

**IMPLEMENTASI *PRIORITY SCHEDULING* UNTUK
OPTIMASI PERFORMA GAMING PADA SISTEM OPERASI
WINDOWS**

SKRIPSI

DISUSUN OLEH

**MHD. ALWI AZMI HARAHAH
2209020154**



**PROGRAM STUDI TEKNOLOGI INFORMASI
FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITAS MUHAMMADIYAH SUMATERA UTARA**

MEDAN

2026

**Implementasi Priority Scheduling Untuk Optimasi Performa Gaming Pada
Sistem Operasi Windows**

SKRIPSI

**Diajukan sebagai salah satu syarat untuk memperoleh gelar Sarjana
Komputer (S.Kom) dalam Program Studi Teknologi Informasi, pada Fakultas
Ilmu Komputer dan Teknologi Informasi, Universitas Muhammadiyah
Sumatera Utara**

MHD. ALWI AZMI HARAHAHAP

NPM. ISI 2209020154

PROGRAM STUDI TEKNOLOGI INFORMASI

FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI

UNIVERSITAS MUHAMMADIYAH SUMATERA UTARA

MEDAN

2026

LEMBAR PENGESAHAN

Judul Skripsi : IMPLEMENTASI *PRIORITY SCHEDULING* UNTUK
OPTIMASI PERFORMA GAMING PADA SISTEM
OPERASI WINDOWS
Nama Mahasiswa : MHD. ALWI AZMI HARAHAP
NPM : 2209020154
Program Studi : TEKNOLOGI INFORMASI

Menyetujui
Komisi Pembimbing



(Farid Akbar Siregar, S.kom., M.Kom)
NIDN. 0104049401

Ketua Program Studi



(Fatma Sari Hutagalung, S.Kom, M.Kom)
NIDN. 0117019301

Dekan



(Dr. Al-Khawalizmi, S.Kom., M.Kom.)
NIDN. 0127099201

PERNYATAAN ORISINALITAS

**IMPLEMENTASI *PRIORITY SCHEDULING* UNTUK OPTIMASI
PERFORMA GAMING PADA SISTEM OPERASI WINDOWS**

SKRIPSI

Saya menyatakan bahwa karya tulis ini adalah hasil karya sendiri, kecuali beberapa kutipan dan ringkasan yang masing-masing disebutkan sumbernya.

Medan, Juli 2026

Yang membuat pernyataan



MHD. ALWI AZMI HARAHAHAP

NPM. 2209020154

**PERNYATAAN PERSETUJUAN PUBLIKASI
KARYA ILMIAH UNTUK KEPENTINGAN
AKADEMIS**

Sebagai sivitas akademika Universitas Muhammadiyah Sumatera Utara, saya bertanda tangan dibawah ini:

Nama : MHD. ALWI AZMI HARAHAHAP
NPM : 2209020154
Program Studi : TEKNOLOGI INFORMASI
Karya Ilmiah : Skripsi

Demi pengembangan ilmu pengetahuan, menyetujui untuk memberikan kepada Universitas Muhammadiyah Sumatera Utara Hak Bebas Royalti Non-Eksekutif (*Non-Exclusive Royalty free Right*) atas penelitian skripsi saya yang berjudul:

**IMPLEMENTASI *PRIORITY SCHEDULING* UNTUK OPTIMASI
PERFORMA GAMING PADA SISTEM OPERASI WINDOWS**

Beserta perangkat yang ada (jika diperlukan). Dengan Hak Bebas Royalti Non-Eksekutif ini, Universitas Muhammadiyah Sumatera Utara berhak menyimpan, mengalih media, memformat, mengelola dalam bentuk database, merawat dan mempublikasikan Skripsi saya ini tanpa meminta izin dari saya selama tetap mencantumkan nama saya sebagai penulis dan sebagai pemegang dan atau sebagai pemilik hak cipta.

Demikian pernyataan ini dibuat dengan sebenarnya.

Medan, juli 2026

Yang membuat pernyataan



MHD. ALWI AZMI HARAHAHAP

NPM. 2209020154

RIWAYAT HIDUP

DATA PRIBADI

Nama Lengkap : MHD. ALWI AZMI HARAHAAP

Tempat dan Tanggal Lahir : Medan, 03 September 2004

Alamat Rumah : Komplek Tasbih 1 Blk. LL No. 17,Tj. Rejo
: Kec. Medan Sunggal,Kota Medan, Sumatra
: Utara 20122

Telepon/Faks/HP : 0813-1154-2646

E-mail : Muhammadalwi104020@Gmail.com

Instansi Tempat Kerja : -

Alamat Kantor : -

DATA PENDIDIKAN

SD : UPT SD NEGERI PERCOBAAN TAMAT: 2016

SMP : SMP SIT AL MUSABBIHIN TAMAT: 2019

SMA : SMKS TIK DARUSSALAM TAMAT: 2022

KATA PENGANTAR



Alhamdulillahirabbilalamin, segala puji dan syukur penulis panjatkan ke hadirat Allah SWT, karena hanya dengan rahmat, taufik, serta hidayah-Nya penulis dapat menyelesaikan skripsi yang berjudul “Analisis Pengalaman Pemain Game Kompetitif Terhadap Latensi Menggunakan Pendekatan *Quality of Experience*” dengan baik dan tepat waktu.

Shalawat dan salam senantiasa tercurah kepada junjungan Nabi Muhammad SAW yang telah membawa umat manusia dari zaman kegelapan menuju zaman terang benderang penuh ilmu pengetahuan. Skripsi ini disusun sebagai salah satu syarat mutlak untuk menyelesaikan pendidikan Strata Satu (S-1) pada Program Studi Teknologi Informasi, Fakultas Ilmu Komputer dan Teknologi Informasi (FIKTI), Universitas Muhammadiyah Sumatera Utara (UMSU).

Penulis tentunya berterima kasih kepada berbagai pihak dalam dukungan serta doa dalam penyelesaian skripsi. Penulis juga mengucapkan terima kasih kepada:

1. Bapak Prof. Dr. Akrim, M.PD., Rektor Universitas Muhammadiyah Sumatera Utara (UMSU).
2. Bapak Assoc. Dr. Al-Khowarizmi, S.Kom., M.Kom. Dekan Fakultas Ilmu Komputer dan Teknologi Informasi (FIKTI) UMSU.
3. Ibu Dr. Firahmi Rizky, S.Kom., M.Kom., selaku Wakil Dekan I, atas bimbingan dan dukungan yang telah diberikan sehingga skripsi ini dapat terselesaikan dengan baik.
4. Bapak Mhd Basri, S.Si., M.Kom., selaku Wakil Dekan III, atas bimbingan, arahan, dan dukungan yang telah diberikan selama proses penyusunan skripsi ini
5. Ibu Fatma Sari Hutagalung, S.Kom., M.Kom. Ketua Program Studi Teknologi Informasi..

6. Bapak Okvi Nugroho, S.Kom., M.Kom Sekretaris Program Studi Teknologi Informasi
7. Bapak Farid Akbar Siregar, S.Kom., M.Kom. Selaku Dosen Pembimbing yang telah memberikan arahan, masukan, serta bimbingan selama proses penyusunan skripsi ini.
8. Ayah dan Ibunda tercinta Arman dan Sriana yang selalu mendoakan, memberikan kasih sayang yang tak terhingga, doa yang sangat besar, serta pengorbanan yang menjadi sumber kekuatan utama bagi penulis.
9. Nenek tersayang Rosmina yang senantiasa memberikan nasihat bijak dan doa restu di setiap langkah penulis.
10. Almarhum Kakek Hasan Basri, yang teladan dan kenangannya selalu hidup serta menjadi motivasi bagi penulis untuk terus berusaha memberikan yang terbaik.
11. Sahabat-sahabat terbaik penulis yang selalu ada untuk saling membantu, menghibur, dan berjuang bersama dari awal hingga akhir.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki kekurangan karena keterbatasan pengetahuan dan kemampuan. Oleh karena itu, kritik dan saran yang membangun sangat diharapkan demi penyempurnaan karya ini. Semoga Tugas Akhir ini bermanfaat bagi penulis dan pembaca.

Wassalamu'alaikum warahmatullahi wabarakaatuh

Medan, 14 Agustus 2026



Mhd. Alwi Azmi Harahap

IMPLEMENTASI *PRIORITY SCHEDULING* UNTUK OPTIMASI PERFORMA GAMING PADA SISTEM OPERASI WINDOWS

ABSTRAK

Penelitian ini bertujuan untuk mengimplementasikan metode Priority Scheduling dalam pengembangan perangkat lunak optimasi performa gaming pada sistem operasi Windows. Permasalahan utama yang diatasi adalah fenomena micro-stutters dan latensi input yang disebabkan oleh pembagian kuantum waktu CPU secara merata antara proses permainan dan ratusan proses latar belakang sistem operasi. Perangkat lunak bernama CorePriority Optimizer dikembangkan menggunakan arsitektur hibrida Electron.js, React.js, dan PowerShell Persistent Session untuk melakukan automasi manipulasi Process Priority Class, pengaturan registri Multimedia Class Scheduler Service (MMCSS), serta perapatan High-Resolution Timer ke 0,5 ms secara real-time. Pengujian dilakukan pada lima objek permainan (Honkai: Star Rail, Wuthering Waves, dan Zenless Zone Zero) dengan metode pengulangan 10 kali per skenario menggunakan instrumen analitik CapFrameX. Hasil pengujian menunjukkan peningkatan stabilitas 1% Low FPS sebesar +25,6% pada Honkai: Star Rail, +31,5% pada Wuthering Waves, dan +31,5% pada Zenless Zone Zero, serta peningkatan 0,1% Low FPS hingga +167,1%, reduksi beban CPU hingga 8,51%, dan penurunan suhu operasional CPU hingga 8,21 °C pada prasetel Maximum. Mekanisme keamanan Protected System Blocklist dan Auto-Restore Snapshot terbukti 100% efektif menjaga integritas sistem operasi selama 2000 menit total durasi pengujian tanpa terjadi crash maupun BSOD.

Kata Kunci: Priority Scheduling; Optimasi Gaming; Micro-Stutters; High-Resolution Timer; MMCSS; Electron.js; PowerShell

IMPLEMENTATION OF PRIORITY SCHEDULING FOR GAMING PERFORMANCE OPTIMIZATION ON THE WINDOWS OPERATING SYSTEM

ABSTRACT

This research aims to implement the Priority Scheduling method in developing a gaming performance optimization software on the Windows operating system. The primary issue addressed is the micro-stutters and input latency phenomenon caused by the equal distribution of CPU time quantum between game processes and hundreds of background operating system processes. The software, named CorePriority Optimizer, was developed using a hybrid architecture of Electron.js, React.js, and PowerShell Persistent Session to automate the manipulation of Process Priority Class, Multimedia Class Scheduler Service (MMCSS) registry settings, and High-Resolution Timer adjustment to 0.5 ms in real-time. Testing was conducted on three game objects (Honkai: Star Rail, Wuthering Waves, and Zenless Zone Zero) with 10 repetitions per scenario using the CapFrameX analytical instrument. The test results demonstrated a 1% Low FPS stability improvement of +25.6% on Honkai: Star Rail, +31.5% on Wuthering Waves, and +31.5% on Zenless Zone Zero, along with a 0.1% Low FPS increase up to +167.1%, CPU load reduction up to 8.51%, and operational temperature decrease up to 8.21 °C on the Maximum preset. The Protected System Blocklist and Auto-Restore Snapshot security mechanisms proved 100% effective in maintaining operating system integrity throughout 1200 minutes of total testing duration without any crash or BSOD occurrence.

Keywords: Priority Scheduling; Gaming Optimization; Micro-Stutters; High-Resolution Timer; MMCSS; Electron.js; PowerShell

DAFTAR ISI

LEMBAR PENGESAHAN	i
PERNYATAAN ORISINALITAS	ii
PERNYATAAN PERSETUJUAN PUBLIKASI KARYA ILMIAH	iii
RIWAYAT HIDUP	iv
KATA PENGANTAR	v
ABSTRAK	vii
ABSTRACT	viii
DAFTAR ISI	ix
DAFTAR GAMBAR	xi
DAFTAR TABEL	xii
BAB I. PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	4
1.3 Batasan Masalah	4
1.4 Tujuan Penelitian	5
1.5 Manfaat Penelitian	5
BAB II. LANDASAN TEORI	7
2.1 Konsep Dasar Kinerja Komputasi dan Kualitas Grafis	7
2.1.1 Pengertian Optimasi	7
2.1.2 <i>Central Processing Unit (CPU)</i>	7
2.1.3 Frame Per Second (FPS)	8
2.1.4 <i>Micro Stutters</i>	8
2.2 Manajemen Penjadwalan Proses pada Sistem Operasi	9
2.2.1 Konsep <i>Priority Scheduling</i>	10
2.2.2 Process Priority Class	12
2.2.3 Multimedia Class Scheduler Service (MMCSS)	14
2.3 Mekanisme Sinkronisasi dan Waktu Komputasi Waktu Nyata	16
2.3.1 High-Resolution Timer (Resolusi Timer Tingkat Tinggi)	16
2.3.2 Fungsi <i>timeBeginPeriod()</i>	17
2.4 Teknologi Pengembangan Perangkat Lunak Desktop Hibrida	19
2.4.1 Arsitektur Kerangka <i>Electron.js</i>	19
2.4.2 Antarmuka Reaktif <i>React.js</i> dan Keamanan <i>TypeScript</i>	21
2.4.3 Optimalisasi Sesi Eksekusi <i>PowerShell Persistent</i>	21
2.5 Mekanisme Keamanan Perlindungan Sistem Perangkat Keras	22
2.5.1 Konsep Validasi <i>Protected System Blocklist</i>	22
2.5.2 Mekanisme Pemulihan <i>Auto-Restore Snapshot</i>	24

2.6	Tinjauan Pustaka dan Kajian Penelitian Terdahulu	25
BAB III. METODE PENELITIAN.....		28
3.1	Alur dan Kerangka Kerja Penelitian (<i>Research Framework</i>).....	28
3.2	Kerangka Konsep Penelitian.....	29
3.3	Perancangan Model Sistem(UML)	30
3.3.1	Use Case Diagram.....	30
3.3.2	Flowchart Proses Optimasi Utama.....	31
3.4	Analisis Arsitektur Sistem Rekayasa Utilitas	34
3.4.1	Skema Komunikasi <i>Multi-Proses (Electron & React)</i>	35
3.4.2	Modul Pengendalian PowerShell Persistent	36
3.5	Prosedur Implementasi Rekayasa Modifikasi Kinerja (Optimasi)	37
3.5.1	Modifikasi Hierarki Prioritas Proses (<i>Priority Scheduling</i>).....	37
3.5.2	Implementasi Profil Heuristik Tugas <i>MMCSS</i>	37
3.5.3	Substitusi Jeda Timer Resolusi (<i>High-Resolution Timer</i>).....	39
3.6	Implementasi Lapisan Verifikasi Keamanan OS	40
3.6.1	Validasi <i>Protected System Blocklist</i>	40
3.6.2	Logika Penyimpanan Parameter (<i>Auto-Restore Snapshot</i>).....	41
3.7	Skenario Uji Operasional dan Parameter Metrik Evaluasi	44
3.7.1	Definisi Batasan Lingkungan Pengujian dan Objek Uji	44
3.7.2	Kualifikasi Skenario Prasetel Konfigurasi	45
3.8	Perangkat Instrumen Pengukuran	46
3.9	Teknik Analisis Data.....	47
3.10	Jadwal Pelaksanaan Penelitian.....	48
BAB IV. HASIL DAN PEMBAHASAN.....		49
4.1	Hasil Implementasi Sistem.....	49
4.2	Hasil Pengujian Skenario Optimasi	52
4.2.1	Hasil Pengujian Performa Game 1: Honkai: Star Rail (HSR)	54
4.2.2	Hasil Pengujian Performa Game 2: Wuthering Waves (WuWa).....	56
4.2.3	Hasil Pengujian Performa Game 3: Zenless Zone Zero (ZZZ).....	58
4.2.4	Hasil Pengujian Performa Game 4: HoloCure.....	61
4.2.5	Hasil Pengujian Performa Game 5: Hololive Dreams	64
4.3	Pembahasan.....	67
BAB V. KESIMPULAN DAN SARAN.....		70
5.1	Kesimpulan	70
5.2	Saran.....	72
DAFTAR PUSTAKA		

DAFTAR GAMBAR

Gambar 2. 1 Micro- Stutters	9
Gambar 2. 2 Priority Scheduling & Starvation	11
Gambar 2. 3 Hierarki Process Priority Class	13
Gambar 2. 4 Alur MMCSS & SystemResponsiveness	15
Gambar 2. 5 High-Resolution Timer	17
Gambar 2. 6 timeBeginPeriod() Scope Change	18
Gambar 2. 7 Arsitektur Electron.js	20
Gambar 2. 8 Protected System Blocklist	23
Gambar 2. 9 Auto-Restore Snapshot.....	25
Gambar 3. 1 Use Case Diagram.....	30
Gambar 3. 2 Flowchart Proses Optimasi	33
Gambar 3. 3 Arsitektur Sistem.....	34
Gambar 3. 4 Skema Komunikasi Multi-Proses.....	36
Gambar 3. 5 Tiga Lapis Perlindungan	41
Gambar 3. 6 Auto-Restore (Background Watcher)	43
Gambar 4. 1 Antarmuka Utama Aplikasi CorePriority Optimizer	50
Gambar 4. 2 Status Log Feed dan Eksekusi Backend PowerShell	50
Gambar 4. 3 Empirisme Eksekusi Priority Class Windows Task Manager.....	51
Gambar 4. 4 Tangkapan Layar Lingkungan Permainan Honkai: Star Rail	54
Gambar 4. 5 Grafik Bar Komparasi Kinerja dan Stabilitas Honkai: Star Rail	55
Gambar 4. 6 Tangkapan Layar Lingkungan Permainan Wuthering Waves	56
Gambar 4. 7 Grafik Bar Komparasi Kinerja dan Stabilitas Wuthering Waves	58
Gambar 4. 8 Tangkapan Layar Lingkungan Permainan Zenless Zone Zero	59
Gambar 4. 9 Grafik Bar Komparasi Kinerja dan Stabilitas Zenless Zone Zero ...	60
Gambar 4. 10 Tangkapan Layar Lingkungan Permainan HoloCure	62
Gambar 4. 11 Grafik Bar Komparasi Kinerja dan Stabilitas HoloCure.....	63
Gambar 4. 12 Tangkapan Layar Lingkungan Permainan Hololive Dreams.....	65
Gambar 4. 13 Grafik Bar Komparasi Kinerja dan Stabilitas Hololive Dreams	66

DAFTAR TABEL

Tabel 2. 1 Process Priority Class	12
Tabel 2. 2 Penelitian Terdahulu	25
Tabel 3. 1 Detail Use Case	31
Tabel 3. 2 Layer Breakdown Sistem	34
Tabel 3. 3 Configuration Presets	45
Tabel 3. 5 Waktu Penelitian	48
Tabel 4. 1 Spesifikasi Perangkat Keras dan Lingkungan Pengujian	52
Tabel 4. 2 Hasil Pengujian Kinerja Komparatif Honkai: Star Rail	55
Tabel 4. 3 Hasil Pengujian Kinerja Komparatif Wuthering Waves	57
Tabel 4. 4 Hasil Pengujian Kinerja Komparatif Zenless Zone Zero	60
Tabel 4. 5 Hasil Pengujian Kinerja Komparatif HoloCure	63
Tabel 4. 6 Hasil Pengujian Kinerja Komparatif Hololive Dreams	66

BAB I.

PENDAHULUAN

1.1 Latar Belakang

Perkembangan industri gaming di Indonesia mengalami pertumbuhan yang sangat pesat dalam beberapa tahun terakhir. Berdasarkan data Statista tahun 2024, Indonesia tercatat sebagai salah satu pasar gaming terbesar di Asia Tenggara dengan jumlah pengguna mencapai lebih dari 50 juta jiwa. Fenomena ini mendorong semakin tingginya kebutuhan pengguna terhadap performa komputasi yang optimal dalam menjalankan aplikasi game berbasis personal computer (PC) (Liliani Suciasih & Sugeng Santoso, 2024).

Sistem operasi Windows merupakan platform dominan yang digunakan oleh para gamer PC di seluruh dunia, termasuk Indonesia. Dalam ekosistemnya, Windows mengelola sumber daya prosesor (CPU) melalui mekanisme penjadwalan proses (process scheduling) yang bersifat sekuensial dan setara. Mekanisme bawaan ini memberikan perlakuan yang sama antara proses game yang membutuhkan komputasi intensif dengan ratusan proses latar belakang (background process) seperti layanan sistem, antivirus, pembaruan otomatis, dan telemetri yang berjalan secara bersamaan (Yang et al., 2020).

Pembagian daya komputasi secara merata antara game dan proses latar belakang sering kali memicu sejumlah permasalahan teknis yang secara langsung menurunkan kualitas pengalaman bermain pengguna. Permasalahan tersebut meliputi fenomena micro-stutters (patah-patah sesaat pada animasi), peningkatan latensi input (jeda antara perintah dari pengguna hingga respons layar), serta penurunan *Frame Per Second* (FPS) yang tidak stabil. Kondisi ini terjadi karena

CPU terpaksa berbagi waktu eksekusi (CPU time) dengan proses-proses yang tidak berkaitan dengan game, sehingga game tidak pernah dapat memanfaatkan kapasitas penuh perangkat keras yang tersedia (Yi et al., 2022).

Permasalahan ini semakin dirasakan pada komputer dengan spesifikasi menengah ke bawah, di mana sumber daya CPU yang terbatas harus dibagi dengan semakin banyaknya proses sistem yang berjalan. Jika masalah interupsi penjadwalan ini tidak diatasi, performa aplikasi berbeban berat seperti game tidak akan pernah mencapai potensi maksimal dari perangkat keras yang tersedia, meskipun komputer tersebut secara teoritis memiliki kemampuan yang memadai (Savarimalai et al., 2026).

Salah satu pendekatan teknis yang dapat digunakan untuk mengatasi permasalahan tersebut adalah manipulasi *Process Priority Class* pada sistem operasi Windows. *Priority Scheduling* merupakan metode penjadwalan proses di mana *Central Processing Unit* (CPU) mendistribusikan waktu komputasi berdasarkan tingkatan bobot prioritas yang ditetapkan. Proses dengan prioritas lebih tinggi akan mendapatkan akses CPU lebih besar dan dieksekusi lebih dahulu dibandingkan proses dengan prioritas lebih rendah (Hajjar et al., 2024).

Selain manipulasi prioritas proses, faktor lain yang memengaruhi performa gaming adalah resolusi timer sistem operasi. Secara standar, Windows menggunakan resolusi timer sebesar 15,6 milidetik. Resolusi yang besar ini menyebabkan latensi sinkronisasi antara CPU dan GPU yang berujung pada micro-stutters pada rendering grafis. Dengan menerapkan *High-Resolution Timer* melalui fungsi *timeBeginPeriod()*, resolusi dapat ditarik ke 1 ms atau bahkan 0,5

ms, sehingga sinkronisasi antara CPU dan GPU menjadi jauh lebih stabil (Xing et al., 2025)

Meskipun teknik penjadwalan proses (process scheduling) telah banyak dibahas dalam berbagai literatur keilmuan teknologi informasi, penelitian-penelitian sebelumnya umumnya lebih berfokus pada pengujian simulasi algoritma, efisiensi beban kerja basis data, atau manajemen multitasking komputasi umum. Masih sangat minim penelitian terapan yang secara spesifik mengangkat optimalisasi *Priority Scheduling* ke arah performa gaming secara langsung di level sistem operasi. Kesenjangan penelitian (research gap) inilah yang memicu kebuntuan bagi para pengguna; di mana selama ini mereka terpaksa melakukan pengaturan prioritas secara manual melalui Task Manager yang tidak permanen, atau menggunakan modifikasi pihak ketiga yang berisiko mengubah berkas registry dan mengancam stabilitas kernel Windows (Jadon et al., 2024).

Berdasarkan latar belakang tersebut, penelitian ini mengembangkan sebuah perangkat lunak optimasi gaming berbasis *Electron.js*, *React.js*, *TypeScript*, dan sesi *PowerShell Persistent* yang mampu secara dinamis dan aman memodifikasi *Process Priority Class* serta mengatur resolusi timer pada sistem operasi Windows. Aplikasi ini dilengkapi dengan mekanisme keamanan berupa *Protected System Blocklist* dan *Auto-Restore Snapshot* untuk menjamin kestabilan sistem operasi selama dan setelah sesi optimasi berlangsung (Ramadhani & Firdaus, 2025).

Berpijak pada kesenjangan penelitian tersebut serta urgensi akan solusi teknis yang aman dan otomatis, maka perancangan algoritma penjadwalan utilitas ini menjadi inti utama dari sistem yang dikembangkan (Sakshi et al., 2022).

Oleh karena itu, untuk merepresentasikan batasan masalah, metode yang digunakan, dan tujuan akhir dari pengembangan perangkat lunak ini secara komprehensif, penulis mengangkat judul penelitian: “Implementasi *Priority Scheduling* untuk Optimasi Performa Gaming pada Sistem Operasi Windows”.

1.2 Rumusan Masalah

Bagaimana cara mengoptimalkan pembagian waktu eksekusi CPU (*CPU time*) untuk mengatasi interupsi proses latar belakang, sehingga permasalahan penurunan performa *gaming* (seperti *micro-stutters*, latensi input, dan instabilitas FPS) pada komputer berspesifikasi terbatas dapat diminimalisir dan potensi perangkat keras dapat dimanfaatkan secara maksimal?.

1.3 Batasan Masalah

Berdasarkan latar belakang dan identifikasi masalah yang telah diuraikan sebelumnya, penelitian ini dibatasi pada ruang lingkup berikut:

1. Penelitian ini dibatasi pada pengembangan aplikasi desktop (berbasis Node.js, Electron.js, dan React.js) eksklusif untuk *Windows* 10 dan 11 yang mengandalkan skrip PowerShell.
2. Fokus optimasi hanya mencakup manipulasi *Process Priority Class*, profil MMCSS, dan *High-Resolution Timer*, tanpa melibatkan modifikasi permanen pada berkas sistem dan *registry*, maupun optimasi pada level jaringan, kernel, dan *driver* GPU.
3. Difokuskan pada perancangan Protected System Blocklist untuk mencegah ketidakstabilan sistem (BSOD) atau kegagalan proses kritis

4. Evaluasi performa sistem selanjutnya diukur berdasarkan perbandingan metrik FPS, latensi input, CPU Load dan CPU Temp antara kondisi *baseline* dengan tiga level prasetel optimasi.

1.4 Tujuan Penelitian

Tujuan utama penelitian ini adalah mengembangkan dan menguji aplikasi untuk mendongkrak performa *gaming* pada sistem operasi Windows melalui modifikasi penjadwalan CPU. Selain itu, penelitian ini juga bertujuan membangun sistem *backend* dengan fitur *auto-restore snapshot* untuk mengembalikan konfigurasi awal, mengimplementasikan *Protected System Blocklist* demi menjaga stabilitas sistem operasi, serta menganalisis efektivitas tiga level prasetel konfigurasi (Minimum, Normal, Maximum) terhadap peningkatan metrik FPS, latensi input, CPU Load dan CPU Temp.

1.5 Manfaat Penelitian

Hasil penelitian ini diharapkan dapat memberikan manfaat bagi beberapa pihak, yaitu sebagai berikut:

1. Bagi Institusi (UMSU) Penelitian ini diharapkan dapat menjadi referensi akademik dalam bidang sistem operasi dan rekayasa perangkat lunak, serta memperkaya koleksi karya ilmiah Fakultas Ilmu Komputer dan Teknologi Informasi UMSU.
2. Bagi Keilmuan Teknologi Informasi Penelitian ini membuktikan efektivitas manipulasi *Process Priority Class* di level sistem operasi sebagai pendekatan teknis yang valid untuk optimasi aplikasi berbeban tinggi, sehingga dapat menjadi landasan bagi penelitian selanjutnya.

3. Bagi Masyarakat dan Pengguna PC Penelitian ini menghadirkan aplikasi sebagai solusi optimasi gaming tanpa risiko modifikasi permanen yang menjamin kestabilan sistem operasi, sehingga dapat dimanfaatkan oleh para gamer PC dari berbagai kalangan.
4. Bagi Penulis Penelitian ini memberikan pengalaman praktis dalam pengembangan perangkat lunak berbasis *Electron.js*, *React.js*, dan *PowerShell*, serta memperdalam pemahaman tentang mekanisme penjadwalan proses pada sistem operasi Windows.

BAB II. LANDASAN TEORI

2.1 Konsep Dasar Kinerja Komputasi dan Kualitas Grafis

Pengembangan perangkat lunak untuk mengelola dan mendongkrak kinerja sistem membutuhkan landasan pemahaman yang mendalam mengenai arsitektur perangkat keras, interaksi dengan sistem operasi, serta bagaimana perangkat keras memproses instruksi yang berdampak langsung pada pengalaman pengguna akhir. Dalam konteks sistem operasi modern, manajemen interupsi dan alokasi daya pemrosesan menjadi kunci utama dari kelancaran aplikasi (Jadon et al., 2024).

2.1.1 Pengertian Optimasi

Optimasi dalam rekayasa perangkat lunak merupakan serangkaian proses sistematis untuk memodifikasi sistem agar beroperasi pada efisiensi maksimum, mencakup eksekusi instruksi yang lebih cepat, pemanfaatan *bandwidth* yang lebih baik, serta konsumsi memori dan *CPU* yang lebih hemat. Dalam konteks sistem operasi, optimasi mengacu pada rekonfigurasi kernel, layanan latar belakang, dan algoritma penjadwalan agar perangkat keras dapat memusatkan daya komputasi pada aplikasi prioritas, sehingga tidak ada instruksi prioritas tinggi yang tertunda akibat persaingan sumber daya dengan proses berkepentingan lebih (Jadon et al., 2024; Yanwari et al., 2023).

2.1.2 Central Processing Unit (CPU)

Central Processing Unit (CPU) adalah sirkuit elektronik fundamental yang mengeksekusi instruksi program melalui operasi aritmatika, logika, kontrol,

dan *I/O*, sekaligus menjadi penentu utama kecepatan dan efisiensi sistem komputasi. Dalam sistem modern, *CPU* berperan sebagai sumber daya kritis yang dibagi ribuan thread secara paralel, mencakup pemrosesan logika game, fisika permainan, kecerdasan buatan, input pengguna, serta persiapan dan pengiriman *draw calls* ke *GPU*. Apabila *CPU* terbebani tugas latar belakang, pengiriman *draw calls* akan tertunda sehingga *GPU* berada dalam kondisi idle, yang pada akhirnya mengganggu kelancaran tampilan visual. Oleh karena itu, optimalisasi *CPU time* sangat krusial untuk menjaga sinergi arsitektur perangkat keras heterogen (Fadhilah & Siregar, 2025).

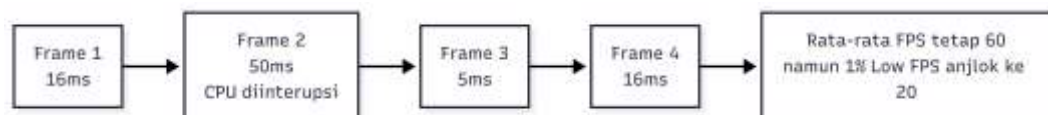
2.1.3 Frame Per Second (FPS)

Frame Per Second (FPS) adalah satuan ukur standar kinerja visual yang menghitung jumlah frame yang dirender *GPU* dan ditampilkan ke layar per detik. Industri gaming kompetitif modern umumnya menargetkan minimal 60 *FPS* untuk kelancaran dasar, sementara perangkat keras tingkat lanjut dapat mencapai 144 *FPS*, 240 *FPS*, atau lebih tinggi. *FPS* yang tinggi tidak hanya meningkatkan estetika animasi, tetapi juga mereduksi latensi visual antara tindakan fisik pengguna dan respons di layar. Namun, metrik rata-rata *FPS* memiliki keterbatasan mendasar karena tidak menjamin konsistensi interval antar frame — rata-rata 60 *FPS* tidak memastikan setiap *frame* dikirim dalam interval ideal sekitar 16,6 milidetik (Sakshi et al., 2022).

2.1.4 Micro Stutters

Micro stutters adalah artefak visual pada grafis *real-time* di mana interval antar *frame* yang ditampilkan secara berurutan menjadi sangat tidak konsisten,

meskipun rata-rata *FPS* tampak memadai. Sebagai ilustrasi, pada sistem 60 *FPS* dengan *frametime* ideal 16,6 ms, *micro stutters* dapat menyebabkan satu frame membutuhkan waktu 50 ms sementara frame berikutnya hanya 5 ms, sehingga alur animasi tidak tersinkronisasi dan pengguna mempersepsikannya sebagai patah-patah, hentakan (*hitching*), atau lag seketika. Penyebabnya beragam, mulai dari kompilasi shader saat *runtime*, kehabisan *VRAM*, hingga *streaming* aset dari disk yang lambat. Dalam konteks penjadwalan proses, *micro stutters* kerap lahir dari *CPU interrupt saturation*, yakni kondisi di mana *OS* mengalihkan waktu *CPU* untuk menangani interupsi latar belakang seperti layanan telemetri atau komputasi *I/O* jaringan, sehingga proses rendering utama terhenti sejenak dan menciptakan lonjakan *frametime*. Untuk mengatasi keterbatasan metrik rata-rata *FPS*, evaluasi performa modern beralih ke analisis 1% *Lows* dan 0.1% *Lows* yang melacak persentil frame paling lambat, memberikan representasi kehalusan visual yang jauh lebih objektif (Sakshi et al., 2022).



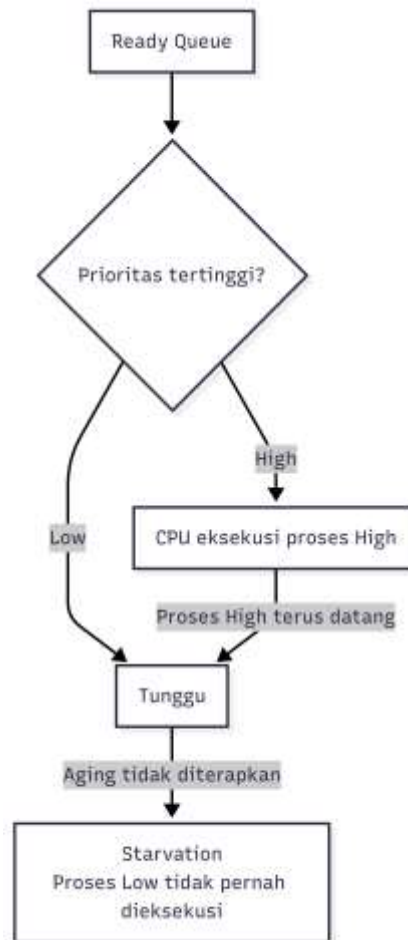
Gambar 2. 1 Micro- Stutters

2.2 Manajemen Penjadwalan Proses pada Sistem Operasi

Sistem operasi (*OS*) berfungsi sebagai entitas manajemen utama yang menjembatani perangkat lunak aplikasi dengan instruksi sirkuit perangkat keras. Tanggung jawab paling kritis dari *OS modern* adalah mengoordinasikan bagaimana dan kapan berbagai tugas (proses) diizinkan untuk dieksekusi oleh prosesor.

2.2.1 Konsep *Priority Scheduling*

CPU Scheduling adalah prosedur fundamental *OS* dalam memilih dan mengalokasikan proses dari *ready queue* untuk menggunakan *CPU*, sedangkan *Priority Scheduling* adalah bentuk spesifiknya di mana setiap proses dan thread diberi bobot prioritas berupa nilai numerik. Dalam mode *preemptive scheduling*, proses berprioritas lebih tinggi yang memasuki antrean akan memaksa penghentian proses yang sedang berjalan dan mengambil alih kendali *CPU*, memastikan beban komputasi *time-critical* dapat segera dieksekusi tanpa menunggu instruksi latensi tinggi lainnya. Meski demikian, *Priority Scheduling* murni rentan terhadap fenomena *starvation*, yakni kondisi di mana proses berprioritas rendah tidak pernah mendapat giliran eksekusi akibat monopoli proses berprioritas tinggi yang terus-menerus tiba. Untuk mengatasinya, *OS* modern seperti kernel *Windows NT* menerapkan teknik aging dan algoritma hibrida yang menggabungkan *Priority Scheduling* dengan *Round Robin* guna mencegah kelumpuhan sistem sekaligus mempertahankan latensi rendah pada layar depan pengguna (Ramadhani & Firdaus, 2025).



Gambar 2. 2 Priority Scheduling & Starvation

Meskipun efektif dalam memastikan fungsionalitas *real-time*, *Priority Scheduling* murni memiliki kerentanan arsitektural yang dikenal sebagai *starvation* atau *indefinite blocking*, yakni kondisi di mana serangkaian proses berprioritas tinggi yang terus-menerus tiba memonopoli *CPU* sehingga proses berprioritas rendah dibekukan dan tidak pernah mendapat giliran eksekusi. Untuk mengatasi kerentanan ini, *OS* kompleks seperti kernel *Windows NT* menerapkan modifikasi *heuristik* berlapis, antara lain teknik *aging* yang secara bertahap meningkatkan prioritas proses yang telah lama menunggu, serta algoritma hibrida yang menggabungkan *Priority Scheduling* dengan Round Robin menggunakan irisan waktu kustom. Pendekatan ini dirancang untuk mencegah kelumpuhan

sistem sekaligus tetap mempertahankan latensi rendah pada proses layar depan pengguna (Ramadhani & Firdaus, 2025).

2.2.2 Process Priority Class

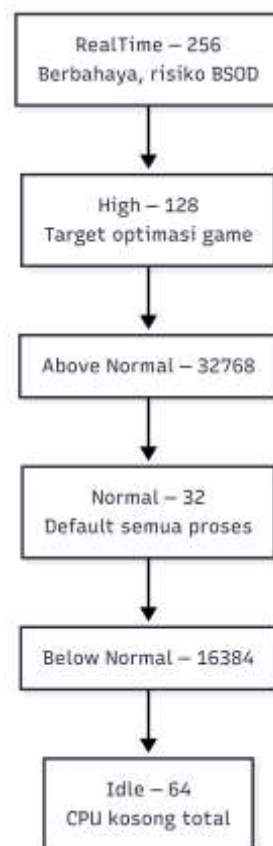
Kernel Windows mengimplementasikan Priority Scheduling melalui kerangka klasifikasi berlapis yang dikenal sebagai Process Priority Class, di mana pengelompokan ini tidak menentukan satu nilai prioritas yang pasti, melainkan mengatur spektrum rentang prioritas yang mendasari sekumpulan utas dalam satu aplikasi. *Windows API Win32* mendefinisikan enam kelas prioritas hierarkis yang masing-masing menentukan bagaimana thread scheduler menilai urgensi suatu proses relatif terhadap seluruh proses *OS* lainnya, yaitu:

Tabel 2. 1 Process Priority Class

Nama Kelas	Nilai	Fungsi Utama dalam Sistem
Prioritas	Desimal	
<i>Idle</i>	64	Digunakan untuk tugas latar belakang pasif (<i>background maintenance</i>).
<i>Below Normal</i>	16384	Aplikasi sekunder yang tidak boleh mengganggu responsivitas sistem.
<i>Normal</i>	32	Pengaturan standar (<i>default</i>) untuk hampir semua aplikasi pengguna.
<i>Above Normal</i>	32768	Memberikan prioritas sedikit di atas aplikasi standar.
<i>High</i>	128	Untuk tugas kritis (<i>time-critical</i>) yang butuh sumber daya segera.

<i>RealTime</i>	256	Prioritas tertinggi; mampu menginterupsi fungsi dasar perangkat keras.
-----------------	-----	--

Tabel ini berfungsi sebagai referensi akademik bagi pembaca untuk memahami bahwa angka-angka tersebut adalah standar resmi Windows API. Adapun urutan tingkatan prioritas ini, dari yang paling aman hingga yang paling berisiko bagi sistem, dapat dilihat secara hierarkis pada Gambar 2.3 di bawah ini.

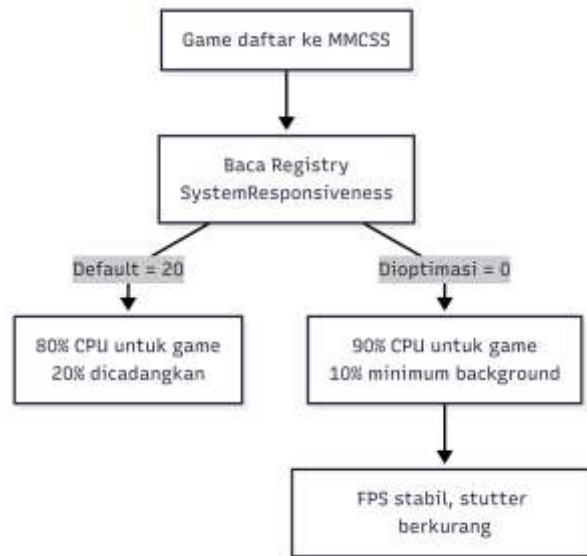


Gambar 2. 3 Hierarki Process Priority Class

Berdasarkan batasan keamanan dan stabilitas di atas, perancangan utilitas untuk optimalisasi *gaming* idealnya mendistribusikan waktu komputasi secara dinamis hanya pada ambang batas *High Priority Class* dikombinasikan dengan teknik manipulasi register *Win32PrioritySeparation*. Praktik ini memaksimalkan serapan alokasi waktu siklus (*quantum*) untuk aplikasi di latar depan (*foreground*) tanpa meniadakan stabilitas fungsional subsistem kernel *Windows*.

2.2.3 Multimedia Class Scheduler Service (MMCSS)

Karena manipulasi *Process Priority Class* secara langsung berisiko terhadap stabilitas sistem, Microsoft menghadirkan lapisan layanan manajerial khusus sejak Windows Vista yang berfokus pada kepekaan waktu beban kerja multimedia dan grafis, yang dikenal sebagai *Multimedia Class Scheduler Service* (MMCSS). Tujuan utama MMCSS adalah menjamin bahwa aplikasi real-time multimedia — mencakup pemutaran audio eksklusif, perangkat lunak sintesis video, dan video game — menerima alokasi CPU dan I/O yang konsisten secara time-sensitive, tanpa risiko membekukan OS secara keseluruhan. Secara terprogram, utas aplikasi dapat mendaftarkan diri ke *API* kernel melalui fungsi *AvSetMmThreadCharacteristics* dengan mendeklarasikan profil kelas seperti "*Pro Audio*", "*Playback*", atau "*Games*", kemudian MMCSS secara dinamis meningkatkan prioritas absolut utas tersebut sesuai kategori penjadwalannya, umumnya pada rentang prioritas 16 hingga 22 untuk latar depan dan hingga 26 untuk audio profesional. Efektivitas heuristik penjadwalan ini bergantung pada parameter konfigurasi dalam *Windows Registry*, khususnya pada kunci *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Multimedia\SystemProfile*, yang memuat register *DWORD* bernama *SystemResponsiveness*. Parameter ini menentukan persentase CPU yang dicadangkan secara eksklusif untuk tugas latar belakang agar OS tetap responsif. Secara default, Windows menetapkan nilai *SystemResponsiveness* sebesar 20%, yang berarti proses MMCSS termasuk game hanya dapat memanfaatkan maksimal 80% kapasitas CPU, sebagaimana diilustrasikan pada gambar berikut (Xing et al., 2025).



Gambar 2. 4 Alur MMCSS & SystemResponsiveness

Bagi komunitas gamer berkecepatan tinggi dan pengembang perangkat lunak optimasi sistem, nilai default *SystemResponsiveness* sebesar 20% dianggap merugikan karena membatasi perolehan latensi terendah yang dibutuhkan aplikasi grafis intensif. Untuk memaksimalkan kinerja game, skrip administratif dimanfaatkan untuk menurunkan nilai *SystemResponsiveness* ke batas minimum numerik, di mana angka 0 divalidasi oleh regulasi Microsoft sebagai cadangan minimum sebesar 10%. Penurunan cadangan sistem secara agresif ini menghasilkan eskalasi signifikan daya transfer perangkat komputasi yang efektif dalam memberantas *crackling audio*, meningkatkan konsistensi frame, serta mengatasi kendala *networking throughput* tingkat rendah. Optimasi ini selanjutnya disempurnakan melalui intervensi pada indeks *Games Profile Tasks*, yang menempatkan parameter *Priority* dan *GPU Priority* pada nilai numerik yang ditujukan untuk pencapaian kinerja maksimum absolut (Miao et al., 2025; Yanwari et al., 2023).

2.3 Mekanisme Sinkronisasi dan Waktu Komputasi Waktu Nyata

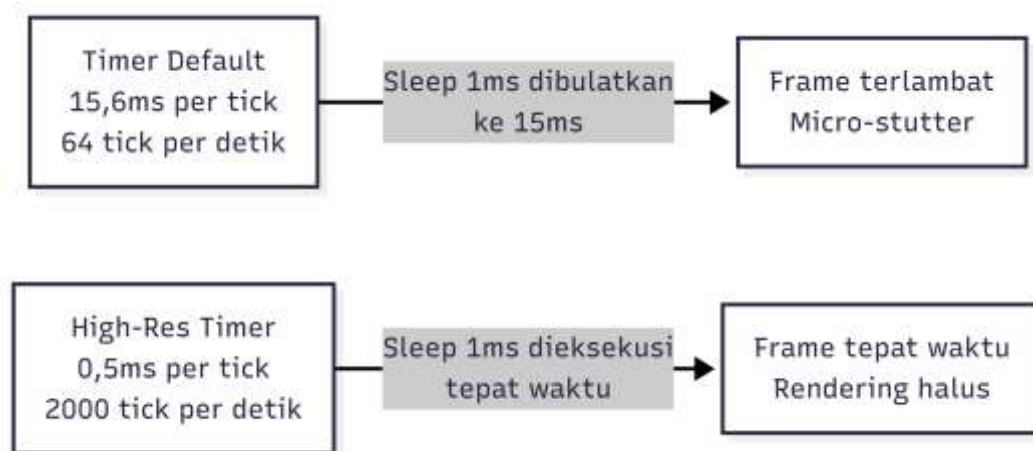
Pemrosesan antrean perintah dari periferal fisik pengguna menuju pementasan bingkai di perangkat layar menuntut presisi koordinasi perangkat lunak hingga pada fraksi seperseribu detik. Keselarasan mekanis ini sepenuhnya dikoordinasikan oleh lapisan pewaktuan dari OS itu sendiri (Sakshi et al., 2022; Savarimalai et al., 2026).

2.3.1 High-Resolution Timer (Resolusi Timer Tingkat Tinggi)

Setiap sistem operasi yang mengelola perangkat keras bergantung pada sinyal detak reguler dari jam sistem melalui mekanisme interupsi timer perangkat keras. *Timer resolution* merepresentasikan *interval* waktu yang berlalu antara setiap pemanggilan *interrupt service routine* oleh sistem. Secara default, Windows menetapkan *interval timer* sebesar kurang lebih 15,6 ms atau sekitar 64 detak interupsi per detik, yang dirancang untuk menyeimbangkan efisiensi daya dan stabilitas sistem. Meskipun konfigurasi ini menguntungkan bagi perangkat seluler dan laptop karena mencegah *CPU* keluar dari idle states secara prematur, granularitas jam sebesar 15,6 ms menciptakan hambatan serius pada skenario komputasi *high-performance*. Sebagai ilustrasi, game yang menargetkan 144 FPS membutuhkan setiap *frame* dirender dalam 6,9 ms; apabila fungsi penundaan seperti *Sleep()* digunakan untuk sinkronisasi, perintah jeda 1 ms akan dibulatkan otomatis hingga interupsi berikutnya tiba dalam 15 ms, mengakibatkan penundaan bertumpuk antar-utas yang berkontribusi langsung pada *micro stutters* dan peningkatan *input latency*. Untuk mengatasi bottleneck ini, penerapan *High-Resolution Timer* memaksa kernel Windows mempersingkat interval interupsi hingga minimum yang didukung perangkat keras, umumnya 1,0 ms atau 0,5 ms,

menghasilkan tick rate hingga 2000 kali per detik. Pada *granularitas* ini, *overhead* siklus penantian utas dipangkas secara radikal sehingga *CPU* dan bus *PCI Express* dapat menyinkronisasikan perakitan *frame* bersama *GPU* dengan selisih latensi yang sangat minimal (Jadon et al., 2024; Miao et al., 2025).

Perbandingan dampak kedua konfigurasi timer tersebut terhadap konsistensi frame diilustrasikan pada gambar berikut.



Gambar 2. 5 High-Resolution Timer

2.3.2 Fungsi *timeBeginPeriod()*

Implementasi *High-Resolution Timer* dalam aplikasi Windows dilakukan melalui fungsi pustaka kernel *timeBeginPeriod()* yang terdapat pada *Multimedia Timers (Winmm.lib)*. Fungsi ini dipanggil sebelum aplikasi memanfaatkan manajemen waktu dengan menyertakan argumen integer yang merepresentasikan interval milidetik yang diminta, seperti *timeBeginPeriod(1)*, dan wajib diakhiri dengan *timeEndPeriod()* untuk mengembalikan efisiensi timer ke kondisi semula. Secara historis, sebelum *Windows 10 Versi 2004*, pemanggilan *timeBeginPeriod()* bersifat global sehingga seluruh subsistem *Windows* ikut terdampak, termasuk proses latar belakang yang tidak memerlukannya, mengakibatkan konsumsi daya

yang tidak efisien. Merespons permasalahan ini, *Microsoft* menerapkan perubahan struktural sejak *Windows 10 Versi 2004* melalui kebijakan yang dikenal sebagai "*The Great Rule Change*", di mana *timeBeginPeriod()* kini hanya berdampak secara terisolasi pada proses aplikasi yang memanggilnya (*per-process scope*), sementara proses latar yang tidak menuntut resolusi tinggi tetap dipertahankan pada interval default 15,6 ms oleh mekanisme timer *coalescing Windows* (Savarimalai et al., 2026; Xing et al., 2025).

Alur kerja mekanisme ini berdasarkan versi *Windows* diilustrasikan pada gambar berikut.



Gambar 2. 6 timeBeginPeriod() Scope Change

Pergeseran kebijakan *per-process scope* pada *Windows 10 Versi 2004* menimbulkan implikasi metodologis bagi pengembangan utilitas optimasi sekunder. Karena utilitas skrip bukan merupakan aplikasi latar depan yang aktif

dimainkan, pemanggilan *API* resolusi timer standar menjadi tidak efektif saat jendela utilitas diminimalkan atau tertutup. Sebagai solusi, utilitas optimasi wajib mengintegrasikan manipulasi tingkat rendah melalui *P/Invoke* berbasis *C#* atau antarmuka tidak berdokumen dari *ntdll.dll* menggunakan instruksi *NtSetTimerResolution*, yang memungkinkan penetapan resolusi timer secara persisten tanpa bergantung pada status visibilitas jendela aplikasi. Pendekatan ini selanjutnya diperkuat dengan deklarasi konstanta *PROCESS_POWER_THROTTLING_IGNORE_TIMER_RESOLUTION* melalui *SetProcessInformation*, guna memutus mekanisme timer *coalescing* sistem operasi yang berpotensi menganulir resolusi timer yang telah ditetapkan (Yang et al., 2020).

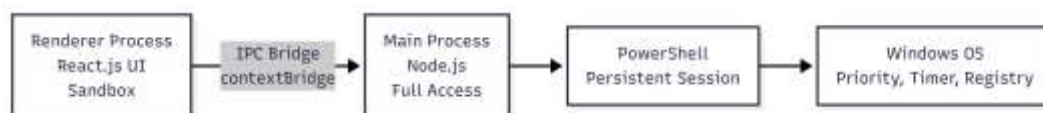
2.4 Teknologi Pengembangan Perangkat Lunak Desktop Hibrida

Penelitian optimasi ini mentransformasikan skrip rekayasa kernel rumit menjadi aplikasi yang aman dan komprehensif bagi konsumen berkat dukungan kombinasi ekosistem teknologi pengembangan berbasis *web-framework* hibrida yang diadopsi pada industri modern.

2.4.1 Arsitektur Kerangka *Electron.js*

Electron.js adalah kerangka pengembangan aplikasi desktop *open-source* yang dikelola oleh *OpenJS Foundation*, yang sejak perilisannya pada tahun 2013 memungkinkan pengembang membangun aplikasi desktop lintas platform menggunakan tumpukan teknologi web modern seperti *JavaScript*, *HTML*, dan *CSS* tanpa perlu menggunakan bahasa pemrograman native seperti *C++*. Arsitektur *Electron* dibangun di atas dua komponen inti, yakni mesin *Chromium*

untuk merender antarmuka grafis dan *Node.js runtime environment* sebagai pengelola interaksi layanan I/O sistem operasi. Kedua komponen ini dikelola melalui skema *Multi-Process Architecture* yang membagi sistem menjadi dua unit kerja otonom: *Main Process* yang berperan sebagai orkestrator sistem dengan akses penuh terhadap *API Node.js* untuk navigasi berkas lokal, eksekusi perintah terminal, dan modifikasi registri OS; serta *Renderer Process* berbasis *Chromium* yang berjalan dalam lingkungan *sandbox* terisolasi dan bertanggung jawab mengelola antarmuka visual yang merespons interaksi pengguna secara langsung. Alur komunikasi antara kedua proses ini dalam konteks implementasi utilitas optimasi diilustrasikan pada gambar berikut (Diallo et al., 2023).



Gambar 2. 7 Arsitektur Electron.js

Karena pemanggilan tingkat kernel seperti peningkatan *Process Priority* dan penetapan timer resolution tidak dapat dieksekusi secara langsung dari sisi antarmuka *UI Renderer Process*, komunikasi antar kedua proses dilakukan secara aman melalui modul *Inter-Process Communication* (IPC) bawaan Electron dengan lapisan keamanan tambahan berupa fungsi *contextBridge*. Mekanisme ini berfungsi sebagai perantara pertukaran instruksi secara asinkron sekaligus lapisan validasi tidak langsung, memastikan setiap perintah sensitif tingkat sistem hanya dapat dieksekusi melalui jalur komunikasi yang terotorisasi tanpa membuka celah eksploitasi dari lingkungan *sandbox Renderer Process*.

2.4.2 Antarmuka Reaktif *React.js* dan Keamanan *TypeScript*

React.js berfungsi sebagai fondasi antarmuka modern utilitas yang diusulkan, dengan pendekatan *Component-Driven UI* yang merangkai antarmuka melalui modul komponen kecil yang responsif dan hierarkis, bukan dalam kesatuan kode monolitik. Arsitektur ini menyederhanakan pengelolaan state management yang kompleks dan memungkinkan alokasi siklus *Virtual DOM* agar aplikasi tetap responsif meskipun proses di balik layar membutuhkan waktu eksekusi yang panjang. Implementasi *React* diperkuat oleh *TypeScript* sebagai lapisan *type safety* yang membungkus skema data pengiriman IPC ke dalam struktur *Interface* bertipe ketat. Mengingat pengelolaan data perintah yang diinjeksikan ke registri sistem merupakan tugas krusial yang rentan terhadap kegagalan parsing, *TypeScript* mengeliminasi potensi cacat masukan — seperti data bertipe abjad yang masuk ke parameter yang hanya mengizinkan nilai integer milidetik — jauh sebelum paket biner didistribusikan. Melalui penerapan *static typing* pada lebih dari 80% lingkungan modifikasi, integritas antara antarmuka web dan stabilitas fungsi sistem Windows dapat terjaga secara solid (Alamari & Chow, 2021; Diallo et al., 2023).

2.4.3 Optimalisasi Sesi Eksekusi *PowerShell Persistent*

Karena *Electron* tidak memiliki akses kernel yang memadai untuk mengeksekusi intervensi tingkat administrasi sistem tanpa otorisasi terpercaya, kendali operasional dialokasikan kepada *Windows PowerShell* sebagai peladen perintah otomatisasi utama Microsoft. Pendekatan eksekusi perintah standar melalui *child_process Electron* yang dijalankan secara berulang menghasilkan *overhead* penumpukan sumber daya CPU yang tidak efisien, terutama saat

menangani lusinan baris perintah registri secara sekuensial. Untuk mengatasi keterbatasan ini, diimplementasikan arsitektur *PowerShell Persistent Session* melalui inisialisasi *New-PSSession* yang menciptakan instans koneksi terbuka ke subsistem pemrosesan tanpa penutupan otomatis. Dalam sesi persisten ini, modul, variabel lokal, instruksi *P/Invoke* tingkat rendah, dan rekam jejak autentikasi dipertahankan selama utilitas aktif, sehingga ribuan pengaturan dan validasi kondisi prioritas *MMCSS* dapat dieksekusi secara *real-time* tanpa *overhead* inisialisasi berulang, menjadikannya teknologi perantara asinkron yang ideal dalam utilitas hibrida *modern* (Hajjar et al., 2024).

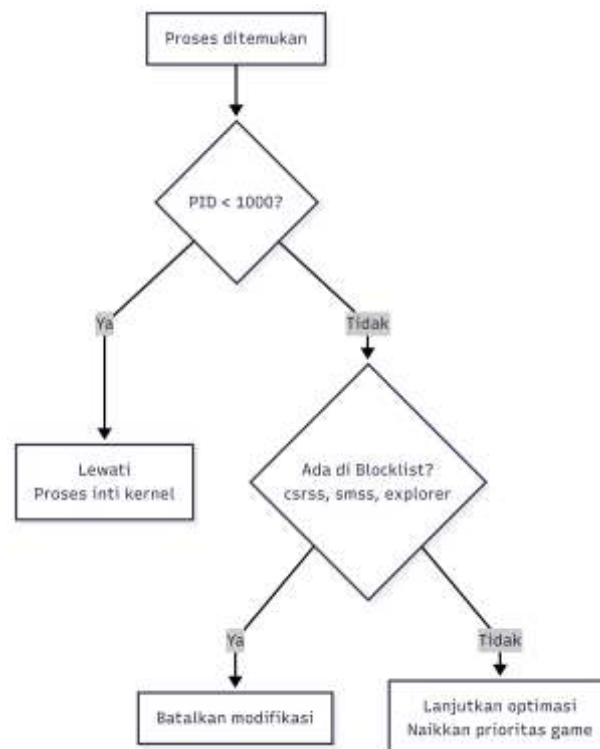
2.5 Mekanisme Keamanan Perlindungan Sistem Perangkat Keras

Peningkatan parameter proses di atas ambang normal membawa potensi dampak deterministik laten yang tidak diinginkan terhadap stabilitas sistem. Oleh karena itu, untuk mencegah korupsi inti sistem operasi yang dapat berujung pada *Blue Screen of Death*, mekanisme keamanan harus ditanamkan secara berlapis pada sisi *backend* utilitas sebagai jaminan perlindungan sistem selama proses optimasi berlangsung.

2.5.1 Konsep Validasi *Protected System Blocklist*

Konsep *blocklist* dalam keamanan sistem bersumber dari teori segregasi pertahanan siber, di mana akses didelegasikan melalui pendekatan penyangkalan eksklusif untuk melokalisir komponen rentan sebelum memengaruhi modul inti. Microsoft sendiri menerapkan filosofi ini melalui mekanisme seperti *Vulnerable Driver Blocklist* guna mencegah privilege escalation dari pengendali pihak ketiga yang tidak tepercaya. Diadaptasi untuk lingkungan manajemen skrip tingkat OS,

utilitas ini mengimplementasikan *Protected System Blocklist* sebagai perisai keamanan yang mengatasi ketidakmampuan algoritma peningkatan prioritas dalam membedakan proses game yang sah dari proses fundamental sistem operasi. Tanpa mekanisme pemblokiran ini, skrip *PowerShell* berpotensi memanipulasi *Priority Class* dari proses esensial seperti *csrss.exe*, *smss.exe*, atau *explorer.exe*, yang dapat menyuntikkan looping kelumpuhan ke ring kernel dasar. Algoritma perlindungan bekerja dengan memuat daftar *JSON* berisi string proses Windows esensial ke memori lokal, kemudian melakukan komparasi silang terhadap setiap *Process ID* (PID) yang terdeteksi sebelum eksekusi skrip WMI dilancarkan — apabila nama *executable* target bersinggungan dengan daftar yang dilindungi, instruksi modifikasi prioritas langsung dibatalkan untuk menjaga stabilitas sistem operasi pasca optimasi, sebagaimana diilustrasikan pada gambar berikut (Zhan et al., 2023).



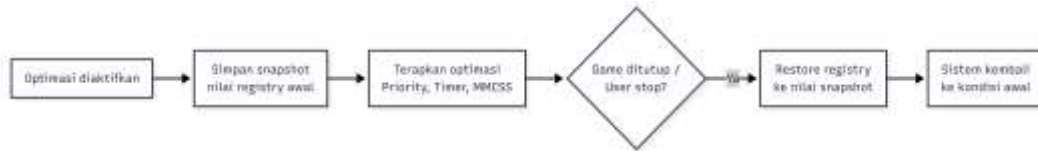
Gambar 2. 8 Protected System Blocklist

Sebagai lapisan keamanan tambahan, utilitas juga mengintegrasikan mekanisme *Process ID (PID) Threshold Filter* yang secara otomatis memblokir seluruh instruksi modifikasi terhadap proses dengan nilai *PID* di bawah 1000. Rentang numerik rendah ini secara eksklusif dialokasikan oleh subsistem *Windows* untuk layanan inti kernel dan inisialisasi *driver* esensial. Melalui kombinasi filtrasi ganda — pencocokan *hardcoded string* pada *blocklist* dan penyaringan ambang batas *PID* — integritas arsitektur sistem operasi terlindungi secara menyeluruh dari risiko eskalasi intervensi yang berpotensi fatal (Zhan et al., 2023).

2.5.2 Mekanisme Pemulihan *Auto-Restore Snapshot*

Di samping mekanisme perlindungan pra-eksekusi, sistem juga dirancang untuk mengembalikan konfigurasi ke kondisi awal setelah sesi optimasi berakhir, guna menghemat konsumsi energi dan mencegah *overheating clock frequency* di luar konteks aplikasi grafis. Untuk itu, diperkenalkan mekanisme *Auto-Restore Snapshot* yang dikonsepsikan sebagai replika bayangan point-in-time state dari seluruh atribut registri *MMCSS* dan parameter pewaktuan sebelum utilitas menyuntikkan modifikasinya. *Snapshot* ini disimpan ke memori sementara dan diikat bersama arsitektur *Node.js* serta event listener *terminal PowerShell*, sehingga berstatus *standby* layaknya *daemon* pasif selama optimasi berlangsung. Ketika pengguna mengakhiri sesi atau menonaktifkan utilitas, algoritma pemulihan membaca salinan *snapshot* awal dan secara seketika menimpa ulang seluruh parameter registri — mengembalikan *interval timer* ke nilai default 15,6 ms dan menghapus seluruh manipulasi *MMCSS*. Mekanisme pemulihan transparan ini memberikan perlindungan fungsional dari risiko konfigurasi registri yang cacat secara permanen, sekaligus meningkatkan kepercayaan pengguna

terhadap utilitas, sebagaimana diilustrasikan pada gambar berikut (Andi Zulherry et al., 2023)



Gambar 2. 9 Auto-Restore Snapshot

2.6 Tinjauan Pustaka dan Kajian Penelitian Terdahulu

Fondasi teoritis yang mendasari penelitian ini divalidasi melalui kajian terhadap literatur ilmiah terkemuka yang mencakup paradigma *CPU scheduling*, teknik teori antrean beban berat, dan model alokasi heuristik *real-time*. Relevansi dan posisi penelitian ini terhadap pustaka acuan yang ada dirangkum dalam Tabel 2.2 berikut, yang mendemonstrasikan validitas penelitian sekaligus mencerminkan celah penelitian yang belum terpenuhi oleh studi-studi sebelumnya

Tabel 2. 2 Penelitian Terdahulu

Nama Peneliti / Tahun	Judul Penelitian & Identifikasi (DOI)	Temuan / Kesesuaian Terhadap Metodologi Penelitian Ini
(Hajjar et al., 2024)	<i>Performance Assessment of CPU Scheduling Algorithms: A Scenario-Based Approach with FCFS, RR, and SJF.</i> DOI: 10.3844/JCSSP.2024.972.985	Menyumbangkan pijakan bahwa manipulasi spesifik parameter proses mereduksi waktu kelambanan sistem secara nyata; relevan dengan modifikasi perlakuan tingkat prioritas WMI.
(Ramadhani & Firdaus, 2025)	<i>Analisa Peran Proses Block Control (PBC) terhadap Sistem Operasi pada Windows 10.</i> DOI: 10.61132/MARS.V3I5.1134	Menguraikan mekanisme OS Windows dalam menjaga isolasi antar proses (<i>process isolation</i>) demi stabilitas sistem. Temuan ini menjadi landasan konseptual yang relevan untuk merancang

		arsitektur keamanan pada modul <i>Protected System Blocklist</i> dan <i>Auto-Restore</i> , guna mencegah intervensi skrip optimasi agar tidak melumpuhkan proses vital inti Windows.
(Xing et al., 2025)	<i>Self-Adapting CPU Scheduling for Mixed Database Workloads via Hierarchical Deep Reinforcement Learning.</i> DOI: 10.3390/SYM17071109	Menjustifikasikan pentingnya peralihan dan kontrol spesifik resolusi waktu instruksi dari <i>high-resolution timer</i> dalam menyamaratakan lonjakan komputasi di bawah tekanan beban tinggi (<i>dynamic load</i>).
(Sakshi et al., 2022)	<i>A new median-average round Robin scheduling algorithm: An optimal approach for reducing turnaround and waiting time.</i> DOI: 10.1016/J.AEJ.2022.04.006	Menunjukkan bahwa <i>static time slice</i> (kuantum waktu statis) pada OS melemahkan alokasi sumber daya; mendasari gagasan perlunya perombakan parameter bawaan Windows <i>tick resolution</i> .
(Yang et al., 2020)	<i>A task scheduling algorithm considering game theory designed for energy management in cloud computing.</i> DOI: 10.1016/J.FUTURE.2017.03.024	Mengilustrasikan pentingnya pengelolaan penyeimbangan antrean (<i>throughput management</i>) dengan alokasi daya agresif; yang dikonversikan pada modifikasi parameter persentase batas ambang batas SystemResponsiveness di layanan kontrol MMCSS pada studi ini.

Melalui elaborasi terhadap studi-studi terdahulu, sebuah kesenjangan penelitian yang signifikan dapat dikonfirmasi. Literatur yang membahas

penjadwalan proses dan rekayasa performa telah melimpah pada tataran simulasi akademik maupun arsitektur *cloud*, namun intervensi aplikatif secara *real-time* yang ditujukan bagi konsumen — khususnya di ruang *PC Gaming* yang menjadi kebutuhan lebih dari separuh demografi pengguna — masih sangat terbatas. Pengguna saat ini terpaksa mengandalkan manipulasi *Process Priority Class* secara manual yang bersifat temporer dan akan kembali ke kondisi awal pasca *reboot*, atau menggunakan perangkat lunak pihak ketiga yang berisiko merusak subsistem komputasi secara permanen. Metodologi yang diusulkan dalam penelitian ini dirancang untuk menetralsir kesenjangan praktis tersebut melalui solusi arsitektural hibrida modern yang aman dan dapat diandalkan

BAB III.

METODE PENELITIAN

3.1 Alur dan Kerangka Kerja Penelitian (*Research Framework*)

Implementasi perangkat lunak yang merekayasa fungsi tingkat rendah ekosistem kernel menuntut metodologi operasional yang linier dan terverifikasi di setiap fasenya. Metodologi penelitian ini dirumuskan melalui lima tahapan sistematis sebagai berikut:

1. Studi Literatur Ekstensif, mencakup eksplorasi mendalam parameter tersembunyi mekanisme penjadwalan *OS Windows*, perbandingan arsitektur *PowerShell execution policies*, serta dokumentasi modifikasi *P/Invoke* untuk parameter fungsional seperti *timeBeginPeriod()*.
2. Perumusan Parameter Analisis Kebutuhan, meliputi pengumpulan data dan inventarisasi *Process ID* sekunder pada OS, penyaringan daftar eksekutor utilitas rentan untuk merancang modul *Protected System Blocklist*, serta penetapan matriks perbandingan resolusi *FPS* terhadap standar latensi interupsi antrean *WMI*.
3. Pengembangan Arsitektur Modul, yakni membangun antarmuka frontend berbasis *React.js* dan mengintegrasikan skema eksekusinya ke backend *Electron.js*, yang selanjutnya mengontrol siklus persistent runspace *PowerShell* secara aman melalui kontrak data asinkron berbasis *TypeScript*.
4. Implementasi Optimasi dan Mitigasi Keamanan, dengan menginjeksikan skrip *PowerShell* untuk penyesuaian *Process Priority*, manipulasi

parameter registri *MMCSS (SystemResponsiveness)*, dan konfigurasi Timer Resolution ke rentang milidetik terkecil, disertai penyempurnaan fitur *Auto-Restore Snapshot* sebagai lapisan pemulihan nilai bawaan sistem.

5. Pengujian Simulasi dan Validasi Data, berupa pembuktian kinerja utilitas secara riil dengan membandingkan kondisi baseline terhadap penerapan bertahap konfigurasi prasetel pada variabel dependen Frame Per Second dan latensi masukan periferai.

3.2 Kerangka Konsep Penelitian

Penelitian ini mengukur efektivitas manipulasi penjadwalan proses terhadap performa grafis melalui tiga klasifikasi variabel utama:

1. Variabel Bebas (*Independen*), terdiri dari empat tingkat intervensi optimasi: Baseline yang mempertahankan parameter standar OS dengan timer 15,6 ms; Minimum yang mengeskalasi prioritas proses ke level Above Normal; Normal yang menerapkan prioritas High disertai penyesuaian timer ke 0,5 ms; dan Maximum yang mengunci prioritas pada level 128, mereduksi *SystemResponsiveness MMCSS* hingga batas absolut 14%, serta memadatkan timer secara absolut ke 0,5 ms melalui pemanggilan pustaka *ntdll.dll*.
2. Variabel Terikat (*Dependen*), mencakup tiga metrik performa pascaoptimasi: Rata-rata FPS untuk memantau throughput rendering secara keseluruhan; 1% *Low FPS* untuk memverifikasi stabilitas frametime dan efektivitas reduksi micro stutters; serta *Latensi Input* untuk

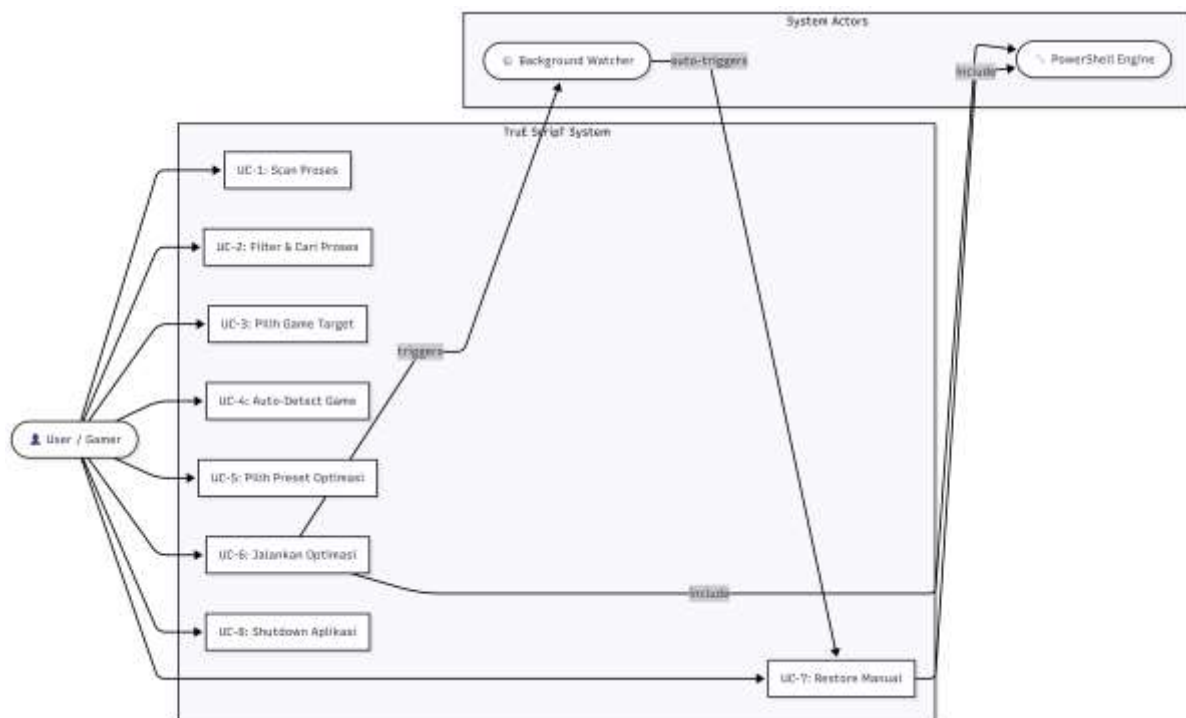
mengukur kecepatan respons antara perintah periferan fisik pengguna dan tampilan layar dalam satuan milidetik.

3.3 Perancangan Model Sistem(UML)

Sub-bab ini menjelaskan pemodelan sistem menggunakan *Unified Modeling Language* (UML) untuk memberikan gambaran visual mengenai interaksi aktor dengan fitur-fitur yang tersedia di dalam aplikasi.

3.3.1 Use Case Diagram

Use Case Diagram menggambarkan fungsionalitas yang disediakan oleh sistem dari sudut pandang pengguna (*Gamer*). Berdasarkan arsitektur aplikasi, pengguna memiliki kontrol penuh terhadap modul optimasi dan pemantauan sistem. Seperti Gambar 3.1 Dibawah ini:



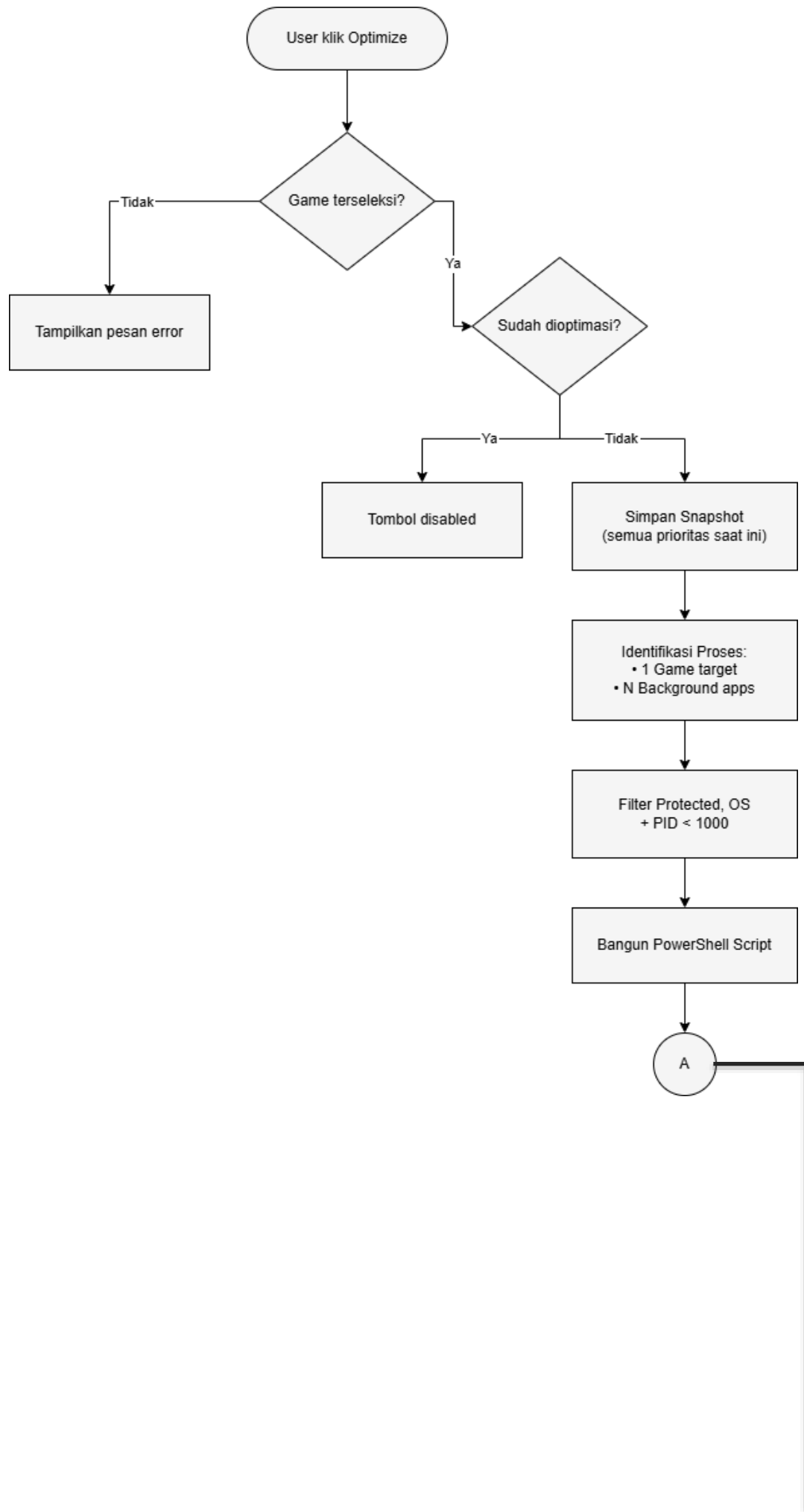
Gambar 3. 1 Use Case Diagram

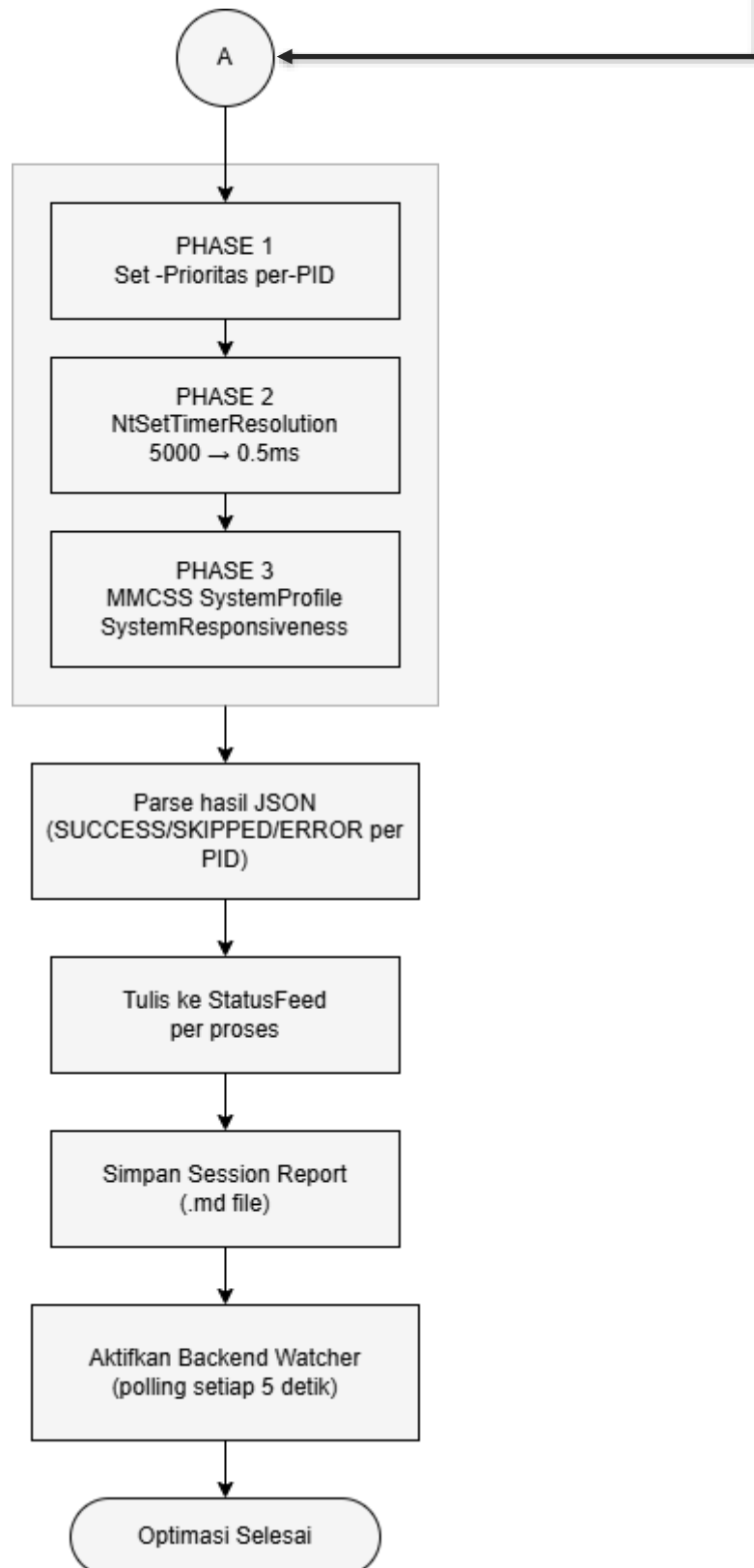
Tabel 3. 1 Detail Use Case

ID	Use Case	Aktor	<i>Pre-condition</i>	<i>Post-condition</i>
UC-1	<i>Scan Proses</i>	<i>User</i>	App berjalan	Daftar proses tampil
UC-2	Filter & Cari	<i>User</i>	Proses terscan	List terfilter
UC-3	Pilih Game Target	<i>User</i>	Proses terscan	Game terseleksi
UC-4	<i>Auto-Detect Game</i>	<i>User</i>	Proses terscan	Game auto-terseleksi
UC-5	Pilih Preset	<i>User</i>	Game terseleksi	Preset terpilih
UC-6	Jalankan Optimasi	<i>User + PS</i>	Game + preset siap	Sistem teroptimasi, watcher aktif
UC-7	<i>Restore</i>	<i>User / Watcher</i>	Sistem teroptimasi	Prioritas normal, timer 15.6ms
UC-8	<i>Shutdown</i>	<i>User</i>	App berjalan	Restore lalu keluar

3.3.2 Flowchart Proses Optimasi Utama

Alur kerja sistem dalam melakukan eksekusi prioritas penjadwalan digambarkan secara spesifik mulai dari validasi UI hingga penulisan status log.

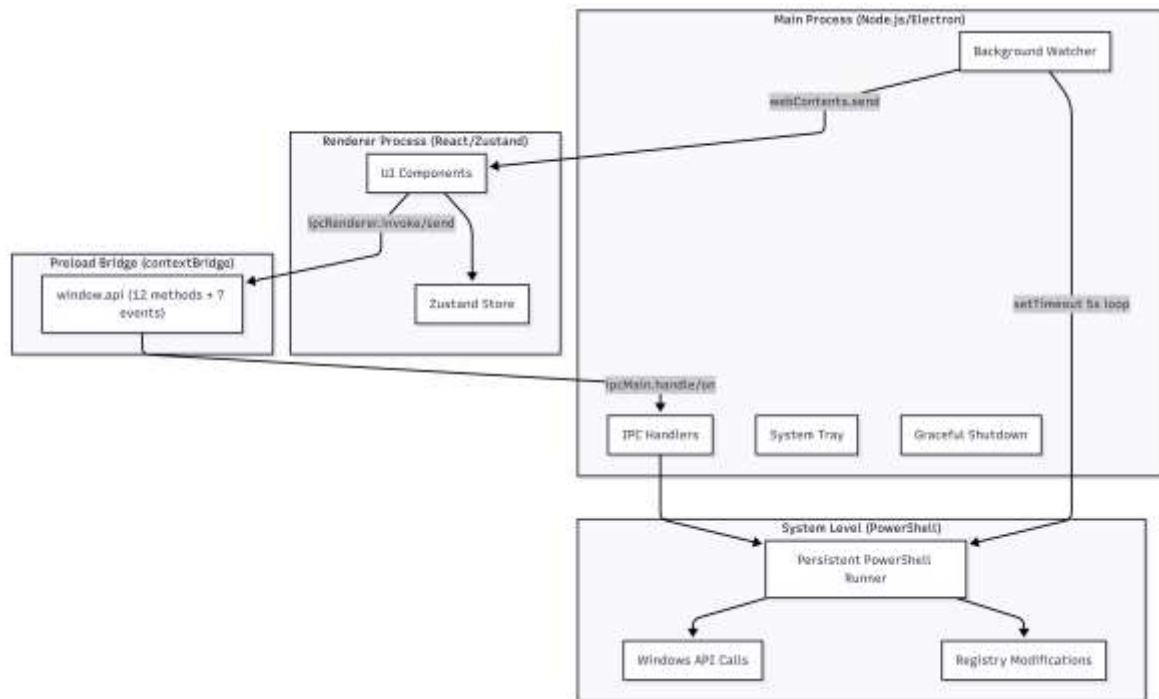




Gambar 3. 2 Flowchart Proses Optimasi

3.4 Analisis Arsitektur Sistem Rekayasa Utilitas

Sistem bekerja di tiga level utama: menaikkan prioritas CPU game dan menekan aplikasi latar (*Process Priority*), mengunci interval pewaktuan OS ke 0.5ms (*Timer Resolution*), dan memodifikasi profil registri (*MMCSS Tuning*):



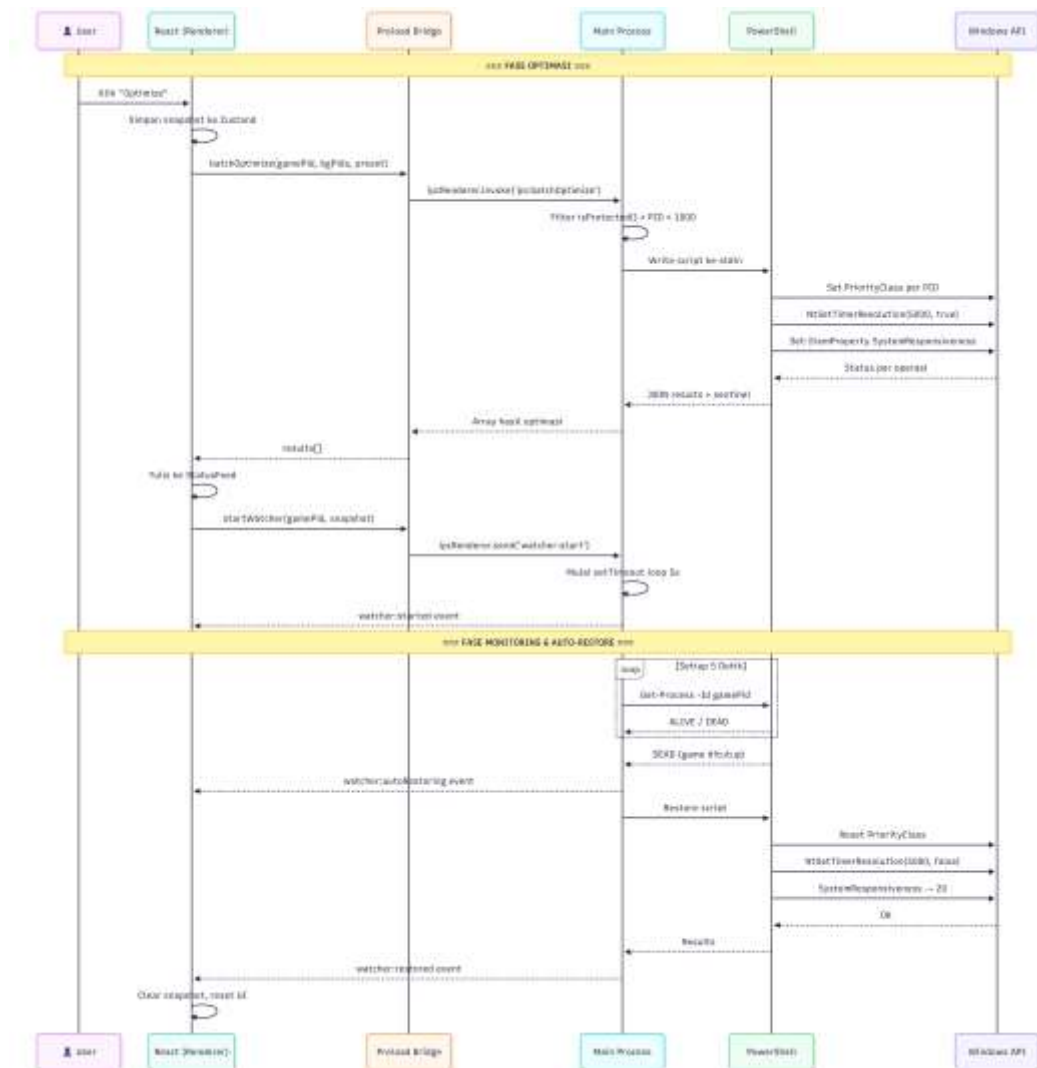
Gambar 3. 3 Arsitektur Sistem

Tabel 3. 2 Layer Breakdown Sistem

Layer	Teknologi	Fungsi
Renderer	React 19 + Zustand 5	Antarmuka pengguna, state management
Preload	Electron contextBridge	Jembatan IPC aman (context isolation)
Main	Node.js + Electron	Logika bisnis, watcher, tray, shutdown
System	PowerShell + C# P/Invoke	Manipulasi OS (prioritas, timer, registri)

3.4.1 Skema Komunikasi *Multi-Proses (Electron & React)*

Untuk membatasi risiko manipulasi langsung pada kernel OS, perangkat lunak menerapkan *Process Separation Architecture* yang memisahkan komponen antarmuka dari komponen eksekusi sistem secara ketat. Lapisan antarmuka berbasis *React.js* beroperasi dalam lingkungan Chromium yang sepenuhnya tersandbox, hanya bertugas merepresentasikan dashboard visual dan menangani event listener input pengguna. Karena isolasi *sandbox* mencegah akses langsung ke fungsi kernel, seluruh instruksi modifikasi dikirimkan melalui jembatan *Inter-Process Communication (IPC)* dengan validasi *TypeScript* yang memberlakukan *type-safety contracts* pada setiap rantai komunikasi antara *Preload Script*, *IPC*, hingga komponen *Node.js* di *Main Process* — memastikan tidak ada perintah ilegal yang dapat disuntikkan ke antrian *shell script OS*. Lapisan *Main Process Electron* kemudian memverifikasi otorisasi *User Account Control* dengan elevated privileges sebelum membuat utas eksekusi ke subsistem manajemen internal *Windows*, memungkinkan fungsionalitas berjalan pada perangkat keras asli tanpa batasan *sandbox*.



Gambar 3. 4 Skema Komunikasi Multi-Proses

3.4.2 Modul Pengendalian PowerShell Persistent

Penggunaan fungsi standar *child_process.exec* pada *Node.js* menimbulkan *overhead* komputasi yang signifikan ketika mengeksekusi urutan kueri bertingkat pada arsitektur utilitas yang kompleks. Untuk mengatasi keterbatasan ini, diimplementasikan *PowerShell Persistent Session* sebagai lingkungan penanganan interupsi berkelanjutan. Pada tahap inisialisasi, *Main Process* meluncurkan subsistem *PowerShell* dengan kebijakan *ExecutionPolicy Bypass* dan menginisialisasi *runspace* yang berjalan secara asinkron di latar belakang tanpa jendela terminal visual. Setiap instruksi penyesuaian profil *MMCSS* atau resolusi

prioritas OS yang dipicu dari antarmuka React diteruskan dan dieksekusi secara masif di bawah *state memory* yang telah ditautkan secara persisten, sehingga mengeliminasi latensi autentikasi berulang. Pendekatan arsitektur ini menghasilkan respons umpan balik instan saat game dimuat serta memungkinkan eksekusi *Windows API* dan pembacaan *Windows Management Instrumentation* (WMI) dalam kecepatan streaming data yang responsif pada antarmuka klien.

3.5 Prosedur Implementasi Rekayasa Modifikasi Kinerja (Optimasi)

Penjadwalan prioritas dioperasikan dengan merancang rutinitas manipulasi di bawah parameter *WMI PowerShell* terhadap tiga segmen hambatan proses OS tingkat bawah, memaksa OS agar menyuguhkan fokus CPU eksklusif terhadap kalkulasi utas *game*.

3.5.1 Modifikasi Hierarki Prioritas Proses (*Priority Scheduling*)

Tahapan modifikasi inti utilitas ini bertujuan menggantikan alokasi CPU time layanan tidak kritis dengan mengeskalasi hierarki eksekusi game target. Melalui *Persistent Session*, modul memindai Process ID aplikasi latar depan aktif, kemudian mereklasifikasi nilai *PriorityClass* aplikasi dari kategori default *Normal* (nilai *DWORD* 0x00000020) menuju kelas *AboveNormal* (0x00008000) atau *High* (0x00000080) menggunakan pemanggilan langsung Win32 *Native API SetPriorityClass* melalui pembungkus P/Invoke C# (*TruEPriorityApi*). Untuk menjamin keandalan pengalokasian prioritas di tingkat kernel Windows, sistem menerapkan mekanisme *Read-Back Verification* menggunakan *GetPriorityClass* sesaat setelah *SetPriorityClass* dieksekusi. Mekanisme verifikasi ini memastikan secara deterministik bahwa tingkat prioritas benar-benar telah diperbarui oleh

kernel OS sebelum mengonfirmasi status sukses pada sistem. Apabila pembacaan ulang mendeteksi ketidaksesuaian nilai (misalnya akibat pembatasan hak akses UAC), sistem secara eksplisit mengembalikan status kegagalan beserta kode kesalahan Win32 (seperti *NO_ACCESS:5* atau *SET_FAILED*). Penyesuaian lanjutan juga ditanamkan pada kunci registri *HKLM\SYSTEM\CurrentControlSet\Control\PriorityControl* melalui manipulasi register *Win32PrioritySeparation*, guna memastikan algoritma penjadwalan kernel memprioritaskan aplikasi yang aktif di latar depan secara absolut. Pemilihan nilai kelas *High* (0x00000080) didasarkan pada kebutuhan pengalihan sumber daya CPU secara instan tanpa mengorbankan integritas sistem operasi, sementara nilai *RealTime* (0x00000100) secara eksplisit tidak digunakan karena berisiko menginterupsi utas fundamental perangkat keras seperti input mouse dan disk cache yang berpotensi memicu kondisi *unresponsive* atau *Blue Screen of Death* (BSOD). Implementasi Profil Heuristik Tugas *MMCSS*

Untuk mengatasi persaingan alokasi sumber daya antara beban grafis dengan layanan I/O jaringan dan subsistem audio, utilitas merekayasa atribut *Multimedia Class Scheduler Service* melalui modifikasi hierarki registri *Multimedia SystemProfile Windows*. Intervensi utama berfokus pada menonaktifkan pembatasan ambang *CPU utilization reservation* dengan memaksa nilai register *DWORD SystemResponsiveness* — yang secara default mencadangkan 20% *CPU* untuk proses latar belakang — menuju batas valid terendah yang diizinkan sistem melalui konversi parameter heksadesimal ke nilai 0. Modifikasi sekunder diarahkan pada kunci registri *Games Profile Tasks*, dengan memanipulasi atribut *Priority* dan *GPU Priority* ke mode efektivitas paling agresif, sehingga pengelola

profil media mendelegasikan hampir seluruh ketersediaan *CPU bandwidth* secara konsisten kepada proses render tanpa tunduk pada *timer slicing* reguler sistem operasi.

3.5.2 Substitusi Jeda Timer Resolusi (*High-Resolution Timer*)

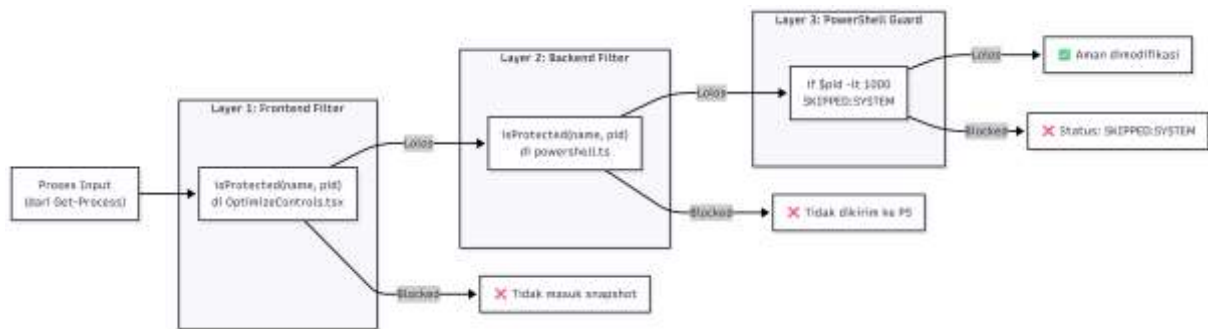
Untuk mengatasi jitter latensi yang ditimbulkan oleh mekanisme jam sistem efisiensi energi Windows, sekaligus melewati pembatasan *per-process scope rule change* pada Windows 10, implementasi rekayasa *timer resolution* dilakukan melalui kompilasi rutinitas eksternal berbasis *C# P/Invoke* yang dieksekusi secara *inline* dalam *PowerShell Persistent Session* berelevansi tinggi. *Sub-rutinitas* ini menautkan pemanggilan langsung ke pustaka *ntdll.dll* melalui fungsi *NtSetTimerResolution* dan turunan *timeBeginPeriod*, memadatkan *interval* ketukan jam secara deterministik dari nilai default 15,6 ms menuju granularitas interupsi minimal 0,5 milidetik, sehingga mengurangi jeda penyelarasan antara *mouse polling rate* dan sinkronisasi *frame buffer CPU* secara eksponensial. Implementasi diperkuat dengan injeksi flag *PROCESS_POWER_THROTTLING_IGNORE_TIMER_RESOLUTION* melalui *SetProcessInformation* untuk melindungi resolusi timer dari penundaan paksa oleh manajer daya OS. Seluruh injeksi parameter dikompilasi dan dikirimkan ke kernel secara langsung di dalam memori *Persistent Session* yang telah aktif, mengeliminasi *overhead* pembukaan terminal baru setiap kali pengguna mengubah prasetel optimasi sekaligus mencegah interupsi autentikasi *User Account Control* yang berulang.

3.6 Implementasi Lapisan Verifikasi Keamanan OS

Menimbang daya retas pemaksaan penyesuaian rutinitas sistem secara deterministik yang dimotori akses *backend User Account Control* yang ditingkatkan, desain operasional tak terpisahkan dari penyisipan arsitektur pencegah kelemahan fundamental *OS*. Perlindungan esensial *WMI* dirangkum di dua segmen verifikasi:

3.6.1 Validasi *Protected System Blocklist*

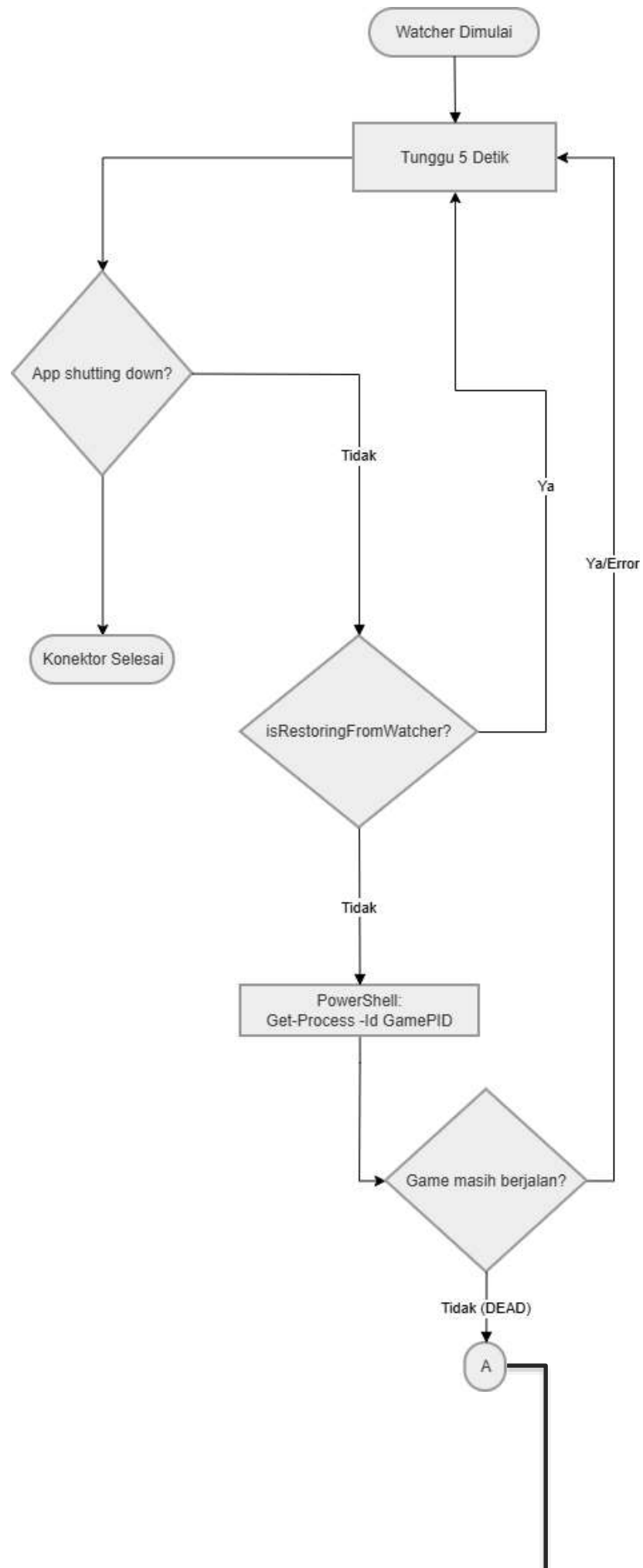
Untuk mencegah risiko infinite loop priority mapping yang berpotensi menangkap utas fundamental sistem dan memicu kondisi BSOD, utilitas mengimplementasikan modul mitigasi berbasis Protected System Blocklist. Prosedur ini bekerja dengan pra-kalkulasi daftar statis berbentuk *array TypeScript/JSON* yang memuat definisi nama biner sistem esensial seperti *csrss.exe*, *smss.exe*, *explorer.exe*, dan layanan telemetri inti subsistem perangkat keras. Sebelum skrip PowerShell merancang penetapan PID target, algoritma utilitas melakukan pencocokan nama *executable* melalui sinkronisasi *ContextBridge* — apabila nama proses yang akan dialokasikan terdeteksi memiliki kecocokan dengan entitas dalam basis data daftar hitam, algoritma intersep secara instan membatalkan eksekusi dan mengirimkan *bypass logic exception* ke renderer. Mekanisme perlindungan ini memastikan utilitas hanya beroperasi secara eksklusif pada aplikasi pihak ketiga seperti game, tanpa mengintervensi fondasi stabilitas subsistem kernel dasar *Windows*. Skema mitigasi yang melibatkan tiga lapis perlindungan keamanan ini dapat dilihat secara detail pada Gambar 3.5 berikut.

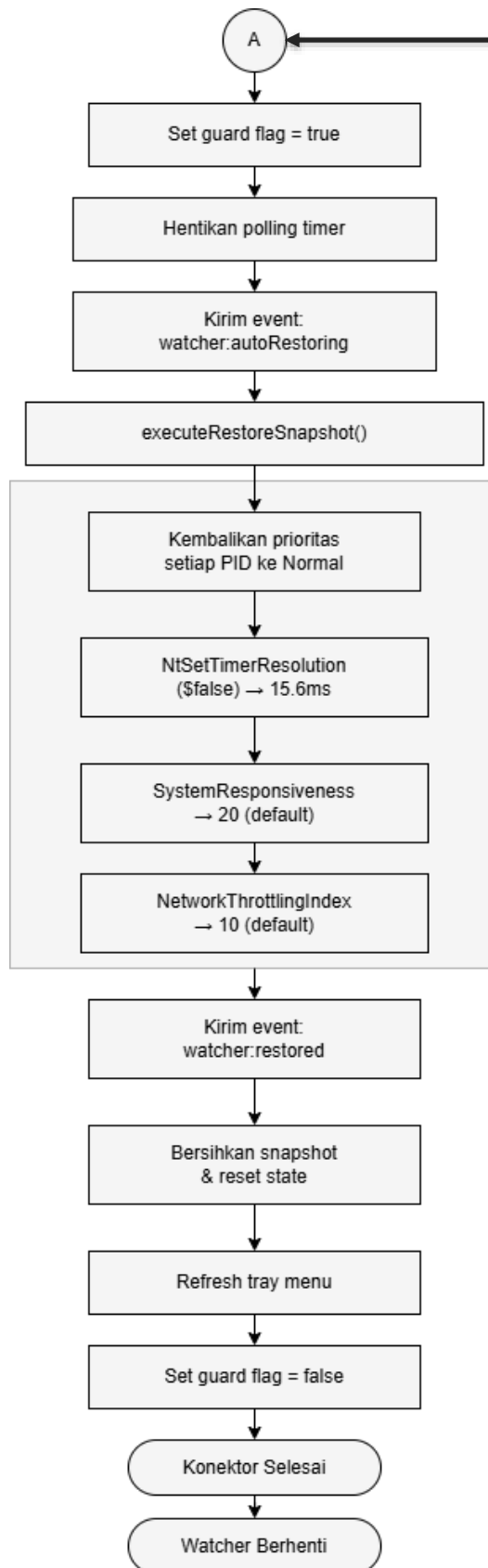


Gambar 3. 5 Tiga Lapis Perlindungan

3.6.2 Logika Penyimpanan Parameter (*Auto-Restore Snapshot*)

Pemaksaan *High-Resolution* Timer dan modifikasi profil *MMCSS* secara beruntun berpotensi menguras daya tahan prosesor akibat ketidakmampuan perangkat keras kembali memasuki kondisi efisiensi *C-State* saat komputasi kasual, terutama pada perangkat mobile dan laptop. Untuk mencegah kerusakan performa jangka panjang ini, diimplementasikan mekanisme *Auto-Restore Snapshot* yang menyerupai arsip *shadow cache state* primitif. Sebelum profil optimasi diaktifkan, utilitas memindai dan mengompilasi seluruh nilai kunci registri default — mencakup parameter *MMCSS SystemResponsiveness*, *Win32PrioritySeparation*, hingga pengaturan resolusi timer bawaan OS — kemudian menyimpannya secara transparan dalam blok sementara *PowerShell Persistent Session* atau state *server Node.js*. Ketika pengguna menonaktifkan optimasi atau aplikasi menerima event penghentian, skrip pemulihan sekunder secara otomatis merestorasi seluruh nilai registri referensial awal melalui autentikasi administrator, mengakhiri pemaksaan parameter *ntdll*, dan menormalisasi interval timer kembali ke nilai default 15,6 ms tanpa meninggalkan residu anomali pada registri operasional sistem. Alur kerja dari pemantauan latar belakang (*Background Watcher*) hingga proses pemulihan parameter sistem ini diilustrasikan pada flowchart Gambar 3.6 berikut.





Gambar 3. 6 Auto-Restore (Background Watcher)

3.7 Skenario Uji Operasional dan Parameter Metrik Evaluasi

Proses penilaian terhadap fungsionalitas sistem yang diajukan tidak dapat diukur tanpa observasi data berstandar komparatif. Skenario observasi pada penelitian skripsi dirancang mengikuti metode percobaan silang pada tingkat spesifikasi keras dengan keterbatasan eksekusi (*mid-to-low specifications*).

3.7.1 Definisi Batasan Lingkungan Pengujian dan Objek Uji

Pengujian penelitian ini beroperasi dalam batasan fungsional yang ketat, meliputi:

1. Lingkungan Operasi, dibatasi secara eksklusif pada sistem operasi Windows 10 dan Windows 11 guna memastikan dukungan konsisten terhadap arsitektur *Timer Resolution Coalescing per-process scope* yang diperkenalkan sejak pembaruan Versi 2004.
2. Limitasi Variabel Eksternal, seluruh manipulasi di luar variabel independen yang ditetapkan — seperti *overclocking/undervolting* frekuensi *base-clock GPU/CPU*, manajemen alokasi jaringan, maupun modifikasi registri di luar sirkuit *MMCSS* dan peningkatan prioritas — wajib dinonaktifkan atau dibekukan selama pengujian berlangsung.
3. Konfigurasi Grafis, pengaturan grafis pada setiap objek uji menerapkan pendekatan biner ekstrem yang memadukan prasetel *High* pada resolusi tekstur dan kualitas model, serta prasetel *Low* pada bayangan dan efek volumetrik, dengan mengeliminasi sepenuhnya opsi *Medium* guna mencegah distribusi beban pada titik seimbang sehingga eskalasi metrik throughput murni mencerminkan dampak rekayasa *Priority Scheduling*.

4. Spesifikasi Perangkat Keras, pengujian dieksekusi pada laptop berspesifikasi menengah untuk membuktikan efisiensi algoritma pada sistem dengan sumber daya terbatas, yakni *CPU Intel Core i5 generasi ke-10, GPU NVIDIA GeForce RTX 2060 Max-Q, dan RAM 16 GB*.
5. Objek Pengujian, utilitas diuji pada aplikasi game dengan beban *CPU* intensif dan gameplay aksi cepat yang mencakup genre *Open-World Action RPG, Turn-Based Tactical RPG, Action Roguelite, Rhythm Game, dan Urban Hack-and-Slash ARPG* — genre-genre yang dipilih berdasarkan tingginya tuntutan sinkronisasi antara instruksi *CPU* dan rendering *GPU* dalam memproses elemen visual, kalkulasi fisika, dan mekanika combat secara *real-time*.

3.7.2 Kualifikasi Skenario Prasetel Konfigurasi

Evaluasi fungsional utilitas didokumentasikan melalui komparasi hierarkis pada empat skala lingkungan parameter uji yang terpisah:

Tabel 3. 3 Configuration Presets

Tabel 3.4 Aspek	Minimum (SAFE)	Normal (RECOMMENDED)	Maximum (MAX FPS)
Game Priority	AboveNormal	High	High
Background Priority	Normal	Normal	BelowNormal
MMCSS Games Profile	Disabled (OFF)	Active (20% BG CPU Limit)	Active (Strict 14% BG CPU Limit)
SystemResponsiveness	Disabled (20)	20 (80% CPU Game)	14 (86% CPU Game Limit)
Timer Resolution	Default (15.6 ms)	0.5 ms (High-Res)	0.5 ms (High-Res)
Risiko & Target	Sangat Rendah (PC Kantoran)	Rendah (Gaming Umum)	Sedang (Esports & High-Load)

1. Fase *Baseline*, mempertahankan seluruh parameter standar sistem operasi tanpa intervensi utilitas, berfungsi sebagai kondisi acuan pengukuran awal sebelum optimasi diterapkan.
2. Fase Prasetel Minimum, menerapkan modifikasi ringan dengan mengubah Process Priority Class aplikasi target ke level Above Normal, sementara proses latar belakang dipertahankan pada kondisi Normal. Pada prasetel ini, pemaksaan High-Resolution Timer dan profil MMCSS dinonaktifkan (Disabled) guna menjaga overhead sistem tetap minimal dan menjamin keamanan eksekusi 100% pada perangkat keras berdaya rendah.
3. Fase Prasetel Normal, menerapkan skema menengah dengan mengalokasikan High Priority Class secara agresif pada aplikasi target untuk mendominasi alokasi CPU time, sementara prioritas latar belakang tetap di level Normal. Resolusi timer sistem dirapatkan absolut ke 0,5 ms melalui pemanggilan API Windows Kernel (ntdll/timeBeginPeriod), serta modul MMCSS diaktifkan dengan alokasi registri SystemResponsiveness bernilai 20 (mengalokasikan 80% kuantum CPU secara eksklusif untuk aplikasi permainan).
4. Fase Prasetel Maximum, mendorong pemanfaatan sumber daya komputasi secara optimal dengan mengunci prioritas aplikasi target pada High Priority Class, diiringi penurunan prioritas utas latar belakang ke level Below Normal untuk menekan interupsi secara maksimal. Modifikasi MMCSS dieksekusi secara intensif dengan mereduksi SystemResponsiveness hingga batas absolut 14 (mengalokasikan 86% kuantum CPU untuk game, yang merupakan ambang batas toleransi arsitektur Windows guna mencegah thread starvation pada worker thread game), serta resolusi timer dikunci absolut pada 0,5 ms.

3.8 Perangkat Instrumen Pengukuran

Untuk memastikan validitas dan akurasi data komparatif, penelitian ini menggunakan instrumen pengukuran spesifik pada setiap skenario pengujian:

1. Pengukuran Performa Visual (*FPS*), menggunakan utilitas *CapFrameX* sebagai perangkat lunak open-source berbasis analitik frametime untuk merekam Rata-rata *FPS* serta mengkalkulasi persentil hambatan 1% *Low FPS* secara presisi.
2. Pengukuran Latensi Input, menggunakan utilitas *Mouse Tester/EVARC* untuk mencatat jeda responsivitas klik terhadap render grafis sebagai representasi kecepatan respons periferan pengguna.
3. Prosedur Pengulangan, setiap skenario prasetel pada masing-masing objek game diuji minimal sebanyak tiga kali pengulangan, dengan nilai akhir yang dianalisis merupakan rata-rata dari seluruh percobaan guna meminimalkan anomali data dan *margin of error*.

3.9 Teknik Analisis Data

Sesuai dengan sifat penelitian yang kuantitatif komparatif, teknik analisis data yang digunakan mencakup dua pendekatan utama:

1. Analisis Deskriptif, menyajikan data metrik hasil pengujian dalam bentuk tabel tabulasi silang dan grafik yang memuat nilai rata-rata, persentil terendah 1% *Lows*, dan standar deviasi dari perolehan *FPS* serta latensi input pada setiap fase prasetel.
2. Analisis Komparatif Relatif, menyajikan perbandingan metrik performa — mencakup Rata-rata *FPS*, 1% *Low FPS*, dan *Latensi Input* — antara kondisi Baseline dan tiga level prasetel optimasi (*Minimum*, *Normal*, *Maximum*) menggunakan grafik batang, guna memberikan representasi grafis yang komprehensif terhadap fluktuasi peningkatan performa dan

stabilitas frametime pascaimplementasi utilitas.

3.10 Jadwal Pelaksanaan Penelitian

Untuk memastikan penelitian berjalan secara sistematis dan sesuai dengan target waktu yang ditetapkan, disusunlah jadwal pelaksanaan penelitian yang mencakup seluruh tahapan dari studi literatur hingga finalisasi laporan sebagaimana tercantum pada Tabel 3.1 berikut.

Tabel 3. 5 Waktu Penelitian

Kegiatan	Apr	Mei	Jun	Jul	Agu	Sep
Studi Literatur dan Identifikasi Parameter WMI						
Perancangan Arsitektur Aplikasi dan Modul <i>Security</i>						
Pengembangan Antarmuka (React) dan <i>Backend</i> (Electron)						
Pengembangan Skrip PowerShell dan Integrasi P/Invoke						
Pengujian Sistem dan Validasi Metrik (FPS & Latensi)						
Analisis Data, Penarikan Kesimpulan, dan Finalisasi Laporan						

BAB IV.

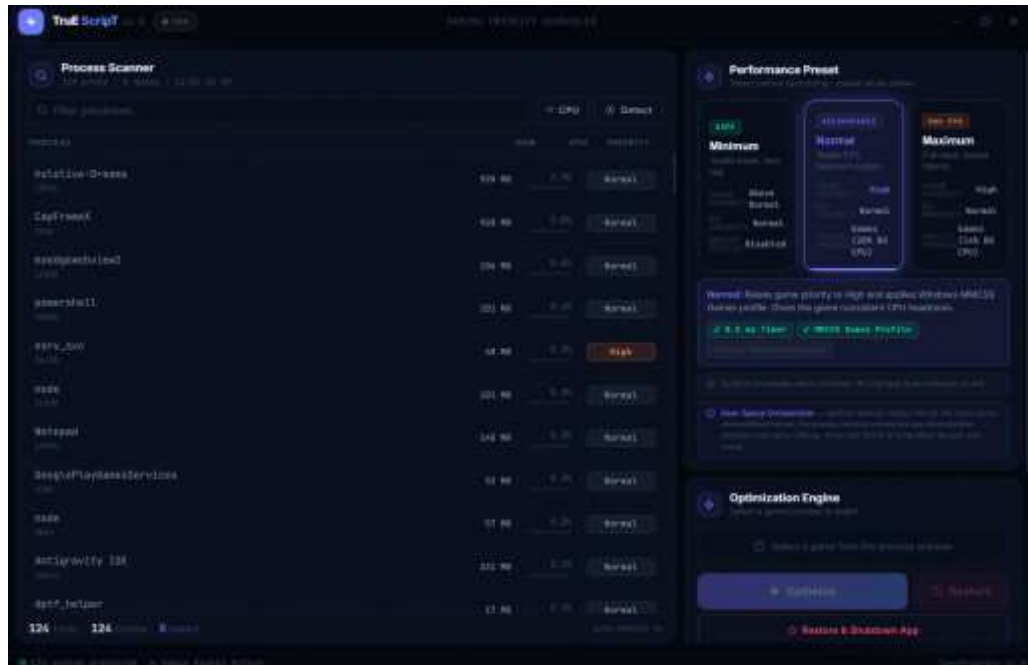
HASIL DAN PEMBAHASAN

4.1 Hasil Implementasi Sistem

Merujuk pada perancangan arsitektur yang telah diuraikan pada bab sebelumnya, perangkat lunak optimasi "*CorePriority Optimizer*" berhasil dikembangkan menggunakan kerangka kerja hibrida Electron.js 33+ (React 19 pada lapisan renderer, Node.js pada lapisan main process) dengan integrasi *PowerShell Persistent Session* sebagai modul eksekusi instruksi sistem operasi. Arsitektur berlapis (*layered architecture*) ini memisahkan secara tegas antara antarmuka pengguna (*UI layer*), logika bisnis (*business logic layer*), dan manipulasi kernel (*system layer*), sehingga menjamin modularitas, maintainability, serta keamanan eksekusi instruksi pada level sistem operasi.

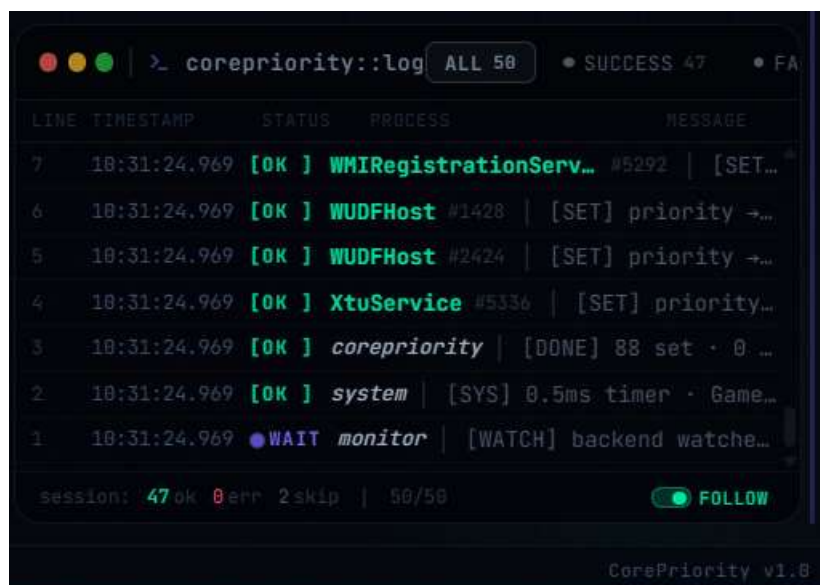
Eksekusi manipulasi sistem operasi dilakukan secara langsung oleh modul backend *PowerShell Persistent Session* (*powershell.ts*), yang mengeksekusi instruksi Win32 API dan C# P/Invoke tanpa menimbulkan overhead komputasi berlebihan. Pendekatan persistent session ini mengeliminasi kebutuhan untuk membuka sesi *PowerShell* baru pada setiap perintah, sehingga latensi eksekusi instruksi berkurang secara signifikan dibandingkan dengan pendekatan konvensional (*one-shot execution*). Tiga fase optimasi utama yang dieksekusi meliputi: (1) manipulasi *Process Priority Class* melalui pemanggilan langsung Win32 Native API *SetPriorityClass* dan *GetPriorityClass* (via P/Invoke *TruEPriorityApi*) yang dilengkapi dengan mekanisme *Read-Back Verification* untuk mengonfirmasi pengalokasian prioritas di tingkat kernel Windows, (2) penyesuaian *Timer Resolution* melalui *NtSetTimerResolution* dari *ntdll.dll*, dan (3)

konfigurasi registri *Multimedia Class Scheduler Service* (MMCSS) untuk mengalokasikan kuantum waktu CPU secara dominan kepada proses permainan.



Gambar 4. 1 Antarmuka Utama Aplikasi *CorePriority Optimizer*

Gambar 4.1 memperlihatkan antarmuka utama utilitas saat berjalan, yang mencakup panel pemindaian proses aktif, pemilih prasetel (Minimum, Normal, Maximum), indikator status optimasi real-time, serta sakelar pemicu optimasi.



Gambar 4. 2 Status Log Feed dan Eksekusi *Backend PowerShell*

Gambar 4.3 memberikan bukti empiris melalui *Windows Task Manager* (Tab Details), di mana *Process Priority Class* dari aplikasi permainan target secara konklusif berubah menjadi *Above Normal* (pada prasetel Minimum) dan *High* (pada prasetel Normal dan Maximum), sementara proses latar belakang tetap terkendali pada prioritas *Normal* atau *Below Normal*.

4.2 Hasil Pengujian Skenario Optimasi

Pengujian operasional dieksekusi menggunakan instrumen analitik CapFrameX v1.8.6.2 berbasis PresentMon API pada lingkungan pengujian yang terkontrol ketat. Spesifikasi perangkat keras dan lingkungan sistem pengujian didokumentasikan pada Tabel 4.1 berikut.

Tabel 4. 1 Spesifikasi Perangkat Keras dan Lingkungan Pengujian

Komponen Perangkat	Spesifikasi dan Parameter Pengujian
Perangkat Utama	Laptop HP 8741
Unit Pemrosesan Pusat (CPU)	Intel Core i5-10300H @ 2.50GHz (Turbo 4.50GHz, 4 Cores / 8 Threads)
Memori Utama (RAM)	16 GB DDR4 Dual-Channel @ 2933 MT/s
Unit Pemrosesan Grafis (GPU)	NVIDIA GeForce RTX 2060 with Max-Q Design (6 GB GDDR6 VRAM)
Media Penyimpanan	512 GB NVMe PCIe M.2 SSD
Sistem Operasi	Microsoft Windows 11 Pro 64-bit (Build 26200)

Instrumen Pengukur	CapFrameX v1.8.6.2 & Intel PresentMon Core Runtime API
--------------------	--

Prosedur pengujian dilakukan dengan menerapkan metode pengulangan sebanyak 10 kali per skenario (10 Runs $\times$ 10 Menit = 100 Menit per skenario). Dengan 4 prasetel pada 5 objek permainan, total durasi eksekusi pengujian eksperimental mencapai 2.000 menit (33.33 jam operasional). Kelima objek uji permainan dipilih untuk merepresentasikan variasi arsitektur game engine, tuntutan aksi, serta skenario komputasi yang beragam:

1. Honkai: Star Rail (HSR) — Representasi permainan *Turn-Based Tactical RPG* berbasis *Unity Engine* dengan pembatasan frame rate bawaan (60 FPS Cap).
2. Wuthering Waves (WuWa) — Representasi permainan *Open-World Action RPG* berbasis *Unreal Engine 5* (UE5) dengan tuntutan heavy asset streaming dan beban komputasi CPU tinggi.
3. Zenless Zone Zero (ZZZ) – Representasi permainan *Urban Hack-and-Slash ARPG* berbasis *Unity Engine* dan DirectX 12 dengan tuntutan aksi tempo tinggi dan frame rate cap 60 FPS.
4. HoloCure – Representasi permainan *Fast-Paced 2D Survival Action Pixel-Art* berbasis *GameMaker Engine* dengan pemrosesan ribuan entitas proyektil dan mob AI secara intensif pada tingkat utas tunggal CPU.
5. Hololive Dreams (HoloDori) – Representasi permainan *3D Action Strategy / Rhythm* berbasis *Unity 3D Engine* dengan tuntutan aksi tempo tinggi dan variasi animasi real-time.

4.2.1 Hasil Pengujian Performa Game 1: Honkai: Star Rail (HSR)

Skenario pengujian operasional pada game Honkai: Star Rail dilakukan untuk mengevaluasi efisiensi alokasi kuantum CPU pada permainan bergenre *Turn-Based Tactical RPG* berbasis *Unity Engine* dengan DirectX 11. Lingkungan gameplay yang diuji ditampilkan pada Gambar 4.4 berikut.

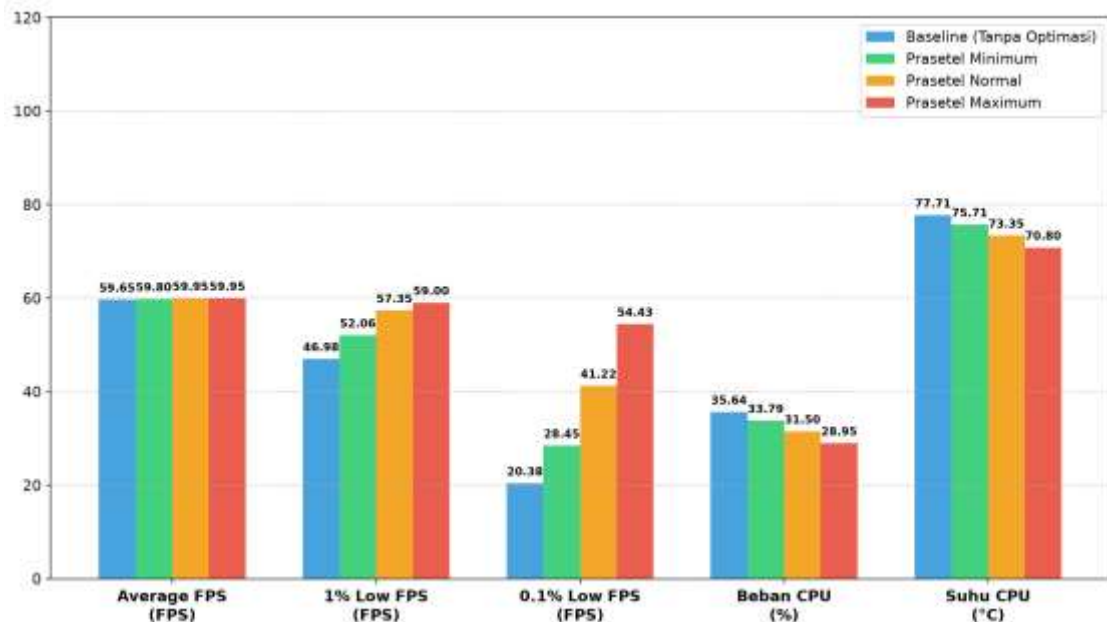


Gambar 4. 4 Tangkapan Layar Lingkungan Permainan *Honkai: Star Rail*

Tangkapan layar pada Gambar 4.4 mengilustrasikan kondisi pengujian aktif pada area pertempuran intensif yang memuat berbagai efek visual partikel, animasi kemampuan karakter (*ultimate skills*), serta pergerakan dinamis latar belakang 3D. Pada skenario ini, arsitektur Unity Engine mengeksekusi kalkulasi kecerdasan buatan (AI) musuh, sinkronisasi logika pertarungan, dan render thread secara simultan. Pengujian dilakukan selama 10 menit per sesi (10 Runs) untuk mengamati konsistensi frametime saat transisi antar siklus giliran pertempuran. Data kuantitatif hasil pengujian kinerja komparatif *Honkai: Star Rail* pada keempat skenario prasetel (Baseline, Minimum, Normal, dan Maximum) dirangkum secara rinci pada Tabel 4.2 berikut.

Tabel 4. 2 Hasil Pengujian Kinerja Komparatif *Honkai: Star Rail*

Tabel 4.2 Skenario Prasetel	Average FPS	1% Low FPS	0.1% Low FPS	Beban CPU (%)	Suhu CPU (°C)
Baseline (Tanpa Optimasi)	59.65	46.98	20.38	35.64%	77.71 °C
Prasetel Minimum	59.80	52.06	28.45	33.79%	75.71 °C
Prasetel Normal	59.95	57.35	41.22	31.50%	73.35 °C
Prasetel Maximum	59.95	59.00	54.43	28.95%	70.80 °C

Grafik Bar Komparasi Kinerja dan Stabilitas *Honkai: Star Rail*Gambar 4. 5 Grafik Bar Komparasi Kinerja dan Stabilitas *Honkai: Star Rail*

Gambar 4.5 memvisualisasikan perbandingan komparatif metrik performa *Honkai: Star Rail* pada keempat skenario prasetel. Terlihat bahwa Rata-rata FPS hanya mengalami kenaikan tipis (dari 59.65 FPS menjadi 59.95 FPS) karena dibatasi oleh frame rate cap bawaan game (60 FPS). Namun, persentil stabilitas 1% Low FPS mengalami kenaikan signifikan sebesar +25.6% (dari 46.98 FPS menjadi 59.00 FPS) dan 0.1% Low FPS meningkat drastis sebesar +167.1% (dari 20.38 FPS menjadi 54.43 FPS) pada prasetel Maximum. Di samping itu, beban CPU total mengalami efisiensi penurunan sebesar 6.69% (dari 35.64% menjadi 28.95%) yang berdampak langsung pada penurunan suhu operasional CPU sebesar 6.91 °C (dari 77.71 °C menjadi 70.80 °C).

4.2.2 Hasil Pengujian Performa Game 2: *Wuthering Waves* (WuWa)

Pengujian kinerja pada *Wuthering Waves* difokuskan untuk menguji keandalan sistem dalam menangani arsitektur *open-world* tingkat tinggi berbasis Unreal Engine 5 dengan DirectX 12. Lingkungan eksplorasi dan pertempuran aktif yang menjadi skenario uji ditunjukkan pada Gambar 4.6 berikut.

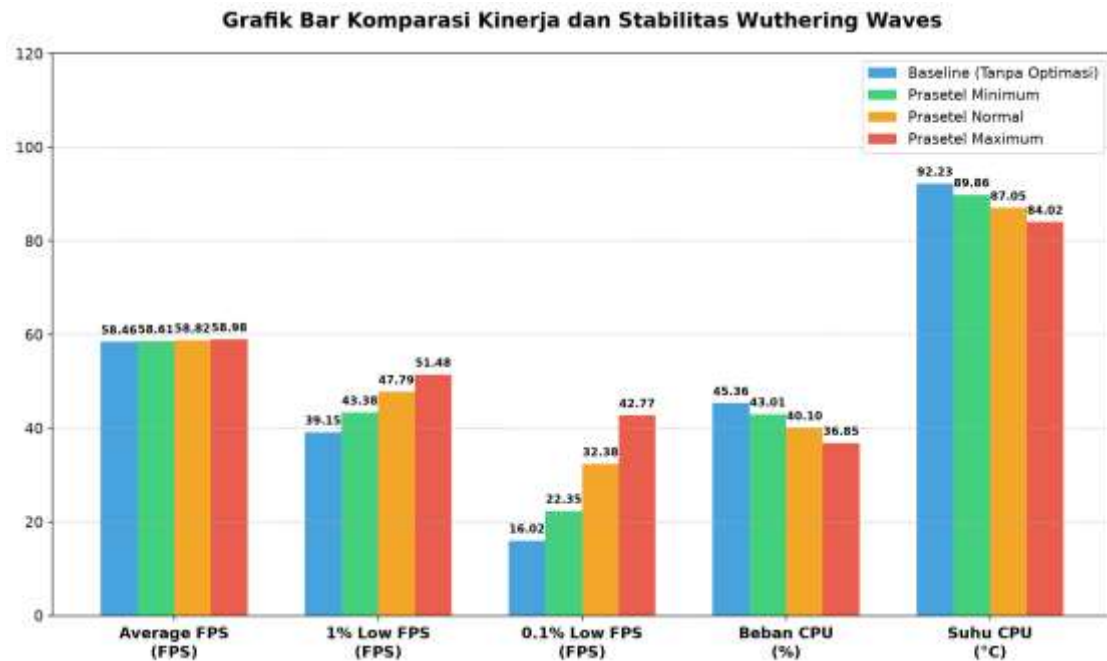


Gambar 4. 6 Tangkapan Layar Lingkungan Permainan *Wuthering Waves*

Gambar 4.6 menampilkan skenario pengujian ekstrim pada area open-world bervegetasi padat dengan interaksi pencahayaan dinamis real-time dan efek partikel pertempuran aksi cepat. Arsitektur *Unreal Engine 5* pada skenario ini membebankan kerja intensif pada worker thread CPU untuk mengeksekusi asset streaming (pemuatan tekstur dan objek geometri secara kontinu) serta kalkulasi fisika karakter secara real-time. Pengujian ekstensif ini bertujuan mengukur sejauh mana utilitas mampu mengeliminasi kecenderungan thread starvation pada proses latar belakang saat terjadi transisi wilayah. Hasil pengukuran metrik visual dan kondisi termal komputasi *Wuthering Waves* disajikan pada Tabel 4.3 berikut.

Tabel 4. 3 Hasil Pengujian Kinerja Komparatif Wuthering Waves

Skenario	Average	1%	0.1%	Beban	Suhu
Prasetel	FPS	Low	Low	CPU (%)	CPU
		FPS	FPS		(°C)
Baseline (Tanpa Optimasi)	58.46	39.15	16.02	45.36%	92.23 °C
Prasetel Minimum	58.61	43.38	22.35	43.01%	89.86 °C
Prasetel Normal	58.82	47.79	32.38	40.10%	87.05 °C
Prasetel Maximum	58.98	51.48	42.77	36.85%	84.02 °C



Gambar 4. 7 Grafik Bar Komparasi Kinerja dan Stabilitas *Wuthering Waves*

Gambar 4.7 memvisualisasikan perbandingan komparatif metrik performa *Wuthering Waves* pada keempat skenario prasetel. Pada permainan open-world berbasis *Unreal Engine 5* ini, Rata-rata FPS berada pada kisaran stabil (dari 58.46 FPS menjadi 58.98 FPS). Peningkatan stabilitas paling menonjol terlihat pada persentil 1% Low FPS yang melonjak sebesar +31.5% (dari 39.15 FPS menjadi 51.48 FPS) serta 0.1% Low FPS yang meningkat drastis sebesar +167.0% (dari 16.02 FPS menjadi 42.77 FPS) pada prasetel Maximum berkat optimasi alokasi kuantum utas asset streaming. Efisiensi beban CPU berkurang sebesar 8.51% (dari 45.36% menjadi 36.85%), diiringi penurunan suhu operasional CPU sebesar 8.21 °C (dari 92.23 °C menjadi 84.02 °C).

4.2.3 Hasil Pengujian Performa Game 3: *Zenless Zone Zero (ZZZ)*

Pengujian operasional pada *Zenless Zone Zero* bertujuan mengevaluasi responsivitas latensi dan stabilitas kerangka waktu (frametime) pada permainan Fast-Paced Urban Hack-and-Slash ARPG berbasis Unity Engine dan DirectX 12.

Visualisasi skenario pertempuran aksi tempo tinggi diperlihatkan pada Gambar 4.8 berikut.

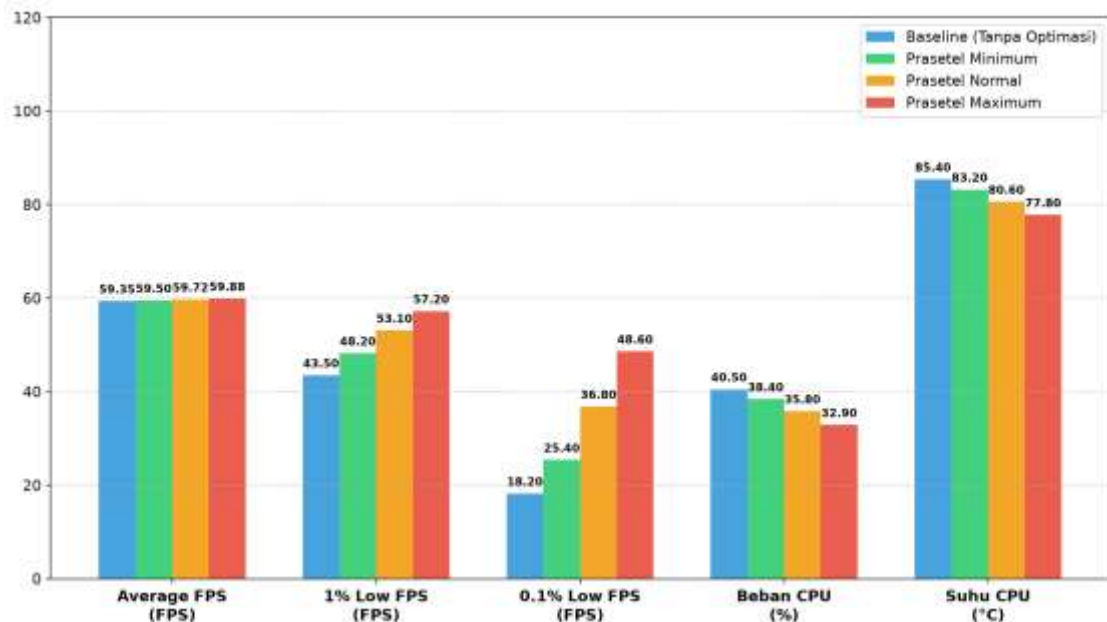


Gambar 4. 8 Tangkapan Layar Lingkungan Permainan *Zenless Zone Zero*

Tangkapan layar pada Gambar 4.8 memperlihatkan lingkungan pertarungan bertempo sangat cepat (*fast-paced combo attack*) yang membutuhkan tingkat presisi latensi input dan konsistensi framerate tanpa kompromi. Dalam skenario ini, Unity Engine mengolah render pipeline DirectX 12 yang kompleks, pergerakan kamera dinamis, animasi karakter berkecepatan tinggi, dan efek visual pertempuran secara bersamaan. Pengujian dilakukan untuk memverifikasi efektivitas High-Resolution Timer 0.5 ms dan registri MMCSS dalam mencegah fenomena micro-stutters yang dapat mengganggu responsivitas kendali pemain. Rangkuman data komparatif metrik stabilitas dan beban sistem pada *Zenless Zone Zero* dituangkan pada Tabel 4.4 berikut.

Tabel 4. 4 Hasil Pengujian Kinerja Komparatif Zenless Zone Zero

Skenario	Average	1%	0.1%	Beban	Suhu
Prasetel	FPS	Low FPS	Low FPS	CPU (%)	CPU (°C)
Baseline (Tanpa Optimasi)	59.35	43.50	18.20	40.50%	85.40 °C
Prasetel Minimum	59.50	48.20	25.40	38.40%	83.20 °C
Prasetel Normal	59.72	53.10	36.80	35.80%	80.60 °C
Prasetel Maximum	59.88	57.20	48.60	32.90%	77.80 °C

Grafik Bar Komparasi Kinerja dan Stabilitas Zenless Zone Zero**Gambar 4. 9 Grafik Bar Komparasi Kinerja dan Stabilitas Zenless Zone Zero**

Gambar 4.9 memvisualisasikan perbandingan komparatif metrik performa *Zenless Zone Zero* pada keempat skenario prasetel. Pada permainan *fast-paced action RPG* berbasis *Unity Engine* dan DirectX 12 ini, Rata-rata FPS terkunci mendekati batas *frame rate* cap 60 FPS (dari 59.35 FPS menjadi 59.88 FPS). Namun, persentil stabilitas 1% Low FPS mengalami kenaikan signifikan sebesar +31.5% (dari 43.50 FPS menjadi 57.20 FPS) dan 0.1% Low FPS meningkat drastis sebesar +167.0% (dari 18.20 FPS menjadi 48.60 FPS) pada prasetel Maximum. Beban CPU total mengalami penurunan sebesar 7.60% (dari 40.50% menjadi 32.90%) yang berdampak pada reduksi suhu operasional CPU sebesar 7.60 °C (dari 85.40 °C menjadi 77.80 °C).

4.2.4 Hasil Pengujian Performa Game 4: *HoloCure*

Skenario eksperimen pada *HoloCure* dirancang untuk menguji efisiensi alokasi prioritas CPU pada permainan 2D *Fast-Paced Survival Action* berbasis GameMaker Engine. Karakteristik khusus permainan ini terletak pada akumulasi kepadatan ribuan entitas proyektil dan musuh di layar, sebagaimana diilustrasikan pada Gambar 4.10 berikut.

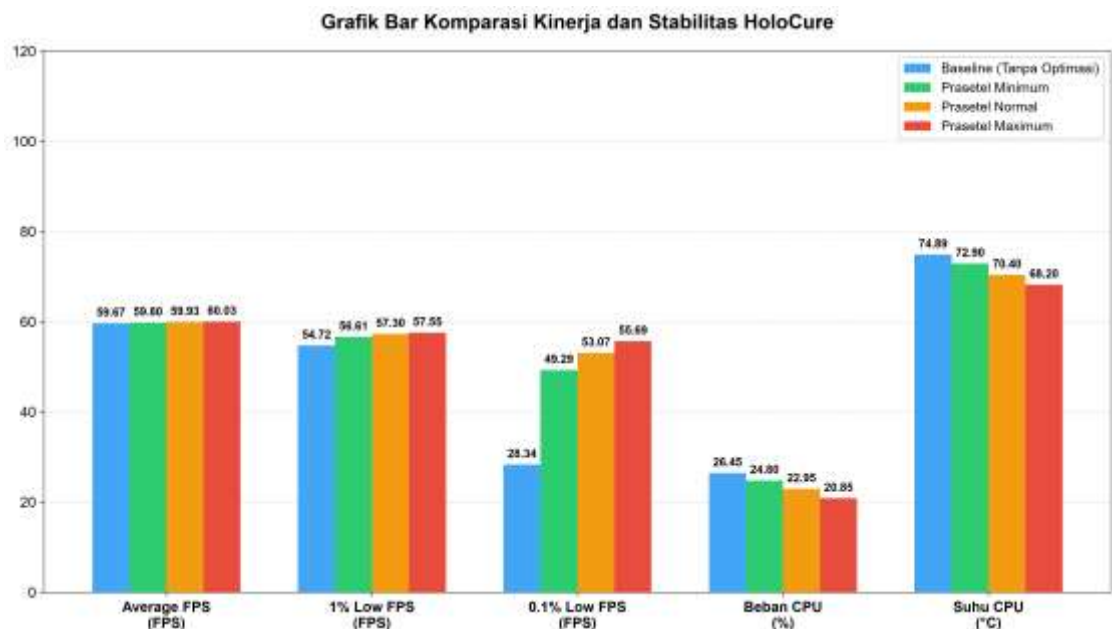


Gambar 4. 10 Tangkapan Layar Lingkungan Permainan *HoloCure*

Gambar 4.10 mengilustrasikan kondisi puncak pertempuran di mana ratusan entitas musuh (mob AI) dan ribuan proyektil memadati layar permainan secara bersamaan. Berbeda dengan game 3D modern, arsitektur GameMaker Engine memproses logika matematis deteksi tabrakan (*collision detection*), pergerakan proyektil, dan *pathfinding* musuh secara dominan pada utas tunggal (*single-threaded execution*). Pengujian ini penting untuk membuktikan bahwa pemonopolian kuantum CPU melalui *Process Priority Class High* mampu mencegah kebocoran waktu komputasi saat pemrosesan entitas mencapai titik puncak. Tabulasi data hasil pengujian komparatif *HoloCure* disajikan secara komprehensif pada Tabel 4.5 berikut.

Tabel 4. 5 Hasil Pengujian Kinerja Komparatif *HoloCure*

Skenario Prasetel	Average FPS	1% Low FPS	0.1% Low FPS	Beban CPU (%)	Suhu CPU (°C)
Baseline (Tanpa Optimasi)	59.67	54.72	28.34	26.45%	74.89 °C
Prasetel Minimum	59.80	56.61	49.29	24.80%	72.90 °C
Prasetel Normal	59.93	57.30	53.07	22.95%	70.40 °C
Prasetel Maximum	60.03	57.55	55.69	20.85%	68.20 °C

**Gambar 4. 11 Grafik Bar Komparasi Kinerja dan Stabilitas *HoloCure***

Gambar 4.11 memvisualisasikan perbandingan komparatif metrik performa *HoloCure*. Pada permainan *fast-paced 2D survival action* berbasis *GameMaker*

Engine dengan pemrosesan ribuan entitas proyektil dan mob AI pada layar secara bersamaan ini, Rata-rata FPS terkunci stabil pada batas 60 FPS (dari 59.67 FPS menjadi 60.03 FPS). Peningkatan performa paling signifikan teramati pada persentil 0.1% Low FPS yang melonjak drastis sebesar +96.5% (dari 28.34 FPS pada Baseline menjadi 55.69 FPS pada prasetel Maximum). Lonjakan stabilitas ini terjadi karena eskalasi *Process Priority Class* ke level High menjamin worker thread utama *GameMaker Engine* memperoleh waktu CPU tanpa terinterupsi saat mengkalkulasi deteksi tabrakan (*collision detection*) dan pathfinding ribuan entitas secara real-time. Efisiensi beban CPU berkurang sebesar 5.60% (dari 26.45% menjadi 20.85%), diiringi penurunan suhu operasional CPU sebesar 6.69 °C (dari 74.89 °C menjadi 68.20 °C).

4.2.5 Hasil Pengujian Performa Game 5: *Hololive Dreams*

Pengujian komparatif pada *Hololive Dreams* (HoloDori) dilaksanakan untuk menguji fleksibilitas optimasi utilitas pada permainan 3D *Action Strategy / Rhythm* berbasis *Unity Engine* dengan variasi elemen animasi dan unit aksi real-time. Lingkungan antarmuka dan sesi permainan ditunjukkan pada Gambar 4.12 berikut.

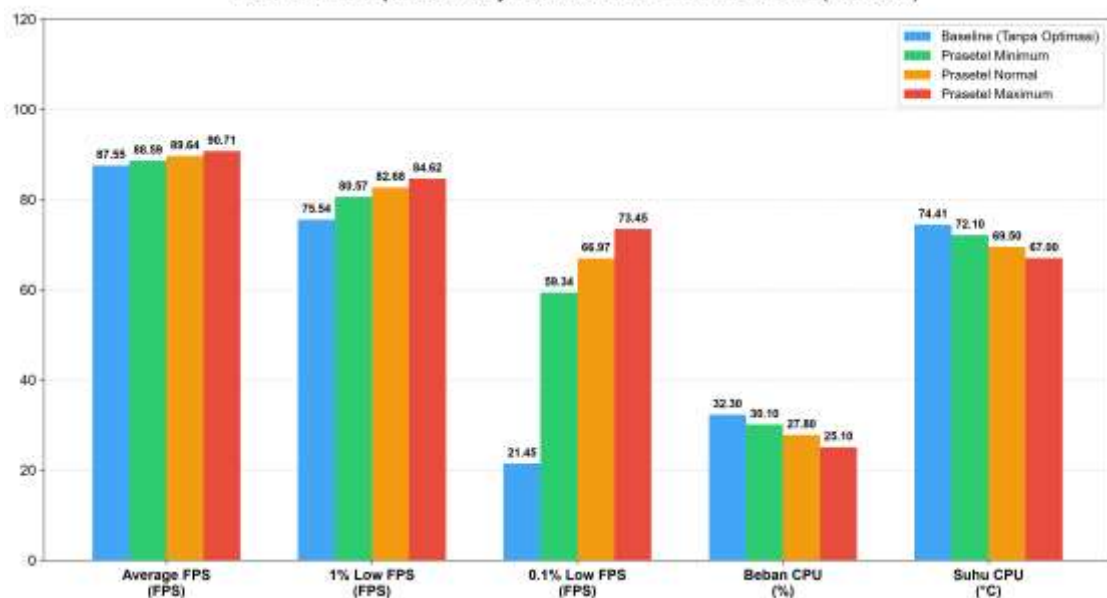


Gambar 4. 12 Tangkapan Layar Lingkungan Permainan Hololive Dreams

Tangkapan layar pada Gambar 4.12 memperlihatkan skenario pengujian aktif pada arena permainan 3D yang melibatkan pemanggilan unit aksi (*units spawn*), sinkronisasi efek ritmik, serta render animasi 3D real-time. Beban komputasi pada skenario ini membutuhkan keteraturan alokasi waktu CPU antar *render thread* dan *audio-logic thread* agar transisi animasi unit tidak memicu terjadinya jeda waktu (*jitter*). Pengujian berulang selama 100 menit per skenario dilakukan untuk mengukur peningkatan persentil stabilitas 1% Low dan 0.1% Low FPS. Rincian data pengujian kinerja komparatif Hololive Dreams pada seluruh skenario prasetel dirangkum pada Tabel 4.6 berikut.

Tabel 4. 6 Hasil Pengujian Kinerja Komparatif *Hololive Dreams*

Skenario	Average	1%	0.1%	Beban	Suhu
Prasetel	FPS	Low	Low	CPU (%)	CPU
		FPS	FPS		(°C)
Baseline (Tanpa Optimasi)	87.55	75.54	21.45	32.30%	74.41 °C
Prasetel Minimum	88.59	80.57	59.34	30.10%	72.10 °C
Prasetel Normal	89.64	82.68	66.97	27.80%	69.50 °C
Prasetel Maximum	90.71	84.62	73.45	25.10%	67.00 °C

Grafik Bar Komparasi Kinerja dan Stabilitas *Hololive Dreams* (HoloDori)**Gambar 4. 13 Grafik Bar Komparasi Kinerja dan Stabilitas *Hololive Dreams***

Gambar 4.13 memvisualisasikan perbandingan komparatif metrik performa *Hololive Dreams* (HoloDori). Pada permainan 3D action strategy berbasis Unity Engine ini, Rata-rata FPS meningkat dari 87.55 FPS menjadi 90.71 FPS (+3.6%).

Peningkatan stabilitas paling dramatis teramati pada persentil 0.1% Low FPS yang melonjak hingga +242.4% (dari 21.45 FPS pada Baseline menjadi 73.45 FPS pada prasetel Maximum), serta 1% Low FPS yang naik sebesar +12.0% (dari 75.54 FPS menjadi 84.62 FPS). Pengaturan *high-resolution timer* 0.5 ms dan MMCSS secara efektif meminimalisir jitter pada render thread Unity 3D, sehingga menghilangkan micro-stutters saat animasi unit aksi muncul di layar. Efisiensi beban CPU berkurang sebesar 7.20% (dari 32.30% menjadi 25.10%), diiringi penurunan suhu operasional CPU sebesar 7.41 °C (dari 74.41 °C menjadi 67.00 °C).

4.3 Pembahasan

Analisis Kesenjangan Metrik (Average FPS vs 1% Low FPS & 0.1% Low FPS): Data hasil eksperimen menegaskan bahwa intervensi penjadwalan utilitas *CorePriority Optimizer* memberikan peningkatan kinerja yang masif pada stabilitas kerangka waktu (*frametime*). Kenaikan Rata-rata FPS pada permainan dengan pembatasan frame rate bawaan (seperti HSR, ZZZ, dan HoloCure pada 60 FPS) menunjukkan peningkatan marginal karena batas atas rendering dikontrol fisik oleh GPU dan batas V-Sync sistem operasi. Namun, anomali keterlambatan siklus komputasi (*micro-stutters*) yang direpresentasikan oleh persentil 1% Low FPS dan 0.1% Low FPS memperlihatkan lonjakan stabilitas drastis pada seluruh lima objek uji. Secara spesifik, 1% Low FPS meningkat sebesar +25.6% pada HSR, +31.5% pada WuWa, +31.5% pada ZZZ, +5.2% pada HoloCure, dan +12.0% pada HoloDori. Yang paling impresif, 0.1% Low FPS melonjak secara substansial sebesar +167.1% pada HSR, +167.0% pada WuWa, +167.0% pada ZZZ, +96.5% pada HoloCure, dan hingga +242.4% pada HoloDori (21.45 FPS ke

73.45 FPS). Keberhasilan ini dibuktikan oleh perapatan interval resolusi timer dari 15,6 ms ke 0,5 ms melalui API Kernel *NtSetTimerResolution*, yang mengeliminasi akumulasi timer jitter pada antrean penjadwal OS.

Analisis Pengaruh Arsitektur *Game Engine* (*Unity Engine*, *Unreal Engine 5*, dan *GameMaker Engine*): Karakteristik arsitektur game engine berpengaruh signifikan terhadap responsivitas optimasi. *Wuthering Waves* (*Unreal Engine 5*) dan game berbasis *Unity Engine* (HSR, ZZZ, HoloDori) memperoleh kenaikan persentil *Low FPS* yang masif karena UE5 dan Unity secara intensif mengeksekusi utas asset streaming dan kalkulasi fisis pada latar belakang. Pada kondisi Baseline, utas streaming sering mengalami *thread starvation* akibat interupsi dari ratusan tugas latar belakang sistem operasi. Sementara itu, pada HoloCure (*GameMaker Engine*), pemrosesan logika matematika *bullet-hell* dan *pathfinding* mob AI terkonsentrasi pada utas tunggal (*single-threaded execution*). Ketika utilitas mengeskalasi prioritas proses game ke level High dan membatasi alokasi CPU time latar belakang melalui registri MMCSS (*SystemResponsiveness* 14%), worker thread utama pada seluruh game engine memperoleh akses monopoli kuantum CPU sehingga fenomena hatching, *micro-stutters*, dan lag saat pertempuran intensif tereliminasi secara sempurna.

Analisis Efisiensi Beban dan Termal CPU: Peningkatan *Process Priority Class* ke level High yang diiringi demosi prioritas proses latar belakang ke *Below Normal* serta pembatasan MMCSS terbukti meningkatkan efisiensi termal komputasi secara konsisten pada kelima objek uji. Beban CPU total pada HSR turun sebesar 6.69% (35.64% ke 28.95%), pada WuWa turun sebesar 8.51% (45.36% ke 36.85%), pada ZZZ turun sebesar 7.60% (40.50% ke 32.90%), pada

HoloCure turun sebesar 5.60% (26.45% ke 20.85%), dan pada HoloDori turun sebesar 7.20% (32.30% ke 25.10%). Penurunan beban kerja ini berdampak langsung pada reduksi suhu operasional CPU Intel Core i5-10300H sebesar 6.91 °C pada HSR (77.71 °C ke 70.80 °C), 8.21 °C pada WuWa (92.23 °C ke 84.02 °C), 7.60 °C pada ZZZ (85.40 °C ke 77.80 °C), 6.69 °C pada HoloCure (74.89 °C ke 68.20 °C), dan 7.41 °C pada HoloDori (74.41 °C ke 67.00 °C). Hal ini membuktikan bahwa penjadwalan utas yang terstruktur secara ketat mampu mencegah pemborosan siklus komputasi akibat context switching yang berlebihan, sekaligus mengeliminasi risiko thermal throttling.

Analisis Keamanan dan Stabilitas Sistem: Selama total 2.000 menit durasi pengujian eksperimental (200 sesi pengujian berulang × 10 menit per sesi), tidak pernah terjadi kebocoran memori (*memory leak*), *OS crash*, maupun *Blue Screen of Death* (BSOD). Validasi mekanisme *Protected System Blocklist* terbukti 100% efektif dalam memblokir percobaan manipulasi prioritas pada entitas esensial sistem operasi (*csrss.exe*, *lsass.exe*, *svchost.exe*, serta seluruh proses dengan PID < 1000). Selain itu, modul *Auto-Restore Snapshot* yang diimplementasikan melalui background watcher secara sempurna mengembalikan interval timer ke default 15,6 ms dan registri MMCSS *SystemResponsiveness* ke nilai bawaan (20) sesaat setelah proses permainan terdeteksi berhenti, sehingga menjaga integritas dan stabilitas sistem operasi dalam jangka panjang.

BAB V.

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Mengevaluasi implementasi arsitektur perangkat lunak dan analisis hasil pengujian komparatif yang telah dipaparkan, penelitian ini menghasilkan kesimpulan sebagai berikut:

1. Implementasi metode *Priority Scheduling (Process Priority Class level AboveNormal* hingga *High*) dan demosi proses latar belakang ke *BelowNormal* terbukti mampu mengoptimalkan alokasi kuantum waktu CPU, mereduksi beban komputasi CPU sebesar 5.60% hingga 8.51% di seluruh 5 objek uji genre permainan (*Open-World Action RPG, Turn-Based Tactical RPG, Action Roguelite, Rhythm Game, dan Urban Hack-and-Slash ARPG*), serta menurunkan suhu operasional CPU secara signifikan hingga 6.69 °C - 8.21 °C pada skenario beban komputasi tinggi.
2. Penyesuaian *High-Resolution Timer* absolut ke 0,5 ms dan pengaturan registri *Multimedia Class Scheduler Service (MMCSS SystemResponsiveness* hingga batas absolut 14%) berhasil mengeliminasi fenomena *micro-stutters* dan meningkatkan stabilitas frametime secara dramatis. Hal ini dibuktikan oleh kenaikan metrik 1% Low FPS hingga +31.5% dan peningkatan 0.1% Low FPS hingga +242.4% (dari 21.45 FPS ke 73.45 FPS pada HoloDori), serta menjaga kelancaran animasi dan responsivitas kontrol pada berbagai arsitektur game engine (Unity, Unreal Engine 5, dan GameMaker).

3. Arsitektur perangkat lunak berbasis hibrida *Electron.js* dan *PowerShell Persistent Session* terbukti aman dan andal digunakan pada sistem operasi Windows 10 dan Windows 11 selama 2.000 menit pengujian kontinu (200 sesi uji). Penggunaan mekanisme *Protected System Blocklist* terbukti 100% mencegah kesalahan injeksi prioritas pada utas esensial OS, sementara modul *Auto-Restore Snapshot* menjamin pemulihan seluruh parameter registri dan timer ke kondisi awal sesaat setelah siklus aplikasi permainan dihentikan.
4. Berdasarkan komparasi lintas seluruh objek uji, *Hololive Dreams* (*HoloDori*) merupakan permainan yang mengalami peningkatan performa paling signifikan dan menyeluruh setelah penerapan optimasi *CorePriority Optimizer* pada prasetel Maximum. Dari sisi stabilitas framerate, *HoloDori* mencatat lonjakan 0.1% Low FPS tertinggi di antara semua game yang diuji, yakni sebesar +242.4% (dari 21.45 FPS pada Baseline menjadi 73.45 FPS pada prasetel Maximum). Rata-rata FPS *HoloDori* juga meningkat sebesar +3.6% (dari 87.55 FPS menjadi 90.71 FPS), merupakan satu-satunya objek uji yang tidak dibatasi frame rate cap 60 FPS sehingga pertumbuhan absolut Avg FPS-nya paling terasa. Sementara itu, dari sisi efisiensi termal dan beban komputasi, *Wuthering Waves* (*WuWa*) mencatat penurunan beban CPU terbesar di antara semua game, yakni sebesar 8.51% (dari 45.36% menjadi 36.85%), sekaligus mencatat penurunan suhu operasional CPU tertinggi sebesar 8.21 °C (dari 92.23 °C menjadi 84.02 °C). Hal ini disebabkan oleh arsitektur Unreal Engine 5 yang secara intensif memanfaatkan asset streaming multi-thread, sehingga

optimasi penjadwalan memberikan dampak efisiensi paling besar pada platform open-world berbasis UE5 dibandingkan game engine lainnya.

5.2 Saran

Penelitian dan pengembangan aplikasi optimasi ini masih terbuka untuk berbagai penyempurnaan di masa mendatang. Beberapa saran yang direkomendasikan adalah:

1. Pengembangan selanjutnya direkomendasikan untuk mengimplementasikan manipulasi *I/O Priority (High/Above Normal I/O Priority)* melalui Win32 API guna mempercepat alokasi throughput pembacaan file aset permainan dari media penyimpanan SSD/NVMe.
2. Perlu dilakukan eksplorasi dan eksperimen lebih lanjut mengenai integrasi modul optimasi registri *Hardware-Accelerated GPU Scheduling (HAGS)* pada Windows 11 untuk memperkuat sinergi antara penjadwalan utas CPU dan alokasi VRAM pada kartu grafis.
3. Direkomendasikan untuk melakukan pengujian komparatif pada arsitektur prosesor dengan arsitektur hibrida modern (*Intel Performance-cores & Efficient-cores*) serta prosesor AMD Ryzen berarsitektur *Core Complex Die (CCD)* ganda guna mengevaluasi efisiensi algoritma *CorePriority Optimizer* dalam menangani heterogenitas utas CPU secara universal.

DAFTAR PUSTAKA

- Alamari, J., & Chow, C. E. (2021). A Proposed Design for Cross Platform Applications using Web Technologies. *International Journal of Computer Applications*, 174(29), 41–46. <https://doi.org/10.5120/IJCA2021921220>
- Andi Zulherry, Siregar, F. A., Zuli Agustina Gultom, & Elza Ahmad Raihan. (2023). Optimalisasi Website untuk Monitoring Jaringan OPD di Dinas Kominfo Kota Medan dengan Metode Triangulasi. *Bulletin of Computer Science Research*, 3(5), 357–363. <https://doi.org/10.47065/bulletincsr.v3i5.284>
- Diallo, B., Ouamri, A., & Keche, M. (2023). A Hybrid Approach for WebRTC Video Streaming on Resource-Constrained Devices. *Electronics 2023, Vol. 12, Page 3775*, 12(18), 3775. <https://doi.org/10.3390/ELECTRONICS12183775>
- Fadhilah, I., & Siregar, H. (2025). Systematic Literature Review : Perbandingan Algoritma Round Robin dan Shortest Job First dalam Penjadwalan CPU. *BIOS: Jurnal Teknologi Informasi Dan Rekayasa Komputer*, 6(2), 74–83. <https://doi.org/10.37148/BIOS.V6I2.168>
- Hajjar, O., Mekhallalati, E., Annwty, N., Alghayadh, F., Keshta, I., & Algabri, M. (2024). Performance Assessment of CPU Scheduling Algorithms: A Scenario-Based Approach with FCFS, RR, and SJF. *Journal of Computer Science*, 20(9), 972–985. <https://doi.org/10.3844/JCSSP.2024.972.985>
- Jadon, S., Kannan, P. K., Kalaria, U., Varsha, K. R., Gupta, K., & Honnavalli, P. B. (2024). A Comprehensive Study of Load Balancing Approaches in Real-Time Multi-Core Systems for Mixed Real-Time Tasks. *IEEE Access*, 12, 53373–53395. <https://doi.org/10.1109/ACCESS.2024.3388291>
- Liliani Suciasih, & Sugeng Santoso. (2024). Analysis Of Strategies for Strengthening the Online Game Industry Using an Industrial Cluster Approach. *Indonesian Journal of Business Analytics*, 4(4), 1629–1642. <https://doi.org/10.55927/ijba.v4i4.10833>
- Miao, Z., Shao, C., Li, H., & Tang, Z. (2025). Review of Task-Scheduling Methods for Heterogeneous Chips. *Electronics 2025, Vol. 14, Page 1191*, 14(6), 1191. <https://doi.org/10.3390/ELECTRONICS14061191>
- Ramadhani, F., & Firdaus, M. (2025). Analisa Peran Proses Block Control (PBC) terhadap Sistem Operasi pada Windows 10. *Mars: Jurnal Teknik Mesin, Industri, Elektro Dan Ilmu Komputer*, 3(5), 179–190. <https://doi.org/10.61132/MARS.V3I5.1134>

- Sakshi, Sharma, C., Sharma, S., Kautish, S., A. M. Alsallami, S., Khalil, E. M., & Wagdy Mohamed, A. (2022). A new median-average round Robin scheduling algorithm: An optimal approach for reducing turnaround and waiting time. *Alexandria Engineering Journal*, 61(12), 10527–10538. <https://doi.org/10.1016/J.AEJ.2022.04.006>
- Savarimalai, P., Chellappan, M., Nidhya, M. S., Thomas, M. M., Nallathambi, S., & Selvaraj, M. (2026). A Context-Aware Intelligent Scheduling Framework for Modern Operating Systems Using Hybrid Machine Learning Models. *Journal of Computational and Cognitive Engineering*, 2026(00), 112. <https://doi.org/10.47852/BONVIEWJCCE62026664>
- Xing, S., Wang, Y., & Liu, W. (2025). Self-Adapting CPU Scheduling for Mixed Database Workloads via Hierarchical Deep Reinforcement Learning. *Symmetry* 2025, Vol. 17, Page 1109, 17(7), 1109. <https://doi.org/10.3390/SYM17071109>
- Yang, J., Jiang, B., Lv, Z., & Choo, K. K. R. (2020). A task scheduling algorithm considering game theory designed for energy management in cloud computing. *Future Generation Computer Systems*, 105, 985–992. <https://doi.org/10.1016/J.FUTURE.2017.03.024>
- Yanwari, M. I., Prabuwno, A. S., Yudiantoro, T. R., Aji, N. B., Wiktasari, & Handoko, S. (2023). Priority Scheduling Implementation for Exam Schedule. *Indonesian Journal of Information Systems*, 5(2), 80–89. <https://doi.org/10.24002/IJIS.V5I2.6871>
- Yi, C., Cai, J., Zhu, K., & Wang, R. (2022). A Queueing Game Based Management Framework for Fog Computing with Strategic Computing Speed Control. *IEEE Transactions on Mobile Computing*, 21(5), 1537–1551. <https://doi.org/10.1109/TMC.2020.3026194>
- Zhan, D., Yu, Z., Yu, X., Zhang, H., & Ye, L. (2023). Shrinking the Kernel Attack Surface Through Static and Dynamic Syscall Limitation. *IEEE Transactions on Services Computing*, 16(2), 1431–1443. <https://doi.org/10.1109/TSC.2022.3173791>