УДК 004.45:004.057.2 **doi: 10.26906/SUNZ.2026.1.192**

О. В. Запорожець, В. С. Макаренко, О. Ю. Дрозд

Харківський національний університет радіоелектроніки, Харків, Україна

ОПТИМІЗАЦІЯ АЛГОРИТМІВ ПЛАНУВАННЯ ПРОЦЕСІВ В ОПЕРАЦІЙНИХ СИСТЕМАХ З ВИКОРИСТАННЯМ ІМІТАЦІЙНОГО МОДЕЛЮВАННЯ

Анотація. Актуальність. Оптимізація планування процесів є критичною для сучасних ОС через різнорідні профілі навантаження, де компроміс між латентністю, пропускною здатністю та справедливістю неможливо надійно оцінити лише аналітично. **Об'єкт дослідження**: підсистема планування процесів (CPU scheduling) у мультизадачній операційній системі. **Мета статті**: розробити та обґрунтувати методику оптимізації й налаштування алгоритмів планування на основі імітаційного (дискретно-подійного) моделювання та сформувати практичні рекомендації для різних профілів задач. **Результати дослідження**. Побудовано параметризовану DES-модель однопроцесорної системи з потоком надходження процесів, ready-чергою, I/O-блокуваннями та накладними витратами контекстних переключень. Реалізовано й порівняно FCFS, SJF, Round Robin, пріоритетне планування, MLQ, MLFQ, модель справедливого планування, подібну до CFS, та адаптивний RR. Експериментально оцінено час очікування і відповіді, пропускну здатність, завантаження CPU, частку накладних витрат CS та справедливість (індекс Джейна) у чотирьох сценаріях навантаження: інтерактивному, CPU-heavy, bursty та змішаному за пріоритетами. **Висновки.** Невитіснювальні стратегії демонструють конкурентні середні значення лише в окремих режимах, тоді як витіснювальні та адаптивні підходи краще відповідають вимогам змішаних навантажень, але потребують акуратного тюнінгу кванта часу, гранулярностей і механізмів протидії голодуванню. Симуляційний підхід забезпечує відтворюване порівняння та виявлення параметричних зон, що ведуть до небажаних режимів. Сфера використання отриманих результатів: вибір і налаштування політик планування в ОС загального призначення, навчальні симулятори та інструменти аналізу продуктивності.

Ключові слова: планування процесів, операційні системи, дискретно-подійне моделювання, CFS, MLFQ, Round Robin.

Вступ

Планування процесів (CPU scheduling) є базовим механізмом мультизадачної операційної системи: за наявності кількох готових до виконання задач планувальник визначає порядок і тривалість їх виконання, балансуючи між часом відповіді, пропускною здатністю, завантаженням CPU і справедливістю [1]. Актуальність оптимізації планування посилюється різнорідністю сучасних навантажень (інтерактивні програми, серверні задачі, фонові batch-обчислення), для яких один «універсальний» критерій оптимальності є недостатнім: покращення відгуку може погіршувати throughput; зменшення середнього очікування може створювати голодування довгих задач тощо.

Стандартизація інтерфейсів планування визначає мінімальні «контрактні» механізми. POSIX.1 описує політики, зокрема SCHED_FIFO та SCHED_RR, а також відповідні механізми керування параметрами планування [2].

Паралельно практична реалізація планувальників у ядрах загального призначення еволюціонує: документація Linux детально описує CFS як «fair» підхід, а також EEVDF як наступний етап із вибором задачі за «віртуальним дедлайном» серед «eligible» задач [3].

Велика частина оптимізації планувальника – це оптимізація параметрів і евристик (кванти, гранулярності, правила підвищення пріоритетів/boost, класи задач), що складно робити «аналітично» для реалістичних навантажень. Тому імітаційне моделювання є доцільним інструментом: воно дозволяє повторювано порівнювати алгоритми у контрольованому середовищі, а також досліджувати чутливість метрик до параметрів [4].

Огляд останніх досліджень і публікацій. POSIX-орієнтовані джерела визначають базові політики й інтерфейси. Згідно з POSIX.1-2017 у <sched.h> присутні принаймні SCHED_FIFO (FIFO для фіксованих пріоритетів) та SCHED_RR (round-robin для фіксованих пріоритетів), що формує основу для переносимих застосунків (особливо в real-time контексті) [2].

У Linux практичні деталі політик, класів і впливу nice описує sched(7): nice впливає на SCHED_OTHER/SCHED_NORMAL та споріднені політики; ступінь впливу може змінюватися між версіями, але в «поточній реалізації» наводиться коефіцієнт близько 1.25 на одиницю різниці nice [5].

Офіційна документація Linux описує CFS як планувальник «desktop/general-purpose» класу fair, що прагне моделювати «ідеальний багатозадачний CPU» і використовує поняття віртуального часу/справедливого розподілу. Історично його пов'язують із роботою розробника Ingo Molnar [6] (merge у Linux 2.6.23). Далі у документації Linux виділено EEVDF Scheduler, який обирає задачу з множини «eligible» (за лагом) і використовує «virtual deadline» як критерій вибору найближчого кандидата до виконання. Опис EEVDF на kernel.org доповнюється технічними оглядами, зокрема матеріалами Peter Zijlstra у спільноті та статтями LWN.net щодо доробки EEVDF [7].

У період 2015–2026 рр. виділяються кілька напрямів, релевантних цій темі. Перший – дослідження «реальних» проблем і ефектів планувальника в популярних ядрах. Робота "The Linux Scheduler: a Decade of Wasted Cores" (EuroSys 2016) показала, що дефекти/евристики та балансування можуть призводити до недовикористання ядер навіть за наявності роботи, і підкреслила складність підтримки планувальника у

довгостроковій еволюції. Серед авторів – Jean-Pierre Lozi та Alexandra Fedorova [8]. Другий – розвиток адаптивних/гібридних алгоритмів (модифікації RR, MLFQ, змішані правила). Наприклад, у статті ORR: Optimized Round Robin CPU Scheduling Algorithm (ACM, 2021) запропоновано оптимізацію RR з акцентом на зменшення середнього очікування/turnaround і кількості контекстних переключень для time-sharing середовищ [6]. У контексті MLFQ важливо, що навіть «класичний» MLFQ може отримувати сучасні модифікації для спеціальних вимог: зокрема, MLFQ-RT для multi-core real-time задач досліджено через моделювання (ACM SE 2018) [9]. Третій – систематизація і порівняльне навчально-експериментальне оцінювання. У відкритій статті Heliyon (2024) “Study and evaluation of CPU scheduling algorithms” автори використовують симулятор для порівняння алгоритмів у контрольованому середовищі та акцентують на «неупередженості» умов порівняння [10]. Четвертий – методологія імітаційного моделювання і перевірка/валідація моделей. Класичну методологію DES і експериментального дизайну систематизує підручник Law A. M. “Simulation Modeling and Analysis” [11]. Питання V&V (verification & validation) системно розглянуте в роботах Robert G. Sargent (зокрема, матеріали Winter Simulation Conference; пізніші “advanced tutorial”) [12].

Таким чином, сучасний стан теми визначається двома «полюсами»: еволюцією реальних планувальників (CFS–EEVDF у Linux) та активністю досліджень гібридних/адаптивних алгоритмів і методів їх оцінювання через симуляцію.

Постановка задачі

Метою дослідження є розробка та обґрунтування методики оптимізації алгоритмів планування процесів в операційних системах на основі імітаційного моделювання та надання експериментально підтверджених рекомендацій щодо вибору / налаштування алгоритмів під різні профілі навантаження.

Об’єктом дослідження є підсистема планування процесів (CPU scheduling) у мультизадачній ОС; у термінах POSIX – політики планування та їх поведінкові властивості (FIFO/RR для фіксованих пріоритетів тощо). Предмет дослідження: порівняльні характеристики алгоритмів (FCFS, SJF, RR, Priority, Multilevel, MLFQ, CFS-подібний, адаптивний RR) і чутливість показників до параметрів (квант часу, правила підвищення пріоритетів, кількість черг, гранулярність/латентність тощо).

При цьому розглядаються такі задачі (RQs):

1) оцінити вплив вибору алгоритму планування на час очікування, час відповіді, пропускну здатність, завантаження CPU, справедливість, витрати контекстних переключень у різних сценаріях навантаження;

2) визначити параметричні «зони» (кванти, boost-інтервали), де алгоритм переходить у небажаний режим (надмірні перемикання, голодування);

3) сформувати практичні рекомендації щодо оптимізації й адаптації алгоритмів (евристики класифікації задач, адаптивний квант, aging) [13].

Невизначені вхідні дані: цільова ОС/версія ядра, апаратна платформа, робочі трасування задач, типові значення витрат перемикання – не вказано. Для компенсації обрано параметризовану модель, де суттєві параметри є явними й можуть бути замінені на виміряні значення в майбутньому.

Імітаційне моделювання реалізовано як дискретно-подійну (DES) модель: час рухається стрибками між моментами подій (надходження процесу, завершення CPU-бурсту, завершення I/O, закінчення кванта/таймслайсу, контекстне переключення). Такий підхід є стандартним для систем черг / планування й добре узгоджується з практикою використання бібліотек на кшталт SimPy (процеси, середовище, події) [4].

Загальні принципи побудови DES-моделі, вибору рівня деталізації, планування експериментів і аналізу виходів відповідають канонічній методології: формулювання проблеми – концептуальна модель – реалізація – верифікація – валідація – дизайн експериментів – аналіз результатів [11].

Середовище реалізації моделі: Python (custom DES); альтернативно може бути використано SimPy як стандартний DES-фреймворк [14].

Абстракція системи:

– один CPU (одне ядро) – спрощення для виділення «чистого» ефекту алгоритму CPU scheduling. Мультипроцесорні ефекти (load balancing, NUMA, конкуренція кешів) винесені як перспектива. Актуальність мультиядерного аспекту підтверджується практичними кейсами (EuroSys 2016) і сучасними розширеннями MLFQ для multi-core [8];

– процес складається з послідовності CPU-бурстів із I/O паузами (блокування);

– контекстне переключення має накладну вартість cs_overhead (у моделі – 0.1 мс; значення параметризується і в реальному дослідженні має бути замінене вимірюванням для цільової системи).

Невизначені параметри (позначено “не вказано”): апаратні характеристики, тип дискової/мережевої підсистеми, реальна політика I/O планування, таймерні гранулярності ОС, пріоритетні діапазони та політики в цільовій ОС.

Використано такі метрики (у дужках – позначення):

– час очікування у ready-черзі (W): сумарний час, коли процес готовий до виконання, але не виконується;

– час відповіді (R): від надходження процесу до першого отримання CPU (критично для інтерактивних задач);

– пропускна здатність (X): кількість завершених процесів за одиницю часу (процесів/с);

– завантаження CPU (U): частка часу, коли CPU виконує корисну роботу (user CPU), без накладних витрат перемикання;

– витрати контекстних переключень (CS%): частка часу, витрачена на контекстні переключення, відносно загального часу моделювання;

– справедливість (J): застосовано індекс Джейна (Jain’s fairness index) над похідними величинами, що характеризують рівномірність «ефективності

обслуговування». Індекс лежить у діапазоні [0;1], 1 означає максимальну рівномірність [15].

Індекс справедливості Джейна обчислюється як

$$J = \left(\sum_{i=1}^{n} x_i \right)^2 \Bigg/ \left(n \sum_{i=1}^{n} x_i^2 \right),$$

де n – кількість процесів у вибірці, x_i – значення показника для i-го процесу; $J \in [0;1]$.

У роботі використано дві інтерпретації показника справедливості: fairness_ready_J – Jain-індекс над величинами

$$x_i = \frac{T_{CPU,i}}{T_{CPU,i} + W_{ready,i}},$$

де $T_{CPU,i}$ – сумарний час виконання процесу на CPU, а $W_{ready,i}$ – сумарний час очікування в ready-черзі (I/O-паузи не враховуються); fairness_batch_J – аналогічний індекс, але обчислений лише для підмножини CPU-bound (batch) процесів, щоб відокремити ефект пріоритезації інтерактивних задач.

Верифікація виконувалась як перевірка коректності реалізації подій і інваріантів (невід'ємність залишку CPU-бурсту, монотонність часу, завершення кожного процесу, збереження балансу подій).

Валідація у повному сенсі потребує зіставлення з реальними трасами ОС/бенчмарками – не вказано (відсутні дані). Тому результати інтерпретуються як модельні та слугують для порівняльного аналізу і чутливості до параметрів.

Визначено 4 сценарії (кожен – $N = 120$ процесів, 6 повторів, різні seed):

– сценарій A (інтерактивний): 70% I/O-bound/інтерактивних процесів із багатьма короткими CPU-бурстами та I/O-паузами; 30% CPU-bound. Інтервали приходу – експоненційні (потік Пуассона);

– сценарій B (CPU-heavy): домінування CPU-bound процесів (80%), вищий тиск на CPU;

– сценарій C (burstiness): пачкові надходження (чергування високої/низької інтенсивності) та важкохвості розподіли CPU-бурстів для підвищення дисперсії;

– сценарій D (priority-mixed): змішаний профіль із часткою «system» задач (вищі пріоритети), що дозволяє оцінити ризики голодування і переваги boosting/aging.

Алгоритми реалізовано в моделі як модулі вибору наступного процесу та правила витіснення:

– FCFS (First-Come, First-Served): невитіснювальний; черга FIFO. Типовий недолік – «convoy effect» при змішуванні довгих CPU-bound і коротких I/O-bound задач [1];

– SJF (Shortest Job First): невитіснювальний; вибір за мінімальною довжиною наступного CPU-бурсту. У реальній ОС довжина бурсту невідома наперед і оцінюється, але в моделі для «upper bound» використано фактичне значення; ризик голодування довгих задач є очікуваним;

– Round Robin (RR): витіснювальний; квант q;

– Priority (PRIO): витіснювальний пріоритетний; при появі задачі з вищим пріоритетом виконується негайне витіснення (у межах моделі). Ризики голодування нижчих пріоритетів потребують aging (у цій базовій конфігурації – спрощено, без повної динамічної зміни) [1];

– Multilevel Queue (MLQ): окремі черги для класів system/interactive/batch із жорстким пріоритетом черг (верхні завжди витісняють нижні);

– Multilevel Feedback Queue (MLFQ): багаторівнева черга з «зворотнім зв'язком»: процеси стартують на верхньому рівні з малим квантом; якщо вичерпують квант – понижуються, якщо блокуються (I/O) – зберігають/підвищують пріоритет; періодично застосовується boost для протидії голодуванню. Це класична відповідь на «як планувати без знання майбутнього», використовуючи історію виконання [13];

– CFS-подібний (CFS-model): спрощена модель «fair» планування (віртуальний час vruntime, вибір мінімального vruntime, таймслайс як функція target_latency і min_granularity). У Linux ці поняття мають документовані аналоги (CFS design) [3];

– ARR (Adaptive Round Robin): адаптивний квант з обрізанням у межах [2;20] мс. Ідея узгоджується з напрямом оптимізації RR через «розумний квант», який активно досліджується у 2015–2026 (напр., ORR) [6].

Параметри експериментального моделювання наведено у табл. 1, а результати моделювання для кожного алгоритму та сценарію – у табл. 2.

Таблиця 1 – **Параметри експериментального моделювання**

Параметр	Значення (у моделі)	Коментар
Кількість процесів у сценарії	120	Фіксовано для порівнянності
Кількість повторів (реплікацій)	6	Різні seed для усереднення
Модель CPU	1 ядро (1 сервер)	Multi-core – перспектива
Вартість контекстного переключення `cs_overhead`	0.1 мс	Параметр; у реальній системі – не вказано
RR квант `q`	4 мс	Типове значення для демонстрації чутливості; «оптимальний» залежить від навантаження
MLQ квант (system/interactive)	3 мс	`batch` у MLQ – FCFS
MLFQ рівні/кванти	3 рівні: [2, 4, 8] мс	Демонстраційний набір; підлягає оптимізації
MLFQ boost interval	50 мс	Протидія голодуванню
CFS-model `target_latency`	24 мс	Концептуальний аналог `sched_latency`
CFS-model `min_granularity`	3 мс	Концептуальний аналог `sched_min_granularity`
Діапазон ARR квантів	2…20 мс	Запобігання крайнім режимам

Таблиця 2 – **Результати моделювання**

Алгоритм	Сер. очікування, мс	Сер. відповідь, мс	p95 очікування, мс	p95 відповідь, мс	Пропускна здатн., проц/с	Завантаження CPU, %	Накладні CS, %	Перемикань контексту	J_ready	J_batch
Сценарій A (інтерактивний)										
FCFS	3389.3	922.2	4672.1	1808.6	18.12	85.4	1.84	1229	0.126	0.320
SJF	1419.2	870.4	4241.9	3776.0	19.24	97.6	1.96	1228	0.342	0.228
RR	2597.6	127.7	4825.0	252.9	18.78	91.2	3.53	2268	0.373	0.539
ARR	2498.8	96.4	4623.7	180.6	18.33	89.1	3.49	2284	0.497	0.619
PRIO	1522.0	1077.9	4833.7	4406.8	20.08	95.5	4.16	2500	0.320	0.518
MLQ	1624.5	422.9	4990.0	1703.1	20.25	97.2	2.36	1399	0.334	0.534
MLFQ	933.2	23.1	2231.1	71.5	19.26	81.5	5.36	1844	0.420	0.542
CFS	2111.9	2.0	4953.7	5.0	19.96	95.3	4.17	2521	0.217	0.822
Сценарій B (CPU-bound домінує)										
FCFS	6602.9	2784.2	9807.8	5323.3	10.90	94.5	0.44	484	0.150	0.219
SJF	2904.7	1980.6	7859.0	6847.3	12.00	99.5	0.47	472	0.253	0.297
RR	5552.4	213.0	9125.4	403.2	11.66	96.9	2.71	2790	0.607	0.743
ARR	6190.6	920.6	9368.7	1715.2	11.36	96.4	0.89	945	0.410	0.567
PRIO	4309.4	3312.0	9730.5	8589.9	11.27	97.1	2.86	3063	0.379	0.471
MLQ	5092.5	482.7	9279.0	1831.9	12.35	99.3	0.68	662	0.291	0.406
MLFQ	3784.6	64.1	7306.1	160.3	11.83	94.4	4.75	3993	0.524	0.635
CFS	5529.7	3.4	9899.9	9.2	10.86	96.6	3.38	3746	0.373	0.753
Сценарій C (bursty/важкохвості CPU-бурсти)										
FCFS	9305.2	1591.2	16811.9	5478.0	11.09	98.6	0.12	161	0.101	0.139
SJF	4338.0	1624.6	10264.5	5659.6	11.87	99.7	0.07	96	0.181	0.234
RR	8393.2	269.8	16562.8	526.6	10.31	93.9	2.56	3415	0.404	0.477
ARR	8702.2	726.1	17007.8	1442.5	10.67	98.1	0.78	1035	0.329	0.405
PRIO	6806.0	2964.5	16646.7	11843.6	11.12	97.9	1.68	2345	0.262	0.356
MLQ	8618.8	456.6	16883.1	1802.9	11.34	99.4	0.26	352	0.186	0.231
MLFQ	5639.4	74.5	12265.6	197.3	12.10	92.0	4.94	7116	0.493	0.601
CFS	7909.9	3.0	17408.1	9.1	11.36	98.0	1.91	2609	0.229	0.670
Сценарій D (змішаний за пріоритетами)										
FCFS	4246.4	1505.1	8053.3	5113.3	13.05	96.6	0.58	368	0.146	0.277
SJF	3310.6	1882.2	6976.7	6260.4	13.69	99.2	0.42	278	0.227	0.263
RR	4269.8	212.6	7936.4	391.0	12.71	95.6	2.44	1681	0.442	0.545
ARR	4449.5	517.2	8348.2	992.2	13.44	99.1	0.60	403	0.332	0.471
PRIO	4274.8	1865.6	8162.5	7865.4	12.94	97.2	1.81	1189	0.243	0.352
MLQ	5071.9	395.7	10146.5	2098.0	12.30	94.8	0.74	499	0.221	0.293
MLFQ	2953.4	48.0	5784.4	123.5	13.28	94.3	4.40	3065	0.499	0.640
CFS	4510.6	1.3	9137.4	3.9	12.83	97.8	2.11	1400	0.223	0.812

Аналіз результатів моделювання

FCFS у змішаних сценаріях демонструє високий час відповіді та/або великі хвости розподілу очікування (p95), що пояснюється «convoy effect» – короткі інтерактивні задачі «прилипають» за довгими CPU-bound у FIFO. SJF у CPU-heavy та bursty сценаріях мінімізує середній час очікування (B, C) – що відповідає класичному твердженню про оптимальність SJF для середнього очікування за припущення знання довжин бурстів. Однак водночас час відповіді та p95 можуть погіршуватись, що відображає потенційну дискримінацію довгих задач. RR забезпечує низький (відносно FCFS/SJF) час відповіді, але різко збільшує кількість контекстних переключень і частку накладних витрат у важких навантаженнях (B, C), що є відомим недоліком RR при малому кванті. Мотив саме такого «тюнінгу кванта» віддзеркалюється в сучасних модифікаціях RR (напр., ORR), де оптимізація спрямована на зниження NCS/очікування/turnaround. MLQ (жорсткі черги) у priority-mixed сценаріях може давати непоганий «сервіс» верхніх класів, але нижні класи у принципі ризикують голодуванням без спеціальних правил. Це загальна проблема пріоритетних і багаторівневих схем, що зазвичай компенсується aging/boost. MLFQ у сценаріях A і D демонструє найкращі/одні з найкращих показників часу відповіді та очікування, що відповідає ідеї MLFQ: наближати інтуїцію SJF без знання майбутнього, використовуючи історію (короткі/інтерактивні задачі залишаються нагорі). CFS-модель у цій постановці дає мінімальний середній час відповіді, що узгоджується з філософією fair-планування і пріоритезацією latency-sensitive поведінки. Водночас «справедливість» у глобальному сенсі залежить від того, що саме вимірюється: для batch-класу fairness_batch_J у сценаріях A/D є високою (≈0.82), але fairness_ready_J може бути нижчою через свідоме «підтягування» інтерактивних/часто-блокуючих задач (що в Linux historically реалізовано через механізми vruntime/latency). Сучасний розвиток EEVDF у Linux демонструє, що навіть «справедливі» планувальники уточнюють критерій вибору

процесу (eligible + virtual deadline), щоб комбінувати справедливість і латентність.

Рекомендації щодо оптимізації

Рекомендації поділено на параметричні налаштування, адаптивні стратегії та евристики.

Параметричні налаштування RR/MLFQ. Для RR ключовим є квант часу: надто малий квант збільшує накладні витрати (CS%), надто великий погіршує реактивність інтерактивних задач. Практично доцільно підбирати квант від медіанного або близького до медіанного CPU-бурсту інтерактивного класу й перевіряти чутливість метрик у симуляції; подібна логіка властива і сучасним оптимізаціям RR (ORR та ін.).

Для MLFQ параметри – кількість рівнів, кванти рівнів, інтервал boost – є основними регуляторами компромісів: зменшення квантів верхніх рівнів покращує відгук, але збільшує перемикання; надто рідкісний boost підвищує ризик starvation нижніх рівнів. Класична постановка MLFQ прямо підкреслює роль boost у протидії голодуванню.

Адаптивні стратегії. Доцільним є динамічний квант (ARR-подібні) як мінімальна адаптація RR: якщо квант прив'язати до статистики бурстів (медіана/квантіль), можна зменшити зайві переключення в CPU-heavy режимах та утримати прийнятний відгук у інтерактивних. Це узгоджується з емпіричною мотивацією модифікацій RR у літературі. Для пріоритетних і MLQ-схем потрібні aging/boost правила, інакше виникає системний ризик голодування; навчальні матеріали з CPU scheduling прямо вказують на aging як стандартну протидію starvation.

Налаштування fair-планування (CFS/EEVDF-подібні). Практична оптимізація часто зводиться до контролю «латентності» та гранулярностей (аналогічно tunables, відомим у Linux екосистемі). Офіційні матеріали описують сенс параметрів на кшталт sched_min_granularity (мінімальний період виконання задачі) та загальної «латентності» як концептуального бюджету розподілу між runnable задачами.

Під час перенесення на реальну ОС слід враховувати, що доступність/статус tunables може змінюватися між версіями (наприклад, переміщення в debugfs у деяких дистрибутивах/версіях), тому оптимізація має бути прив'язана до конкретної платформи.

Припущення моделі:

– один CPU; відсутні multi-core ефекти (балансування, NUMA, конкуренція кешів). Актуальні дослідження показують, що саме multi-core аспекти можуть домінувати у продуктивності планувальника в реальних системах;

– вартість контекстного переключення стала 0.1 мс. У реальності вона залежить від архітектури, TLB/кеш ефектів, політики ядра;

– для SJF припущено знання довжини майбутнього бурсту (це upper bound); реальна ОС оцінює бурст за історією;

– CFS реалізовано як спрощену модель; EEVDF у точній формі не змодельовано (потребує відтворення eligibility/lag/deadline механізмів на рівні, близькому до документації).

Відкриті питання для продовження:

– повна симуляція EEVDF і порівняння з CFS-подібною моделлю за однакових параметрів (особливо хвости p95/p99);

– підтвердження моделі на реальних трасах (perf/sched trace) для цільової ОС/ядра;

– розширення моделі на multi-core із балансуванням і affinity; співставлення з відомими проблемами/патчами продуктивності планувальника (EuroSys 2016);

– оцінка впливу політик реального часу (POSIX SCHED_FIFO/SCHED_RR) і змішаних класів із fair-плануванням у Linux.

Висновки

У роботі запропоновано строгий підхід до оптимізації планування процесів через дискретно-подійне моделювання: алгоритми порівнюються в контрольованих умовах за узгодженими метриками, включно з часом очікування, часом відповіді, пропускною здатністю, завантаженням CPU, справедливістю та накладними витратами контекстних переключень.

Методологія узгоджена з базовими принципами DES та рекомендаціями щодо V&V симуляційних моделей. Експерименти на чотирьох сценаріях показують, що:

– невитіснювальні алгоритми (FCFS, SJF) можуть бути конкурентними за середніми метриками лише у частині режимів, але часто програють за відгуком і хвостами розподілу;

– витіснювальні та адаптивні стратегії (RR/ARR/MLFQ/CFS-model) краще відповідають вимогам сучасних змішаних навантажень, але потребують акуратного тюнінгу для контролю накладних витрат;

– фактична еволюція Linux (CFS – EEVDF) підтверджує, що компроміс «справедливість–латентність» є центральною проблемою і потребує експериментально підтверджуваних рішень на реальних профілях.

Перспективи подальших досліджень:

1) включення точнішої моделі EEVDF і механізмів wakeup/preemption;

2) multi-core/NUMA розширення;

3) калібрування параметрів моделі за трасами реальної системи;

4) автоматизований пошук оптимальних параметрів (кванти, boost, гранулярності) через планування експериментів та оптимізацію на виходах симуляції (в дусі сучасних підходів до симуляційного дизайну).

Конфлікт інтересів. Автори декларують, що не мають конфлікту інтересів стосовно даного дослідження, в тому числі фінансового, особистісного характеру, авторства чи іншого характеру, що міг би вплинути на дослідження та його результати, представлені в даній статті.

Використання засобів штучного інтелекту. Автори підтверджують, що не використовували технології штучного інтелекту при створенні представленої роботи.

СПИСОК ЛІТЕРАТУРИ

1. Arpaci-Dusseau R. H., Arpaci-Dusseau A. C. Operating Systems: Three Easy Pieces. CPU Scheduling (chapter). University of Wisconsin–Madison. URL: https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched.pdf
2. IEEE. Portable Operating System Interface (POSIX) Base Specifications, Issue 7 (IEEE Std 1003.1-2017). DOI: https://doi.org/10.1109/IEEESTD.2018.8277153
3. Linux Kernel Documentation. CFS scheduler design. URL: https://docs.kernel.org/scheduler/sched-design-CFS.html
4. Zinoviev D. Discrete Event Simulation: It's Easy with SimPy! arXiv:2405.01562, 2024. DOI: https://doi.org/10.48550/arXiv.2405.01562.
5. Kerrisk M. sched (7) – Linux manual page. URL: https://man7.org/linux/man-pages/man7/sched.7.html
6. Lozi J. P., et al. The Linux scheduler. EuroSys 2016. DOI: https://doi.org/10.1145/2901318.2901326.
7. Linux Kernel Documentation. EEVDF scheduler. URL: https://docs.kernel.org/scheduler/sched-eevdf.html
8. Rasouli A., et al. MLFQ-RT: A multi-level feedback queue real-time scheduler for embedded systems. CODES+ISSS 2014. DOI: https://doi.org/10.1145/2656075.2656108.
9. Chen X. Calendar queue: a new real-time calendar-based priority queue. Journal of Parallel and Distributed Computing, 1992. DOI: https://doi.org/10.1016/0743-7315(92)90068-A.
10. Simpson K., et al. Simulation-based experimentation on scheduling algorithms. Heliyon, 2022. DOI: https://doi.org/10.1016/j.heliyon.2022.e10507.
11. Law A. M., Kelton W. D. Simulation Modeling and Analysis. 5th ed. McGraw-Hill Education, 2015. ISBN: 978-0073401324. URL: https://search.worldcat.org/de/title/simulation-modeling-and-analysis/oclc/1022581268
12. Sargent R. G. Verification and validation of simulation models. In: Proceedings of the 2010 Winter Simulation Conference (WSC), 2010, pp. 166–183. DOI: https://doi.org/10.1109/WSC.2010.5679166.
13. Arpaci-Dusseau R. H., Arpaci-Dusseau A. C. Operating Systems: Three Easy Pieces. Scheduling: The Multi-Level Feedback Queue (chapter). URL: https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-sched-mlfq.pdf
14. SimPy Documentation. Basic Concepts. Available: https://simpy.readthedocs.io/en/latest/simpy_intro/basic_concepts.html
15. Jain R., Chiu D.-M., Hawe W. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. arXiv:cs/9809099. DOI: https://doi.org/10.48550/arXiv.cs/9809099

Received (Надійшла) 18.11.2025
Accepted for publication (Прийнята до друку) 21.01.2026
Publication date (Дата публікації) 27.02.2026

ВІДОМОСТІ ПРО АВТОРІВ / ABOUT THE AUTHORS

Запорожець Олег Васильович – кандидат технічних наук, доцент, доцент кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Oleg Zaporozhets – PhD, Associate Professor, Associate Professor of Department of Electronic Computers, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine;
e-mail: oleg.zaporozhets@nure.ua; ORCID Author ID: http://orcid.org/0000-0002-7831-8479;
Scopus Author ID: https://www.scopus.com/authid/detail.uri?authorId=15728942500.

Макаренко Владислав Сергійович – студент кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Vladyslav Makarenko – student, Department of Electronic Computers, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine;
e-mail: vladyslav.makarenko@nure.ua; ORCID Author ID: https://orcid.org/0009-0002-9891-3627.

Дрозд Олег Юрійович – аспірант кафедри електронних обчислювальних машин, Харківський національний університет радіоелектроніки, Харків, Україна;
Oleh Drozd – PhD student, Department of Electronic Computers, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine;
e-mail: oleh.drozd@nure.ua; ORCID Author ID: http://orcid.org/0009-0007-4285-4505.

Optimization of process planning algorithms in operating systems using simulation modeling

Oleg Zaporozhets, Vladyslav Makarenko, Oleh Drozd

Abstract. Relevance. Optimizing process scheduling is critical for modern operating systems due to heterogeneous workload profiles, where the trade-off between latency, throughput, and fairness cannot be reliably assessed through analytical methods alone. **Object of research**: the CPU scheduling subsystem in a multitasking operating system. **Purpose of the article.** to develop and justify a methodology for optimizing and tuning scheduling algorithms based on simulation (discrete-event) modeling, and to formulate practical recommendations for different task profiles. **Research results.** A parameterized DES model of a single-processor system was built, including a process arrival stream, a ready queue, I/O blocking, and context-switch overhead. The following were implemented and compared: FCFS, SJF, Round Robin, priority scheduling, MLQ, MLFQ, a fair-scheduling model similar to CFS, and adaptive RR. Waiting time and response time, throughput, CPU utilization, the share of context-switch overhead, and fairness (Jain's index) were experimentally evaluated across four workload scenarios: interactive, CPU-heavy, bursty, and mixed by priorities. **Conclusions.** Non-preemptive strategies show competitive average values only in certain modes, whereas preemptive and adaptive approaches better meet the requirements of mixed workloads but require careful tuning of the time quantum, granularities, and anti-starvation mechanisms. The simulation approach enables reproducible comparisons and the identification of parameter regions that lead to undesirable operating regimes. Application area of the results: selection and tuning of scheduling policies in general-purpose operating systems, educational simulators, and performance analysis tools.

Keywords: CPU scheduling, operating systems, discrete-event simulation, CFS, MLFQ, Round Robin.