 유닛 유형 목록 또는 로드된 및 마스킹 과 같은 단위 로드 상태가 필요합니다.

추가 리소스

- **RHEL**에서 [기본 시스템 설정 구성](#)

- **systemd.resource-control(5), systemd.exec(5) manual pages**

22.9. SYSTEMD 제어 그룹 계층 구조 보기

다음 절차에서는 특정 **cgroup**에서 실행되는 제어 그룹(**cgroup**) 계층 구조와 프로세스를 표시하는 방법을 설명합니다.

절차

- 시스템에 **cgroup** 전체 계층 구조를 표시하려면 다음을 실행합니다 # **systemd-cgls**:

```
# systemd-cgls
Control group /:
-.slice
├─user.slice
│ └─user-42.slice
│   └─session-c1.scope
│     ├── 965 gdm-session-worker [pam/gdm-launch-environment]
│     └─1040 /usr/libexec/gdm-x-session gnome-session --autostart
│       /usr/share/gdm/greeter/autostart
...
├─init.scope
│ └─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
└─system.slice
...
├─example.service
│ ├── 6882 /bin/bash /home/jdoe/example.sh
│ └─6902 sleep 1
├─systemd-journald.service
│ └─629 /usr/lib/systemd/systemd-journald
...
```

예제 출력은 가장 높은 수준이 슬라이스 에 의해 형성되는 전체 **cgroup** 계층 구조를 반환합니다.

- 리소스 컨트롤러에서 필터링한 **cgroup** 계층 구조를 표시하려면 # **systemd-cgls <resource_controller>** 를 실행합니다.

```
# systemd-cgls memory
Controller memory; Control group /:
├─1 /usr/lib/systemd/systemd --switched-root --system --deserialize 18
├─user.slice
│ └─user-42.slice
│   └─session-c1.scope
│     └─ 965 gdm-session-worker [pam/gdm-launch-environment]
```

```

...
└─system.slice
  /
  ...
  └─chronyd.service
    └─844 /usr/sbin/chronyd
  └─example.service
    └─8914 /bin/bash /home/jdoe/example.sh
    └─8916 sleep 1
  ...

```

위의 명령의 예제 출력은 선택한 컨트롤러와 상호 작용하는 서비스를 나열합니다.

- 특정 유닛 및 **cgroup** 계층 구조의 일부에 대한 자세한 정보를 표시하려면 **# systemctl status <system_unit>**:을 실행합니다.

```

# systemctl status example.service
● example.service - My example service
   Loaded: loaded (/usr/lib/systemd/system/example.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2019-04-16 12:12:39 CEST; 3s ago
 Main PID: 17737 (bash)
    Tasks: 2 (limit: 11522)
   Memory: 496.0K (limit: 1.5M)
    CGroup: /system.slice/example.service
            └─17737 /bin/bash /home/jdoe/example.sh
            └─17743 sleep 1

Apr 16 12:12:39 redhat systemd[1]: Started My example service.
Apr 16 12:12:39 redhat bash[17737]: The current time is Tue Apr 16 12:12:39 CEST 2019
Apr 16 12:12:40 redhat bash[17737]: The current time is Tue Apr 16 12:12:40 CEST 2019

```

추가 리소스

- [커널 리소스 컨트롤러란](#)
- [systemd.resource-control\(5\), cgroups\(7\) 도움말 페이지](#)

22.10. 리소스 컨트롤러 보기

다음 절차에서는 어떤 프로세스에서 어떤 리소스 컨트롤러를 사용하는지 학습하는 방법을 설명합니다.

절차

1.

프로세스가 상호 작용하는 리소스 컨트롤러를 보려면 **# cat proc/<PID>/cgroup** 명령을 실행합니다.

```
# cat /proc/11269/cgroup
12:freezer:/
11:cpuset:/
10:devices:/system.slice
9:memory:/system.slice/example.service
8:pids:/system.slice/example.service
7:hugetlb:/
6:rdma:/
5:perf_event:/
4:cpu,cpuacct:/
3:net_cls,net_prio:/
2:blkio:/
1:name=systemd:/system.slice/example.service
```

예제 출력은 관심 있는 프로세스와 관련이 있습니다. 이 경우 **PID 11269** 로 식별되며 **example.service** 유닛에 속하는 프로세스입니다. 프로세스가 **systemd** 유닛 파일 사양에 정의된 대로 올바른 제어 그룹에 배치되었는지 확인할 수 있습니다.



참고

기본적으로 모든 기본 리소스 컨트롤러를 자동으로 마운트하므로 리소스 컨트롤러 목록의 항목과 순서는 **systemd** 에서 시작한 모든 유닛에서 동일합니다.

추가 리소스

- **cgroups(7)** 도움말 페이지
- **/usr/share/doc/kernel-doc-<kernel_version>/Documentation/cgroups-v1/** 디렉터리에 있는 설명서

22.11. 리소스 사용량 모니터링

다음 절차에서는 현재 실행 중인 컨트롤 그룹(**cgroup**) 목록과 해당 리소스 사용을 실시간으로 확인하는 방법을 설명합니다.

절차

1.

현재 **cgroups** 를 실행 중인 동적 계정을 보려면 **# systemd-cgtop** 명령을 실행합니다.

```
# systemd-cgtop
Control Group           Tasks %CPU  Memory Input/s Output/s
/                        607 29.8  1.5G   -      -
/system.slice           125 -    428.7M -      -
/system.slice/ModemManager.service    3 -    8.6M  -      -
/system.slice/NetworkManager.service  3 -   12.8M -      -
/system.slice/accounts-daemon.service  3 -    1.8M -      -
/system.slice/boot.mount               - -   48.0K -      -
/system.slice/chronyd.service           1 -    2.0M -      -
/system.slice/cockpit.socket            - -    1.3M -      -
/system.slice/colord.service            3 -    3.5M -      -
/system.slice/crond.service             1 -    1.8M -      -
/system.slice/cups.service              1 -    3.1M -      -
/system.slice/dev-hugepages.mount       - -   244.0K -      -
/system.slice/dev-mapper-rhel\x2dswap.swap - -   912.0K -      -
/system.slice/dev-mqueue.mount          - -    48.0K -      -
/system.slice/example.service           2 -    2.0M -      -
/system.slice/firewalld.service         2 -   28.8M -      -
...
```

예제 출력에는 리소스 사용량(CPU, 메모리, 디스크 I/O 로드)에 따라 정렬된 **cgroup** 이 현재 실행 중인 것으로 표시됩니다. 이 목록은 기본적으로 1초마다 새로 고쳐집니다. 따라서 각 제어 그룹의 실제 리소스 사용량에 대한 동적 인사이트를 제공합니다.

추가 리소스

- [systemd-cgtop\(1\) manual page](#)

23장. SYSTEMD와 함께 CGROUP 버전 2를 사용하여 리소스 관리 구성

systemd의 핵심은 서비스 관리 및 제어입니다. **systemd**는 부팅 프로세스 중 올바른 순서로 올바른 서비스가 시작되도록 합니다. 서비스가 실행되면 기본 하드웨어 플랫폼을 최적으로 사용하기 위해 원활하게 실행해야 합니다. 따라서 **systemd**는 리소스 관리 정책을 정의하고 다양한 옵션을 튜닝하는 기능도 제공하므로 서비스의 성능을 향상시킬 수 있습니다.

23.1. 사전 요구 사항

- [Linux cgroup](#) 하위 시스템에 대한 기본 지식.

23.2. 리소스 배포 모델 소개

리소스 관리를 위해 **systemd**는 **cgroup v2** 인터페이스를 사용합니다.

RHEL 8에서는 기본적으로 **cgroup v1**을 사용합니다. 따라서 **systemd**가 리소스 관리에 **cgroup v2** 인터페이스를 사용할 수 있도록 **cgroup v2**를 활성화해야 합니다. **cgroup v2**를 활성화하는 방법에 대한 자세한 내용은 [cgroups-v2를 사용하여 애플리케이션에 CPU 제한 설정](#)을 참조하십시오.

시스템 리소스의 배포를 수정하려면 다음 리소스 배포 모델 중 하나 이상을 적용할 수 있습니다.

가중치

리소스는 모든 하위 그룹의 가중치를 추가하고 각 하위 그룹에 합계에 대한 비율과 일치하는 비율을 제공하여 배포됩니다.

예를 들어 각각 값이 100인 **cgroup** 10개가 있는 경우 합계는 1000이고 각 **cgroup**은 리소스의 1/10을 받습니다.

가중치는 일반적으로 상태 비저장 리소스를 배포하는 데 사용됩니다. **CPUWeight=** 옵션은 이 리소스 배포 모델을 구현한 것입니다.

제한

cgroup은 구성된 리소스 양까지 사용할 수 있지만 리소스를 과다 할당할 수도 있습니다. 따라서 하위 그룹 제한 합계는 상위 **cgroup**의 제한을 초과할 수 있습니다.

MemoryMax= 옵션은 이 리소스 배포 모델을 구현한 것입니다.

보호

cgroup에 대해 보호되는 리소스 양을 설정할 수 있습니다. 리소스 사용량이 보호 경계보다 낮은 경우 커널은 동일한 리소스에 대해 경쟁하는 다른 **cgroup**에 대해 이 **cgroup**을 위반하지 않도록 합니다. 오버 커밋도 허용됩니다.

MemoryLow= 옵션은 이 리소스 배포 모델을 구현한 것입니다.

할당

제한된 리소스의 절대적인 배타적 할당. 오버 커밋은 허용되지 않습니다. **Linux**에서 이 리소스 유형의 예로 실시간 예산이 있습니다.

추가 리소스

- **systemd**로 CPU 관리
- **systemd**를 사용하여 메모리 리소스 할당
- **systemd**를 사용하여 I/O 대역폭 구성

23.3. SYSTEMD를 사용하여 CPU 리소스 할당

systemd에서 관리하는 시스템에서 각 시스템 서비스는 **cgroup**에서 시작됩니다. **CPU cgroup** 컨트롤러에 대한 지원을 활성화하면 시스템은 프로세스별 배포 대신 **CPU** 리소스의 서비스 인식 배포관을 사용합니다. 서비스 인식 배포에서는 각 서비스는 서비스를 구성하는 프로세스 수에 관계없이 시스템에서 실행 중인 다른 모든 서비스에 비해 약 동일한 양의 **CPU** 시간을 받습니다.

특정 서비스에 더 많은 **CPU** 리소스가 필요한 경우 서비스의 **CPU** 시간 할당 정책을 변경하여 부여할 수 있습니다.

절차

systemd를 사용할 때 **CPU** 시간 할당 정책 옵션을 설정하려면 다음을 수행합니다.

1.

선택한 서비스에서 할당된 **CPU** 시간 할당 정책 옵션 값을 확인합니다.

```
$ systemctl show --property <CPU time allocation policy option> <service name>
```

2.

CPU 시간 할당 정책 옵션의 필수 값을 루트로 설정합니다.

```
# systemctl set-property <service name> <CPU time allocation policy option>=<value>
```



참고

cgroup 속성은 설정된 직후 적용됩니다. 따라서 서비스를 다시 시작할 필요가 없습니다.

cgroup 속성은 설정된 직후 적용됩니다. 따라서 서비스를 다시 시작할 필요가 없습니다.

검증 단계

•

서비스에 필요한 **CPU** 시간 할당 정책 옵션 값을 변경했는지 확인하려면 다음 명령을 실행합니다.

```
$ systemctl show --property <CPU time allocation policy option> <service name>
```

추가 리소스

•

[systemd의 CPU 시간 할당 정책 옵션](#)

•

[리소스 배포 모델 소개](#)

23.4. SYSTEMD의 CPU 시간 할당 정책 옵션

가장 자주 사용되는 **CPU** 시간 할당 정책 옵션은 다음과 같습니다.

CPUWeight=

다른 모든 서비스에 대해 특정 서비스에 더 높은 우선 순위를 할당합니다. 1 - 10,000 간격에서 값을 선택할 수 있습니다. 기본값은 100입니다.

예를 들어 **httpd.service** 에 다른 모든 서비스에 비해 두 배 많은 **CPU**를 제공하려면 값을 **CPUWeight=200** 으로 설정합니다.

CPUWeight= 는 운영 체제에 과부하가 있는 경우에만 적용됩니다.

CPUQuota=

절대 **CPU** 시간 할당량을 서비스에 할당합니다. 이 옵션의 값은 서비스가 수신할 **CPU** 시간의 최대 백분율을 지정합니다(예: **CPUQuota= 30%**).

CPUQuota= 는 리소스 배포 모델 소개에 설명된 특정 리소스 배포 모델의 제한 값을 나타냅니다.

CPUQuota= 에 대한 자세한 내용은 **systemd.resource-control(5)** 도움말 페이지를 참조하십시오.

추가 리소스

- [리소스 배포 모델 소개](#)
- [systemd를 사용하여 CPU 리소스 할당](#)

23.5. SYSTEMD를 사용하여 메모리 리소스 할당

이 섹션에서는 메모리 구성 옵션(**MemoryMin**, **MemoryLow**, **Memory High**, **Memory Max**, **Memory SwapMax**)을 사용하여 **systemd**를 사용하여 메모리 리소스를 할당하는 방법을 설명합니다.

절차

systemd를 사용할 때 메모리 할당 구성 옵션을 설정하려면 다음을 수행합니다.

1. 선택한 서비스에서 메모리 할당 구성 옵션의 할당된 값을 확인합니다.

```
$ systemctl show --property <memory allocation configuration option> <service name>
```

2.

메모리 할당 구성 옵션의 필요한 값을 루트로 설정합니다.

```
# systemctl set-property <service name> <memory allocation configuration option>=<value>
```



참고

cgroup 속성은 설정된 직후 적용됩니다. 따라서 서비스를 다시 시작할 필요가 없습니다.

검증 단계

•

서비스에 필요한 메모리 할당 구성 옵션 값을 변경했는지 확인하려면 다음 명령을 실행합니다.

```
$ systemctl show --property <memory allocation configuration option> <service name>
```

추가 리소스

•

[systemd의 메모리 할당 구성 옵션](#)

•

[리소스 배포 모델 소개](#)

23.6. SYSTEMD의 메모리 할당 구성 옵션

systemd를 사용하여 시스템 메모리 할당을 구성할 때 다음 옵션을 사용할 수 있습니다.

MemoryMin

하드 메모리 보호. 메모리 사용량이 제한보다 작으면 **cgroup** 메모리가 회수되지 않습니다.

MemoryLow

소프트 메모리 보호. 메모리 사용량이 제한 미만인 경우, 메모리가 보호되지 않은 **cgroup**에서 회수되지 않는 경우에만 **cgroup** 메모리를 회수할 수 있습니다.

MemoryHigh

메모리 제한. 메모리 사용량이 제한을 초과하면 **cgroup**의 프로세스가 제한되어 높은 회수 부담이 발생합니다.

MemoryMax

메모리 사용량에 대한 절대 제한입니다. 킬로(K), 메가(M), 기가(G), 테라(T) 접미사를 사용할 수 있습니다(예: **MemoryMax=1G**).

MemorySwapMax

스왑 사용 하드 제한.



참고

메모리 제한을 소진하면 **OOM**(메모리 부족) 종료자가 실행 중인 서비스를 중지합니다. 이 문제를 방지하려면 **OOMScoreAdjust**= 값을 줄여 메모리 허용 오차를 늘립니다.

추가 리소스

- [systemd를 사용하여 메모리 리소스 할당](#)
- [리소스 배포 모델 소개](#)

23.7. SYSTEMD를 사용하여 I/O 대역폭 구성

RHEL 8에서 특정 서비스의 성능을 개선하기 위해 **systemd**를 사용하여 **I/O** 대역폭 리소스를 해당 서비스에 할당할 수 있습니다.

이를 위해 다음 **I/O** 구성 옵션을 사용할 수 있습니다.

- **IOWeight**
- **IODeviceWeight**
- **IOReadBandwidthMax**
- **IOWriteBandwidthMax**

- ***IOReadIOPSMax***
- ***IOWriteIOPSMax***

절차

systemd를 사용하여 I/O 대역폭 구성 옵션을 설정하려면 다음을 수행합니다.

1. 선택한 서비스에서 할당된 I/O 대역폭 구성 옵션 값을 확인합니다.

```
$ systemctl show --property <I/O bandwidth configuration option> <service name>
```

2. I/O 대역폭 구성 옵션의 필수 값을 루트로 설정합니다.

```
# systemctl set-property <service name> <I/O bandwidth configuration option>=<value>
```

cgroup 속성은 설정된 직후 적용됩니다. 따라서 서비스를 다시 시작할 필요가 없습니다.

검증 단계

- 서비스에 필요한 I/O 대역폭 구성 옵션 값을 변경했는지 확인하려면 다음 명령을 실행합니다.

```
$ systemctl show --property <I/O bandwidth configuration option> <service name>
```

추가 리소스

- [systemd에 대한 I/O 대역폭 구성 옵션](#)
- [리소스 배포 모델 소개](#)

23.8. SYSTEMD에 대한 I/O 대역폭 구성 옵션

systemd를 사용하여 블록 계층 I/O 정책을 관리하려면 다음 구성 옵션을 사용합니다.

IOWeight

기본 I/O 가중치를 설정합니다. **weight** 값은 서비스가 다른 서비스와 관련하여 수신되는 실제 I/O 대역폭의 양을 계산하기 위한 기준으로 사용됩니다.

IODeviceWeight

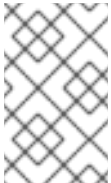
특정 블록 장치에 대한 I/O 가중치를 설정합니다.

예를 들어 **IODeviceWeight=/dev/disk/by-id/dm-name-rhel-root 200** 입니다.

IOReadBandwidthMax, IOWriteBandwidthMax

장치 또는 마운트 지점당 절대 대역폭을 설정합니다.

예를 들어 **IOWriteBandwidth=/var/log 5M** 입니다.



참고

systemd는 **file-system-to-device** 변환을 자동으로 처리합니다.

IOReadIOPSMax, IOWriteIOPSMax

이전 옵션과 유사한 옵션: **IOPS**(초당 입/출력 작업)에서 절대 대역폭을 설정합니다.



참고

가중치 기반 옵션은 블록 장치가 **CFQ I/O** 스케줄러를 사용하는 경우에만 지원됩니다. 장치가 다중 대기열 블록 I/O 대기열 메커니즘을 사용하는 경우 옵션이 지원되지 않습니다.

추가 리소스

- [systemd를 사용하여 I/O 대역폭 구성](#)
- [리소스 배포 모델 소개](#)

23.9. SYSTEMD를 사용하여 CPUSET 컨트롤러 구성

systemd 리소스 관리 **API**를 사용하면 사용자가 서비스에서 사용할 수 있는 **CPU** 및 **NUMA** 노드 집합에 대한 제한을 구성할 수 있습니다. 이 제한은 프로세스에서 사용하는 시스템 리소스에 대한 액세스를 제한합니다. 요청된 구성은 **cpuset.cpus** 및 **cpuset.mems** 로 작성됩니다. 그러나 상위 **cgroup** 은 **cpus** 또는 **mems** 를 제한하므로 요청된 구성을 사용할 수 없습니다. 현재 구성에 액세스하려면 **cpuset.cpus.effective** 및 **cpuset.mems.effective** 파일을 사용자에게 내보냅니다.

절차

- **AllowedCPUs** 를 설정하려면 다음을 수행합니다.

```
# systemctl set-property service_name.service AllowedCPUs=value
```

예를 들면 다음과 같습니다.

```
systemctl set-property service_name.service AllowedCPUs=0-5
```

- **AllowedMemoryNodes** 를 설정하려면 다음을 수행합니다.

```
# systemctl set-property service_name.service AllowedMemoryNodes=value
```

예를 들면 다음과 같습니다.

```
# systemctl set-property service_name.service AllowedMemoryNodes=0
```

24장. SYSTEMD를 사용하여 CPU 유사성 및 NUMA 정책 구성

CPU 관리, 메모리 관리 및 I/O 대역폭 옵션은 사용 가능한 리소스를 파티셔닝하는 데 사용됩니다.

24.1. SYSTEMD를 사용하여 CPU 선호도 구성

CPU 선호도 설정을 사용하면 특정 프로세스의 액세스를 일부 CPU로 제한할 수 있습니다. 효과적으로 CPU 스케줄러는 프로세스의 선호도 마스크에 없는 CPU에서 실행되도록 프로세스를 예약하지 않습니다.

기본 CPU 선호도 마스크는 **systemd**에서 관리하는 모든 서비스에 적용됩니다.

특정 **systemd** 서비스에 대한 CPU 선호도 마스크를 구성하기 위해 **systemd**는 **CPUAffinity=**를 유닛 파일 옵션으로, **/etc/systemd/system.conf** 파일에서 관리자 구성 옵션으로 제공합니다.

CPUAffinity= 유닛 파일 옵션은 병합되고 선호도 마스크로 사용되는 CPU 또는 CPU 범위 목록을 설정합니다. **/etc/systemd/system.conf** 파일의 **CPUAffinity** 옵션은 PID(프로세스 식별 번호) 1 및 PID1에서 분기된 모든 프로세스에 대한 선호도 마스크를 정의합니다. 그런 다음 서비스별로 **CPUAffinity**를 덮어쓸 수 있습니다.



참고

특정 **systemd** 서비스에 대한 CPU 선호도 마스크를 구성한 후 시스템을 다시 시작하여 변경 사항을 적용해야 합니다.

절차

CPU Affinity 유닛 파일 옵션을 사용하여 **particular systemd** 서비스의 CPU 선호도 마스크를 설정하려면 다음을 수행합니다.

1. 선택한 서비스에서 **CPUAffinity** 유닛 파일 옵션의 값을 확인합니다.

```
$ systemctl show --property <CPU affinity configuration option> <service name>
```

2. 루트로 선호도 마스크로 사용되는 CPU 범위에 필요한 **CPUAffinity** 장치 파일 옵션 값을 설정합니다.

```
# systemctl set-property <service name> CPUAffinity=<value>
```

3. 서비스를 다시 시작하여 변경 사항을 적용합니다.

```
# systemctl restart <service name>
```

manager 구성 옵션을 사용하여 특정 **systemd** 서비스에 대한 **CPU** 선호도 마스크를 설정하려면 다음을 수행합니다.

1. **/etc/systemd/system.conf** 파일을 편집합니다.

```
# vi /etc/systemd/system.conf
```

2. **CPUAffinity=** 옵션을 검색하고 **CPU** 번호 설정
3. 편집한 파일을 저장하고 서버를 다시 시작하여 변경 사항을 적용합니다.

24.2. SYSTEMD를 사용하여 NUMA 정책 구성

NUMA(Non-Uniform Memory Access)는 메모리 액세스 시간이 프로세서와 관련된 실제 메모리 위치에 따라 달라지는 컴퓨터 메모리 하위 시스템 설계입니다.

CPU에 가까운 메모리는 다른 **CPU**(정규 메모리)에 대해 로컬 메모리보다 대기 시간(로컬 메모리)이 낮거나 **CPU** 세트 간에 공유됩니다.

Linux 커널의 관점에서 **NUMA** 정책은 커널이 프로세스에 물리적 메모리 페이지를 할당하는 위치(예: **NUMA** 노드의 위치)를 관리합니다.

systemd 는 유닛 파일 옵션 **NUMAPolicy** 및 **NUMAMask** 를 제공하여 서비스의 메모리 할당 정책을 제어합니다.

절차

NUMA Policy 유닛 파일 옵션을 통해 **NUMA** 메모리 정책을 설정하려면 다음을 수행합니다.

1.

선택한 서비스에서 **NUMAPolicy** 장치 파일 옵션의 값을 확인합니다.

```
$ systemctl show --property <NUMA policy configuration option> <service name>
```

2.

루트로 **NUMAPolicy** 장치 파일 옵션의 필요한 정책 유형을 설정합니다.

```
# systemctl set-property <service name> NUMAPolicy=<value>
```

3.

서비스를 다시 시작하여 변경 사항을 적용합니다.

```
# systemctl restart <service name>
```

관리자 구성 옵션을 통해 글로벌 **NUMAPolicy** 설정을 설정하려면 다음을 수행합니다.

1.

/etc/systemd/system.conf 파일에서 **NUMAPolicy** 옵션을 검색합니다.

2.

정책 유형을 편집하고 파일을 저장합니다.

3.

systemd 구성을 다시 로드합니다.

```
# systemd daemon-reload
```

4.

서버를 재부팅합니다.

중요

엄격한 **NUMA** 정책을 구성할 때(예: 바인딩) **CPUAffinity=** 장치 파일 옵션을 적절하게 설정해야 합니다.

추가 리소스



[systemd의 NUMA 정책 구성 옵션](#)

- **`systemd.resource-control(5)`, `systemd.exec(5)`, `set_mempolicy(2)` manual pages.**

24.3. SYSTEMD의 NUMA 정책 구성 옵션

systemd는 **NUMA** 정책을 구성하는 다음 옵션을 제공합니다.

NUMAPolicy

실행된 프로세스의 **NUMA** 메모리 정책을 제어합니다. 다음과 같은 정책 유형이 가능합니다.

- **default**
- 선호하는
- **bind**
- 인터리브
- 로컬

NUMAMask

선택한 **NUMA** 정책과 연결된 **NUMA** 노드 목록을 제어합니다.

다음 정책에는 **NUMAMask** 옵션을 지정할 필요가 없습니다.

- **default**
- 로컬

기본 정책의 경우 목록은 단일 **NUMA** 노드만 지정합니다.

추가 리소스

- [systemd.resource-control\(5\), systemd.exec\(5\) 및 set_mempolicy\(2\) 매뉴얼 페이지](#)
- [systemd를 사용하여 NUMA 구성](#)

24.3.1. BPF Compiler Collection을 사용하여 시스템 성능 분석

시스템 관리자는 **BCC(BPF Compiler Collection)** 라이브러리를 사용하여 **Linux** 운영 체제의 성능 분석 및 정보 수집을 위한 툴을 생성할 수 있습니다. 이러한 툴은 다른 인터페이스를 통해 확보하기 어려울 수 있습니다.

24.3.1.1. BCC 소개

BCC(BPF Compiler Collection)는 **eBPF(extended Berkeley Packet Filter)** 프로그램을 쉽게 생성할 수 있는 라이브러리입니다. **eBPF** 프로그램의 주요 유틸리티는 오버헤드나 보안 문제가 발생하지 않고 **OS** 성능 및 네트워크 성능을 분석하는 것입니다.

BCC는 사용자가 **eBPF**의 심층적인 기술 세부 사항을 알고 있어야 하는 필요성을 없애고, 미리 생성된 **eBPF** 프로그램을 사용하는 **bcc-tools** 패키지와 같이 많은 즉시 사용 가능한 시작점을 제공합니다.



참고

eBPF 프로그램은 디스크 I/O, TCP 연결 및 프로세스 생성과 같은 이벤트에서 트리거됩니다. 커널의 안전한 가상 시스템에서 실행되므로 커널이 충돌하거나 반복하거나 응답하지 않게 되는 프로그램이 발생할 가능성이 낮습니다.

24.3.1.2. bcc-tools 패키지 설치

이 섹션에서는 **BCC(BPF Compiler Collection)** 라이브러리도 종속성으로 설치하는 **bcc-tools** 패키지를 설치하는 방법에 대해 설명합니다.

절차

1. **bcc-tools** 를 설치합니다.

```
# yum install bcc-tools
```

BCC 툴은 `/usr/share/bcc/tools/` 디렉토리에 설치됩니다.

2.

선택적으로 툴을 검사합니다.

```
# ll /usr/share/bcc/tools/
...
-rwxr-xr-x. 1 root root 4198 Dec 14 17:53 dcsnoop
-rwxr-xr-x. 1 root root 3931 Dec 14 17:53 dcstat
-rwxr-xr-x. 1 root root 20040 Dec 14 17:53 deadlock_detector
-rw-r--r--. 1 root root 7105 Dec 14 17:53 deadlock_detector.c
drwxr-xr-x. 3 root root 8192 Mar 11 10:28 doc
-rwxr-xr-x. 1 root root 7588 Dec 14 17:53 execsnoop
-rwxr-xr-x. 1 root root 6373 Dec 14 17:53 ext4dist
-rwxr-xr-x. 1 root root 10401 Dec 14 17:53 ext4slower
...
```

위의 목록의 **doc** 디렉토리에는 각 툴에 대한 문서가 포함되어 있습니다.

24.3.1.3. 선택한 **bcc-tools**를 성능 분석에 사용

이 섹션에서는 **BCC(BPF Compiler Collection)** 라이브러리에서 미리 생성된 특정 프로그램을 사용하여 이벤트별로 효율적이고 안전하게 시스템 성능을 분석하는 방법에 대해 설명합니다. **BCC** 라이브러리에서 미리 생성된 프로그램 집합은 추가 프로그램 생성의 예제 역할을 할 수 있습니다.

사전 요구 사항

- [설치된 **bcc-tools** 패키지](#)
- 루트 권한

execsnoop를 사용하여 시스템 프로세스 검사

1.

하나의 터미널에서 **execsnoop** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/execsnoop
```

2.

예를 들어 다른 터미널 실행에서 다음을 실행합니다.

```
$ ls /usr/share/bcc/tools/doc/
```

위의 경우 **ls** 명령의 수명이 짧은 프로세스가 생성됩니다.

3.

execsnoop 를 실행하는 터미널에는 다음과 유사한 출력이 표시됩니다.

```
PCOMM PID  PPID  RET ARGS
ls 8382 8287  0 /usr/bin/ls --color=auto /usr/share/bcc/tools/doc/
...
```

execsnoop 프로그램은 시스템 리소스를 사용하는 각 새 프로세스의 출력 행을 인쇄합니다. **ls** 와 같이 매우 빨리 실행되는 프로그램의 프로세스도 감지하며 대부분의 모니터링 틀은 등록되지 않았습니다.

execsnoop 출력에는 다음 필드가 표시됩니다.

- **PCOMM** - 부모 프로세스 이름입니다(**ls**)
- **PID** - 프로세스 ID입니다. (**8382**)
- **PPID** - 상위 프로세스 ID입니다. (**8287**)
- **RET** - 프로그램 코드를 새 프로세스로 로드하는 **exec()** 시스템 호출(**0**)의 반환 값입니다.
- **ARGS** - 인수가 포함된 시작된 프로그램의 위치입니다.

execsnoop 에 대한 자세한 내용, 예제 및 옵션을 보려면 `/usr/share/bcc/tools/doc/execsnoop_example.txt` 파일을 참조하십시오.

exec() 에 대한 자세한 내용은 **exec(3)** 도움말 페이지를 참조하십시오.

opensnoop를 사용하여 명령이 여는 파일을 추적합니다.

1.

하나의 터미널에서 **opensnoop** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/opensnoop -n uname
```

위의 출력은 **uname** 명령의 프로세스에 의해서만 열려 있는 파일의 출력을 인쇄합니다.

2.

다른 터미널에서 다음을 실행합니다.

```
$ uname
```

위의 명령은 다음 단계에서 캡처된 특정 파일을 엽니다.

3.

opensnoop를 실행하는 터미널에는 다음과 유사한 출력이 표시됩니다.

```
PID  COMM  FD ERR PATH
8596  uname 3  0  /etc/ld.so.cache
8596  uname 3  0  /lib64/libc.so.6
8596  uname 3  0  /usr/lib/locale/locale-archive
...
```

opensnoop 프로그램은 전체 시스템에서 **open()** 시스템 호출을 감시하고, 그 과정에서 **uname** 이 열려고 시도한 각 파일의 출력을 출력합니다.

opensnoop 출력에는 다음 필드가 표시됩니다.

- **PID** - 프로세스 ID입니다. (8596)
- **COMM** - 프로세스 이름입니다(동일하지 않음)
- **fd** - 파일 설명자 - **open()**가 열려 있는 파일을 참조하도록 반환되는 값입니다. (3)
- **ERR** - 모든 오류.
- **PATH** - 열기를 시도한 파일의 위치입니다.

명령이 존재하지 않는 파일을 읽으려고 하면 **FD** 열은 -1 을 반환하고 **ERR** 열은 관련

오류에 해당하는 값을 출력합니다. 그 결과 **opensnoop** 는 제대로 작동하지 않는 애플리케이션을 식별하는 데 도움이 될 수 있습니다.

opensnoop 에 대한 자세한 내용, 예제 및 옵션을 보려면 `/usr/share/bcc/tools/doc/opensnoop_example.txt` 파일을 참조하십시오.

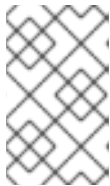
open() 에 대한 자세한 내용은 **open(2)** 매뉴얼 페이지를 참조하십시오.

biotop을 사용하여 디스크의 I/O 작업 검사

1. 하나의 터미널에서 **biotop** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/biotop 30
```

명령을 사용하면 디스크에서 I/O 작업을 수행하는 최상위 프로세스를 모니터링할 수 있습니다. 인수를 사용하면 명령이 30초 요약을 생성할 수 있습니다.



참고

인수를 제공하지 않으면 기본적으로 출력 화면이 1초마다 새로 고쳐집니다.

2. 다른 터미널의 예를 들면 다음과 같습니다.

```
# dd if=/dev/vda of=/dev/zero
```

위의 명령은 로컬 하드 디스크 장치에서 콘텐츠를 읽고 `/dev/zero` 파일에 출력을 씁니다. 이 단계에서는 **biotop** 을 설명하는 특정 I/O 트래픽을 생성합니다.

3. **biotop** 을 실행하는 터미널은 다음과 유사한 출력을 보여줍니다.

```
PID  COMM      D MAJ MIN DISK   I/O Kbytes  AVGms
9568 dd        R 252 0  vda    16294 14440636.0 3.69
48   kswapd0   W 252 0  vda    1763 120696.0 1.65
7571 gnome-shell R 252 0  vda     834 83612.0 0.33
1891 gnome-shell R 252 0  vda    1379 19792.0 0.15
7515 Xorg       R 252 0  vda     280 9940.0 0.28
7579 llvmpipe-1 R 252 0  vda     228 6928.0 0.19
9515 gnome-control-c R 252 0  vda      62 6444.0 0.43
```

```

8112 gnome-terminal- R 252 0 vda      67 2572.0  1.54
7807 gnome-software R 252 0 vda      31 2336.0  0.73
9578 awk             R 252 0 vda      17 2228.0  0.66
7578 llvmpipe-0      R 252 0 vda      156 2204.0  0.07
9581 pgrep           R 252 0 vda      58 1748.0  0.42
7531 InputThread     R 252 0 vda      30 1200.0  0.48
7504 gdbus           R 252 0 vda      3 1164.0  0.30
1983 llvmpipe-1      R 252 0 vda      39 724.0   0.08
1982 llvmpipe-0      R 252 0 vda      36 652.0   0.06
...

```

biotop 출력에는 다음 필드가 표시됩니다.

- **PID** - 프로세스 ID입니다. (9568)
- **COMM** - 프로세스 이름입니다. (dd)
- **DISK** - 읽기 작업을 수행하는 디스크입니다 (vda)
- **I/O** - 수행된 읽기 작업 수입니다. (16294)
- **kbytes** - 읽기 작업에서 얻은 K바이트 양입니다. (14,440,636)
- **mms** - 평균 읽기 작업 I/O 시간입니다. (3.69)

biotop에 대한 자세한 내용, 예제 및 옵션을 보려면 `/usr/share/bcc/tools/doc/biotop_example.txt` 파일을 참조하십시오.

dd에 대한 자세한 내용은 **dd(1)** 도움말 페이지를 참조하십시오.

여기저기 않게 느린 파일 시스템 작업 노출을 위해 **xfsslower** 사용

1. 한 터미널에서 **xfsslower** 프로그램을 실행합니다.

```
# /usr/share/bcc/tools/xfsslower 1
```

위의 명령은 **XFS** 파일 시스템이 읽기, 쓰기, 열기 또는 동기화(**fsync**) 작업을 수행하는 데 소비하는 시간을 측정합니다. **1** 인수를 사용하면 프로그램이 **1ms**보다 느린 작업만 표시합니다.



참고

인수를 지정하지 않으면 기본적으로 **xfsslower**는 **10ms**보다 느리게 작업을 표시합니다.

2.

다른 터미널에서 다음을 실행합니다.

```
$ vim text
```

위의 명령은 **vim** 편집기에서 텍스트 파일을 생성하여 **XFS** 파일 시스템과의 특정 상호 작용을 시작합니다.

3.

xfsslower를 실행하는 터미널에는 이전 단계에서 파일을 저장할 때 유사한 내용이 표시됩니다.

```
TIME  COMM      PID  T BYTES  OFF_KB  LAT(ms)  FILENAME
13:07:14 b'bash'   4754  R 256    0        7.11 b'vim'
13:07:14 b'vim'   4754  R 832    0        4.03 b'libgpm.so.2.1.0'
13:07:14 b'vim'   4754  R 32     20       1.04 b'libgpm.so.2.1.0'
13:07:14 b'vim'   4754  R 1982   0        2.30 b'vimrc'
13:07:14 b'vim'   4754  R 1393   0        2.52 b'getscriptPlugin.vim'
13:07:45 b'vim'   4754  S 0      0        6.71 b'text'
13:07:45 b'pool'  2588  R 16     0        5.58 b'text'
...
```

위의 각 행은 파일 시스템에서 특정 임계값보다 더 많은 시간이 걸리는 작업을 나타냅니다. **xfsslower**는 예기치 않게 느린 작업을 수행할 수 있는 가능한 파일 시스템 문제를 노출하는 데 효과적입니다.

xfsslower 출력에는 다음 필드가 표시됩니다.

- **COMM** - 프로세스 이름입니다. (**b'bash'**)
- **t** - 작업 유형입니다. (**R**)

- **읽기**
- **w rite**
- **s ync**
- **OFF_KB - KB 단위의 파일 오프셋. (0)**
- **FILENAME - 읽기, 쓰기 또는 동기화되는 파일입니다.**

xfsslower에 대한 자세한 내용, 예제 및 옵션을 보려면 `/usr/share/bcc/tools/doc/xfsslower_example.txt` 파일을 참조하십시오.

fsync에 대한 자세한 내용은 **fsync(2)** 매뉴얼 페이지를 참조하십시오.

24.3.2. 커널 무결성 하위 시스템을 사용하여 보안 강화

커널 무결성 하위 시스템의 구성 요소를 활용하여 시스템 보호를 강화할 수 있습니다. 다음 섹션에서는 관련 구성 요소를 소개하고 해당 구성에 대한 지침을 제공합니다.



참고

암호화 서명과 관련된 기능은 **Red Hat** 내부용으로만 사용할 수 있습니다. 이는 **Red Hat** 서명 키의 인증서만 커널 키링에 존재하기 때문입니다. 사용자는 모든 해시 기능을 사용할 수 있지만 완전한 변조 메커니즘은 성공적으로 완료되지 않습니다.

24.3.2.1. 커널 무결성 하위 시스템

무결성 하위 시스템은 전체 시스템 데이터 무결성을 유지 관리하는 데 사용되는 커널의 일부입니다. 이 하위 시스템은 구축된 시점과 특정 시스템의 상태를 동일하게 유지하는 데 도움이 되므로 특정 시스템 파일에서 원하지 않는 수정을 방지합니다.

커널 무결성 하위 시스템은 다음 두 가지 주요 구성 요소로 구성됩니다.

무결성 측정 아키텍처(IMA)

- 암호화 키로 해시 또는 서명을 사용하여 실행하거나 열 때마다 파일 내용을 측정합니다. 키는 커널 키링 하위 시스템에 저장됩니다.
- 측정된 값을 커널의 메모리 공간에 배치하여 시스템 사용자의 수정을 방지합니다.
- 로컬 및 원격 당사자가 측정된 값을 확인할 수 있도록 합니다.
- 커널 메모리 내의 측정 목록에 이전에 저장된 값에 대해 파일의 현재 콘텐츠의 로컬 유효성 검사를 제공합니다. 이 확장 기능은 현재 작업이 있고 이전 작업이 일치하지 않는 경우 특정 파일에서 작업을 수행할 것을 금지합니다.

EVM(Extended Verification Module)

- IMA 측정 및 SELinux 속성과 같은 시스템 보안과 관련된 확장된 파일(xattr라고도 함)을 보호합니다. 구성 요소는 해당 값을 암호화 방식으로 해시하거나 암호화 키를 사용하여 서명합니다. 키는 커널 키링 하위 시스템에 저장됩니다.

커널 무결성 하위 시스템은 신뢰할 수 있는 플랫폼 모듈(TPM)을 활용하여 시스템 보안을 더욱 강화할 수 있습니다. TPM은 중요한 암호화 기능에 대한 MTC(신뢰할 수 있는 컴퓨팅 그룹)의 사양입니다. TPMS는 일반적으로 플랫폼의 마더보드에 연결된 전용 하드웨어로 구축되며 하드웨어 칩의 보호되고 변조되는 영역에서 암호화 기능을 제공하여 소프트웨어 기반 공격을 방지합니다. TPM 기능 중 일부는 다음과 같습니다.

- 임의 번호 생성기
- 암호화 키용 생성 및 보안 스토리지
- 해시 생성기
- 원격 인증

추가 리소스

- [무결성 측정 아키텍처\(IMA\)](#)
- [신뢰할 수 있는 컴퓨팅 그룹 리소스](#)
- [보안 강화](#)
- [SELinux\(Security-Enhanced Linux\)의 기본 및 고급 구성](#)

24.3.2.2. 무결성 측정 아키텍처

무결성 측정 아키텍처(IMA)는 커널 무결성 하위 시스템의 구성 요소입니다. IMA는 로컬 파일의 내용을 유지 관리하는 것을 목표로 합니다. 특히 IMA는 파일 해시에 액세스하기 전에 파일 해시를 측정, 저장 및 감사하므로 신뢰할 수 없는 데이터의 읽기 및 실행을 방지합니다. 따라서 디지털 서명과 함께 IMA를 사용하면 시스템의 보안이 향상되고 데이터 위조를 측정할 수 없습니다. 이 시나리오에서는 공격자가 원래 서명 키를 소유하고 있지 않다고 가정합니다.

추가 리소스

- [무결성 측정 아키텍처\(IMA\)](#)
- [보안 강화](#)
- [SELinux\(Security-Enhanced Linux\)의 기본 및 고급 구성](#)

24.3.2.3. 확장 검증 모듈

EVM(Extended Verification Module)은 파일 확장 속성(xattr)의 변경 사항을 모니터링하는 커널 무결성 하위 시스템의 구성 요소입니다. 많은 보안 지향 기술은 콘텐츠 해시 및 서명과 같은 중요한 파일 정보를 파일의 확장된 속성에 저장합니다. EVM은 한 단계 더 나아가 해시를 진행하거나 확장된 속성에 서명합니다. 확장 특성을 사용할 때마다 결과 데이터의 유효성을 검사합니다. 예를 들어, IMA가 파일을 평가할 때.

추가 리소스

- [무결성 측정 아키텍처\(IMA\)](#)

보안 강화

SELinux(Security-Enhanced Linux)의 기본 및 고급 구성

24.3.2.4. 신뢰할 수 있는 암호화된 키

다음 섹션에서는 시스템 보안을 강화하는 데 중요한 일환으로 신뢰할 수 있고 암호화된 키를 소개합니다.

신뢰할 수 있는 암호화된 키는 커널 인증 키를 사용하는 커널에서 생성하는 가변 길이 대칭 키입니다. 이러한 유형의 키 유형이 암호화되지 않은 형식의 사용자 공간에 표시되지 않는다는 것은 해당 키의 무결성을 확인할 수 있음을 의미하며, 이는 예를 들어 실행 중인 시스템의 무결성을 확인하고 확인하는 데 **EVM(Extended verification)** 모듈을 통해 사용할 수 있음을 의미합니다. 사용자 수준 프로그램은 암호화된 **Blob** 형태의 키에만 액세스할 수 있습니다.

신뢰할 수 있는 키에는 하드웨어 구성 요소가 필요합니다. 이 칩은 키를 만들고 암호화하는 데 사용됩니다. **TPM**은 **SRK**(스토리지 루트 키)라는 **2048비트 RSA** 키를 사용하여 키를 봉인합니다.

참고

TPM 1.2 사양을 사용하려면 시스템 펌웨어의 설정을 통해 또는 **tpm-tools** 유틸리티 패키지에서 **tpm_setactive** 명령을 사용하여 활성화하고 활성화합니다. 또한 **TrouSers** 소프트웨어 스택을 설치해야 하며 **TPM**(전용 하드웨어)과 통신하기 위해 **tcscd** 데몬을 실행해야 합니다. **tcscd** 데몬은 **TrouSers** 제품군의 일부입니다. 이 제품군은 구성요소 패키지를 통해 사용할 수 있습니다. 최신 버전과 호환되지 않는 **TPM 2.0**은 **tpm2-tools** 또는 **ibm-tss** 유틸리티가 전용 하드웨어에 대한 액세스를 제공하는 다른 소프트웨어 스택을 사용합니다.

또한 사용자는 신뢰할 수 있는 키를 특정 **TPM**의 **PCR**(플랫폼 구성 레지스터) 값 세트로 봉인할 수 있습니다. **PCR**에는 펌웨어, 부트 로더 및 운영 체제를 반영하는 무결성-관리 값 집합이 포함되어 있습니다. 즉, **PCR** 봉인된 키는 암호화된 동일한 시스템에서 **TPM**으로만 해독할 수 있습니다. 그러나 **PCR** 봉인된 신뢰할 수 있는 키가 로드되고(키링에 추가) 관련 **PCR** 값이 확인되면 새 (또는 향후) **PCR** 값으로 업데이트할 수 있으므로 새 커널(예:)을 부팅할 수 있습니다. 단일 키도 여러 개의 **Blob**으로 저장할 수 있으며 각각 다른 **PCR** 값이 있습니다.

암호화된 키는 **TPM**이 필요하지 않습니다. 이는 커널 **AES(Advanced Encryption Standard)**를 사용하여므로 신뢰할 수 있는 키보다 빠릅니다. 암호화된 키는 커널에서 생성된 임의의 숫자를 사용하여 생성되며 사용자 공간 **Blob**으로 내보낼 때 마스터 키로 암호화됩니다. 마스터 키는 신뢰할 수 있는 키 또는 사용자 키입니다. 마스터 키가 신뢰할 수 없는 경우 암호화된 키는 암호화에 사용되는 사용자 키만큼 안전합니다.

24.3.2.5. 신뢰할 수 있는 키 작업

다음 섹션에서는 시스템 보안을 개선하기 위해 **keyctl** 유틸리티를 사용하여 신뢰할 수 있는 키를 생성, 내보내기, 로드 또는 업데이트하는 방법을 설명합니다.

사전 요구 사항

- **64비트 ARM** 아키텍처 및 **IBM Z**의 경우 신뢰할 수 있는 커널 모듈을 로드해야 합니다. 커널 모듈을 로드하는 방법에 대한 자세한 내용은 커널 모듈 [관리를 참조하십시오](#).
- 신뢰할 수 있는 **TPM(Platform Module)**을 활성화하고 활성화해야 합니다. **TPM**에 대한 자세한 내용은 [커널 무결성 하위](#) 시스템과 [신뢰할 수 있고 암호화된 키](#)를 참조하십시오.

절차

1. **TPM**을 사용하여 신뢰할 수 있는 키를 생성하려면 다음을 실행합니다.

```
# keyctl add trusted <name> "new <key_length> [options]" <key_ring>
```

구문에 따라 다음과 같이 예제 명령을 구성합니다.

```
# keyctl add trusted kmk "new 32" @u
642500861
```

이 명령은 길이 **32바이트(256비트)**를 사용하여 **kmk** 라는 신뢰할 수 있는 키를 생성하고 사용자 인증 키(**@u**)에 배치합니다. 키의 길이는 **32~128바이트(256비트에서 1024비트)**일 수 있습니다.

2. 커널 인증 키의 현재 구조를 나열하려면 다음을 수행합니다.

```
# keyctl show
Session Keyring
-3 --alswrv 500 500 keyring: ses 97833714 --alswrv 500 -1 \ keyring: uid.1000
642500861 --alswrv 500 500 \ trusted: kmk
```

3. 키를 사용자 공간 **Blob**으로 내보내려면 다음을 실행합니다.

```
# keyctl pipe 642500861 > kmk.blob
```

명령은 **pipe** 하위 명령과 **kmk**의 일련 번호를 사용합니다.

4.

사용자 공간 **Blob**에서 신뢰할 수 있는 키를 로드하려면 **blob**을 인수로 **add** 하위 명령을 사용합니다.

```
# keyctl add trusted kmk "load `cat kmk.blob`" @u
268728824
```

5.

TPM으로 봉인된 신뢰할 수 있는 키를 기반으로 보안 암호화 키를 생성합니다.

```
# keyctl add encrypted <pass:quotes[name]> "new [format] <pass:quotes[key_type]>:
<pass:quotes[primary_key_name]> <pass:quotes[keylength]>"
<pass:quotes[key_ring]>
```

-

구문에 따라 이미 생성된 신뢰할 수 있는 키를 사용하여 암호화된 키를 생성합니다.

```
# keyctl add encrypted encr-key "new trusted:kmk 32" @u
159771175
```

명령은 이전 단계에서 생성된 **TPM** 봉인된 신뢰할 수 있는 키(**kmk**)를 암호화된 키를 생성하기 위한 기본 키로 사용합니다.

추가 리소스

-

keyctl(1) 매뉴얼 페이지

-

신뢰할 수 있는 암호화된 키

-

커널 키 보존 서비스

-

커널 무결성 하위 시스템

24.3.2.6. 암호화된 키 작업

다음 섹션에서는 **TPM**(신뢰할 수 있는 플랫폼 모듈)을 사용할 수 없는 시스템에서 시스템 보안을 개선하기 위해 암호화된 키를 관리하는 방법을 설명합니다.

사전 요구 사항

- 64비트 **ARM** 아키텍처 및 **IBM Z**의 경우 **encrypted-keys** 커널 모듈을 로드해야 합니다. 커널 모듈을 로드하는 방법에 대한 자세한 내용은 커널 모듈 [관리를 참조하십시오](#).

절차

- 임의의 숫자 시퀀스를 사용하여 사용자 키를 생성합니다.

```
# keyctl add user kmk-user "$(dd if=/dev/urandom bs=1 count=32 2>/dev/null)" @u
427069434
```

명령은 기본 키 역할을 하고 실제 암호화된 키를 봉인하는 데 사용되는 **kmk-user** 라는 사용자 키를 생성합니다.

- 이전 단계의 기본 키를 사용하여 암호화된 키를 생성합니다.

```
# keyctl add encrypted encr-key "new user:kmk-user 32" @u
1012412758
```

- 선택적으로 지정된 사용자 인증 키의 모든 키를 나열합니다.

```
# keyctl list @u
2 keys in keyring:
427069434: --alswrv 1000 1000 user: kmk-user
1012412758: --alswrv 1000 1000 encrypted: encr-key
```

중요

신뢰할 수 있는 기본 키로 봉인되지 않은 암호화된 키는 암호화하는 데 사용된 사용자 기본 키(임의의 숫자 키)만큼 안전합니다. 따라서 기본 사용자 키를 최대한 안전하게 로드하고 부팅 프로세스 중 조기에 로드해야 합니다.

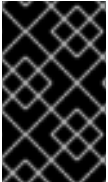
추가 리소스

- keyctl(1) 매뉴얼 페이지**

커널 키 보존 서비스

24.3.2.7. 무결성 측정 아키텍처 및 확장 검증 모듈 활성화

무결성 측정 아키텍처(IMA) 및 EVM(확장 검증 모듈)은 커널 무결성 하위 시스템에 속하고 다양한 방법으로 시스템 보안을 강화합니다. 다음 섹션에서는 운영 체제의 보안을 개선하기 위해 IMA 및 EVM을 활성화하고 구성하는 방법을 설명합니다.



중요

이 절차는 Red Hat에서 발행한 인증서와 키를 사용하는 경우에만 작동합니다.

사전 요구 사항

- **securityfs** 파일 시스템이 **/sys/kernel/security/** 디렉터리 및 **/sys/kernel/security/integrity/ima/** 디렉터리에 마운트되었는지 확인합니다.

```
# mount
```

```
...
securityfs on /sys/kernel/security type securityfs (rw,nosuid,nodev,noexec,relatime)
...
```

- **systemd** 서비스 관리자가 부팅 시 IMA 및 EVM을 지원하도록 이미 패치되었는지 확인합니다.

```
# dmesg | grep -i -e EVM -e IMA
```

```
[ 0.000000] Command line: BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-167.el8.x86_64 root=/dev/mapper/rhel-root ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet
[ 0.000000] kvm-clock: cpu 0, msr 23601001, primary cpu clock
[ 0.000000] Using crashkernel=auto, the size chosen is a best effort estimation.
[ 0.000000] Kernel command line: BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-167.el8.x86_64 root=/dev/mapper/rhel-root ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet
[ 0.911527] ima: No TPM chip found, activating TPM-bypass!
[ 0.911538] ima: Allocated hash algorithm: sha1
[ 0.911580] evm: Initialising EVM extended attributes:
[ 0.911581] evm: security.selinux
[ 0.911581] evm: security.ima
[ 0.911582] evm: security.capability
[ 0.911582] evm: HMAC attrs: 0x1
[ 1.715151] systemd[1]: systemd 239 running in system mode. (+PAM +AUDIT +SELINUX +IMA -APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP +GCRYPT +GNUTLS +ACL +XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN2 -IDN +PCRE2 default-hierarchy=legacy)
```

```
[ 3.824198] fbcon: qxldrmfb (fb0) is primary device
[ 4.673457] PM: Image not found (code -22)
[ 6.549966] systemd[1]: systemd 239 running in system mode. (+PAM +AUDIT
+SELINUX +IMA -APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP
+GCRYPT +GNUTLS +ACL +XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN2 -
IDN +PCRE2 default-hierarchy=legacy)
```

절차

1.

다음 커널 명령줄 매개변수를 추가합니다.

```
# grubby --update-kernel=/boot/vmlinuz-$(uname -r) --args="ima_policy=appraise_tcb
ima_appraise=fix evm=fix"
```

이 명령을 사용하면 현재 부팅 항목에 대한 수정 모드에서 **IMA** 및 **EVM**을 활성화하고 사용자가 **IMA** 측정값을 수집하고 업데이트할 수 있습니다.

The **ima_policy=appraise_tcb** 커널 명령줄 매개 변수는 커널이 기본 **MTCB**(신뢰할 수 있는 컴퓨팅 기반) 측정 정책 및 평가 단계를 사용하도록 합니다. 평가 부분은 이전 및 현재 측정값이 일치하지 않는 파일에 대한 액세스를 금지합니다.

2.

재부팅하여 변경 사항을 적용합니다.

3.

선택적으로 매개변수가 커널 명령줄에 추가되었는지 확인합니다.

```
# cat /proc/cmdline
BOOT_IMAGE=(hd0,msdos1)/vmlinuz-4.18.0-167.el8.x86_64 root=/dev/mapper/rhel-root
ro crashkernel=auto resume=/dev/mapper/rhel-swap rd.lvm.lv=rhel/root
rd.lvm.lv=rhel/swap rhgb quiet ima_policy=appraise_tcb ima_appraise=fix evm=fix
```

4.

커널 마스터 키를 생성하여 **EVM** 키를 보호합니다.

```
# keyctl add user kmk "$((dd if=/dev/urandom bs=1 count=32 2> /dev/null))" @u
748544121
```

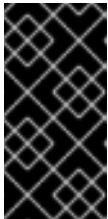
커널 마스터 키(**kmk**)는 커널 공간 메모리에 전적으로 유지됩니다. 커널 마스터 키 **kmk**의 32바이트 긴 값은 **/dev/urandom** 파일에서 임의의 바이트에서 생성되어 사용자(**@u**) 인증 키에 배치됩니다. 키 일련 번호는 이전 출력의 두 번째 행에 있습니다.

5.

kmk 키를 기반으로 암호화된 **EVM** 키를 만듭니다.

```
# keyctl add encrypted evm-key "new user:kmk 64" @u
641780271
```

명령은 **kmk** 를 사용하여 64바이트의 긴 사용자 키(**evm-key**)를 생성 및 암호화하여 사용자(@u) 인증 키에 배치합니다. 키 일련 번호는 이전 출력의 두 번째 행에 있습니다.



중요

이 이름은 **EVM** 하위 시스템 이름이 필요하며 작업 중이므로 사용자 키의 이름을 **evm-key** 로 지정해야 합니다.

6.

내보낸 키의 디렉터리를 만듭니다.

```
# mkdir -p /etc/keys/
```

7.

kmk 키를 검색하고 파일로 값을 내보냅니다.

```
# keyctl pipe $(keyctl search @u user kmk) > /etc/keys/kmk
```

명령은 이전에 정의된 위치(/etc/keys/)의 파일에 커널 마스터 키(**kmk**)의 암호화되지 않은 값을 배치합니다.

8.

evm-key 사용자 키를 검색하고 해당 값을 파일로 내보냅니다.

```
# keyctl pipe $(keyctl search @u encrypted evm-key) > /etc/keys/evm-key
```

명령은 사용자 **evm-key** 키의 암호화된 값을 임의의 위치 파일에 배치합니다. **evm-key** 는 이전에 커널 마스터 키로 암호화되었습니다.

9.

선택적으로 새로 생성된 키를 확인합니다.

```
# keyctl show
Session Keyring
974575405 --alswrv 0 0 keyring: ses 299489774 --alswrv 0 65534 \ keyring:
```

```
uid.0 748544121 --alswrv 0 0 \ user: kmk
641780271 --alswrv 0 0 \ encrypted: evm-key

# ls -l /etc/keys/
total 8
-rw-r--r--. 1 root root 246 Jun 24 12:44 evm-key
-rw-r--r--. 1 root root 32 Jun 24 12:43 kmk
```

비슷한 출력이 표시됩니다.

10.

EVM을 활성화합니다.

```
# echo 1 > /sys/kernel/security/evm
```

검증 단계

-

다음을 입력하여 **EVM**이 초기화되었는지 확인할 수 있습니다.

```
# dmesg | tail -1
[...] evm: key initialized
```

참고

시스템을 재부팅하면 인증 키에서 키가 제거됩니다. 이러한 경우 이미 내보낸 **kmk** 및 **evm-key** 키를 가져올 수 있습니다.

절차

1.

사용자 **kmk** 키(단계 7에서 `/etc/keys/kmk` 파일에 준비됨)를 추가합니다.

```
# keyctl add user kmk "$(cat /etc/keys/kmk)" @u
451342217

# keyctl show
Session Keyring
 695566911 --alswrv 0 0 keyring: ses 58982213 --alswrv 0 65534 \
keyring: uid.0 451342217 --alswrv 0 0 \ user: kmk
```

2.

사용자 **evm-key** 키(예: 8단계에서 `/etc/keys/evm-key` 파일로 내보내기)를 가져옵니다.

```
# keyctl add encrypted evm-key "load $(cat /etc/keys/evm-key)" @u
924537557

# keyctl show
Session Keyring
 695566911 --alswrv 0 0 keyring: ses 58982213 --alswrv 0 65534 \
keyring: uid.0 451342217 --alswrv 0 0 \ user: kmk
 924537557 --alswrv 0 0 \ encrypted: evm-key
```

추가 리소스

- [확장 검증 모듈](#)
- [무결성 측정 아키텍처](#)
- [커널 무결성 하위 시스템](#)
- [신뢰할 수 있는 키](#)

24.3.2.8. 무결성 측정 아키텍처를 사용하여 파일 해시 수집

무결성 측정 아키텍처(IMA)의 첫 번째 단계는 파일 해시를 생성하고 해당 파일의 확장 속성(xattrs)으로 저장할 수 있는 측정 단계입니다. 다음 섹션에서는 파일의 해시를 생성하고 검사하는 방법을 설명합니다.

사전 요구 사항

- 무결성 측정 아키텍처(IMA) 및 확장된 검증 모듈 [활성화에 설명된 대로 무결성 측정 아키텍처\(IMA\) 및 EVM\(Extended verification module\)](#)을 활성화합니다.
- `ima-evm-utils`, `attr` 및 `keyutils` 패키지가 이미 설치되어 있는지 확인합니다.

```
# yum install ima-evm-utils attr keyutils
Updating Subscription Management repositories.
This system is registered to Red Hat Subscription Management, but is not receiving updates. You can use subscription-manager to assign subscriptions.
Last metadata expiration check: 0:58:22 ago on Fri 14 Feb 2020 09:58:23 AM CET.
Package ima-evm-utils-1.1-5.el8.x86_64 is already installed.
Package attr-2.4.48-3.el8.x86_64 is already installed.
Package keyutils-1.5.10-7.el8.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
```

절차

1. 테스트 파일을 생성합니다.

```
# echo <Test_text> > test_file
```

IMA 및 EVM은 `test_file` 예제 파일에 확장 속성으로 저장된 해시 값을 할당했는지 확인합니다.

2. 파일의 확장된 속성을 검사합니다.

```
# getfattr -m . -d test_file
# file: test_file
security.evm=0sAnDly4VPA0HArpPO/EqiuTnNyBqI
security.ima=0sAQOEdeuUnWzwwKYk+n66h/vby3eD
security.selinux="unconfined_u:object_r:admin_home_t:s0"
```

예제 출력에는 SELinux 및 IMA 및 EVM 해시 값과 관련된 확장된 속성이 표시되어 있습니다. EVM은 다른 특성과 관련된 `security.evm extended` 속성을 추가합니다. `security.evm` 에서

evmctl 유틸리티를 사용하여 **RSA** 기반 디지털 서명 또는 해시 기반 메시지 인증 코드(**HMAC-SHA1**)를 생성할 수 있습니다. 이전 작업을 성공적으로 수행하려면 커널 인증 키에 저장된 유효한 신뢰할 수 있거나 암호화된 키가 필요합니다. 이 구성을 사용하면 확장된 특성에 대한 오프라인 변조 공격을 방지할 수 있습니다.

추가 리소스

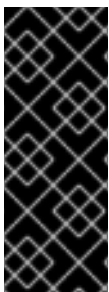
- [보안 강화](#)
- [확장 검증 모듈](#)
- [무결성 측정 아키텍처](#)

24.3.3. Ansible 역할을 사용하여 커널 매개 변수 영구 구성

Red Hat Ansible에 대한 지식이 있는 숙련된 사용자는 커널 설정 역할을 사용하여 여러 클라이언트에서 한 번에 커널 매개 변수를 구성할 수 있습니다. 이 솔루션은 다음과 같습니다.

- 효율적인 입력 설정을 갖춘 친숙한 인터페이스를 제공합니다.
- 의도한 모든 커널 매개 변수를 한 위치에 유지합니다.

제어 머신에서 커널 설정 역할을 실행하면 커널 매개변수가 관리 시스템에 즉시 적용되고 재부팅 시 지속됩니다.



중요

RHEL 채널을 통해 제공되는 **RHEL** 시스템 역할은 기본 **AppStream** 리포지토리의 **RPM** 패키지로 **RHEL** 고객이 사용할 수 있습니다. **RHEL** 시스템 역할은 **Ansible Automation Hub**를 통해 **Ansible** 서브스크립션을 사용하는 고객에 대한 컬렉션으로도 사용할 수 있습니다.

24.3.3.1. 커널 설정 역할 소개

RHEL 시스템 역할은 여러 시스템을 원격으로 관리할 수 있도록 일관된 구성 인터페이스를 제공하는 역할 집합입니다.

RHEL 시스템 역할은 커널 설정 시스템 역할을 사용하여 커널 자동 구성을 위해 도입되었습니다. **rhel-system-roles** 패키지에는 이 시스템 역할과 참조 문서도 포함됩니다.

자동화된 방식으로 하나 이상의 시스템에 커널 매개변수를 적용하려면 플레이북에서 선택한 역할 변수 중 하나 이상과 함께 커널 설정 역할을 사용합니다. 플레이북은 사람이 읽을 수 있으며 **YAML** 형식으로 작성된 하나 이상의 플레이 목록입니다.

인벤토리 파일을 사용하여 **Ansible**이 플레이북에 따라 구성하려는 시스템 집합을 정의할 수 있습니다.

커널 설정 역할을 사용하여 다음을 구성할 수 있습니다.

- **kernel_settings_sysctl** 역할 변수를 사용하는 커널 매개변수
- **kernel_settings_sysfs** 역할 변수를 사용하는 다양한 커널 하위 시스템, 하드웨어 장치 및 장치 드라이버
- **systemd** 서비스 관리자의 **CPU** 선호도 및 **kernel_settings_systemd_cpu_affinity** 역할 변수를 사용하여 포크 처리
- 커널 메모리 하위 시스템은 **kernel_settings_transparent_hugepages** 및 **kernel_settings_transparent_hugepages_defrag** 역할 변수를 사용하여 **hugepages**를 투명하게 합니다.

추가 리소스

- **/usr/share/doc/rhel-system-roles/kernel_settings/ 디렉토리의 README.md 및 README.html 파일**
- [플레이북 작업](#)
- [재고를 구축하는 방법](#)

24.3.3.2. 커널 설정 역할을 사용하여 선택한 커널 매개변수 적용

다음 단계에 따라 **Ansible** 플레이북을 준비하고 적용하여 여러 관리 운영 체제에 미치는 영향을 유지하여 커널 매개 변수를 원격으로 구성합니다.

사전 요구 사항

- **root** 권한이 있습니다.
- **RHEL** 서브스크립션에서 권한을 부여하면 컨트롤 시스템에 **ansible-core** 및 **rhel-system-roles** 패키지를 설치했습니다.
- 관리 호스트의 인벤토리는 제어 시스템에 있으며 **Ansible**은 연결할 수 있습니다.

중요

RHEL 8.0 - 8.5에서는 **Ansible**을 기반으로 자동화를 위해 **Ansible Engine 2.9**가 포함된 별도의 **Ansible** 리포지토리에 대한 액세스 권한을 제공했습니다. **Ansible Engine**에는 **ansible**, **ansible-playbook**, **docker** 및 **podman** 과 같은 커넥터, 플러그인 및 모듈 전체와 같은 명령줄 유틸리티가 포함되어 있습니다. **Ansible Engine**을 확보하고 설치하는 방법에 대한 자세한 내용은 [How do I download and Install Red Hat Ansible Engine](#) 을 참조하십시오.

RHEL 8.6 및 **9.0**에서는 **Ansible Core**(**Ansible** 명령줄 유틸리티, 명령 및 소규모 **Ansible** 플러그인 세트가 포함된 **Ansible** 코어 RPM 제공)를 도입했습니다. **AppStream** 리포지토리는 제한된 지원 범위가 있는 **ansible-core** 를 제공합니다. [RHEL 9 AppStream](#) 에 포함된 **ansible-core** 패키지에 대한 지원 범위를 검토하여 자세한 내용을 확인할 수 있습니다.

절차

1.

필요한 경우 그림 목적으로 인벤토리 파일을 검토합니다.

```
# cat /home/jdoe/<ansible_project_name>/inventory
[testingservers]
pdoe@192.168.122.98
fdoe@192.168.122.226

[db-servers]
db1.example.com
db2.example.com

[webservers]
```

```
web1.example.com
web2.example.com
192.0.2.42
```

파일은 **[testingservers]** 그룹 및 기타 그룹을 정의합니다. 이를 통해 특정 시스템 집합에 대해 **Ansible**을 더 효과적으로 실행할 수 있습니다.

2.

구성 파일을 생성하여 **Ansible** 작업에 대한 기본값 및 권한 에스컬레이션을 설정합니다.

a.

새 **YAML** 파일을 생성하고 텍스트 편집기에서 엽니다. 예를 들면 다음과 같습니다.

```
# vi /home/jdoe/<ansible_project_name>/ansible.cfg
```

b.

파일에 다음 내용을 삽입합니다.

```
[defaults]
inventory = ./inventory

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = true
```

[defaults] 섹션은 관리 호스트의 인벤토리 파일의 경로를 지정합니다.

[privilege_escalation] 섹션은 지정된 관리 호스트에서 사용자 권한을 **root**로 전환하도록 정의합니다. 커널 매개 변수를 성공적으로 구성하려면 이 작업이 필요합니다. **Ansible** 플레이북이 실행되면 사용자 암호를 묻는 메시지가 표시됩니다. 사용자는 관리 호스트에 연결한 후 **sudo**를 통해 자동으로 **root**로 전환합니다.

3.

커널 설정 역할을 사용하는 **Ansible** 플레이북을 생성합니다.

a.

새 **YAML** 파일을 생성하고 텍스트 편집기에서 엽니다. 예를 들면 다음과 같습니다.

```
# vi /home/jdoe/<ansible_project_name>/kernel-roles.yml
```

이 파일은 플레이북을 나타내며 일반적으로 인벤토리 파일에서 선택한 특정 관리 호스트에 대해 실행되는 플레이 라고도 하는 정렬된 작업 목록을 포함합니다.

b.

파일에 다음 내용을 삽입합니다.

```
---
-
  hosts: testingservers
  name: "Configure kernel settings"
  roles:
    - rhel-system-roles.kernel_settings
  vars:
    kernel_settings_sysctl:
      - name: fs.file-max
        value: 400000
      - name: kernel.threads-max
        value: 65536
    kernel_settings_sysfs:
      - name: /sys/class/net/lo/mtu
        value: 65000
    kernel_settings_transparent_hugepages: madvise
```

name 키는 선택 사항입니다. 임의의 문자열과 플레이를 레이블로 연결하고 플레이의 용도를 식별합니다. 플레이의 **hosts** 키는 플레이를 실행할 호스트를 지정합니다. 이 키의 값 또는 값은 관리 호스트의 개별 이름으로 제공되거나 인벤토리 파일에 정의된 호스트 그룹으로 제공할 수 있습니다.

vars 섹션은 선택한 커널 매개 변수 이름과 설정해야 하는 값을 포함하는 변수 목록을 나타냅니다.

roles 키는 **vars** 섹션에 언급된 매개 변수 및 값을 구성하기 위해 수행할 시스템 역할을 지정합니다.



참고

필요에 맞게 플레이북에서 커널 매개변수와 해당 값을 수정할 수 있습니다.

4.

필요한 경우 플레이의 구문이 올바른지 확인합니다.

```
# ansible-playbook --syntax-check kernel-roles.yml
```

```
playbook: kernel-roles.yml
```

이 예는 플레이북의 성공적인 확인을 보여줍니다.

5.

플레이북을 실행합니다.

```
# ansible-playbook kernel-roles.yml
```

```
...
```

```
BECOME password:
```

```
PLAY [Configure kernel settings]
```

```
*****
```

```
PLAY RECAP
```

```
*****
```

```
fdoe@192.168.122.226 : ok=10 changed=4 unreachable=0 failed=0 skipped=6
rescued=0 ignored=0
pdoe@192.168.122.98 : ok=10 changed=4 unreachable=0 failed=0 skipped=6
rescued=0 ignored=0
```

Ansible이 플레이북을 실행하기 전에 암호를 입력하라는 메시지가 표시되고 관리 호스트의 사용자가 커널 매개 변수를 구성하는 데 필요한 **root**으로 전환할 수 있습니다.

recap 섹션에는 플레이가 모든 관리 호스트에 대해 성공적으로 완료(**failed=0**)되고 4개의 커널 매개 변수가 적용되었음을 보여줍니다(**changed=4**).

6.

관리 호스트를 다시 시작하고 영향을 받는 커널 매개 변수를 확인하여 변경 사항이 적용되었는지 확인하고 재부팅 후에도 지속되는지 확인합니다.

추가 리소스

-

RHEL 시스템 역할 시작하기

-

/usr/share/doc/rhel-system-roles/kernel_settings/ 디렉토리의 README.html 및 README.md 파일

-

인벤토리 빌드

- *Ansible 구성*
- *플레이북으로 작업*
- *변수 사용*
- *역할*