Electron: Jurnal Ilmiah Teknik Elektro
p-ISSN 2830–523X | e-ISSN 2622–6588
Volume 7 Nomor 1 Mei 2026

Simulasi Komparatif: Empat Algoritma Penjadwalan CPU dengan Pemodelan Interupsi I/O Menggunakan OS-SIM

Comparative Simulation: Four CPU Scheduling Algorithms with I/O Interruption Modeling Using OS-SIM

Mufid Athooyaa[1], Mirza Putra Firmansyah[2], Djuniadi Djuniadi[3], Alfian Ardhiansyah[4]

[1,2,3,4] Jurusan Teknik Elektro, Fakultas Teknik, Universitas Negeri Semarang; email: [1]mufidathooyaa13@students.unnes.ac.id, [2]mirzafirmansyah66@students.unnes.ac.id, [3]djuniadi@mail.unnes.ac.id, [4]alfianardhiansyah@mail.unnes.ac.id

[Dikirimkan: 1 Januari 2026, Direvisi: 13 Maret 2026, Diterima: 31 Mei 2026]
Corresponding Author: Mufid Athooyaa

INTISARI — Penjadwalan CPU merupakan komponen fundamental dalam sistem operasi yang menentukan efisiensi pemanfaatan prosesor dan responsivitas sistem. Tujuan penelitian ini adalah melakukan perbandingan kinerja empat algoritma klasik penjadwalan CPU, yaitu First Come First Served (FCFS) multiprogramming, Preemptive Shortest Job First (SRTF), Preemptive Priority, dan Median Round Robin dengan mempertimbangkan pengaruh interupsi I/O. Metode penelitian menggunakan simulasi eksperimental melalui OS-SIM dengan sampel uji 12 proses yang mencakup karakteristik CPU bound dan I/O bound. Evaluasi kinerja dilakukan berdasarkan parameter Average Waiting Time (AWT), Average Turnaround Time (ATT), Average Response Time (ART), efisiensi CPU, dan throughput. Hasil simulasi menunjukkan bahwa algoritma Preemptive SJF (SRTF) menghasilkan kinerja paling efisien dengan AWT sebesar 15.5 *s* dan ATT sebesar 20.5 *s*, lebih rendah dibandingkan algoritma lainnya. FCFS dan Median Round Robin unggul dalam memberikan respons awal yang cepat dengan ART masing-masing 10.58 *s* dan 10.42 *s*, namun menghasilkan waktu tunggu yang lebih tinggi. Preemptive Priority berada pada posisi menengah dengan AWT 18 *s* dan ATT 23 *s*, tetapi berpotensi menyebabkan starvation pada proses prioritas rendah. Kebaruan penelitian ini terletak pada pemodelan interupsi I/O secara eksplisit dalam lingkungan multiprogramming, suatu aspek yang belum dipertimbangkan secara sistematis dalam studi komparatif sebelumnya. Penelitian ini memberikan kontribusi berupa panduan pemilihan algoritma penjadwalan yang optimal berdasarkan karakteristik beban kerja sistem, di mana SRTF direkomendasikan untuk skenario batch processing, sementara FCFS dan MRR lebih sesuai untuk lingkungan interaktif yang mengutamakan responsivitas.

ABSTRACT — *CPU scheduling is a fundamental component in operating systems that determines processor utilization efficiency and system responsiveness. The objective of this research is to comparatively evaluate four classic CPU scheduling algorithms, namely First Come First Served (FCFS) multiprogramming, Preemptive Shortest Job First (SRTF), Preemptive Priority, and Median Round Robin, considering the effects of I/O interruption. The research method employs experimental simulation through OS-SIM with 12 test sample processes covering CPU bound and I/O bound characteristics. Performance evaluation is conducted based on Average Waiting Time (AWT), Average Turnaround Time (ATT), Average Response Time (ART), CPU efficiency, and throughput parameters. Simulation results demonstrate that the Preemptive SJF (SRTF) algorithm produces the most efficient performance with AWT of 15.5 s and ATT of 20.5 s, lower than other algorithms. FCFS and Median Round Robin excel in providing quick initial response with ART of 10.58 s and 10.42 s respectively, but result in higher waiting times. Preemptive Priority stands in the middle position with AWT of 18 s and ATT of 23 s, but potentially causes starvation for low-priority processes. The novelty of this study lies in the explicit modeling of I/O interruption within a multiprogramming environment, an aspect that has not been systematically addressed in previous comparative studies. This research contributes a guideline for selecting optimal CPU scheduling algorithms based on system workload characteristics, recommending SRTF for batch processing scenarios, while FCFS and MRR are more suitable for interactive environments that prioritize responsiveness.*

KATA KUNCI — Penjadwalan CPU, Algoritma Penjadwalan, OS-SIM, I/O Interruption, Multiprogramming, Studi Komparatif, CPU Burst Model, Metrik Kinerja Sistem.

I. PENDAHULUAN

Penjadwalan CPU merupakan salah satu bagian dasar yang sangat penting dalam sebuah sistem operasi. Penjadwalan ini berperan penting dalam meningkatkan efisiensi pemanfaatan prosesor, responsivitas sistem, serta keadilan dalam pembagian sumber daya [1]. Berbagai algoritma penjadwalan klasik seperti *First Come First Served* (FCFS), *Shortest Job First* (SJF), *Round Robin* (RR), dan *Priority* telah banyak digunakan dan dikaji untuk meningkatkan kinerja sistem berbasis komputasi

multiprogramming [2]. Umumnya, evaluasi kinerja algoritma klasik dilakukan menggunakan parameter *waiting time, response time, throughput*, serta jumlah *context switching* sebagai indikator performa sistem [3].

Sejumlah penelitian terdahulu berfokus pada optimalisasi dan perbandingan performa algoritma *scheduling* berdasarkan karakteristik CPU *burst* dan waktu kuantum. Penelitian mengenai optimasi RR melalui pendekatan statistik menunjukkan bahwa pemilihan kuantum berbasis nilai median mampu meningkatkan *throughput* dan menurunkan *waiting time* serta *context switching* secara signifikan [4]. Sementara itu, terdapat beberapa penelitian yang menggunakan *time quantum* (TQ) yang bervariasi di tiap iterasinya, seperti penggunaan *Median Average Round Robin* (MARR) dan *Average Max Round Robin* (AMRR), yang menghasilkan peningkatan performa penjadwalan terutama pada skenario beban kerja yang berbeda [5], [6]. Variasi nilai TQ yang dihitung secara dinamis pada setiap putaran terbukti mampu menyeimbangkan responsivitas sistem dan efisiensi pemrosesan, meskipun terdapat kompleksitas perhitungan serta *overhead* penjadwalan [7]. Selain itu, pengembangan metode *Advanced Round Robin Method* (ARRM) juga terbukti mampu meminimalkan *total waiting time* hingga 90% dibandingkan metode RR konvensional [8]. Selain itu, studi komparatif berbasis simulasi terhadap algoritma FCFS, SJF/SRTF, Priority, Round Robin, dan multilevel queue juga telah dilakukan dengan berbagai pendekatan, termasuk melalui simulasi berbasis skenario dan analisis pemilihan nilai quantum secara dinamis [9]–[11]. Namun, studi-studi tersebut pada umumnya belum mempertimbangkan pemodelan I/O burst secara eksplisit dalam skenario simulasinya

Studi lain menunjukkan bahwa algoritma *Preemptive SJF* atau *Short Remaining Time First* (SRTF) lebih efisien dibandingkan RR dalam menghasilkan nilai *Average Waiting Time* (AWT) atau rerata waktu tunggu dan *Average Turnaround Time* (ATT) atau rerata waktu penyelesaian yang lebih rendah [1], [12], [13]. Di sisi lain, algoritma FCFS mengeksekusi proses berdasarkan urutan kedatangan tanpa mekanisme *preemption*. Hasil studi menunjukkan bahwa FCFS memiliki keunggulan dalam kesederhanaan implementasi, tetapi cenderung menghasilkan AWT dan ATT yang tinggi ketika terdapat proses dengan *burst time* yang tinggi di awal antrian sehingga menimbulkan efek konvoi [14], [15]. Selanjutnya, simulasi terhadap algoritma *Preemptive Priority* dan *Priority Round Robin* menggunakan aplikasi simulator OS-SIM menunjukkan bahwa algoritma berbasis prioritas mampu memberikan respons yang cepat terhadap proses dengan tingkat prioritas tinggi, meskipun berpotensi menyebabkan peningkatan waktu tunggu pada proses dengan prioritas rendah, [16], [17]. Perbandingan kedua algoritma tersebut juga dilakukan melalui simulasi dengan bahasa C yang menunjukkan bahwa algoritma *Preemptive Priority* memiliki kegunaan yang sama. Namun, algoritma RR dinilai lebih sesuai untuk sistem yang menekankan aspek keadilan dan pemerataan waktu proses. Sementara itu, hasil AWT dan ATT menunjukkan bahwa *Preemptive Priority* memiliki kinerja yang lebih cepat dibandingkan RR, [18]. Selain itu, simulasi algoritma *Preemptive SJF* juga membuktikan bahwa proses dengan *burst time* lebih pendek dapat melakukan *preemption* terhadap proses yang lebih panjang sehingga mampu menurunkan waktu tunggu rata-rata secara signifikan [19], [20].

Namun demikian, sebagian besar penelitian tersebut masih menggunakan asumsi bahwa proses bersifat CPU *bound* murni, di mana proses selalu berada dalam kondisi *ready* dan tidak mempertimbangkan adanya interupsi akibat operasi *input/output* (I/O). Ketidakhadiran pemodelan I/O interruption dalam analisis penjadwalan menyebabkan hasil evaluasi algoritma menjadi kurang merepresentasikan kondisi sistem yang sesungguhnya. Penelitian [9] secara eksplisit mengonfirmasi bahwa dalam sebagian besar studi komparatif yang telah dipublikasikan, waktu I/O umumnya diabaikan dan hanya waktu komputasi CPU yang dipertimbangkan dalam perbandingan algoritma, sehingga hasil evaluasi tidak mencerminkan beban kerja sistem operasi secara nyata. Pada kenyataannya, sebagian besar proses dalam sistem operasi modern bersifat I/O-bound, sehingga proses dapat berpindah dari status *running* ke *blocked* ketika menunggu operasi I/O, kemudian kembali ke antrean siap (*ready queue*) setelah terjadi interupsi I/O [21]. Ketidakhadiran pemodelan *I/O interruption* dalam analisis scheduling menyebabkan hasil evaluasi algoritma menjadi kurang merepresentasikan kondisi sistem yang sesungguhnya.

Berdasarkan celah penelitian tersebut, studi ini bertujuan menganalisis perbandingan kinerja algoritma FCFS *multiprogramming*, *Preemptive* SJF atau SRTF, Median Round Robin, dan *Preemptive Priority* dengan mempertimbangkan I/O *interruption* secara eksplisit. Evaluasi dilakukan melalui simulasi sistem dengan memasukkan parameter CPU *burst* dan I/O *burst* secara bersamaan guna merepresentasikan lingkungan eksekusi yang lebih realistis. Hasil penelitian ini diharapkan dapat mendukung pengembangan model evaluasi algoritma penjadwalan yang lebih andal serta menjadi dasar pemilihan algoritma yang optimal pada sistem dengan beban CPU dan I/O.

Penelitian ini memiliki beberapa kontribusi ilmiah yang membedakannya dari studi studi sebelumnya. Pertama, penelitian ini secara eksplisit memodelkan interupsi I/O dalam simulasi penjadwalan CPU, suatu aspek yang kerap diabaikan dalam studi komparatif sebelumnya yang umumnya mengasumsikan proses bersifat CPU bound murni [12], [16], [19]. Kedua, penelitian ini mengevaluasi campuran proses CPU bound dan I/O bound secara bersamaan dalam lingkungan multiprogramming, sehingga representasi kondisi eksekusi lebih realistis. Ketiga, penelitian ini menggunakan Median Round Robin (MRR) sebagai salah satu varian RR yang lebih adaptif dibandingkan RR konvensional. Dengan demikian, penelitian ini berkontribusi dalam penyediaan model evaluasi kinerja algoritma penjadwalan yang lebih komprehensif, yang dapat dijadikan dasar rekomendasi pemilihan algoritma pada sistem dengan karakteristik beban kerja CPU–I/O campuran.

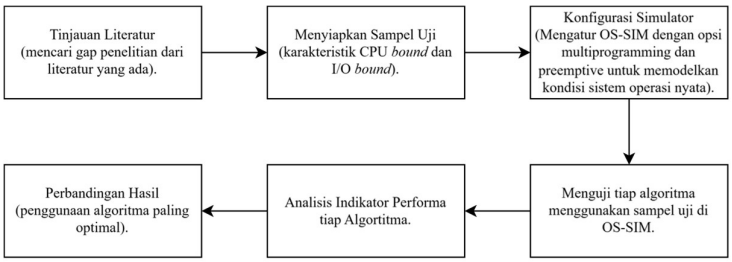
II. METODE PENELITIAN

Penelitian ini menggunakan pendekatan simulasi percobaan. Simulasi percobaan dilaksanakan dengan memanfaatkan simulator sistem operasi sebagai media untuk memodelkan dan menganalisis perilaku algoritma penjadwalan CPU. Saat ini, terdapat berbagai macam simulator operasi sistem yang telah dikembangkan, baik berbasis situs web maupun perangkat lunak [22], [23]. Simulator ini bertujuan mempermudah proses perhitungan simulasi serta analisis kinerja algoritma. Pada penelitian ini, peneliti menggunakan simulator OS-SIM yang dapat melakukan empat fitur simulasi, di antaranya (1) penjadwalan proses, (2) manajemen memori, (3) manajemen sistem berkas, dan (4) manajemen disk [16], [17], [19], [24]. Simulasi dilaksanakan menggunakan OS-SIM versi yang tersedia pada repositori resmi yang telah digunakan dalam beberapa studi sebelumnya [16], [17], [19], [24]. Untuk memastikan keakuratan hasil simulasi, nilai-nilai kinerja yang dihasilkan oleh OS-SIM diverifikasi secara parsial menggunakan perhitungan

manual pada beberapa proses representatif, dan diperoleh hasil yang konsisten. Penggunaan 12 proses sebagai sampel uji diakui sebagai keterbatasan penelitian dalam hal generalisabilitas, namun dianggap cukup untuk tujuan perbandingan algoritma dalam skenario multiprogramming yang terkontrol, sesuai dengan praktik pada studi sejenis [12], [16].

OS-SIM yang digunakan dalam penelitian ini merupakan simulator sistem operasi yang telah divalidasi dan digunakan secara konsisten dalam beberapa studi terdahulu [16], [17], [19], [24], sehingga hasil simulasinya dapat dipertanggungjawabkan secara akademik. Simulator ini mampu merepresentasikan mekanisme multiprogramming beserta interaksi CPU burst dan I/O burst secara bersamaan. Simulasi pada penelitian ini dilakukan secara deterministik dengan menggunakan satu skenario tetap yang mencakup 12 proses, di mana setiap algoritma dieksekusi dengan masukan yang identik untuk memastikan komparabilitas hasil. Pendekatan ini konsisten dengan metodologi yang digunakan pada studi terdahulu berbasis OS-SIM [16], [19]. Meskipun demikian, keterbatasan simulasi tunggal ini diakui sebagai salah satu keterbatasan penelitian yang perlu diatasi pada studi lanjutan melalui pendekatan multiple runs dengan variasi distribusi beban kerja.

OS-SIM digunakan secara khusus untuk mengkomparasikan berbagai algoritma klasik dari *process scheduling*. Penggunaan simulator ini berperan dalam perhitungan secara otomatis untuk mengetahui efisiensi penggunaan prosesor, ATT, AWT, dan *Avarage Response Time* (ART) terhadap keseluruhan proses yang dijalankan. Proses yang dijalankan pada simulasi berisikan CPU *bound*, I/O *bound*, dan campuran keduanya. I/O *bound* merupakan proses yang menghabiskan banyak waktu untuk melakukan operasi I/O, sedangkan CPU *bound* jarang menghasilkan permintaan I/O [21]. Proses berupa sampel yang akan diuji terpapar pada tabel 1. Sampel uji ini akan dijalankan pada tiap algoritma klasik yang telah ditetapkan. Rangkaian tahapan penelitian ditampilkan secara lengkap pada gambar 1.



Gambar 1. Rangkaian Tahapan Penelitian.

A. TINJAUAN LITERATUR

Tahapan awal penelitian ini dilakukan melalui studi literatur yang berfokus pada topik penjadwalan proses. Fokus kajian literatur meliputi prinsip kerja dari algoritma klasik penjadwalan proses, yaitu FCFS, SJF, *Priority*, dan RR. Selain itu, literatur relevan yang terkait tinjauan sistematis dan studi komparatif dari masing-masing algoritma juga digunakan untuk mengetahui kelebihan dan kelemahan setiap metode penjadwalan. Hasil kajian literatur dijadikan sebagai landasan teoritis dalam menganalisis hasil pengujian algoritma yang diteliti.

Untuk memperjelas posisi penelitian ini di antara studi-studi sebelumnya, tabel 1 menyajikan ringkasan komparatif penelitian terdahulu berdasarkan aspek algoritma yang dievaluasi, pemodelan I/O burst, penggunaan simulator, dan metrik evaluasi yang digunakan. Berdasarkan kajian terhadap literatur terkini [9]–[12], [16], [19], teridentifikasi bahwa belum ada studi yang secara bersamaan mengevaluasi keempat algoritma klasik berupa FCFS, SRTF, Preemptive Priority, dan Median Round Robin dalam satu lingkungan simulasi multiprogramming dengan pemodelan I/O interruption yang eksplisit. Penelitian ini mengisi celah tersebut dengan mengintegrasikan CPU burst dan I/O burst secara bersamaan dalam simulasi OS-SIM.

TABEL I
RINGKASAN TINJAUAN LITERATUR PENELITIAN PENJADWALAN CPU

Tinjauan Literatur				
Referensi	**Algoritma**	**Pemodelan I/O Burst**	**Simulator**	**Metrik Evaluasi**
[9]	FCFS, RR, SJF	Tidak	Simulasi	AWT, ATT, RT
[10]	FCFS, SJF, SRTF, RR, Priority, Multilevel	Tidak	Manual	AWT, ATT
[11]	FCFS, RR, SJF	Tidak	Simulasi Java	AWT, RT, CS
[12]	SRTF, RR	Tidak	Manual	AWT, ATT
[16]	Priority, RR	Tidak	OS-SIM	AWT, ATT, ART
[18]	Priority, RR	Tidak	Simulasi C	AWT, ATT
[19]	SRTF	Tidak	OS-SIM	AWT, ATT, ART
Penelitian ini	FCFS, SRTF, Priority, MRR	Ya (eksplisit)	OS-SIM	AWT, ATT, ART, Efisisensi, Throughput

Berdasarkan tabel di atas, terlihat bahwa mayoritas penelitian sebelumnya tidak mempertimbangkan pemodelan I/O burst secara eksplisit dalam skenario simulasi. Penelitian ini mengisi celah tersebut dengan menggabungkan CPU burst dan I/O burst secara bersamaan, sehingga menghasilkan evaluasi yang lebih representatif terhadap kondisi sistem operasi nyata.

B. SAMPEL UJI

Sampel uji pada penelitian ini berupa sekumpulan proses yang digunakan sebagai masukan pada simulasi algoritma penjadwalan proses. Setiap proses direpresentasikan oleh beberapa parameter, yaitu waktu kedatangan (*submission*), CPU *burst time*, prioritas proses, serta karakteristik proses sebagai CPU *bound* atau I/O *bound*. Sampel uji terdiri atas 12 proses, yaitu P1 hingga P12, dengan variasi nilai pada setiap parameter untuk merepresentasikan kondisi beban kerja yang beragam sebagaimana yang terjadi pada sistem operasi nyata. Nilai dari tiap parameter sampel uji terpapar pada tabel 2.

TABEL II
SAMPEL UJI PENJADWALAN PROSES

Sampel Uji				
Proses	***Submission Time***	**Prioritas**	***Burst Time***	**Antrian *Burst***
P1	0	3	4	CPU, CPU, CPU, CPU
P2	1	2	3	CPU, CPU, CPU
P3	2	6	5	CPU, CPU, CPU, CPU, CPU
P4	4	5	2	CPU, CPU
P5	7	4	6	CPU, CPU, CPU, CPU, CPU, CPU
P6	1	1	5	CPU, I/O, CPU, I/O, CPU
P7	3	7	5	CPU, CPU, I/O, CPU, CPU
P8	5	8	6	CPU, I/O, CPU, CPU, I/O, CPU
P9	6	9	4	CPU, CPU, I/O, CPU
P10	2	10	7	CPU, I/O, CPU, I/O, CPU, I/O, CPU
P11	8	3	8	CPU, I/O, CPU, I/O, CPU, I/O, I/O, CPU
P12	9	10	5	CPU, I/O, CPU, I/O, CPU

C. SIMULASI ALGORITMA PENJADWALAN

Sampel uji yang telah disiapkan disimulasikan menggunakan OS-SIM dengan menerapkan penjadwalan proses, yaitu FCFS *multiprogramming*, *preemptive* SJF atau bisa disebut dengan SRTF, *Preemptive Priority*, dan RR dengan nilai TQ yang ditentukan berdasarkan nilai median dari keseluruhan *burst time* (MRR). Algoritma tersebut digunakan karena beberapa studi menyatakan bahwa algoritma tersebut merupakan algoritma yang efektif di kelasnya, sehingga layak digunakan sebagai objek untuk menganalisis efisiensi penjadwalan proses.

1) FCFS MULTIPROGRAMMING

FCFS atau bisa disebut FIFO (*First In First Out*) dalam konteks *multiprogramming* mengeksekusi proses berdasarkan urutan kedatangan tanpa *preemption*. Setiap proses yang pertama masuk ke antrean akan dilayani hingga selesai. Mekanisme ini memiliki keunggulan berupa implementasi sederhana dan bebas dari risiko *starvation*. Namun, kelemahannya muncul ketika proses berdurasi panjang berada di awal antrean, menyebabkan proses lainnya menunggu lebih lama dan memunculkan efek konvoi yang dapat meningkatkan waktu tunggu rata-rata [15].

2) PREEMPTIVE SJF / SRTF

Preemptive SJF (*Short Job First*), juga disebut sebagai *Shortest Remaining Time First* (SRTF), mengeksekusi proses dengan sisa *burst time* paling pendek. Jika proses baru tiba dengan *burst time* yang lebih kecil daripada sisa waktu proses aktif, proses yang sedang berjalan akan dihentikan dan digantikan oleh proses baru. Algoritma ini dapat meminimalkan waktu tunggu rata-rata secara efektif, tetapi memiliki risiko *starvation* terhadap proses berdurasi panjang apabila perbedaan *burst time* antarproses terlalu besar [12], [13].

3) PREEMPTIVE PRIORITY

Pada algoritma ini, proses dipilih berdasarkan prioritas yang telah ditentukan. Proses yang memiliki nilai prioritas tertinggi akan dijalankan terlebih dahulu. *Preemption* terjadi ketika sebuah proses baru memiliki prioritas yang lebih tinggi daripada proses yang sedang berjalan saat ini. Algoritma ini sangat efektif untuk menangani proses-proses kritis yang membutuhkan respons cepat, tetapi memiliki potensi *starvation* pada proses dengan prioritas rendah jika tidak ada mekanisme *aging* [18], [22]. Mekanisme *aging* ini memungkinkan prioritas yang ditingkatkan dari proses yang sedang menunggu dalam antrian (*ready queue*) secara bertahap.

4) MEDIAN ROUND ROBIN

Algoritma MRR membagi jatah eksekusi kepada setiap proses secara bergantian dalam putaran yang teratur. Performa algoritma ini bergantung pada kecocokan nilai kuantum terhadap variasi *burst time*, di mana jika waktu kuantum terlalu besar membuatnya mendekati FCFS, sementara waktu kuantum terlalu kecil meningkatkan frekuensi *context switching* [7]. Pada penelitian ini, nilai waktu kuantum ditentukan berdasarkan median dari keseluruhan *burst time* proses yang ada, sehingga pemilihan waktu kuantum lebih adaptif terhadap karakteristik beban kerja. Pendekatan ini dapat memberikan keseimbangan antara keadilan distribusi CPU dan efisiensi pergerakan konteks [4].

Pada tabel 1, sampel uji terdiri dari dua belas proses yang mencakup proses CPU *bound* dan I/O *bound*. Nilai *burst time* dari keseluruhan proses diurutkan dari nilai terkecil hingga terbesar, berupa 2, 3, 4, 4, 5, 5, 5, 5, 6, 6, 7, 8. Karena data berjumlah genap, median dihitung sebagai rata-rata dua nilai tengah, yaitu lima dan lima. Dengan demikian, waktu kuantum yang digunakan pada penelitian ini adalah lima.

D. INDIKATOR PERFORMA SISTEM

Pada tahap evaluasi, dilakukan perbandingan *Average Waiting Time* (AWT), *Average Turnaround Time* (ATT), dan *Average Response Time* (ART) untuk setiap algoritma penjadwalan, disertai pengukuran *throughput* pada tiap algoritma. Perhitungan indikator-indikator ini bertujuan untuk menilai efisiensi algoritma dalam mengelola antrean proses, mencakup seberapa lama proses menunggu sebelum dieksekusi, seberapa cepat proses diselesaikan sejak kedatangannya, serta seberapa responsif sistem terhadap proses yang baru masuk. Dengan menggunakan metrik tersebut, kinerja keseluruhan algoritma dapat dianalisis secara menyeluruh berdasarkan waktu respons, kecepatan penyelesaian proses, dan efektivitas pemanfaatan CPU.

Evaluasi kinerja algoritma penjadwalan CPU pada penelitian ini didasarkan pada lima metrik utama yang didefinisikan secara matematis sebagai berikut.

5) AVERAGE WAITING TIME (AWT)

Average Waiting Time (AWT) merupakan rata-rata waktu yang dihabiskan proses di dalam ready queue sebelum memperoleh giliran eksekusi CPU, diformulasikan sebagai berikut.

$$AWT = \frac{1}{n} \times \sum_{i} Wi$$

Di mana W_i adalah *waiting time* proses ke-i dan n adalah jumlah total proses.

6) AVERAGE TURNAROUND TIME (ATT)

Average Turnaround Time (ATT) merupakan rata-rata waktu total yang dibutuhkan sejak proses tiba hingga proses selesai dieksekusi, didefinisikan sebagai berikut.

$$ATT = \frac{1}{n} \times \sum_{i} (Ci - Ai)$$

Di mana C_i adalah *completion time,* A_i adalah *arrival time* proses ke-i, dan n adalah jumlah total proses.

7) AVERAGE RESPONSE TIME (ART)

Average Response Time (ART) merupakan rata-rata waktu dari saat proses tiba hingga proses pertama kali mendapatkan akses ke CPU.

$$ART = \frac{1}{n} \times \sum_{i} (Fi - Ai)$$

Di mana F_i adalah waktu pertama kali proses ke-I dieksekusi oleh CPU*,* A_i adalah *arrival time* proses ke-i, dan n adalah jumlah total proses.

1) THROUGHPUT

Throughput didefinisikan sebagai jumlah proses yang berhasil diselesaikan per satuan waktu.

$$Throughput = \frac{n}{T_{total}}$$

Di mana T_{total} adalah total waktu simulasi dari awal hingga seluruh proses selesai dan n adalah jumlah total proses.

2) EFISIENSI CPU

Efisiensi CPU merupakan persentase waktu prosesor yang digunakan secara produktif untuk mengeksekusi proses dibandingkan total waktu simulasi

$$Efisiensi\ CPU\ (\%) = \frac{T_{busy}}{T_{total}} \times 100\%$$

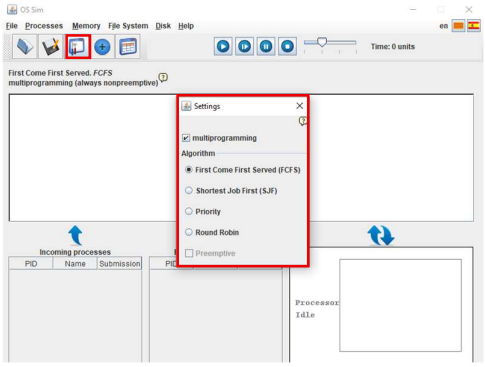
Di mana T_{busy} adalah total waktu CPU dalam kondisi aktif mengeksekusi proses dan T_{total} adalah total waktu simulasi dari awal hingga seluruh proses selesai dan n adalah jumlah total proses.

E. PERBANDINGAN HASIL

Berdasarkan analisis terhadap AWT, ATT, dan ART, dapat diidentifikasi algoritma dengan kinerja paling optimal. Rekomendasi difokuskan pada algoritma yang mampu memberikan AWT paling rendah, ATT lebih singkat, serta ART yang lebih cepat. Evaluasi ini memastikan bahwa algoritma terpilih benar-benar optimal dalam memanfaatkan CPU, mengurangi waktu antrean, dan meningkatkan responsivitas sistem secara keseluruhan.

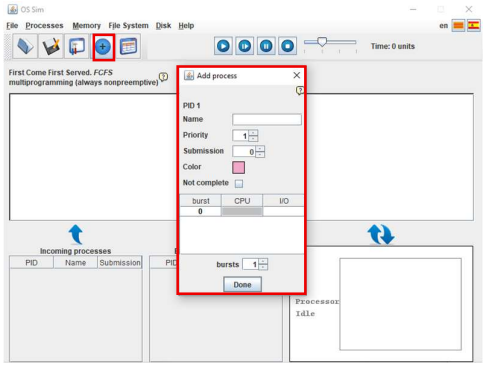
III. HASIL DAN PEMBAHASAN

Simulasi penjadwalan proses pada penelitian ini dilakukan menggunakan OS-SIM, sebuah simulator sistem operasi yang menyediakan lingkungan terkontrol untuk mengamati perilaku algoritma penjadwalan proses secara visual dan terstruktur. Pemilihan OS-SIM didasarkan pada kemampuannya dalam memodelkan mekanisme multiprogramming serta kejadian CPU dan I/O secara bersamaan, sehingga hasil simulasi lebih representatif dibandingkan pendekatan perhitungan manual. Seperti tampak pada gambar 2, antarmuka OS-SIM menyediakan menu pemilihan algoritma penjadwalan, meliputi *First Come First Served* (FCFS), *Shortest Job First* (SJF), *Priority*, dan *Round Robin*. Opsi *multiprogramming* dan *preemptive* juga dapat diaktifkan untuk memastikan proses dapat berpindah antara kondisi *ready*, *running*, dan *blocked* sebagaimana terjadi pada sistem operasi nyata.

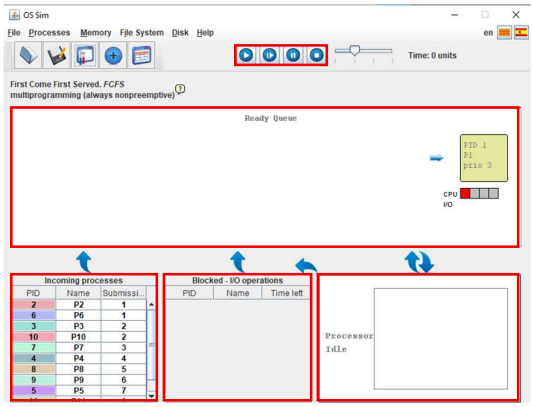


Gambar 2. Konfigurasi OS-SIM

Setelah memilih algoritma yang diuji, proses selanjutnya adalah menambahkan kumpulan proses yang akan menjadi sampel simulasi. Gambar 3 memperlihatkan jendela konfigurasi proses, di mana setiap proses dapat diatur parameter utamanya, seperti nilai prioritas, waktu kedatangan (submission time), warna identifikasi, serta urutan *CPU burst* dan *I/O burst*. Pengaturan ini memungkinkan setiap proses memiliki karakteristik berbeda, termasuk apakah mereka bersifat CPU *bound* maupun I/O *bound*. Ketika seluruh proses telah ditambahkan, OS-SIM menampilkan daftar *incoming processes* dan kondisi awal sistem sebagaimana terlihat pada Gambar 4. Pada tahap ini seluruh proses menunggu di antrean awal sebelum simulasi dijalankan, dan sistem menampilkan bagian *ready queue*, *blocked I/O operations*, serta status prosesor. Tampilan ini menjadi dasar bagi eksekusi simulasi berikutnya, yang kemudian menghasilkan *Gantt chart* dan *Process Scheduling Information* pada setiap algoritma yang diuji.



Gambar 3. Konfigurasi Penjadwalan Proses

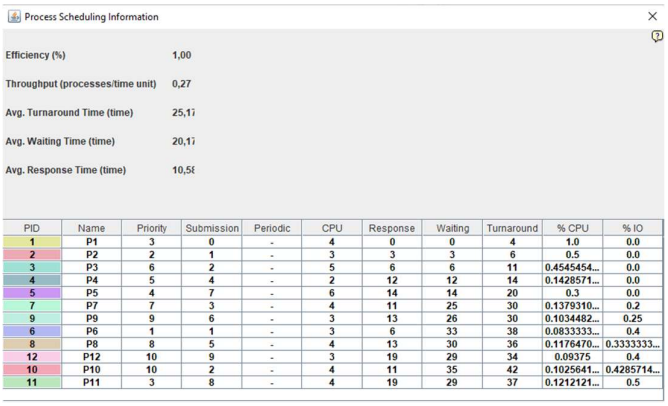


Gambar 4. Hasil Konfigurasi OS-SIM

A. FCFS MULTIPROGRAMMING

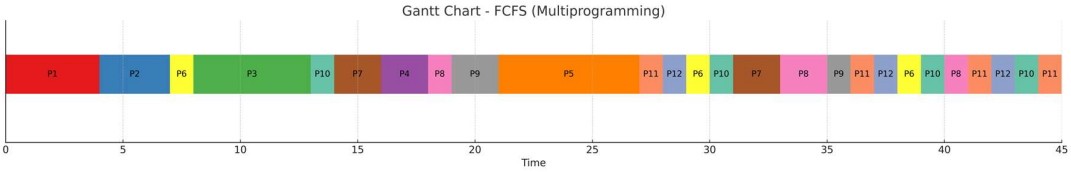
Analisis kinerja algoritma FCFS pada skenario *multiprogramming* menunjukkan bahwa mekanisme penjadwalan ini mengeksekusi proses secara murni berdasarkan urutan kedatangannya tanpa menerapkan *preemption*. Hal tersebut ditunjukkan pada *gantt chart* proses (gambar 6), di mana proses yang tiba paling awal dan memiliki *burst time* besar memperoleh waktu eksekusi terlebih dahulu dan berjalan hingga selesai. Pola ini memunculkan efek konvoi yang berdampak pada meningkatnya waktu tunggu proses-proses berikutnya. Dampak tersebut terlihat pada halaman *process scheduling information* yang ditampilkan pada gambar

5. FCFS menghasilkan AWT sebesar 20.17 dan ATT sebesar 25.17, yang tergolong tinggi dibandingkan algoritma penjadwalan lain. Meskipun demikian, FCFS tetap memiliki kelebihan berupa struktur implementasi yang sederhana serta ketiadaan risiko *starvation*, karena setiap proses dipastikan memperoleh giliran eksekusi sesuai urutan kedatangannya.



PID	Name	Priority	Submission	Periodic	CPU	Response	Waiting	Turnaround	% CPU	% IO
1	P1	3	0	-	4	0	0	4	1.0	0.0
2	P2	2	1	-	3	3	3	6	0.5	0.0
3	P3	6	2	-	5	6	6	11	0.4545454...	0.0
4	P4	5	4	-	2	12	12	14	0.1428571...	0.0
5	P5	4	7	-	6	14	14	20	0.3	0.0
7	P7	7	3	-	4	11	25	30	0.1379310...	0.2
9	P9	9	6	-	3	13	26	30	0.1034482...	0.25
6	P6	1	1	-	3	6	33	38	0.0833333...	0.4
8	P8	8	5	-	4	13	30	36	0.1176470...	0.3333333...
12	P12	10	9	-	3	19	29	34	0.09375	0.4
10	P10	10	2	-	4	11	35	42	0.1025641...	0.4285714...
11	P11	3	8	-	4	19	29	37	0.1212121...	0.5

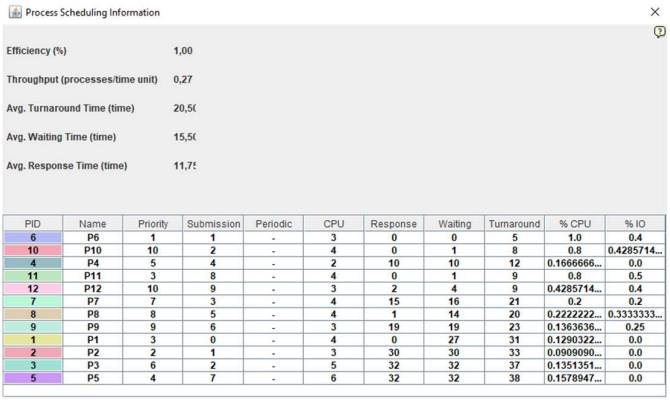
Gambar 5. Hasil performa FCFS Multiprogramming



Gambar 6. Gantt chart FCFS Multiprogramming

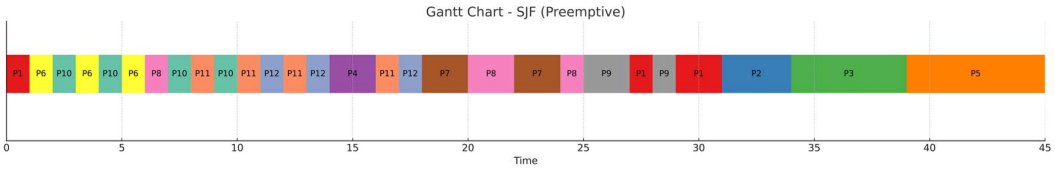
B. PREEMPTIVE SJF / SRTF

Hasil simulasi menunjukkan bahwa algoritma *preemptive* SJF atau SRTF memberikan kinerja paling efisien di antara seluruh algoritma yang diuji. Pola eksekusi pada gambar 8 memperlihatkan bahwa proses dengan sisa *burst time* paling pendek selalu memperoleh prioritas, sehingga terjadi banyak *preemption* ketika proses baru datang dengan durasi yang lebih kecil. Mekanisme ini berdampak langsung pada hasil pengukuran pada gambar 7, di mana nilai AWT turun hingga 15.5 *s* dan ATT mencapai 20.5 *s*, yang merupakan yang terendah di seluruh skenario. Selain itu, nilai ART sebesar 11.7 *s* menunjukkan bahwa proses dapat segera memperoleh eksekusi awal. Kombinasi pola eksekusi yang adaptif dan efisiensi waktu ini konsisten dengan teori bahwa SJF secara efektif meminimalkan waktu tunggu rata-rata, terutama dalam lingkungan *multiprogramming* dengan variasi *burst time* dan interupsi I/O.



PID	Name	Priority	Submission	Periodic	CPU	Response	Waiting	Turnaround	% CPU	% IO
6	P6	1	1	-	3	0	0	5	1.0	0.4
10	P10	10	2	-	4	0	1	8	0.8	0.4285714...
4	P4	5	4	-	2	10	10	12	0.1666666...	0.0
11	P11	3	8	-	4	0	1	9	0.8	0.5
12	P12	10	9	-	3	2	4	9	0.4285714...	0.4
7	P7	7	3	-	4	15	16	21	0.2	0.2
8	P8	8	5	-	4	1	14	20	0.2222222...	0.3333333...
9	P9	9	6	-	3	19	19	23	0.1363636...	0.25
1	P1	3	0	-	4	0	27	31	0.1290322...	0.0
2	P2	2	1	-	3	30	30	33	0.0909090...	0.0
3	P3	6	2	-	5	32	32	37	0.1351351...	0.0
5	P5	4	7	-	6	32	32	38	0.1578947...	0.0

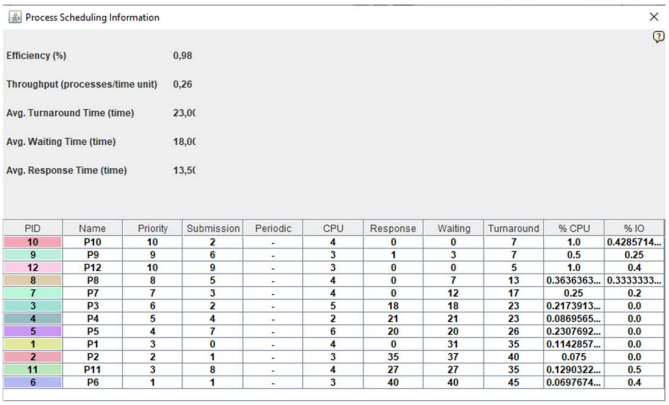
Gambar 7. Gantt chart FCFS Multiprogramming



Gambar 8. Gantt chart FCFS Multiprogramming

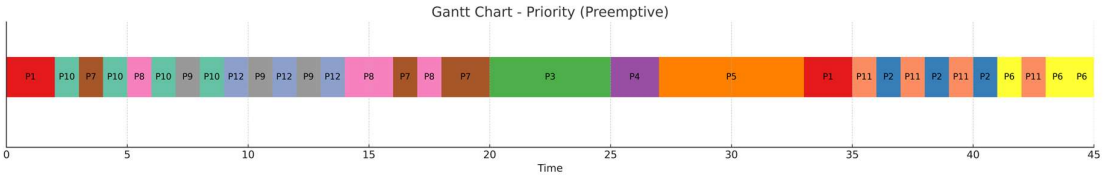
C. PREEMPTIVE PRIORITY

Hasil simulasi memperlihatkan bahwa algoritma *Priority Preemptive* menghasilkan pola eksekusi yang sangat bergantung pada urutan prioritas proses. Pola pada gambar 10 menunjukkan bagaimana proses dengan prioritas lebih tinggi berulang kali melakukan preemption terhadap proses yang sedang berjalan, sehingga alur eksekusi untuk proses berprioritas rendah tampak terputus-putus dan tertunda. Konsekuensi dari mekanisme ini tampak pada gambar 9, di mana rata-rata waktu tunggu dan penyelesaian proses berada pada tingkat yang masih cukup besar, dengan nilai AWT sebesar 18 *s* dan ATT 23 *s*. Selain itu, ART mencapai 13.5 *s*, yang mengindikasikan bahwa tidak semua proses memperoleh respons awal secara cepat, terutama yang berprioritas rendah. Sementara itu, efisiensi sistem berada pada nilai 98% dengan *throughput* 0.26 proses per satuan waktu. Pola ini konsisten dengan karakter algoritma Priority, yang mengutamakan proses kritis namun berpotensi mengorbankan kenyamanan proses lain dalam lingkungan multiprogramming dengan kombinasi beban CPU–I/O.



PID	Name	Priority	Submission	Periodic	CPU	Response	Waiting	Turnaround	% CPU	% IO
10	P10	10	2	-	4	0	0	7	1.0	0.4285714...
9	P9	9	6	-	3	1	3	7	0.5	0.25
12	P12	10	9	-	3	0	0	5	1.0	0.4
8	P8	8	5	-	4	0	7	13	0.3636363...	0.3333333...
7	P7	7	3	-	4	0	12	17	0.25	0.2
3	P3	6	2	-	5	18	18	23	0.2173913...	0.0
4	P4	5	4	-	2	21	21	23	0.0869565...	0.0
5	P5	4	7	-	6	20	20	26	0.2307692...	0.0
1	P1	3	0	-	4	0	31	35	0.1142857...	0.0
2	P2	2	1	-	3	35	37	40	0.075	0.0
11	P11	3	8	-	4	27	27	35	0.1290322...	0.5
6	P6	1	1	-	3	40	40	45	0.0697674...	0.4

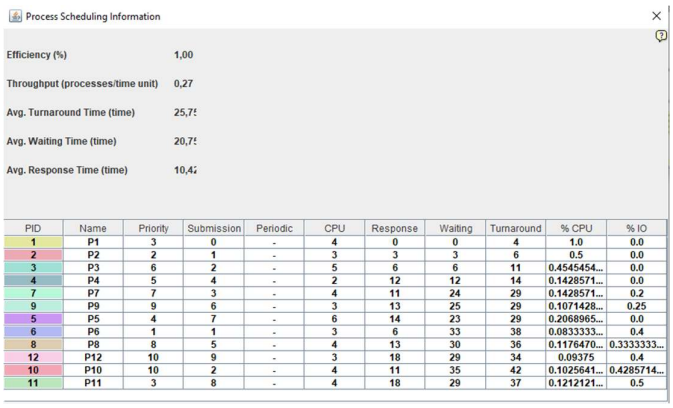
Gambar 9. Gantt chart FCFS Multiprogramming



Gambar 10. Gantt chart FCFS Multiprogramming

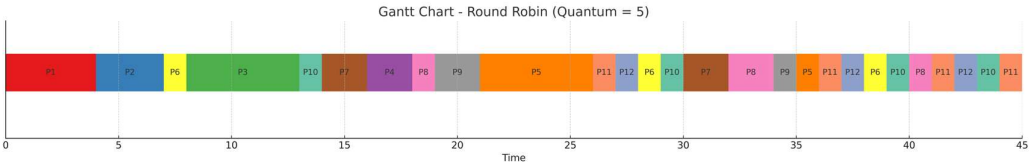
D. MEDIAN ROUND ROBIN

Hasil simulasi menunjukkan bahwa algoritma MRR dengan nilai TQ = 5 *s* memberikan distribusi jatah CPU yang lebih merata di antara proses. Pola pada Gambar 12 memperlihatkan setiap proses dieksekusi secara bergiliran, sehingga alur eksekusi tampak terfragmentasi namun tetap menjaga unsur keadilan. Konsekuensi dari mekanisme ini tercermin pada Gambar 11, di mana efisiensi sistem tetap tinggi dengan nilai 100% dan *throughput* mencapai 0.27 proses per satuan waktu. Meskipun demikian, nilai AWT sebesar 20.7 *s* dan ATT 25.7 *s* menunjukkan bahwa waktu tunggu dan penyelesaian proses masih berada pada kisaran yang relatif tinggi dan mendekati kinerja FCFS. Di sisi lain, nilai ART sebesar 10.4 *s* mengindikasikan bahwa proses memperoleh respons awal yang cukup cepat berkat pembagian quantum yang teratur. Secara keseluruhan, algoritma MRR menawarkan kompromi antara keadilan dan responsivitas, tetapi dari sisi efisiensi waktu tunggu masih belum mampu melampaui performa algoritma SJF.



PID	Name	Priority	Submission	Periodic	CPU	Response	Waiting	Turnaround	% CPU	% IO
1	P1	3	0	-	4	0	0	4	1.0	0.0
2	P2	2	1	-	3	3	3	6	0.5	0.0
3	P3	6	2	-	5	6	6	11	0.4545454...	0.0
4	P4	5	4	-	2	12	12	14	0.1428571...	0.0
7	P7	7	3	-	4	11	24	29	0.1428571...	0.2
9	P9	9	6	-	3	13	25	29	0.1071428...	0.25
5	P5	4	7	-	6	14	23	29	0.2068965...	0.0
6	P6	1	1	-	3	6	33	38	0.0833333...	0.4
8	P8	8	5	-	4	13	30	36	0.1176470...	0.3333333...
12	P12	10	9	-	3	18	29	34	0.09375	0.4
10	P10	10	2	-	4	11	35	42	0.1025641...	0.4285714...
11	P11	3	8	-	4	18	29	37	0.1212121...	0.5

Gambar 11. Hasil performa Round Robin (TQ = 5 s)



Gambar 12. Hasil performa Round Robin (TQ = 5 s)

E. KOMPARASI ALGORITMA

Perbandingan kinerja empat algoritma penjadwalan proses menunjukkan adanya perbedaan karakteristik yang cukup signifikan pada aspek waktu tunggu, waktu penyelesaian, dan kecepatan respons awal proses. Meskipun seluruh algoritma beroperasi dengan tingkat efisiensi CPU yang tinggi berkisar antara 98% hingga 100%, kualitas performa masing-masing pendekatan lebih jelas terlihat melalui indikator AWT, ATT, ART, dan *throughput*. Tabel 2 menyajikan komparasi kinerja algoritma penjadwalan proses yang digunakan pada penelitian ini.

TABEL II
KOMPARASI KINERJA ALGPRITMA PENJADWALAN PROSES

Algoritma	Efisiensi Penggunaan CPU	AWT	ATT	ART	*Troughput*
FCFS *Multiprogramming*	100%	20.17 *s*	25.17 *s*	10.58 *s*	0.27
Preemptive SJF / SRTF	100%	15.5 *s*	20.5 *s*	11.75 *s*	0.27
Preemptive Priority	98%	18 *s*	23 *s*	13.5 *s*	0.26
Median Round Robin	100%	20.75 *s*	25.75 *s*	10.42 *s*	0.27

Algoritma Preemptive SJF (SRTF) menampilkan kinerja paling efisien dalam hal waktu tunggu dan waktu penyelesaian, dengan nilai AWT sebesar 15.5 *s* dan ATT sebesar 20.5 *s*. Keunggulan ini merupakan konsekuensi langsung dari strategi pemilihan proses dengan *burst time* terpendek, yang secara efektif mengurangi total waktu proses harus menunggu di *ready queue*. Namun demikian, nilai ART yang dicapai algoritma ini (11.75 *s*) tidak menjadi yang terbaik, karena proses yang lebih panjang dapat tertunda akibat seringnya *preemption*.

Sementara itu, FCFS *multiprogramming* dan MRR menunjukkan performa yang lebih baik dalam hal respons awal proses. Kedua algoritma ini menghasilkan nilai ART yang relatif rendah, yaitu masing-masing 10.58 *s* dan 10.42 *s*. Meski demikian, peningkatan respons awal ini dibayar dengan waktu tunggu dan waktu penyelesaian yang lebih tinggi. FCFS mencatat AWT sebesar 20.17 *s* dan ATT sebesar 25.17 *s*, yang menjadi indikasi munculnya efek konvoi akibat sifat *non-preemptive*. Pada MRR, pembagian kuantum waktu yang merata menyebabkan frekuensi *context switching* meningkat, sehingga berdampak pada AWT dan ATT yang relatif besar, yaitu 20.75 *s* dan 25.75 *s*.

Algoritma *Preemptive Priority* berada pada posisi menengah, menghasilkan AWT sebesar 18 *s* dan ATT sebesar 23 *s*, lebih baik daripada FCFS dan MRR, tetapi masih di bawah SRTF. Nilai ART yang lebih tinggi 13.5 *s* memperlihatkan potensi penundaan pada proses prioritas rendah, selaras dengan kecenderungan *starvation* jika mekanisme *aging* tidak diterapkan. Efisiensi CPU yang sedikit lebih rendah (98%) juga menunjukkan adanya beban pemrosesan tambahan akibat dinamika prioritas. Di sisi throughput, seluruh algoritma menunjukkan nilai yang relatif seragam (0.26–0.27 proses per satuan waktu), sehingga metrik ini tidak memberikan perbedaan berarti dalam evaluasi performa.

Analisis kuantitatif terhadap selisih kinerja tiap algoritma menunjukkan perbedaan yang signifikan. Algoritma SRTF mampu mereduksi nilai AWT sebesar 23,2% dibandingkan FCFS (dari 20,17 *s* menjadi 15,5 *s*) dan sebesar 25,3% dibandingkan MRR (dari 20,75 *s* menjadi 15,5 *s*). Demikian pula pada ATT, SRTF mencapai pengurangan sebesar 18,5% terhadap FCFS dan 20,2% terhadap MRR. Perbedaan ini secara langsung dipengaruhi oleh keberadaan interupsi I/O yaitu ketika proses I/O bound memasuki kondisi *blocked*, algoritma SRTF secara adaptif memanfaatkan waktu tersebut untuk mengeksekusi proses dengan sisa *burst time* terpendek,

sehingga *idle time* prosesor dapat diminimalkan. Sebaliknya, FCFS tidak dapat melakukan preemption terhadap proses yang sedang berjalan, sehingga periode *blocked* akibat I/O mengakibatkan antrian proses lainnya tertahan lebih lama. Adapun nilai throughput yang relatif seragam (0,26–0,27 proses/satuan waktu) di seluruh algoritma mengindikasikan bahwa perbedaan utama terletak pada distribusi waktu tunggu, bukan pada jumlah proses yang diselesaikan per satuan waktu.

Secara keseluruhan, SRTF menjadi pilihan paling efektif untuk skenario *batch processing*, karena mampu meminimalkan AWT dan ATT secara konsisten. FCFS dan Round Robin lebih unggul dalam menyediakan respons awal yang cepat, sehingga lebih sesuai untuk lingkungan interaktif. Adapun *Preemptive Priority* menawarkan kompromi performa yang cukup baik, namun memerlukan mekanisme pendukung untuk mengatasi risiko *starvation* dalam beban kerja tertentu.

IV. KESIMPULAN

Penelitian ini berhasil melakukan analisis komparatif terhadap empat algoritma klasik penjadwalan CPU dengan mempertimbangkan pengaruh interupsi I/O menggunakan simulator OS-SIM. Berdasarkan hasil simulasi dan pembahasan, dapat disimpulkan bahwa algoritma *Preemptive* SJF (SRTF) menunjukkan kinerja paling efisien dalam hal minimalisasi waktu tunggu dan waktu penyelesaian proses, dengan nilai AWT sebesar 15.5 *s* dan ATT sebesar 20.5 *s*. Keunggulan ini menjadikan SRTF sebagai pilihan optimal untuk lingkungan *batch processing* yang mengutamakan efisiensi waktu pemrosesan.

Algoritma FCFS *multiprogramming* dan MRR lebih unggul dalam memberikan respons awal yang cepat terhadap proses dengan nilai ART masing-masing 10.58 *s* dan 10.42 *s*, sehingga lebih sesuai diterapkan pada sistem interaktif yang membutuhkan responsivitas tinggi. Namun, kedua algoritma ini menghasilkan waktu tunggu dan waktu penyelesaian yang relatif lebih tinggi akibat efek konvoi pada FCFS dan frekuensi *context switching* yang meningkat pada MRR.

Algoritma *Preemptive Priority* berada pada posisi menengah dengan AWT 18 *s* dan ATT 23 *s*, menawarkan kompromi yang cukup baik antara efisiensi dan responsivitas. Namun, algoritma ini memerlukan mekanisme *aging* untuk mencegah potensi starvation pada proses dengan prioritas rendah. Secara umum, pemilihan algoritma penjadwalan yang optimal sangat dipengaruhi oleh karakteristik beban kerja sistem, dengan pertimbangan utama antara efisiensi waktu pemrosesan dan kecepatan respons awal.

Penelitian ini memiliki beberapa keterbatasan yang perlu diakui. Pertama, simulasi dilakukan dengan satu skenario tetap yang terdiri dari 12 proses, sehingga generalisabilitas hasil terhadap skenario beban kerja yang lebih kompleks masih terbatas. Kedua, penelitian ini tidak mencakup analisis kompleksitas waktu algoritma secara formal maupun pengujian pada sistem operasi nyata. Ketiga, nilai time quantum pada MRR ditentukan secara statis berdasarkan median, tanpa mempertimbangkan mekanisme penyesuaian dinamis. Untuk penelitian lanjutan, disarankan (1) memperluas variasi skenario beban kerja dengan distribusi burst time yang berbeda secara statistik; (2) mengimplementasikan dan memvalidasi hasil simulasi pada sistem operasi nyata atau lingkungan embedded; (3) mengeksplorasi algoritma penjadwalan hibrid yang menggabungkan mekanisme prioritas dinamis dengan pendekatan burst-aware scheduling untuk sistem real-time.

REFERENSI

[1] I. Fadhilah, H. Siregar, I. Komputer, U. P. Indonesia, I. Komputer, dan U. P. Indonesia, "Systematic Literature Review : Perbandingan Algoritma Round Robin dan Shortest Job First dalam Penjadwalan CPU," *BIOS J. Teknol. Inf. dan Rekayasa Komput.*, vol. 6, no. 2, hal. 74–83, 2025.

[2] D. N. Al Husaeni dan J. Kusnendar, "Analisis Trend Penelitian Penggunaan Algoritma Penjadwalan serta Faktor yang Mempengaruhinya: Analisis Bibliometrik R dan Pemetaan VOSviewer," *J. Nas. Teknol. dan Sist. Inf.*, vol. 11, no. 01, hal. 57–66, 2025.

[3] T. D. Putra dan R. Purnomo, "Analisis Algoritma Round Robin pada Penjadwalan CPU," *J. Ilm. Teknol. Inf. Asia*, vol. 15, no. 2, hal. 85–90, 2021.

[4] A. R. M. N, K. Al-qawasmi, D. A. Abdulkarim, A. W. Haithem, O. A. A, dan A. Al-momani, "Investigating the Impact of Statistical Measures on Round Robin Quantum Length Selection in Scheduling Algorithms," *WSEAS Trans. Comput. Res.*, vol. 13, hal. 450–461, 2025, doi: 10.37394/232018.2025.13.41.

[5] R. Purnomo dan T. D. Putra, "Median-Average Round Robin (MARR) Algorithm for Optimal CPU Task Scheduling," *Sink. J. dan Penelit. Tek. Inform.*, vol. 9, no. 1, hal. 90–95, 2025.

[6] T. D. Putra dan R. Purnomo, "A Review on AMRR and Improved Round Robin Algorithms: Comparative Study," *Sink. J. dan Penelit. Tek. Inform.*, vol. 8, no. 4, hal. 2354–2360, 2024.

[7] H. A. Syauqi dan H. Siregar, "Systematic Literature Review (SLR): Dampak Modifikasi Algoritma Round- Robin dalam Penjadwalan CPU pada Sistem Operasi," *Digit. Digit. Transform. Technol.*, vol. 5, no. 1, hal. 368–374, 2025.

[8] H. R. Bhapkar, P. R. Chandre, dan P. Mahalle, "Minimizing CPU Utilization for Job Scheduling Problems by the Advanced Round Robin Method: A Pragmatic Perspective," *ASEAN J. Sci. Technol. Dev.*, vol. 42, no. 1, hal. 27–38, 2025, doi: https://doi.org/10.61931/2224-9028.1610.

[9] M. González-Rodríguez, L. Otero-Cerdeira, E. González-Rufino, dan F. J. Rodríguez-Martínez, "Study and evaluation of CPU scheduling algorithms," *Heliyon*, vol. 10, no. 9, hal. e29959, Mei 2024, doi: 10.1016/j.heliyon.2024.e29959.

[10] S. Pemasinghe dan S. Rajapaksha, "Comparison of CPU Scheduling Algorithms: FCFS, SJF, SRTF, Round Robin, Priority Based, and Multilevel Queuing," in *2022 IEEE 10th Region 10 Humanitarian Technology Conference (R10-HTC)*, IEEE, Sep 2022, hal. 318–323. doi: 10.1109/R10-HTC54060.2022.9929533.

[11] O. Hajjar, E. Mekhallalati, N. Annwty, F. Alghayadh, I. Keshta, dan M. Algabri, "Performance Assessment of CPU Scheduling Algorithms: A Scenario-Based Approach with FCFS, RR, and SJF," *J. Comput. Sci.*, vol. 20, no. 9, hal. 972–985, Sep 2024, doi: 10.3844/jcssp.2024.972.985.

[12] R. Purnomo, T. D. Putra, U. Bhayangkara, dan J. Raya, "Comparative Study: Preemptive Shortest Job First and Round Robin Algorithms," *Sink. J. dan Penelit. Tek. Inform.*, vol. 8, no. 2, hal. 756–763, 2024.

[13] R. Belferik dan E. Banjarnahor, "Analisis Average Waiting Time Penjadwalan CPU Menggunakan Algoritma Shortest Remaining First dan Algoritma Round Robin," *JDMIS J. Data Min. Inf. Syst.*, vol. 3, no. 1, hal. 43–53, 2025, doi: 10.54259/jdmis.v3i1.4076.

[14] M. T. D. Putra, H. Hidayat, N. Septian, dan T. Afriani, "Analisis Perbandingan Algoritma Penjadwalan CPU First Come First (FCFS) dan Round Robin," *Build. Informatics, Technol. Sci.*, vol. 3, no. 3, hal. 207–212, 2021, doi: 10.47065/bits.v3i3.1047.

[15] Z. Indra, A. A. Zidan, A. A. Silaen, A. N. Habibi, dan K. P. Sitepu, "Analisis Komparatif dan Evaluatif terhadap Algoritma First-Come First-Served (FCFS) da/lam Penjadwalan CPU di Era Komputasi Modern," *J. Penelit. Inov.*, vol. 5, no. 4, hal. 3197–3206, 2025.

[16] T. D. Putra dan R. Purnomo, "Simulation and Modelling of Pre-emptive Priority CPU Scheduling Algorithm," *Sink. J. dan Penelit. Tek. Inform.*, vol. 8, no. 3, hal. 1631–1640, 2024.

[17] T. D. Putra dan R. Purnomo, "Simulation of Priority Round-Robin Scheduling Algorithm," *Sink. J. dan Penelit. Tek. Inform.*, vol. 6, no. 4, hal. 2170–2181, 2022.

[18] W. Widiarto, R. A. Chaerunnisa, R. A. Tsaqif, dan S. S. Nadia, "Analisis Perbandingan Penjadwalan Proses Menggunakan Algoritma Round Robin dan Priority Preemptive," *Progresif J. Ilm. Komput.*, vol. 20, no. 2, hal. 762–773, 2024.

[19] R. Purnomo dan T. D. Putra, "Simulation of Preemptive Shortest Job First Algorithm," *IJARCCE Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 11, no. 5, hal. 1–11, 2022, doi: 10.17148/IJARCCE.2022.11501.

[20] T. D. Putra, "Analysis of Preemptive Shortest Job First (SJF) Algorithm in CPU Scheduling," *IJARCCE Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 9, no. 4, hal. 41–45, 2020, doi: 10.17148/IJARCCE.2020.9408.

[21] A. Silberschatz, P. B. Galvin, dan G. Gagne, *Operating System Concepts*, 10 ed. New York, NY, USA: Wiley, 2018.

[22] G. A. Rumahorbo, Z. Indra, A. Wijaya, M. D. Putri, dan C. S. Buulolo, "Analisis Perbandingan Algoritma Penjadwalan Prioritas Preemptive dan Non-Preemptive Menggunakan Aplikasi Web Interaktif," *Tek. J. Ilmu Tek. dan Inform.*, vol. 5, no. 2, hal. 150–158, 2025.

[23] R. A. Putri, "APLIKASI SIMULASI ALGORITMA PENJADWALAN SISTEM OPERASI," *Jurti J. Teknol. Inf.*, vol. 5, no. 1, hal. 98–102, 2021.

[24] F. L. Putri dan F. P. Rochim, "Tinjauan Komprehensif : Simulasi Sistem Disk Simulation with Various Algorithms Using OS Sim," *Electron J. Ilm. Tek. Elektro*, vol. 6, no. 1, hal. 47–55, 2025.