گزارش امتیازی ALFS (Heap-based CFS Simulation)

این گزارش سه محور دارد:

1. چرا CFS واقعی از **RB-Tree** استفاده می‌کند و Heap چه trade-offهایی دارد.
2. جایگزین‌های مطرح CFS (به‌ویژه **EEVDF**) و مقایسه‌ی پژوهشی.
3. اثر معماری‌های **asymmetric / big.LITTLE** روی زمان‌بندها و implicationهای طراحی.

> یادآوری: ALFS در این پروژه یک شبیه‌ساز user-space است و دقیقاً کرنل لینوکس نیست.

---

1) Min-Heap: «همه‌چیز فقط extract-min نیست» در برابر CFS در RB-Tree

1.1 CFS دقیقاً از RB-Tree برای چه استفاده می‌کند؟

مستند طراحی CFS در کرنل لینوکس صریحاً می‌گوید runqueue یک **rbtree مرتب‌شده بر اساس** `vruntime` نگه می‌دارد و «leftmost» (کمترین vruntime) را برای اجرا انتخاب می‌کند. ایده‌ی کلیدی هم این است که با پیشرفت زمان، entityهای اجراشده آرام‌آرام به سمت راست می‌روند تا هر task در یک بازه‌ی قابل‌پیش‌بینی، فرصت CPU بگیرد.
منبع: [K1].

1.2 آیا Heap از نظر انتخاب min بهتر نیست؟

محدود نیست. چند peek-min فقط به CFS ،عالی است؛ اما در کرنل واقعی `peek-min` برای Heap عملیات مهم دیگر:

A) حذف/به‌روزرسانی یک task دلخواه (arbitrary removal/update)

- در کرنل، taskها مرتب unblock/block می‌شوند، از runqueue حذف می‌شوند، و دوباره insert می‌شوند.
- در RB-tree: با داشتن اشاره‌گر به node، حذف O(log n) است.
- در heap *ساده* اگر index نگه ندارید، حذف می‌تواند O(n) شود چون باید عنصر را پیدا کنید.
  (در ALFS ما برای حل این مشکل index داخل heap را نگه می‌داریم و حذف را O(log n) می‌کنیم.)
- این نکته در بحث‌های فنی هم مطرح شده که heap بدون “handle” برای حذف دلخواه دردسر دارد. [K2]

B) traversal مرتب (in-order iteration)

در برخی مسیرها/heuristicها، داشتن ability برای حرکت “به ترتیب vruntime” (successor/predecessor) یا کار با rangeها مفید است.

- به شکل طبیعی این قابلیت را می‌دهد RB-tree.

- Heap و order کامل را نمی‌دهد فقط "کمترین" را به شکل مستقیم می‌دهد.

C) SMP / load-balancing

در لینوکس واقعی، زمان‌بندی per-CPU است و load balancing بین runqueueها یک بخش مهم طراحی است. داشتن یک ساختار مرتب می‌تواند طراحی و پیاده‌سازی heuristicهای انتخاب/مهاجرت را ساده‌تر کند. [K1]

D) group scheduling (cgroup)

با group schedulingهای بیشتری داریم و عملیات‌های جانبی بیشتر می‌شود. entity، RB-tree به خاطر traversal مرتب و عملیات‌های استاندارد، انتخاب قابل دفاعی است.

1.3 کجا بهتر است؟ Heap

- heap آرایه‌ای cache-friendly است و پیاده‌سازی‌اش می‌تواند ساده و سریع باشد.
- `extract-min` و `insert` در O(log n) و `peek-min` در O(1).

اما هزینه‌ی heap در workloadهایی که eligibility پیچیده است بالا می‌رود:

- چون heap به شما "کمترین **eligible** برای CPU k" را نمی‌دهد، مجبور می‌شوید pop/push موقت انجام دهید (بدترین حالت نزدیک به O(n) per tick در سناریوهای pathological مثل affinity شدید یا maskهای محدود).

> نکته‌ی عملی: در این پروژه workloadها کوچک‌اند و این worst-case عملاً کمتر رخ می‌دهد؛ اما در کرنل واقعی با هزاران task و topology پیچیده، این هزینه می‌تواند مهم شود.

1.4 نتیجه‌گیری بخش ۱

بلکه برای مجموعه‌ای از min، یک انتخاب "سیستمی" است: نه فقط برای انتخاب CFS برای RB-tree قابل استفاده است، اما Heap (گروه‌ها، SMP، traversal، زیاد/insert/حذف) عملیات‌های سیستم‌عاملی می‌تواند بدترین‌حالت‌های بدی بسازد overhead eligibility.

---

2) جایگزین‌های CFS و تمرکز روی EEVDF

2.1 چیست و چرا مهم است؟ EEVDF

EEVDF (Earliest Eligible Virtual Deadline First) یک الگوریتم proportional share است که ایده‌ی اصلی‌اش این است:

- هر entity یک مفهوم virtual time دارد،
- ولی انتخاب بر اساس "earliest **eligible** virtual deadline" انجام می‌شود، نه صرفاً کمترین vruntime.

دو مرجع پژوهشی مهم:

- Ion Stoica, Hussein Abdel-Wahab, “Earliest Eligible Virtual Deadline First: A Flexible and Accurate Mechanism for Proportional Share Resource Allocation”, TR-95-22, 1995. [E1]
- Scott A. Brandt, James B. Weissman, “EEVDF Proportional Share Resource Allocation Revisited”, RTSS 2000 (PDF proceedings). [E2]

2.2 تفاوت ایده‌ای EEVDF با CFS

- CFS عمدتاً با vruntime کار می‌کند و “کمترین vruntime” را اجرا می‌کند.
- EEVDF علاوه بر virtual time، مفهوم **eligible time** و **virtual deadline** دارد:
  - entityها تا “eligible” نشوند وارد رقابت اصلی نیستند.
  - سپس entity با earliest virtual deadline انتخاب می‌شود.

در لینوکس هم یک مستند رسمی درباره‌ی EEVDF وجود دارد که همین مفاهیم را برای context کرنل توضیح می‌دهد. [K3]

> نکته‌ی مهم: EEVDF صرفاً یک ایده‌ی پژوهشی نیست؛ لینوکس در سال‌های اخیر آن را در چارچوب CFS وارد کرده است تا latency بهتر و fairness دقیق‌تر به‌دست آید. [K3]

2.3 جایگزین‌های دیگر (BFS / MuQSS و ...)

برای یک منبع (interactivity/desktop تمرکز روی) مطرح شدند Con Kolivas توسط MuQSS و BFS استفاده [M1]: را مقایسه کرده، می‌توانید از MuQSS و CFS که (thesis در حد) دانشگاهی قابل استناد کنید.

2.4 جایگاه ALFS

در این پروژه ALFS:

- از CFS “فلسفه‌ی vruntime و وزن nice” را نگه می‌دارد،
- اما ساختار داده‌ی انتخاب را از RB-tree به heap تغییر می‌دهد،
- و (با cgroup) یک مدل ساده از hierarchical fairness را شبیه‌سازی می‌کند.

پس ALFS بیشتر یک **آزمایش مهندسی روی ساختار داده** است تا یک الگوریتم scheduling جدید مثل EEVDF.

---

3) big.LITTLE های ناهمسان یعنیCPU روی fairness: و زمان‌بندها چه؟

3.1 مشکل اصلی big.LITTLE

در big.LITTLE:

- coreها capacity یکسان ندارند.

- یک tick روی big با یک tick روی LITTLE برابر نیست (از نظر "کار انجام‌شده").
- migration بین خوشه‌ها هزینه‌ی cache/warmup دارد و می‌تواند latency را خراب کند.

دو مرجع پژوهشی مفید:

- Vinicius Petrucci, Orlando Loques, "Lucky scheduling for energy-efficient heterogeneous multi-core systems", USENIX HotPower 2012. [B1]
- Agostino Mascitti et al., "Dynamic partitioned scheduling of real-time tasks on ARM big.LITTLE architectures", Journal of Systems Architecture (2020). [B2]

3.2 پیام برای طراحی زمان‌بند

A) fairness باید capacity-aware شود
اگر CPUها همسان نباشند، fairness صرفاً بر حسب زمان (time fairness) می‌تواند غلط‌انداز باشد. بعضی طرح‌ها "work-based" یا "capacity-normalized" را پیشنهاد می‌کنند.

B) energy-aware placement
در لینوکس، Energy Aware Scheduling (EAS) برای topologies ناهمسان اضافه شده و از مدل انرژی برای تصمیم placement استفاده می‌کند. [K4][K5]

C) overhead مهاجرت
بسیاری از سیاست‌ها تلاش می‌کنند migration را کاهش دهند تا latency و thrashing کم شود.

3.3 پیشنهادهای طراحی برای ALFS اگر بخواهیم big.LITTLE-aware شویم (پیشنهادی)

(در این پروژه پیاده نشده، ولی برای دفاع/جمع‌بندی پژوهشی مفید است)

1. تعریف `delta_exec` بر اساس capacity CPU:
   - `effective_exec = quanta_us * cpu_capacity[cpu]`
     - سپس vruntime بر اساس effective_exec update شود.
2. برای مهاجرت penalty:
   - اگر task روی CPU قبلی eligible است، ترجیح به sticky CPU بده.
3. برای quota/shares گروه‌ها، accounting را بر اساس "work" نه صرفاً "time" انجام دهید.

نتیجه‌گیری: آیا ALFS «بهتر» است؟

- برای شبیه‌سازی ساده و آموزش ساختار داده، heap قابل قبول و حتی جذاب است.
- برای کرنل واقعی با load balancing، SMP، و عملیات‌های جانبی زیاد، RB-tree انتخابی قابل‌دفاع است. [K1]
- الگوریتم‌هایی مثل EEVDF تغییر مفهومی عمیق‌تری نسبت به "تعویض ساختار داده" هستند و هدفشان latency بهتر با حفظ proportional fairness است. [E1][E2][K3]

References

Kernel / Docs

- [K1] Linux Kernel Documentation — CFS design: `sched-design-CFS` https://www.kernel.org/doc/html/latest/scheduler/sched-design-CFS.html
- [K2] بحث فنی درباره‌ی RB-tree vs heap در CFS (جمع‌بندی غیررسمی برای intuition؛ نه منبع رسمی کرنل): https://stackoverflow.com/questions/33191110/reason-why-cfs-scheduler-using-red-black-tree
- [K3] Linux Kernel Documentation — EEVDF scheduler: https://www.kernel.org/doc/html/latest/scheduler/sched-eevdf.html
- [K4] Linux Kernel Documentation — Energy Aware Scheduling (EAS): https://www.kernel.org/doc/html/latest/scheduler/sched-energy.html
- [K5] LWN — Energy Aware Scheduling overview (زمینه/تاریخچه): https://lwn.net/Articles/749738/

EEVDF (papers)

- [E1] Ion Stoica, Hussein Abdel-Wahab (1995), TR-95-22 (PDF): https://people.eecs.berkeley.edu/~istoica/papers/eevdf-tr-95.pdf
- [E2] Scott A. Brandt, James B. Weissman (2000), RTSS proceedings paper (PDF): https://users.soe.ucsc.edu/~sbrandt/rtss2000/proceedings/6.pdf

big.LITTLE / Heterogeneous scheduling (papers)

- [B1] Petrucci, Loques (2012), HotPower (PDF): https://www.usenix.org/system/files/conference/hotpower12/hotpower12-final34.pdf
- [B2] Mascitti et al. (2020), Journal of Systems Architecture (ScienceDirect page): https://www.sciencedirect.com/science/article/pii/S0164121220302764

Other scheduler comparisons

- [M1] “CFS and MuQSS Comparison” (DiVA portal thesis PDF): https://www.diva-portal.org/smash/get/diva2:1566002/FULLTEXT01.pdf