Section A — Interruptions sous Linux

Sous Linux, une interruption (IRQ, Interrupt Request) est un signal matériel envoyé au processeur lorsqu'un périphérique a besoin de traitement. Les interruptions permettent au CPU de réagir sans scrutation permanente.

On distingue : interruptions matérielles (hard IRQs) et interruptions logicielles (softIRQs). Les handlers (ISRs) exécutent la routine associée, et les traitements longs sont différés en bottom-half (softIRQ, workqueue).

La latence d'interruption correspond au délai entre la réception de l'IRQ et son traitement. Une latence excessive dégrade les performances (ex. perte de paquets). Le mécanisme NAPI (New API) limite les interruptions réseau à haute charge via le polling.

L'observation des IRQ sous Linux s'effectue via : grep . /proc/interrupts | head -n 40 cat /proc/softirqs

Chaque IRQ réseau (eth0, nvme, etc.) possède une affinité CPU configurable dans /proc/irq/$IRQ/smp_affinity_list. L'assignation d'une IRQ réseau à un CPU local à la file RX/TX réduit la latence. Les tests avec ping -f ou iperf3 montrent l'évolution de NET_RX dans /proc/softirqs.

Le service irqbalance répartit automatiquement les IRQ entre cœurs. Avec taskset, on peut pinner un processus réseau sur un CPU spécifique pour comparer les performances avec/sans co-localisation.

Si un handler est trop long, il bloque d'autres IRQ et augmente la latence globale. Linux délègue alors le traitement à des softIRQs, workqueues ou via NAPI pour préserver la réactivité.

Section B — Caches (CPU/TLB, page cache)

Les caches réduisent l’écart de vitesse entre le CPU, la mémoire et le stockage.

Les caches CPU (L1, L2, L3) stockent les lignes de mémoire fréquemment utilisées. Le TLB (Translation Lookaside Buffer) met en cache les correspondances entre adresses virtuelles et physiques. Le page cache du noyau garde en mémoire les blocs récemment lus depuis le disque.

Un micro-benchmark montre que les accès séquentiels profitent de la localité spatiale et du prefetching matériel, tandis que les accès aléatoires provoquent davantage de cache et TLB misses. La courbe expérimentale illustre un temps d’accès croissant dès que la taille dépasse la capacité du LLC (Last Level Cache).

Lors de lectures disque, le page cache accélère les accès répétés. Sur cache froid, la lecture (dd if=...) dépend du périphérique. Sur cache chaud, elle est quasi instantanée. Le mécanisme de readahead anticipe les blocs voisins pour améliorer la performance.

Le cache CPU et le page cache influencent directement la latence et le débit global du système.

Section C — Virtualisation (KVM/QEMU, Containers)

La virtualisation permet d’exécuter plusieurs environnements isolés sur une même machine. KVM/QEMU (Type 2) utilise le support matériel (Intel VT-x/AMD-V) et les pilotes virtio pour améliorer les E/S.

Les containers (Docker, Podman) n’émulent pas le matériel mais isolent les processus via namespaces et cgroups. Ils démarrent quasi instantanément et partagent le même noyau que l’hôte.

Dans une VM, le démarrage et les E/S sont plus lents (pile virtio, double cache). Exemple : systemd-analyze time montre un temps de boot 2–3× supérieur à l’hôte. Les tests avec stress-ng et fio indiquent une perte de 10–15 % sur les E/S disque.

Les containers, eux, offrent un overhead minimal et une rapidité proche du natif, mais un isolement moindre. L’E/S disque en VM est plus coûteuse car elle traverse plusieurs couches (virtio $\rightarrow$ hyperviseur $\rightarrow$ pilote hôte).

En pratique, les environnements modernes combinent VM + containers pour équilibrer performance et sécurité.

Section D — Multiprocessing / Multicœurs

Le noyau Linux planifie les processus sur les différents cœurs via l'ordonnanceur CFS. L'affinité CPU (taskset) permet de pinner un processus sur un cœur pour réduire les migrations et optimiser la localité du cache.

La synchronisation entre threads repose sur les verrous (mutex, spinlock). Sans verrou, on observe des conditions de course. Les mutex assurent la cohérence au prix d'un léger overhead. Les spinlocks sont adaptés aux sections critiques très courtes.

Le faux-partage (false sharing) se produit lorsque deux threads écrivent dans la même ligne de cache. Le padding de 64 octets entre variables élimine ce problème. Sur systèmes NUMA, il faut lier les threads et la mémoire au même nœud (numactl --cpunodebind --membind).

Une bonne gestion de l'affinité, des verrous et de la mémoire locale NUMA garantit la scalabilité et la stabilité du système multicœur.

Section E — Synthèse transversale

Un serveur web Linux illustre la synergie entre ces mécanismes :

1. Un paquet réseau déclenche une IRQ traitée par le noyau et dispatchée via softIRQ (NET_RX). 2. Le scheduler planifie le thread serveur sur un CPU local. Les données transitent dans les caches CPU et TLB. 3. Les fichiers demandés sont servis via le page cache ; les lectures disque réelles sont rares. 4. L'application peut tourner dans un container (rapide, léger) ou une VM (plus isolée mais plus lente). 5. Le multicœur et les verrous assurent la concurrence entre threads.

Schéma conceptuel : [Carte réseau] → IRQ → softIRQ → Scheduler → Thread appli → Page cache / disque → Réponse NET_TX

La performance dépend donc de l'interaction harmonieuse entre le matériel, le noyau et les logiciels.