IT資格道場

LinuC レベル3 3PS試験 学習用テキスト

予備知識ゼロから合格ラインへ —— 読んで理解する1冊目

無料サンプル(第1章まで)

LPI-Japan公式 LinuC レベル3 3PS試験(LinuC-3PS Version 1.0) 出題範囲 準拠
2026年7月版

本書のご利用にあたって

- 本書は「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルであっても著作物です。本書の転載・改変しての再配布・二次販売を禁じます。
- 本書を AI モデルの学習データとして利用すること(学習目的の入力・データセット化を含む)を禁じます。
- 製品版(学習用テキスト・暗記用ノート・頻出度別ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

不正な配布を見つけた場合は info@shikaku-doujou.com までご連絡ください。

目次

このセクションの使い方

第1部 仮想化基盤（主題 3ps.1）

第1章 仮想化技術の動作原理と最適化（3ps.1.1）

1-1 KVM と QEMU は「どこで動くか」で役割が分かれる

1-2 スケジューリングクラスは3階層で、上から順に探される

1-3 I/O スケジューラーは4つ —— 「何をしてくれるか」で選ぶ

1-4 Virtio —— 「ホストへ制御が移る回数」を減らす仕組み

1-5 GPU パススルー —— 3段で成立する

第1章の混同しやすい対の概念

第1章の試験ポイント

サンプルはここまでです

このテキストは、収録問題集（選択式308問・入力式91問・複数選択21問＝全420問）を解くための知識を、出題範囲の副主題単位で整理したものです。**このテキストの知識だけで、収録問題が解けるようになる**ことを完成の条件として書かれています。

- 対象試験: LinuC レベル3 3PS（プラットフォームスペシャリスト）Version 1.0
- 試験形式: 約60問・90分（アンケート5分込み）。合格点は公式非公表です
- 構成: 全20章（第1〜5章=仮想化基盤／第6〜8章=コンテナ・ストレージ／第9〜12章=コンテナ内部構造とリソース最適化／第13〜15章=高可用性・負荷分散／第16〜20章=自動化・運用）
- 各章は「本文 → 混同しやすい対の概念 → 章の試験ポイント」の順です。入力式で綴りを打つ項目は、章内の綴り表にまとめてあります

このセクションの使い方

LinuC レベル3 3PS試験の主題1「仮想化基盤」と主題2「コンテナ・ストレージ」のうち、**副主題 3ps.1.1〜3ps.1.5 と 3ps.2.1〜3ps.2.3** を担当する範囲です。章立ては副主題そのままで、第1〜5章が主題1、第6〜8章が主題2に対応します。

章	副主題	扱う中心
第1章	3ps.1.1 仮想化技術の動作原理と最適化	KVM/QEMU、スケジューリングクラス、I/O スケジューラー、Virtio、GPU パススルー

章	副主題	扱う中心
第2章	3ps.1.2 仮想ディスクと QEMU の操作	QCOW2／RAW、qemu-img、NUMA、メモリバルーニング、CPU ホットプラグ
第3章	3ps.1.3 仮想化環境のセキュリティ設定	Secure Boot、UEFI 変数、TPM 2.0、仮想 TPM、キー管理
第4章	3ps.1.4 仮想マシンの管理と OVF の活用	virsh、スナップショット、マイグレーション、OVF テンプレート
第5章	3ps.1.5 仮想化プラットフォームの利用	Proxmox VE の特徴、クラスタ、HA、バックアップ／リストア
第6章	3ps.2.1 仮想化基盤のネットワーク構築	ブリッジ／NAT／ホストオンリー、bonding、VLAN、Open vSwitch、性能最適化
第7章	3ps.2.2 クラスタ・分散ストレージの理解	OCFS2、Lustre、DRBD の比較
第8章	3ps.2.3 Ceph の理解と実装	Ceph アーキテクチャ、cephadm、管理コマンド、クライアント

各章は次の3点セットで構成しています。

1. **本文** …… 「そう決まっている」ではなく「なぜその設計になるのか」まで書いています。3PS は暗記型の設問が少なく、**要件を読んで手段を選ばせる**形が中心なので、仕組みごと理解しておくほうが取り違えが起きにくくなります

2. **混同しやすい対の概念** …… この範囲でいちばん多いひっかけは「似た役割の2つを入れ替える」型です。入れ替えられやすい組を表にしてあります
3. **章の試験ポイント** …… 実際に壊されやすい箇所と、その見分け方を並べています

3PS の設問はどう問われるか

特徴	内容
出題形式	四肢択一（正答1つ）が主力。ほかに**複数選択**（「2つ選べ」）と**コマンド入力**（コマンド名・オプションまで打たせる）がある
設問の立て方	状況（台数・容量・制約・許容できる停止時間）を先に示し、**「最も適切なものはどれか」**を選ばせる形が大半。単独の用語定義を聞く設問は少ない
誤り肢の作り	「その技術では実現できない」ではなく、**「別の技術ならできるが、これではできない」**という近接技術の性質を持ち込む形が多い
コマンド入力	名称だけを答えさせる型（`bfq`・`vfio-pci`・`OVF` など）と、**オプションまで含む1行**を答えさせる型（`ovs-vsctl add-port br0 enp3s0 trunks=10,20` など）の2種類がある

壊され方（ひっかけ）は7つに整理できる

本文中の「試験ポイント」では、この番号で型を示します。

型	仕掛け
① 近接概念ペアの入れ替え	対になる2つの中身を丸ごと入れ替える（SCHED_FIFO／SCHED_RR、db／dbx、マニフェスト／証明書、blockcommit／blockpull など。**最頻**）
② 層の取り違え	別の層（CPU／ブロック／ネットワーク／ファイルシステム）の仕組みを持ち込んで「これで解決する」と書く
③「量」と「配置」のすり替え	資源の**量**を変える手段と、同じ量を**どこに置くか**を変える手段を入れ替える（バルーニング／ホットプラグ ⇔ NUMA）
④ 前提条件の付け替え	成立条件を別の機能の条件に差し替える（HA の前提に CPU 世代一致を持ち込む、Secure Boot の前提に TPM を持ち込む など）
⑤ 手順の順序反転	正しい要素を並べたうえで**順番だけ逆**にする（縮小手順、vfio-pci のバインド時期、クラスタ構築の順序）
⑥ 効かない対処	症状は消えるが原因は残る対処を「解決」として書く（閾値を上げる、キャッシュを増やす、流量を絞る）
⑦ 存在しない機能・オプションの捏造	もっともらしいが実在しない指定を書く（`qemu-img convert` で出力サイズ指定、Secure Boot の検証除外指定、PCR への値の書き戻し）

⑦は特に注意してください。この範囲の誤り肢は「日本語として自然で、やりたいことにも合っている」ように書かれます。**実在するかどうか**を思い出せるかが分かれ目になります。

第1部 仮想化基盤（主題 3ps.1）

主題1は「ホスト1台の中で何が起きているか」を扱います。CPU をどう配るか（スケジューリング）、I/O をどう並べるか（I/O スケジューラー）、デバイスをどう見せるか（Virtio・パススルー）、ディスクをどう持つか（QCOW2／RAW）、起動をどう守るか（Secure Boot／TPM）、仮想マシンをどう運ぶか（virsh／OVF）、そしてそれを束ねる製品（Proxmox VE）—— という順で積み上がります。

第1章 仮想化技術の動作原理と最適化（3ps.1.1）

1-1 KVM と QEMU は「どこで動くか」で役割が分かれる

KVM と QEMU は組で使われますが、**動作する場所が違い、それがそのまま役割分担になっています。**

	KVM	QEMU
動く場所	ホストの**カーネル内**（カーネルモジュール）	ホストの**ユーザー空間**（プロセス）
担当	CPU の仮想化支援機能を用いた**ゲスト命令の実行**と**メモリのアドレス変換**	**デバイスのエミュレーション**（ディスク・NIC など）と仮想マシン全体の制御

ゲストの命令は、KVM が CPU の仮想化支援機能（Intel VT-x／AMD-V に相当する機能）を使って**物理 CPU 上で直接実行**させます。QEMU 単体でも1命令ずつ解釈して実行する動作は持っていますが、KVM と組む構成ではその経路を通りません。

一方、ゲストが「ディスクへ書く」「NIC へ送る」といったデバイス操作を行うと、そこで初めて QEMU 側の処理が呼ばれます。**カーネル側＝KVM、ユーザー空間側＝QEMU** という位置の違いを押さえておけば、「KVM がデバイスモデルを提供する」「QEMU がカーネルモジュールとして常駐する」といった入れ替え型の誤り肢はその場で落とせます。

1-2 スケジューリングクラスは3階層で、上から順に探される

Linux のスケジューラは**クラス単位に分かれており、上位のクラスから順に「次に走らせるタスク」を探します。**上位で候補が見つかった時点で探索は終

わります。

```
DEADLINE スケジューリングクラス    ← 最初に探される
        ↓ 候補が無ければ
RT スケジューリングクラス
        ↓ 候補が無ければ
Fair スケジューリングクラス         ← 最後に探される
```

ここで重要なのは次の2点です。

- **クラスごとに実行キューと選び方が分かれている**。優先度と nice 値を共通の尺度へ換算して横並びで比較する仕組みは**無い**
- **クラスの検索順は実装として決まっている**。ブート時のカーネルパラメータで入れ替える設定項目では**ない**
- fair のタスクが長く待たされても、カーネルが一時的に RT へ移してくれることは**無い**。待たされ続ける状態は RT 側の設計で避ける

3つのクラスと5つのポリシー

クラス	ポリシー	静的優先度	順番の決まり方
DEADLINE (EDF)	`SCHED_DEADLINE`	使わない	実行時間・期限・周期の3つから求めた**期限が近いもの**から
RT	`SCHED_FIFO` ／ `SCHED_RR`	**1〜99**（大きいほど先）	静的優先度の大小。同一優先度の扱いで両者が分かれる

クラス	ポリシー	静的優先度	順番の決まり方
Fair (CFS/EEVDF)	`SCHED_OTHER`（= `SCHED_NORMAL`）／`SCHED_BATCH`／`SCHED_IDLE`	**0 に固定**	nice 値による重みと、仮想的な期限

静的優先度 1〜99 を持つのは `SCHED_FIFO` と `SCHED_RR` の2つだけです。`SCHED_DEADLINE` は優先度の数値ではなく3つの時間量で表現し、Fair 側の3つは静的優先度が 0 に固定されているため 1〜99 の指定ができません。ここは複数選択でそのまま問われます。

SCHED_FIFO と SCHED_RR ―― 違いは同一優先度の扱いだけ

両者はどちらも RT クラスで静的優先度 1〜99 を持ちます。**違うのは、同じ優先度のスレッドどうしをどう扱うかの1点だけ**です。

- `SCHED_FIFO` …… **タイムクォンタム（時間量）による切り替えが無い。**自らブロックするか、より高い優先度に横取りされるまで走り続ける
- `SCHED_RR` …… タイムクォンタムを使い切ると**同一優先度の待ち行列の末尾へ回される。**同じ優先度の間で順番が回る

「同じ優先度のスレッド同士を一定の時間量ごとに譲り合わせたい」という要件なら `SCHED_RR` です。ここで `SCHED_FIFO` を選ぶと、CPU を長く占有する処理が入った瞬間に譲り合いが止まります。

Fair クラスへ落として nice 値をいじる案も答えになりません。nice 値を負の方向へ寄せても**変わるのは CPU 時間の取り分だけ**で、**リアルタイム優先度を与えたスレッドの間で順番が回る形そのものは作れません。**時間量ごとの譲り合いは RT クラスの `SCHED_RR` が持つ性質であって、重みの調整で代用できるものではないからです。

この性質は事故の形でも問われます。**vCPU スレッドに `SCHED_FIFO` と高い優先度を与えると、ゲスト内で CPU を回し続ける処理が走った時にホストのコンソール操作まで応答しなくなります。**理由は上の階層の話そのもので、RT クラスに実行可能なスレッドがある限り fair クラスのタスクは選ばれず、`SCHED_FIFO` は時間量で切り替わらないため CPU が返ってこないからです。

SCHED_DEADLINE —— 3つの値とアドミッション制御

`SCHED_DEADLINE` は EDF（Earliest Deadline First）に基づき、**実行時間・期限・周期の3つを指定**します。指定した実行時間を各周期で確保できるかどうかを、**カーネルが受け付け時に検査（アドミッション制御）し、確保できない要求は弾きます。**

- 実行時間と周期は**利用者が指定する値**。これが無ければ足りるかどうかを判断できないため、「期限だけ指定すればカーネルが実測から自動で決める」は誤り
- 1〜99 の静的優先度で順番が決まるのは RT クラス側で、DEADLINE の指定方法ではない
- nice 値の重みの比率で CPU 時間を配分するのは Fair 側の動作

SCHED_BATCH と SCHED_IDLE ——「重み」が違う

どちらも Fair クラスに属し静的優先度は 0 ですが、性質はまったく違います。

	SCHED_BATCH	SCHED_IDLE
CPU 時間の重み	`SCHED_OTHER` と**同じ扱い**（取り分は極端に落ちない）	**最低の扱い**
起床時の横取り	**抑える**（対話的な起床でほかを蹴らない）	—

	SCHED_BATCH	SCHED_IDLE
他に実行可能なタスクがある間	相応に進む	**ほとんど進まない**

「多少時間がかかってもよいが、他に実行可能なタスクがある間ほとんど進まないのでは困る」という要件は `SCHED_BATCH` です。`SCHED_IDLE` は重みそのものが最低なので、時間帯を選んでもその性質は変わりません。逆に `SCHED_OTHER` のまま nice 値を 0 に戻すと、対話的な処理と同じ重みでゲストの vCPU スレッドから実行機会を奪う側に回ります。

コマンド —— chrt

スケジューリングポリシーと優先度の設定・確認には `chrt` を使います。

```
# 現在のポリシーと優先度を見る
chrt -p <PID>

# SCHED_RR・優先度50 で新しいコマンドを起動する
chrt --rr 50 <コマンド>

# 稼働中のプロセスを SCHED_FIFO・優先度80 へ変更する
chrt --fifo -p 80 <PID>

# Fair クラスへ戻す（SCHED_OTHER・優先度は 0 固定）
chrt --other -p 0 <PID>
```

1-3 I/O スケジューラーは4つ ―― 「何をしてくれるか」で選ぶ

ブロックデバイスごとに、`none`・`mq-deadline`・`bfq`・`kyber` の4つから選びます。現行のカーネルはマルチキュー（blk-mq）を前提としているため、シングルキュー時代に公平配分を担っていた `cfq` **は存在しません**（入力式の誤答としてよく置かれます）。

スケジューラー	していること	向く場面
`none`	**並べ替えも期限管理もしない**。要求をそのままデバイスのキューへ渡す（ブロック層が順番に手を入れない）	NVMe SSD など、深いキューと高い並列度を持ち、並べ替えの利得が乏しいデバイス。**要求あたりの CPU 時間が最小**
`mq-deadline`	要求ごとに**期限**を持たせる。期限までは並べ替えでスループットを稼ぎ、期限が来た要求を割り込ませて**飢餓を防ぐ**。読み込みの期限は書き込みより短く取る	回転式 HDD などで、書き込みが続く間に読み込みが待たされる状況
`bfq`	**プロセス（cgroup）単位の重み**でディスク帯域を比例配分する。対話的な I/O を出すプロセスを優遇する	発行元ごとに帯域を切り分けたい場面（VM ごと・利用者ごと）。ただし計算が比較的重い
`kyber`	読み込みと書き込みそれぞれに**目標レイテンシ**を持ち、実測に応じて**デバイスへ送出するキューの深さを絞る**。軽量	高速なフラッシュストレージで応答時間に目標値を置きたい場面

選び分けの軸は「何を約束してくれるか」です。

- 「1件あたりの待ち時間に上限を作りたい」→ `mq-deadline`（期限）

- 「誰にどれだけ配るかを決めたい」→ `bfq`（配分）
- 「応答時間の目標に寄せたい」→ `kyber`（流量の調節）
- 「何もしないでほしい」→ `none`

`kyber` **の向きも取り違えないでください。**kyber がしているのは**キューの深さを絞ること**なので、**深さを大きく取れば待たされる要求はむしろ増えます。**「待たされ続けるのを防ぎたい」場面で kyber のキューを深くする案は、仕組みの向きが逆になっています。

この違いを取り違えると、誤り肢に引っかかります。たとえば「VM 単位で帯域を按分したい」に `mq-deadline` を持ってきても、**個々の要求の待ち時間に上限は作れても VM ごとの取り分は決められません。**逆に「読み込みが待たされ続ける」に `bfq` の重みを揃えても、取り分は均等になるだけで**待ち時間の上限は作られません。**

設定は sysfs から行います。

```
# 現在の I/O スケジューラーを確認する（[] が付いているものが現在値）
cat /sys/block/sda/queue/scheduler

# bfq へ切り替える
echo bfq > /sys/block/sda/queue/scheduler
```

1-4 Virtio ——「ホストへ制御が移る回数」を減らす仕組み

完全エミュレーション方式の仮想デバイスは、**物理デバイスのレジスタ仕様をそのまま再現**します。ゲストのドライバがレジスタを触るたびにホストへ制御が移り、その回数がそのまま CPU 時間として積み上がります。

Virtio は**準仮想化**の方式で、この再現をやめます。ゲスト側の**フロントエンドドライバ**とホスト側の**バックエンド**が**共有のリング**を通じて複数の要求をまとめて受け渡すため、レジスタ操作のたびにホストへ制御が移らずに済みます。同じ通信量でホストの CPU 使用率が下がるのは、この**制御が移る回数の差**が主因です。

誤り肢としてよく置かれるのは次の3つです。いずれも別の仕組みの説明になっています。

- 「物理 NIC のレジスタ仕様を忠実に再現する」→ **エミュレーション方式側**の説明
- 「物理 NIC がゲストのメモリを直接読み書きする」→ **デバイスをパススルーした場合**の形
- 「割り込みを束ねてゲスト内の処理量そのものを減らす」→ ゲスト内のドライバの処理量は減らない

5つの Virtio デバイス

デバイス	提供するもの	押さえどころ
`virtio-net`	準仮想化 NIC	Virtio の動作原理そのものが問われる主役
`virtio-blk`	準仮想化ブロックデバイス	**ディスク1本ごとに1デバイス**。本数が増えるとゲストへ見せるデバイス数がそのまま増える
`virtio-scsi`	準仮想化 SCSI **コントローラ**	**1つのコントローラの配下に多数のディスクをぶら下げられる**。SCSI コマンドをホスト側のデバイスへ渡す使い方にも対応する

デバイス	提供するもの	押さえどころ
`virtio-rng`	準仮想化乱数デバイス	**ホスト側の乱数源をゲストのエントロピープールへ供給**する
`virtio-balloon`	メモリの回収・返却	**ゲスト内のドライバが協調して初めて成立する**

`virtio-blk` と `virtio-scsi` の使い分けは頻出です。1台のゲストへ20本以上の仮想ディスクを接続する計画や、将来ホスト側の SCSI デバイスをゲストへそのまま見せたい場合は `virtio-scsi` を選びます。`virtio-blk` は本数分だけデバイスが増えていきます。

`virtio-rng` が要る場面も具体的です。ゲスト上で TLS の鍵生成を伴うサービスの起動が数十秒待たされるとき、疑うのはゲスト内の**乱数の供給待ち**です。ゲストは自分の中だけでは乱数の材料を集めにくく、その不足が起動の遅延として表面化します。ホスト上のファイルを読ませる形ではエントロピープールへ供給されないので、解決になりません。

virtio-balloon は「協調型」── ここが一番問われる

`virtio-balloon` は、**ゲスト内のバルーンドライバがページを確保（インフレート）し、その位置をホストへ伝え、ホストがその分を他へ回す**という仕組みです。次の3点が誤り肢の材料になります。

1. **ホストが一方的にページを取り上げる仕組みではない**。ドライバが無い、あるいはゲストが応答しない場合は返させられない
2. **ホスト側のスワップとは別物**。スワップはホスト側だけで完結し、ゲストの協調を必要としない

3. **割り当てメモリ量の「上限」を書き換える機能ではない。**バルーンが動かせるのは起動時に決めた上限の**範囲内で実際に使わせる量**であり、増減にゲストの再起動は不要

1-5 GPU パススルー —— 3段で成立する

GPU をゲストへ丸ごと占有させる構成は、次の3段で成り立ちます。**どれか1段でも欠けると成立しません。**

1. ハードウェア設定	IOMMU を有効化する（VT-d / AMD-Vi）
2. カーネル設定	対象 PCI デバイスを vfio-pci にバインドする
3. QEMU/virsh 設定	そのデバイスをゲストへ割り当てる（hostdev）

IOMMU が果たしている役割

IOMMU は、**デバイスが出したアドレスを変換表で照合してホストの物理アドレスへ変換し、変換表に無い領域への DMA を遮断します。**

パススルーではゲストがデバイスを直接操作するため、これが無いとゲストが指定したアドレス次第でホストや他のゲストのメモリを読み書きできてしまいます。誤り肢では、ブロック層の I/O スケジューラ（要求の並べ替え）や、スケジューラ（CPU 時間の割り当て）といった**別の層の役割**が持ち込まれます。

割り込みとの関係も整理しておきます。 IOMMU には割り込みリマップの機能もありますが、**パススルーの文脈で問われるのは DMA のアドレス変換と遮断のほう**です。**デバイスが出した割り込みを受け取って対応する vCPU スレッドへ配送するのはホストのカーネル側の仕事**であり、これを IOMMU の役割として説明する肢は、いま問われている DMA のアドレス変換と保護の説明になっていません。

IOMMU グループは分離の最小単位

VFIO は **IOMMU グループを分離の最小単位**として扱います。グループ内のすべてのデバイスを `vfio-pci` へ預けられない限り、パススルーは成立しません。ホストが使い続けるデバイスが同じグループに残ると DMA の分離が保証できないためです。

したがって、パススルーしたい GPU と同じグループにホスト使用中の USB コントローラが入っていた場合、取るべき判断は「**USB コントローラをホストから切り離せるかを確認する**」です。GPU の映像機能だけをバインドして音声機能を残す形も、PCI アドレスだけを `hostdev` に書く形も成立しません。

- **IOMMU グループの構成はチップセットのトポロジで決まります。** 起動のたびに変わるものではないので、再起動を繰り返しても解決しません。装着するスロットを変えて構成が変わるかを見る手はあります

```
# IOMMU グループとその中のデバイスを一覧する
for g in /sys/kernel/iommu_groups/*; do
  echo "IOMMU Group ${g##*/}:"
  for d in "$g"/devices/*; do lspci -nns "${d##*/}"; done
done
```

vfio-pci のバインドは「先に」

ホストのベンダー提供ドライバが起動時に GPU を掴んでしまい、VM 起動直前に解除する運用では動作が安定しない —— これは典型的な症状です。恒久的な対処は、`vfio-pci` **を initramfs に含めて先に読み込ませ、対象のベンダ ID とデバイス ID を指定してバインドする**ことです。

- 一度ドライバが初期化したデバイスを後から預け替える形では、不安定さの原因が残ったままになります（GPU のように初期化が重いデバイスで顕著）
- 手作業の解除を運用に組み込んでも、掴まれる順番は変わりません
- **IOMMU のカーネルパラメータはアドレス変換の扱いを決めるもので、ホスト側のドライバの読み込み順は制御しません**

```
# カーネルパラメータの例（IOMMU の有効化と vfio-pci へのバインド指定）
intel_iommu=on                  # Intel プラットフォーム
amd_iommu=on                    # AMD プラットフォーム
vfio-pci.ids=10de:xxxx,10de:yyyy   # ベンダID:デバイスID を指定してバインド
```

libvirt の managed 属性

libvirt の定義で PCI デバイスを `<hostdev>` として渡すとき、`managed` 属性で**付け外しを自動化するかどうか**が決まります。

managed	ホスト側の挙動
有効（`yes`）	**VM の起動時**にホスト側のドライバからデバイスを切り離して `vfio-pci` へ預け、**VM の停止時**に元のドライバへ戻す
無効（`no`）	この付け外しを**管理者が事前に済ませておく**ことが前提になる

```
<hostdev mode='subsystem' type='pci' managed='yes'>
  <source>
    <address domain='0x0000' bus='0x01' slot='0x00' function='0x0'/>
  </source>
</hostdev>
```

`managed` が担うのは**この付け外しの自動化だけ**です。ここから、次の3つはいずれも `managed` を有効にした場合の挙動として誤りになります。

- 「ホストの起動時に固定して保持し続ける」→ **無効時の運用**（管理者が事前にバインドを済ませておく形）にあたります
- 「同じ IOMMU グループのデバイスを検出し、その構成を定義ファイルへ書き出す」→ **そうした書き出しは行いません。** グループの構成を調べるのは管理者側の作業です
- 「割り当て状況を監視して複数の VM から順番に使えるように切り替える」→ **パススルーしたデバイスは1つの VM が占有する**もので、切り替える仕組みではありません

第1章の混同しやすい対の概念

	SCHED_FIFO	SCHED_RR
クラス	どちらも RT スケジューリングクラス	
静的優先度	1〜99（同じ）	1〜99（同じ）
同一優先度の扱い	**自らブロックするまで走り続ける**（時間量による切り替えが無い）	**タイムクォンタムを使い切ると待ち行列の末尾へ回る**

	SCHED_FIFO	SCHED_RR
事故の形	CPU を手放さない処理でホストごと固まる	—

	SCHED_BATCH	SCHED_IDLE
クラス	どちらも Fair、静的優先度は 0	
CPU 時間の重み	`SCHED_OTHER` と**同じ**	**最低**
他に実行可能なタスクがある間	相応に進む	**ほとんど進まない**
何を抑えるか	対話的な起床による**横取り**	—

	RT スケジューリングクラス	DEADLINE スケジューリングクラス
指定するもの	静的優先度（1〜99）	**実行時間・期限・周期の3つ**
順番の決め方	優先度の大小	**期限が近いもの**から（EDF）
受け付け時の検査	無い	**アドミッション制御で足りない要求を弾く**

	mq-deadline	bfq
約束するもの	個々の要求の**待ち時間の上限**（期限）	発行元ごとの**帯域の取り分**（比例配分）
単位	要求	プロセス（cgroup）
向く症状	書き込みが続く間に読み込みが待たされる	1つの発行元が帯域を占有して他が止まる

	kyber	none
していること	目標レイテンシに合わせて**送出するキューの深さを絞る**	**何もしない**（そのままデバイスへ渡す）
CPU 負荷	軽量	**最小**
向く場面	応答時間に目標値を置きたい高速デバイス	並べ替えの利得が乏しくCPU 時間を削りたい高速デバイス

	virtio-blk	virtio-scsi
見え方	ディスク**1本ごとに1デバイス**	**1つのコントローラの配下に多数**
本数が増えると	ゲストへ見せるデバイス数がそのまま増える	コントローラは1つのまま
ホストの SCSI デバイス	—	**SCSI コマンドを渡す使い方に対応**

	virtio-balloon	ホスト側のスワップ
ゲストの協調	**必要**（ゲスト内のドライバがページを確保して申告）	不要（ホスト側だけで完結）
動かせるもの	起動時の上限の**範囲内**で実際に使わせる量	—
ゲストの再起動	不要	—

	KVM	QEMU
動く場所	ホストの**カーネル内**	ホストの**ユーザー空間**

	KVM	QEMU
担当	ゲスト命令の実行・メモリのアドレス変換	デバイスのエミュレーション・VM 全体の制御

第1章の試験ポイント

- **FIFO と RR の入れ替え（①型）**：「`SCHED_FIFO` はタイムクォンタムを使い切ると待ち行列の末尾へ回される」は誤り。それは `SCHED_RR` です。**時間量で切り替わらないのが FIFO** で、この1点だけが両者の差です
- **DEADLINE に優先度を持ち込む（①型）**：「`SCHED_DEADLINE` は 1〜99 の静的優先度を指定する」は誤り。3つの時間量で指定します。逆に「静的優先度 1〜99 を持つポリシー」を2つ選ばせる設問では、**FIFO と RR の2つ**が答えで、`SCHED_BATCH`・`SCHED_IDLE` は Fair で優先度 0 固定、`SCHED_DEADLINE` は数値の優先度を使いません
- **BATCH と IDLE の取り違え（①型）**：「他に実行可能なタスクがある間ほとんど進まないのでは困る」という条件が出たら `SCHED_IDLE` は落ちます。**重みが最低なのは IDLE、OTHER と同じなのが BATCH** です
- **クラス階層の作り替え（⑦型）**：「3つのクラスのタスクが1つの実行キューに並ぶ」「クラスの検索順はカーネルパラメータで決まる」「fair のタスクが待たされると RT へ移される」は、いずれも実在しない仕組みです
- **I/O スケジューラーの目的違い（①型）**：**期限＝mq-deadline／配分＝bfq／目標レイテンシ＝kyber／何もしない＝none**。「VM ごとに按分」に mq-deadline、「待たされ続けるのを防ぐ」に bfq を当てる形が定番の壊し方です
- **cfq を混ぜる（⑦型）**：入力式の誤答として `cfq` が置かれます。**マルチキュー（blk-mq）前提の現行カーネルには存在しません**

- **層の取り違え（②型）**：「ホストのコンソールまで応答しない」に I/O スケジューラーを当てる、IOMMU の説明にブロック層の並べ替えを当てる、といった形。**CPU の話か、ブロックの話か、DMA の話か**を先に切り分けてください
- **virtio-balloon を一方的な回収と書く（①型）**：「ゲストの構成に手を入れる必要はない」「ホスト側の指示だけで物理メモリを切り離せる」は誤り。**ゲスト内のドライバの動作が要る協調型**です。ホスト側で完結するのはスワップの説明です
- **virtio-scsi と virtio-blk の入れ替え（①型）**：多数のディスクを束ねる／ホストの SCSI デバイスを見せるなら **virtio-scsi**。virtio-blk は本数分デバイスが増えます
- **IOMMU グループの扱い（③④型）**：「映像機能だけをバインドして音声機能はホストに残す」「PCI アドレスだけを hostdev に書く」は成立しません。**グループ全体が単位**です。「再起動を繰り返せば構成が変わる」も誤りで、構成はチップセットのトポロジで決まります
- **vfio-pci のバインド時期（⑤型）**：ドライバが掴んだ後に上書きする形・手作業で解除する形はどちらも不安定さが残ります。**initramfs に含めて先に読ませる**のが恒久対処です。IOMMU のカーネルパラメータではドライバの読み込み順は変わりません
- **managed の挙動（①型）**：有効なら **VM の起動時にデタッチ、停止時に元へ戻す**。「ホスト起動時に固定して保持し続ける」は無効時の運用です
- **入力式で綴りを問われる語**：`bfq`（`BFQ` ／ `Bfq` も可）、`vfio-pci`（`vfio_pci` も可）。`vfio` **だけではフレームワーク全体の名称**にあたり、バインド先のドライバ名にはなりません

サンプルはここまでです

続き(第2章〜巻末)は、LinuC 3PS 問題集のご購入で、暗記用ノート・頻出度別ノートと合わせて3点すべてダウンロードできます。

LinuC レベル3 3PS試験 学習用テキスト(無料サンプル)

版

2026年7月版(LPI-Japan公式 LinuC レベル3 3PS試験(LinuC-3PS Version 1.0) 出題範囲 準拠)

発行

IT資格道場(https://www.shikaku-doujou.com)

お問い合わせ

info@shikaku-doujou.com