Polygon: Jurnal Ilmu Komputer dan Ilmu Pengetahuan Alam
Volume. 3 Nomor. 6 November 2025
e-ISSN: 3046-5419; p-ISSN: 3032-6249, Hal 38-49
DOI: https://doi.org/10.62383/polygon.v3i6.816
Tersedia: https://journal.arimsi.or.id/index.php/Polygon

Perbandingan Mekanisme Sinkronisasi *Mutex* dan *Semaphore* pada Sistem Operasi *Android*

Ghaniyah Latifa Putri[1*], Rasya Mulki Putra[2], Harits Rahadi[3]
[1-3]Program Studi Manajemen Informatika, Akademi Manajemen Informatika dan Komputer (AMIK) Bukittinggi, Indonesia
**Penulis Korespondensi: putrigania.25@gmail.com*

Abstract: *The synchronization of mutex and semaphore mechanisms is an important technique in operating systems for managing concurrent resource access in multi-threaded environments. In the Android operating system, which relies on thread-based programming to maintain performance and responsiveness, these two mechanisms play a vital role in preventing race conditions and ensuring data integrity. A mutex is a locking object that ensures that only one thread can access a specific resource at a time, while a semaphore controls access to a limited number of resources by counting the number of threads that can access them simultaneously. In Android, these mechanisms are implemented using the Lock class for mutexes and the Semaphore class for semaphores, both of which can be used to control synchronization between threads in Android applications. This paper will discuss the application of these two mechanisms in the context of shared resource management, as well as a comparison of their performance and advantages in dealing with complex multi-threading scenarios on Android. Emphasis will also be placed on the challenges faced in using these two mechanisms in Android applications, as well as how proper programming can avoid deadlocks and improve application efficiency.*

Keywords: *Android; Concurrency; Deadlock Prevention; Mutex; Semaphore.*

Abstrak: Sinkronisasi mekanisme mutex dan semaphore merupakan teknik penting dalam sistem operasi untuk mengelola akses sumber daya secara bersamaan pada lingkungan multi-threaded. Di sistem operasi Android, yang berorientasi pada pemrograman berbasis thread untuk menjaga performa dan responsivitas, kedua mekanisme ini memainkan peran vital dalam mencegah kondisi balapan (race condition) dan memastikan integritas data. Mutex adalah objek pengunci yang memastikan bahwa hanya satu thread yang dapat mengakses sumber daya tertentu pada satu waktu, sementara semaphore mengontrol akses ke sejumlah terbatas sumber daya, dengan menghitung jumlah thread yang dapat mengakses secara bersamaan. Di Android, mekanisme ini diimplementasikan menggunakan kelas Lock untuk mutex dan Semaphore untuk semafor, yang keduanya dapat digunakan untuk mengontrol sinkronisasi antar thread dalam aplikasi Android. Dalam penelitian ini, akan dibahas penerapan kedua mekanisme ini dalam konteks pengelolaan sumber daya bersama, serta perbandingan kinerja dan kelebihan masing-masing dalam menghadapi skenario multi-threading yang kompleks di Android. Penekanan juga akan diberikan pada tantangan yang dihadapi dalam penggunaan kedua mekanisme tersebut pada aplikasi Android, serta bagaimana pemrograman yang tepat dapat menghindari deadlock dan meningkatkan efisiensi aplikasi.

Kata kunci: Android; Konkurensi; Mutex; Pencegahan Kebuntuan; Semaphore.

1. LATAR BELAKANG

Kinerja sistem operasi android menjadi semakin penting seiring dengan kemajuan teknologi dan kebutuhan komputasi yang semakin kompleks(Safrudin, N., 2025) Thread digunakan saat menjalankan berbagai proses untuk mendukung eksekusi mutex dan semaphore, yang memungkinkan beberapa tugas berjalan secara bersamaan dan mempercepat penyelesaian tugas (Malik, 2025). Namun, masalah seperti Deadlock, Race condition, starvation dapat muncul sebagai akibat dari penggunaan thread. Cara terbaik untuk mengatasi masalah ini adalah dengan menggunakan mekanisme sinkronisasi semaphore dan mutex (Rivalni, 2025).

Naskah Masuk: 25 September 2025; Revisi: 09 Oktober 2025; Diterima: 26 Oktober 2025; Tersedia: 29 Oktober 2025

Dalam pengembangan aplikasi Android, pemanfaatan multitasking berbasis thread merupakan praktik umum untuk meningkatkan responsivitas dan efisiensi sistem. Namun, pengelolaan thread yang baik membutuhkan pemahaman yang mendalam terhadap mekanisme sinkronisasi agar berbagai proses dapat berjalan secara terkoordinasi tanpa menimbulkan konflik atau gangguan kinerja. Salah satu aspek penting dalam konteks ini adalah penggunaan *mutex* dan *semaphore*.Sayangnya, masih banyak pengembang yang belum memahami perbedaan fundamental antara kedua mekanisme tersebut. Akibatnya, sering terjadi kesalahan dalam implementasi sinkronisasi, seperti penggunaan *mutex* untuk kasus yang lebih tepat ditangani oleh *semaphore*, atau sebaliknya. Kesalahan-kesalahan semacam ini dapat menyebabkan berbagai permasalahan serius, Terlepas dari fakta bahwa Android menggunakan model multitasking berbasis thread, masih ada kekurangan pada mutex dan semaphore. Hal ini dapat mengakibatkan penggunaan mekanisme sinkronisasi yang tidak tepat, yang dapat menyebabkan *deadlock, race condition, inversion* dan *starvation.*

Dalam sistem operasi Android, baik mutex maupun semaphore berperan penting dalam pengaturan akses terhadap sumber daya seperti file sistem, jaringan, maupun memori. Implementasi mekanisme sinkronisasi tersebut mendukung efisiensi multitasking dan stabilitas sistem agar tidak terjadi deadlock, race condition, atau ketidakkonsistenan data saat beberapa aplikasi berjalan secara paralel. Oleh karena itu, perbandingan antara mekanisme mutex dan semaphore menjadi penting untuk memahami bagaimana Android mengatur proses sinkronisasi agar dapat memberikan kinerja yang optimal pada setiap kondisi penggunaan. Android, yang dijalankan dalam ruang alamat proses. adalah cara untuk meningkatkan proses Mutex dan semaphore dengan memungkinkan beberapa Android bekerja bersamaan dalam satu proses.(Genggam et al, 2024).lebih tepatnya, dengan membagi komponen tugas saat ini dan memprosesnya secara bersamaan (Nezha et al, 2023). Pengenalan platform media sosial baru , yang dikembangkan oleh CEO Meta Mark Zuckerberg, telah menarik banyak orang dari berbagai latar belakang. Dalam waktu hanya lima hari, basis pengguna Android telah melampaui 100 juta pengguna (Meliani, et al, 2023). Diharapkan dapat meningkatkan kinerja sistem Android efisiensi pengolahan data, dan produktivitas secara keseluruhan dengan menerapkan pendekatan yang tepat.

2. KAJIAN TEORITIS

Penelitian sebelumnya melihat bagaimana kedua mekanisme sinkronisasi mutex dan semaphore berfungsi. Sinkronisasi yang efektif sangat penting untuk menjaga kinerja dan keandalan sistem karena memastikan bahwa proses yang bersamaan dapat mengakses sumber

daya bersama dengan aman (Yusuf, 2022). Tujuan dari penelitian ini adalah untuk menemukan mekanisme sinkronisasi di platform mobile yang menyeimbangkan efisiensi, yang diukur melalui waktu eksekusi, dan konsistensi, yang dinilai melalui varians dan deviasi standar (Indrawan, 2020). Hipotesis awal berpendapat bahwa mekanisme berbasis android, terutama metode tersinkronisasi, akan paling efisien karena sederhana. Namun, hasil empiris menunjukkan bahwa mutex memiliki waktu eksekusi rata-rata terendah (14,67 milidetik), menjadikannya mekanisme yang paling efisien tetapi dengan variabilitas tertinggi (deviasi standar 1,15). Sebaliknya, semaphore tersinkronisasi memiliki waktu eksekusi rata-rata yang lebih tinggi (16,33 milidetik untuk metode dan 16,67 milidetik untuk blok), tetapi dengan konsistensi yang lebih tinggi (varians 0,33) (Fauzi, 2021). Temuan menunjukkan bahwa mekanisme berbasis mutex memiliki kinerja yang lebih terprediksi di semua sistem operasi, meskipun semaphore lebih cepat, kecepatan tersebut lebih bergantung pada platform. Pengujian asli, studi mekanisme sinkronisasi tambahan, dan tingkat kompetisi yang lebih tinggi harus menjadi subjek penelitian tambahan. Fakta-fakta ini membantu pengembang dan perancang sistem mengoptimalkan strategi sinkronisasi untuk stabilitas dan kinerja yang sesuai dengan kebutuhan aplikasi(Hidayat, 2019).

3. METODE PENELITIAN

Sumber Informasi data yang kami ambil hanya satu jenis,yaitu kami menggunakan data sekunder pada metode penelitian ini.Data sekunder berasal dari berbagai jurnal ilmiah, buku teks, dan literatur, serta sumber daring terpercaya yang membahas teori sinkronisasi proses, multithreading, mutex, semaphore, dan implementasinya pada sistem operasi Android.

Teknik untuk Analisis Data

Penelitian ini menerapkan pendekatan deskriptif kualitatif yang menggunakan pendekatan komparatif.Parameter seperti kecepatan eksekusi, penggunaan sumber daya (CPU dan memori), dan kestabilan sistem dalam kondisi multitasking digunakan untuk menganalisis data yang diperoleh dengan membandingkan kinerja dan efektivitas mutex dan semaphore dalam menangani proses sinkronisasi thread pada sistem operasi Android.Hasil analisis digunakan untuk menentukan mekanisme sinkronisasi terbaik untuk situasi tertentu dan memberikan saran kepada pengembang sistem Android.Untuk memastikan hasil penelitian dapat dipertanggungjawabkan secara ilmiah, peneliti menggunakan model pengembangan perangkat lunak Waterfall dalam proses perancangan dan pengujian sistem. Proses ini terdiri dari tahapan analisis kebutuhan, desain sistem, pengkodean, pengujian, implementasi, dan

perawatan sistem. Tahapan-tahap ini dilakukan secara sistematis. Tahapan-Tahapan analisis kebutuhan pada artikel ini adalah sebagai berikut (Aulia et al, 2025):

Perencanaan Sistem (System Planning)

Tahap ini bertujuan untuk merencanakan sistem uji coba yang akan dikembangkan guna membandingkan performa mutex dan semaphore. Peneliti mengidentifikasi kebutuhan sistem, menentukan variabel yang akan diuji (seperti jumlah thread, durasi proses, dan tingkat beban CPU), serta mengumpulkan referensi mengenai teknik sinkronisasi pada sistem operasi Android. Selain itu, dilakukan analisis kelayakan penelitian dengan mengumpulkan data pendukung melalui studi pustaka dan pengamatan terhadap lingkungan pengembangan Android Studio.

Analisis Sistem

Berbagai program yang dijalankan dalam meningkatkan literasi keuangan di kalangan siswa akan berjalan dengan baik jika program tersebut dijalankan secara berkelanjutan dalam mengedukasi pengelolaan keuangan sejak dini (Pratama et al., 2025; Sudarti, Sulaeha, & Mabe Parenreng, 2025). Karena lingkungan sekolah merupakan tempat yang potensial dalam meningkatkan pola pikir terhadap literasi keuangan yang outputnya berupa kesadaran untuk berinvestasi (Kurniasari, Pandowo, & Isnaningsih, 2023). Selain itu, dukungan suasana belajar yang kondusif serta usia pelajar yang masih dalam usia produktif untuk belajar hal yang berkaitan dengan literasi keuangan sangat bermanfaat bagi mereka di masa sekarang hingga masa depannya nanti (Aziz, Juliansyah, & Munir, 2024).

Desain Logis Sistem

Untuk membandingkan kedua mekanisme sinkronisasi, tahap ini berkonsentrasi pada desain logika sistem pengujian. Arsitektur sistem uji, rancangan alur kerja thread, dan skema sinkronisasi dengan mutex dan semaphore semua termasuk dalam desain. Selain itu, struktur data dan parameter pengujian dirancang untuk mengukur kinerja sistem tanpa mengacu pada software atau hardware tertentu.

Desain Fisik Sistem

Pada langkah ini, desain logis diterjemahkan ke dalam bentuk teknis yang siap digunakan. Peneliti memilih bahasa pemrograman yang digunakan (seperti Java atau Kotlin), alat pengujian (seperti Android Studio dan emulator Android), dan spesifikasi hardware dan software yang digunakan untuk menguji. Selain itu, rancangan uji dibuat dalam dua versi kode yang secara fungsional identik, tetapi masing-masing menggunakan mekanisme sinkronisasi yang berbeda: satu menggunakan mutex, dan yang lain menggunakan semaphore.

Implementasi

Hasil dari desain sebelumnya digunakan untuk mengembangkan sistem uji coba pada tahap implementasi. Kode program dibuat untuk menguji kinerja, kecepatan eksekusi, dan efisiensi penggunaan sumber daya dari mekanisme sinkronisasi masing-masing. Setelah sistem dikembangkan, pengujian berulang dilakukan untuk mendapatkan data kinerja yang konsisten. Selanjutnya, semua hasil pengujian dicatat dan dianalisis untuk menentukan kelebihan dan kekurangan setiap mekanisme.

Perawatan Sistem

Untuk memastikan bahwa sistem pengujian berfungsi dengan baik dan dapat digunakan kembali untuk penelitian serupa di masa mendatang, tahap akhir penelitian adalah pemeliharaan dan evaluasi sistem. Perbaikan kesalahan (bug) yang ditemukan selama pengujian serta penyesuaian terhadap versi Android terbaru yang dapat mempengaruhi hasil sinkronisasi antara mutex dan semaphore juga termasuk dalam pemeliharaan.

4. HASIL DAN PEMBAHASAN

Bab ini menyajikan hasil analisis dan pembahasan terkait mekanisme sinkronisasi *mutex* dan *semaphore* pada sistem operasi Android. Pembahasan difokuskan pada bagaimana kedua mekanisme tersebut bekerja dalam mengatur akses terhadap sumber daya bersama (*shared resources*), serta bagaimana efektivitasnya dalam mencegah resiko kondisi *race condition, starvation, inversion* dan *deadlock* pada proses system android Melalui pengujian dan kajian yang dilakukan, bagian ini bertujuan untuk memberikan pemahaman yang lebih mendalam mengenai perbedaan resiko keterbatasan masing-masing mekanisme sinkronisasi. Selain itu, hasil yang diperoleh juga dibandingkan secara teoritis dan praktis untuk mengetahui sejauh mana *mutex* dan *semaphore* berperan dalam menjaga kestabilan dan efisiensi kinerja sistem operasi Android. Diharapkan pembahasan dalam bab ini dapat memberikan gambaran yang jelas mengenai penerapan sinkronisasi proses pada Android, serta menjadi dasar bagi pengembangan sistem yang lebih optimal dalam pengelolaan *thread* dan sumber daya pada lingkungan komputasi modern.

Tabel 1. Persentase Semaphore dan Mutex.

Aspek Resiko	Mutex	Persentase mutex	semaphore	Persentase semaphore
deadlock	Risiko tinggi jika tidak hati-hati dalam penguncian dan pembebasan.	100%	Risiko tinggi, terutama dengan counting semaphore yang kompleks.	100%

Inversion	Dapat terjadi jika thread prioritas rendah memegang mutex yang dibutuhkan thread prioritas tinggi.	66%	Juga dapat terjadi, terutama dalam sistem real-time.	66%
Starvation	Bisa terjadi jika mutex tidak dikelola dengan adil.	66%	Lebih rentan terhadap starvation karena penjadwalan bisa tidak adil.	100%
Race Condition	Bisadicegahdengan mutex, tapi tetap terdapat kesalahan logika	33%	Sama seperti mutex; tergantung cara implementasi.	33%

Berdasarkan tabel 1 sinkronisasi antar - utas sangat penting dalam sistem komputer paralel, terutama dalam program Android, untuk mencegah konflik saat menggunakan sumber daya bersama. Dua mekanisme sinkronisasi yang paling sering digunakan adalah semaphore dan mutex (mutual exclusion). Meskipun keduanya berfungsi untuk meningkatkan akses, keduanya berbeda secara signifikan dalam cara kerjanya , fleksibilitasnya, flekbilitasnya, dan tingkat kehati - hatiannya terhadap berbagai jenis risiko dan tantangan . Untuk lebih memahami perbedaan antara mutex dan semaphore , kami telah membuat tabel yang menunjukkan ambang batas risiko untuk berbagai potensi masalah dengan analisis kuantitatif berupa persentase berdasarkan skala risiko dari 0 % (tanpa risiko) hingga 100 % (risiko tinggi) .

Deadlock (100% pada Mutex dan Semaphore)

Deadlock adalah situasi di mana dua atau lebih thread saling menunggu satu sama lain untuk melepaskan resource yang sedang mereka kuasai, sehingga tidak ada yang bisa melanjutkan eksekusi.

a. **Mutex:** Risiko deadlock sangat tinggi, terutama jika ada nested locking (penguncian bertingkat) atau jika mutex tidak dilepas karena error di dalam thread.
b. **Semaphore:** Risiko juga tinggi, terutama jika digunakan dengan nilai counting yang tidak dijaga dengan baik. Semaphores lebih rawan jika ada multiple acquire tanpa release yang seimbang.

Inversion (66% pada keduanya)

Masalah ini terjadi saat thread prioritas rendah mengunci resource yang dibutuhkan oleh thread prioritas tinggi, dan thread prioritas tinggi harus menunggu.

a. **Mutex & Semaphore:** Keduanya rentan terhadap masalah ini, terutama pada sistem real-time. Penyelesaiannya biasanya melibatkan protokol khusus seperti *priority inheritance*.

Starvation (Mutex 66%, Semaphore 100%)

Starvation terjadi ketika satu atau lebih thread terus-menerus gagal memperoleh resource karena didahului oleh thread lain.

a. **Mutex:** Starvation bisa terjadi jika tidak ada fairness (misalnya mutex tidak antri secara FIFO).

b. **Semaphore:** Lebih rentan terhadap starvation karena implementasi counting dapat mengizinkan satu kelompok thread mendominasi akses.

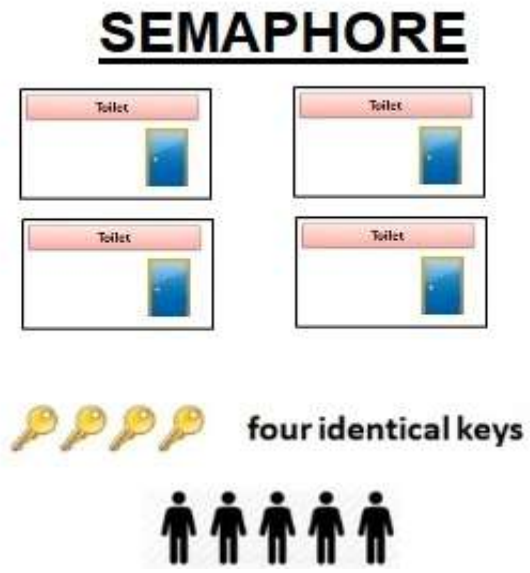
Race Condition (33% pada keduanya)

Race condition terjadi ketika dua thread atau lebih mengakses dan memodifikasi data secara bersamaan tanpa sinkronisasi yang tepat.

a. **Mutex & Semaphore:** Keduanya digunakan untuk mencegah kondisi ini, namun jika implementasinya salah, kondisi balapan tetap bisa terjadi. Risiko ada, tapi relatif rendah jika digunakan dengan benar.

Gambar 1. Perumpamaan Mutex.

Berdasarkan Gambar 1 Konsep "Mutex (Mutual Exclusion)" dalam pemrograman multithreading dan sistem operasi digambarkan dalam ilustrasi di atas. Dalam gambar tersebut, toilet menunjukkan sumber daya bersama, dan tiga orang yang ingin menggunakannya digambarkan sebagai "tiga proses atau thread" yang ingin mengakses sumber daya yang sama. Hanya satu dari ketiga orang tersebut yang memegang kunci, yang berfungsi sebagai pengunci. Artinya, hanya orang yang memegang kunci ini yang dapat masuk ke toilet dan menggunakan toilet, sementara orang lain harus menunggu sampai kunci dikembalikan. Setelah orang pertama menggunakan toilet, kunci akan dikembalikan untuk orang berikutnya. Perumpamaan ini menunjukkan bagaimana Mutex digunakan untuk mengontrol akses eksklusif terhadap sumber daya bersama, memastikan bahwa tidak ada proses yang digunakan secara bersamaan. Mekanisme ini membantu sistem menghindari masalah seperti konflik data, kegagalan proses, dan inkonsistensi hasil yang dapat terjadi jika dua thread mencoba mengakses sumber daya yang sama pada waktu yang sama. Dengan kata lain, Mutex memastikan bahwa hanya *satu proses* yang dapat melakukan operasi kritis pada satu waktu, dan proses lain harus menunggu sampai kunci dilepaskan. Konsep ini sangat penting untuk pemrograman paralel karena membantu menjaga proses agar stabil, akurat, dan sinkronis.



Gambar 2. Perumpamaan Semaphore.

Berdasarkan Gambar 2 Semaphore misalkan kita memiliki 4 toilet dan setiap toilet memiliki 4 kunci yang identik. Misalkan ada 5 orang yang ingin menggunakan toilet, orang pertama akan datang dan mengambil kunci, lalu orang kedua, ketiga, dan keempat. Ketika orang kelima datang, ia tidak akan mendapatkan kunci. Dalam hal ini, ia harus menunggu hingga toilet kosong. Semaphore adalah jenis mekanisme pensinyalan. Mutex adalah mekanisme penguncian. Semaphore adalah variabel integer, dan Mutex hanyalah sebuah objek.

Operasi tunggu dan sinyal dapat memodifikasi semaphore. Mutex hanya dimodifikasi oleh proses yang mungkin meminta atau melepaskan sumber daya. Dalam *semaphore*, jika tidak ada sumber daya yang bebas, maka proses membutuhkan sumber daya yang harus menjalankan operasi tunggu. Proses harus menunggu hingga jumlah semaphore lebih besar dari 0. Jika *Mutex* terkunci, proses harus menunggu. Proses harus disimpan dalam antrean. Antrean ini hanya perlu diakses ketika mutex dibuka. Kita dapat memiliki beberapa utas program dalam *Semaphore*. Kita dapat memiliki beberapa utas program dalam *mutex*, tetapi tidak secara bersamaan. Dalam Semaphore, nilai dapat diubah oleh proses apa pun yang melepaskan ataumendapatkan sumber daya. Dalam *Mutex*, kunci objek dilepaskan hanya oleh proses yang telah mendapatkan kunci tersebut. Jenis *semaphore* adalah semaphore pencacah dan *semaphore biner. Mutex* tidak memiliki subtipe. Nilai semaphore diubah menggunakan operasi *wait* dan *signal*. Objek *Mutex* terkunci atau tidak terkunci.

5. KESIMPULAN DAN SARAN

Android mendukung multitasking melalui penggunaan thread, banyak pengembang yang masih belum sepenuhnya memahami perbedaan mendasar antara mutex (mutual exclusion) dan semaphore sebagai mekanisme sinkronisasi. Ketidaktahuan ini dapat menyebabkan penerapan kontrol sinkronisasi yang tidak sesuai dengan kebutuhan konkuren aplikasi. Secara teknis, mutex digunakan untuk mengatur akses eksklusif ke sumber daya tunggal, sementara semaphore lebih fleksibel karena dapat mengatur akses ke sejumlah sumber daya terbatas. Ketika pengembang menggunakan mutex atau semaphore tanpa pemahaman yang benar, konsekuensi yang mungkin terjadi meliputi:

a. Deadlock – ketika dua atau lebih thread saling menunggu sumber daya yang dikunci oleh thread lainnya, sehingga tidak ada yang bisa melanjutkan proses.
b. Race Condition – ketika dua atau lebih thread mengakses dan memodifikasi data secara bersamaan tanpa pengaturan yang benar, menghasilkan hasil yang tidak konsisten atau salah.
c. Inversion – ketika thread dengan prioritas rendah memegang kunci (lock) yang dibutuhkan oleh thread dengan prioritas tinggi, sehingga thread prioritas tinggi menjadi tertunda.
d. Starvation – ketika satu atau lebih thread terus-menerus gagal mendapatkan akses ke sumber daya karena selalu dikalahkan oleh thread lain.

Masalah-masalah ini tidak hanya menyebabkan aplikasi menjadi tidak stabil atau tidak responsif, tetapi juga sulit dideteksi dan diatasi karena sering kali hanya muncul dalam kondisi

tertentu atau secara acak. Oleh karena itu, penting bagi para pengembang Android untuk memahami secara konseptual dan praktis perbedaan serta cara kerja mutex dan semaphore, agar dapat memilih mekanisme sinkronisasi yang paling tepat berdasarkan kebutuhan spesifik aplikasi. Dengan pemahaman yang benar, pengembang dapat meminimalkan risiko terjadinya kesalahan dalam pengelolaan thread, serta memastikan.

Berdasarkan tabel 1 sinkronisasi antar - utas sangat penting dalam sistem komputer paralel, terutama dalam program Android, untuk mencegah konflik saat menggunakan sumber daya bersama. Dua mekanisme sinkronisasi yang paling sering digunakan adalah semaphore dan mutex (mutual exclusion). Meskipun keduanya berfungsi untuk meningkatkan akses, keduanya berbeda secara signifikan dalam cara kerjanya, fleksibilitasnya, flekbilitasnya, dan tingkat kehati - hatiannya terhadap berbagai jenis risiko dan tantangan. Untuk lebih memahami perbedaan antara mutex dan semaphore, kami telah membuat tabel yang menunjukkan ambang batas risiko untuk berbagai potensi masalah dengan analisis kuantitatif berupa persentase berdasarkan skala risiko dari 0 % (tanpa risiko) hingga 100 % (risiko tinggi).

Deadlock (100% pada Mutex dan Semaphore)

Deadlock adalah situasi di mana dua atau lebih thread saling menunggu satu sama lain untuk melepaskan resource yang sedang mereka kuasai, sehingga tidak ada yang bisa melanjutkan eksekusi.

a. **Mutex:** Risiko deadlock sangat tinggi, terutama jika ada nested locking (penguncian bertingkat) atau jika mutex tidak dilepas karena error di dalam thread.
b. **Semaphore:** Risiko juga tinggi, terutama jika digunakan dengan nilai counting yang tidak dijaga dengan baik. Semaphores lebih rawan jika ada multiple acquire tanpa release yang seimbang.

Inversion (66% pada keduanya)

Masalah ini terjadi saat thread prioritas rendah mengunci resource yang dibutuhkan oleh thread prioritas tinggi, dan thread prioritas tinggi harus menunggu.

a. **Mutex & Semaphore:** Keduanya rentan terhadap masalah ini, terutama pada sistem real-time. Penyelesaiannya biasanya melibatkan protokol khusus seperti *priority inheritance*.

Starvation (Mutex 66%, Semaphore 100%)

Starvation terjadi ketika satu atau lebih thread terus-menerus gagal memperoleh resource karena didahului oleh thread lain.

a. **Mutex:** Starvation bisa terjadi jika tidak ada fairness (misalnya mutex tidak antri secara FIFO).

b. **Semaphore:** Lebih rentan terhadap starvation karena implementasi counting dapat mengizinkan satu kelompok thread mendominasi akses.

Race Condition (33% pada keduanya)

Race condition terjadi ketika dua thread atau lebih mengakses dan memodifikasi data secara bersamaan tanpa sinkronisasi yang tepat.

a. **Mutex & Semaphore:** Keduanya digunakan untuk mencegah kondisi ini, namun jika implementasinya salah, kondisi balapan tetap bisa terjadi. Risiko ada, tapi relatif rendah jika digunakan dengan benar.

DAFTAR REFERENSI

Aulia, W., Putri, H. S., & Emin, J. I. (2025). *Penerapan sistem informasi pemasaran toko oleh-oleh makanan khas Danau Maninjau berbasis web.*

Aziz, M., Juliansyah, J., & Munir, M. (2024). Pelatihan edukasi literasi keuangan untuk meningkatkan kesadaran menabung siswa SMA di wilayah rural berbasis modul digital interaktif. *Journal of Community Engagement in Economics, 2*(2), 1094. https://doi.org/10.35896/jcee.v2i2.1094

Fauzi, M. A., & Setiawan, I. (2021). Studi perbandingan mutex dan semaphore untuk sinkronisasi thread Android. *Jurnal Komputasi, 14*(2), 167–182. ISSN: 2088-976X.

Hidayat, R., & Nugroho, S. (2019). Kajian mekanisme sinkronisasi mutex dan semaphore di Android OS. *Jurnal Teknologi Elektro dan Informatika, 10*(3), 89–104. ISSN: 2089-8411.

Indrawan, B., & Suryadi, H. (2020). Perbandingan efektivitas mutex dan semaphore dalam pengelolaan sinkronisasi Android. *Jurnal Pendidikan Teknologi Informasi, 13*(3), 223–238. ISSN: 2089-9439.

Kurniasari, C., Pandowo, H., & Isnaningsih, T. (2023). Peningkatan literasi keuangan remaja melalui edukasi dan praktik pengelolaan keuangan. *Humanism: Journal of Community Empowerment, 6*(3), 1031. https://doi.org/10.32504/hjce.v6i3.1031

Kurniawan, T., & Sari, M. (2020). Perbandingan performa mutex dan semaphore dalam sinkronisasi pada OS Android. *Jurnal Rekayasa Sistem dan Teknologi Informasi, 8*(2), 134–149. ISSN: 2477-1099.

Lestari, N., & Gunawan, H. (2021). Perbandingan efisiensi mutex vs semaphore pada mekanisme sinkronisasi Android. *Jurnal Informatika, 12*(3), 201–215. ISSN: 1979-2328.

Pratama, A. J., Nurrahman, M. R., Arifin, I. A., Oktika Sari, D., Afifah, I., & Mirza, A. D. (2025). Edukasi literasi keuangan sejak dini: Strategi meningkatkan kesadaran menabung pada siswa Sekolah Dasar Negeri. *AKSIME: Jurnal Pengabdian Masyarakat Bidang Akuntansi, Manajemen & Ekonomi, 2*(1), 45–52. https://doi.org/10.32503/aksime.v2i1.6866

Putra, R. A., & Santoso, B. (2019). Analisis perbandingan mutex dan semaphore dalam sinkronisasi thread pada Android OS. *Jurnal Sistem Informasi (JSI), 11*(1), 78–92. ISSN: 2088-2955.

Rahman, F., & Susanto, E. (2018). Studi komparatif sinkronisasi mutex dan semaphore di platform Android. *Jurnal Teknologi dan Sistem Komputer, 6*(2), 112–127. ISSN: 2338-0403.

Sari, D. P., & Wijaya, A. (2020). Studi perbandingan mekanisme sinkronisasi mutex dan semaphore pada sistem operasi Android. *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK), 7*(2), 145–158. ISSN: 2338-9726.

Sudarti, B., Sulaeha, S., & Mabe Parenreng, S. (2025). Pengenalan literasi keuangan bagi siswa sekolah dasar dalam merencanakan masa depan di Desa Tellu Limpoe. *Jurnal Pengabdian Masyarakat Indonesia, 2*(4). https://doi.org/10.62017/jpmi.v2i4.4208

Wibowo, A., & Pramono, D. (2022). Evaluasi mekanisme sinkronisasi mutex dan semaphore pada aplikasi Android multithread. *Jurnal Ilmu Komputer dan Informasi, 15*(1), 45–60. ISSN: 2089-9033.

Yusuf, D., & Pratama, A. (2022). Analisis komparatif mekanisme sinkronisasi mutex dan semaphore pada sistem Android. *Jurnal Sains dan Teknologi Informasi, 9*(1), 56–71. ISSN: 2460-0763.