LAPORAN SISTEM OPERASI
ANDROID 7

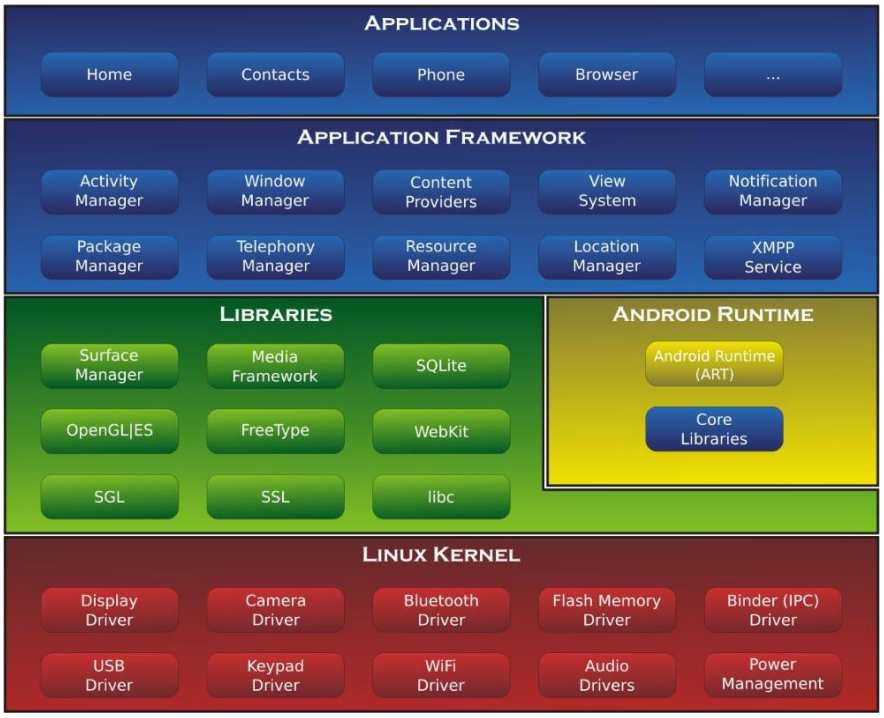
Untuk Memenuhi Tugas Sistem Operasi I
Dosen Pengampu : Drs., Purwadi Budi S, M.T

Disusun Oleh :

Adi Saepuloh	**2341001**
Andini Wijayanti	**2341004**
Dika Hidayat	**2341013**
Mutiara Botani	**2341027**
Winda Aryanti	**2341039**

TEKNIK INFORMATI S-1
SEKOLAH TINGGI TEKNOLOGI MANDALA
2024

A. Arsitektur Sistem Operasi Android



Pada sistem operasi Android, ada 4 lapisan utama yang bekerja sama agar semuanyaberjalan dengan baik:

1. Aplikasi

Di lapisan terluar, kita punya aplikasi-aplikasi yang langsung berinteraksi dengan pengguna,seperti YouTube, WhatsApp, Instagram, dan lainnya. Ini yang kita gunakan setiap hari.

2. Framework Aplikasi

Nah, di balik layar, ada framework aplikasi yang ngurusin banyak hal biar aplikasi bisa berjalandengan lancar. Beberapa bagian pentingnya adalah:

- Activity Manager: Ngatur siklus hidup aplikasi, biar tahu kapan aplikasi dijalankanatau dihentikan.
- Content Providers: Memungkinkan aplikasi berbagi data satu sama lain.
- Resource Manager: Ngatur sumber daya eksternal kayak gambar, layout XML, danteks.
- Location Manager: Ngasih layanan berbasis lokasi, kayak GPS.
- Notification Manager: Ngatur notifikasi yang muncul dari aplikasi.
- View System: Ngasih elemen UI kayak tombol, daftar, dan formulir yang kita lihat diaplikasi.

  Jadi, pengembang tidak perlu nulis kode dari awal buat ngakses fitur perangkat. Ini membuat proses aplikasi jadi lebih cepat.

3. **C++ Library**

Di lapisan lebih dalam, ada yang namanya library C++. Di sini, C++ dipakai buat aplikasi yangbutuh performa tinggi atau akses langsung ke hardware.

- Game: Banyak game di Android yang pakai C++ buat performa lebih baik, biasanya lewat library OpenGL atau Vulkan buat grafis, dan Box2D buat fisika.
- Pemrosesan Multimedia: Aplikasi yang ngolah video dan audio secara real-time juga sering pakai C++ biar lebih efisien, contohnya aplikasi streaming atau editing.
- Porting Aplikasi Lama: Kalau ada kode lama yang ditulis dalam C atau C++ buat platform lain, kita bisa pakai Android NDK buat nge-port kode tersebut ke Android tanpa harus nulis ulang dalam Java. Ini penting buat aplikasi yang butuh keamanan tinggi atau yang ngatur data sensitif.
  Contoh library C++ di Android yang sering dipakai:
  1) OpenGL ES: Buat rendering 2D dan 3D.
  2) FFmpeg: Buat pemrosesan video dan audio.

  Jadi, pakai library C++ bisa bikin performa aplikasi lebih baik, terutama buat aplikasi yangbutuh banyak sumber daya.

4. **Android Runtime (ART)**

ART adalah mesin virtual yang Android pakai buat menjalankan aplikasi. Mulai dari Android7 (Nougat), ART menggabungkan metode AOT (Ahead-of-Time) dan JIT (Just-in-Time) biar aplikasi lebih cepat diluncurkan, performanya makin bagus, dan memori jadi lebih efisien. Intinya, pengalaman pakai aplikasi jadi lebih mulus.

5. **Linux Kernel**

Linux kernel itu semacam "otak" dari sistem operasi Android. Semua perangkat Android sebenarnya berjalan di atas kernel Linux yang sudah dimodifikasi. Kernel ini yang menghubungkan perangkat keras (seperti kamera, prosesor, layar sentuh) dengan sistem dan aplikasi yang kita pakai di Android.

B. Proses Berjalannya Aplikasi pada Sistem Operasi Android

Proses berjalannya aplikasi pada sistem operasi Android 7 (Nougat) melibatkan beberapa tahapan penting yang dikelola oleh komponen inti dari Android. Berikut adalah langkah-langkah umum yang terjadi ketika sebuah aplikasi dijalankan:

1. **Instalasi Aplikasi**
   - Sebelum aplikasi dapat dijalankan, aplikasi harus diinstal terlebih dahulu. Aplikasi Android dikemas dalam format **APK (Android Package)**.
   - Ketika aplikasi diinstal, Android melakukan validasi, mengekstrak konten APK, dan menyimpannya di direktori khusus. Setiap aplikasi juga diberikan **UID (User ID)** yang unik untuk menjaga isolasi dan keamanan antar aplikasi.

2. **Pembuatan Proses**
   - Saat aplikasi dijalankan, Android memulai proses baru dengan menggunakan **Dalvik Virtual Machine (DVM)** atau **Android Runtime (ART)**. Pada Android 7, ART adalah runtime default.
   - Setiap aplikasi berjalan dalam proses terpisah untuk menjaga isolasi dan keamanan. Proses ini dikelola oleh **Zygote**, yang merupakan proses inti yang memuat pustaka umum dan mempersingkat waktu inisialisasi aplikasi baru.

3. **Activity Manager**
   - **Activity Manager** adalah bagian dari **Android Framework** yang mengelola lifecycle dari komponen aplikasi seperti **Activity**, **Service**, dan **BroadcastReceiver**.
   - Saat pengguna memulai aplikasi, Activity Manager akan membuat instance dari **Activity** utama dan memulai **lifecycle** yang sesuai (misalnya, **onCreate()**, **onStart()**, **onResume()**).

4. **Rendering UI**
   - Setelah Activity dibuat, Android menggunakan **View System** untuk merender UI (User Interface). Proses ini melibatkan rendering layout XML dan berinteraksi dengan elemen UI seperti tombol, teks, dan gambar.
   - **SurfaceFlinger** bertanggung jawab untuk komposisi dan rendering grafis aplikasi ke layar.

5. **Interaksi Pengguna dan Input Events**
   - Saat pengguna berinteraksi dengan aplikasi, Android mengirimkan **input events** (seperti klik, sentuhan) ke aplikasi yang sedang berjalan.
   - **Activity** merespons event tersebut melalui callback seperti **onTouchEvent()**, **onClick()**, atau metode lain yang terkait.

6. **Threading dan Background Tasks**
   - Android menggunakan **Main Thread** (juga disebut **UI Thread**) untuk mengelola operasi UI. Untuk tugas berat seperti akses jaringan atau operasi database, Android menganjurkan penggunaan **AsyncTask**, **Threads**, atau **Services** agar aplikasi tetap responsif.
   - **IntentService** atau **JobScheduler** pada Android 7 digunakan untuk menangani operasi background yang lebih panjang atau terjadwal.

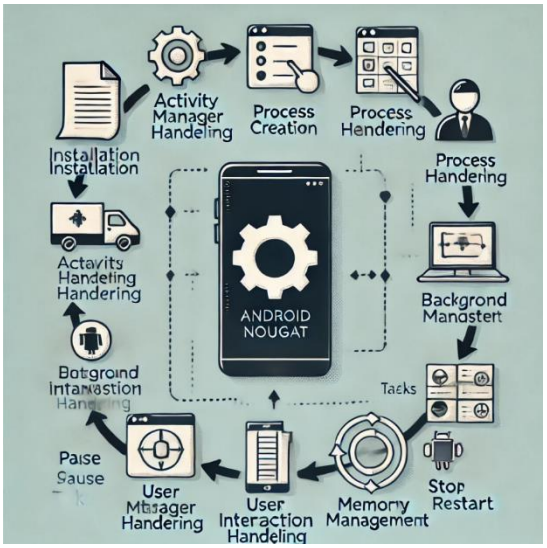
7. **Pengelolaan Memori dan Garbage Collection**
   - Android memiliki mekanisme **garbage collection** untuk mengelola memori secara otomatis. **ART** pada Android 7 meningkatkan performa garbage collection dengan teknik seperti **Concurrent Garbage Collection** untuk mengurangi gangguan pada aplikasi.
   - Jika sebuah aplikasi menggunakan terlalu banyak memori, sistem dapat meminta aplikasi untuk melepaskan memori yang tidak perlu atau bahkan menghentikan aplikasi yang berjalan di latar belakang (background) untuk menghindari kekurangan memori.

8. **Pause dan Stop Aplikasi**
   - Ketika pengguna beralih ke aplikasi lain atau menekan tombol home, aplikasi tidak langsung dihentikan. Sebaliknya, **Activity** masuk ke dalam status **onPause()** atau **onStop()**. Jika dibutuhkan lagi, aplikasi dapat kembali ke status **onResume()** tanpa harus memulai ulang dari awal.

9. **Aplikasi di Latar Belakang (Background)**
   - Pada Android 7, aplikasi yang berjalan di latar belakang dibatasi dengan lebih ketat untuk menghemat baterai dan sumber daya. **Doze Mode** dan **App Standby** diperkenalkan untuk menunda tugas latar belakang ketika perangkat tidak digunakan aktif.

10. **Penghentian dan Pembersihan Proses**
    - Jika aplikasi tidak digunakan dalam waktu lama atau sistem membutuhkan memori tambahan, Android akan menghentikan proses aplikasi sepenuhnya dan membersihkan sumber daya yang digunakan aplikasi tersebut.
    - Aplikasi yang dihentikan dapat dimulai ulang dari awal jika diperlukan kembali.

    Proses ini berulang setiap kali aplikasi diluncurkan dan dikelola oleh sistem operasi Android untuk memastikan performa yang baik, keamanan, dan efisiensi sumber daya.



C. Proses Kerja Multitasking Aplikasi Android

Multitasking pada Android 7 (Nougat) mencakup berbagai proses yang memungkinkan aplikasi berjalan secara efisien di latar depan dan latar belakang, termasuk pengelolaan memori, penjadwalan tugas, dan fitur split-screen yang baru. Berikut adalah penjelasan tentang bagaimana multitasking bekerja di Android 7:

1. Split-Screen Multitasking

Fitur utama yang diperkenalkan di Android 7 adalah split-screen multitasking, yang memungkinkan pengguna menjalankan dua aplikasi secara bersamaan dalam mode layar terpisah. Prosesnya adalah sebagai berikut:

- Pembagian layar: Layar dibagi menjadi dua bagian, di mana setiap aplikasi mendapatkan setengah dari layar atau sesuai dengan pengaturan yang dipilih pengguna.
- Manajemen input: Android menangani input untuk dua aplikasi secara bersamaan. Jika pengguna berinteraksi dengan aplikasi di bagian atas layar, sistem menangguhkan input untuk aplikasi di bagian bawah, dan sebaliknya.
- Pengelolaan memori: Aplikasi yang sedang dibagi di layar diberi prioritas lebih tinggi, dan Android akan mengurangi aktivitas aplikasi latar belakang yang tidak terlihat untuk menghemat memori.

2. Activity Lifecycle dalam Multitasking

Setiap aplikasi di Android dikelola melalui Activity Lifecycle, yaitu serangkaian status yang mengatur kapan aplikasi harus aktif, dihentikan sementara, atau dihentikan sepenuhnya. Ini penting dalam multitasking karena memungkinkan Android untuk memutuskan kapan aplikasi harus terus berjalan atau di-pause:

- Aplikasi aktif di latar depan: Ketika sebuah aplikasi ada di layar, aplikasi ini berada dalam status onResume(), yang berarti aplikasi siap untuk berinteraksi dengan pengguna.
- Aplikasi di latar belakang: Ketika aplikasi dipindahkan ke latar belakang (misalnya, pengguna membuka aplikasi lain), aplikasi akan menerima event onPause() atau onStop(). Dalam status ini, aplikasi tetap ada dalam memori tetapi tidak menerima input pengguna, sehingga tidak menggunakan sumber daya berlebihan.

Pada split-screen, kedua aplikasi berada dalam keadaan resumed, sehingga keduanya aktif tetapi berbagi sumber daya seperti CPU dan memori.

3. Threading dan Asynchronous Tasks

Aplikasi Android menggunakan thread untuk menangani multitasking. Main thread digunakan untuk menjalankan antarmuka pengguna (UI), sementara tugas-tugas berat seperti akses jaringan atau pemrosesan data dilakukan pada background thread menggunakan:

- AsyncTask: Tugas-tugas ringan yang dapat berjalan di latar belakang tanpa membebani UI.
- Services: Proses untuk menangani tugas berat di latar belakang, seperti pemutaran musik atau sinkronisasi data.
- HandlerThread atau ThreadPoolExecutor: Untuk tugas yang lebih kompleks, yang membutuhkan lebih banyak pengaturan dan pengelolaan, thread-thread tambahan dikelola untuk menangani pekerjaan intensif.

4. **Manajemen Memori dan Penghentian Aplikasi**

Android 7 menggunakan manajemen memori adaptif, yang memprioritaskan aplikasi yang sedang digunakan dan secara otomatis menutup aplikasi yang tidak aktif atau jarang digunakan:

- Least Recently Used (LRU) Cache: Sistem ini digunakan untuk melacak aplikasi mana yang terakhir kali digunakan. Aplikasi yang paling lama tidak digunakan akan ditutup lebih dulu jika memori hampir penuh.
- Penghematan memori: Aplikasi di latar belakang diberi prioritas lebih rendah dalam hal penggunaan memori, dan jika memori perangkat terbatas, aplikasi latar belakang ini akan dihentikan oleh sistem secara otomatis.

5. **Doze Mode dan App Standby**

Android 7 meningkatkan fitur Doze Mode, yang pertama kali diperkenalkan di Android 6. Fitur ini menghemat daya dengan menempatkan perangkat ke dalam mode hemat energi saat tidak digunakan dalam waktu lama. Pada Android 7, Doze Mode diaktifkan lebih agresif, bahkan ketika perangkat dalam keadaan bergerak, tetapi layarnya dimatikan.

Selain Doze Mode, App Standby juga membatasi aplikasi yang jarang digunakan. Aplikasi yang berada dalam mode Standby tidak akan diperbolehkan mengakses jaringan atau sumber daya sistem lainnya sampai aplikasi itu dibuka kembali oleh pengguna.

6. **Window Management dan Virtual Display**

Android 7 menggunakan WindowManager dan sistem Virtual Display untuk menangani split-screen dan multitasking. WindowManager bertanggung jawab untuk membagi dan mengelola ukuran jendela aplikasi di layar, sementara Virtual Display memungkinkan sistem untuk menciptakan tampilan virtual sehingga setiap aplikasi bisa berfungsi seolah-olah mendapatkan layar penuh.

Dalam split-screen, setiap aplikasi masih berjalan seolah-olah di layar penuh tetapi memiliki pembagian layar yang terbatas. Hal ini memerlukan manajemen resource yang cermat agar performa tetap stabil.

7. **Penjadwalan Tugas (JobScheduler)**

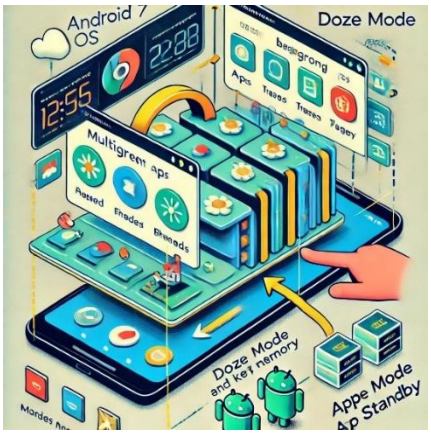
Android 7 juga mengandalkan JobScheduler untuk menangani tugas-tugas di latar belakang. Ini adalah API yang mengizinkan aplikasi untuk menjadwalkan pekerjaan yang dapat dijalankan di masa depan. Misalnya, jika aplikasi perlu memperbarui data dari server, JobScheduler dapat mengatur tugas ini untuk dilakukan ketika perangkat terhubung ke Wi-Fi atau sedang diisi daya, sehingga menghemat baterai dan data seluler.

8. **Recent Apps (Tombol Aplikasi Terbaru)**

Android 7 memperbarui tampilan Recent Apps, yang memungkinkan pengguna untuk beralih antar aplikasi dengan lebih cepat. Recent Apps menunjukkan daftar aplikasi yang baru dibuka, dan pengguna dapat beralih di antaranya dengan mudah, meningkatkan pengalaman multitasking.

Jika sebuah aplikasi ditutup atau dihapus dari daftar Recent Apps, itu menandakan kepada sistem untuk menghapus aplikasi dari memori jika tidak diperlukan lagi.

Berikut ini adalah gambar yang menunjukkan cara kerja multitasking di Android 7:



D. Status Pengerjaan Aplikasi

Status pada aplikasi Android 7 mengacu pada berbagai keadaan atau siklus hidup aplikasi yang menggambarkan bagaimana aplikasi berinteraksi dengan pengguna dan sistem operasi. Berikut adalah penjelasan tentang status atau siklus hidup aplikasi di Android:

1. Status Siklus Hidup Aplikasi

Siklus hidup aplikasi Android terdiri dari beberapa status yang berbeda, yang ditentukan oleh metode callback dalam aktivitas (Activity). Berikut adalah status utamanya:

- onCreate(): Dipanggil saat aktivitas dibuat. Ini adalah tempat menginisialisasi elemen UI dan melakukan setup awal, seperti mengatur layout dengan setContentView().
- onStart(): Dipanggil ketika aktivitas menjadi terlihat oleh pengguna. Pada tahap ini, aktivitas siap untuk berinteraksi dengan pengguna, tetapi mungkin belum aktif.
- onResume(): Dipanggil ketika aktivitas mulai aktif dan dapat berinteraksi dengan pengguna. Pada tahap ini, aktivitas berada di depan dan menjadi fokus utama.
- onPause(): Dipanggil ketika aktivitas tidak lagi berada di depan, tetapi belum sepenuhnya dihentikan. Ini bisa terjadi ketika pengguna beralih ke aktivitas lain atau menutup aplikasi. Di sini, harus menyimpan data penting atau menghentikan animasi yang sedang berlangsung.
- onStop(): Dipanggil ketika aktivitas tidak lagi terlihat oleh pengguna. Pada tahap ini, aktivitas dapat dihentikan dan mungkin akan dihapus dari memori. Ini adalah kesempatan untuk membebaskan sumber daya atau menghentikan proses yang tidak diperlukan.
- onRestart(): Dipanggil ketika aktivitas yang telah dihentikan (onStop) mulai aktif kembali. Ini menandakan bahwa aktivitas akan kembali ke status aktif.
- onDestroy(): Dipanggil sebelum aktivitas dihapus dari memori. Di sini, harus melakukan pembersihan, seperti membebaskan sumber daya yang digunakan.

2. Status berhasil dikelola

Jika status dalam siklus hidup aplikasi Android berhasil dikelola, maka aplikasi

tersebut akan berfungsi secara optimal dan memberikan pengalaman pengguna yang baik. Berikut adalah beberapa aspek yang menunjukkan keberhasilan pengelolaan status tersebut:

- Responsif dan Stabil
  - Interaksi Pengguna yang Lancar: Pengguna dapat berinteraksi dengan aplikasi tanpa adanya lag atau pembekuan. Hal ini berkat pengelolaan thread yang baik, di mana operasi yang memakan waktu tidak dijalankan di thread utama.
  - Stabilitas Aplikasi: Aplikasi tidak mengalami crash atau error saat berpindah antar status. Ini menunjukkan bahwa semua siklus hidup dikelola dengan benar.

- Pengelolaan Sumber Daya yang Efisien
  - Penggunaan Memori yang Optimal: Dengan memanfaatkan metode onPause() dan onStop() untuk membebaskan sumber daya yang tidak lagi digunakan, aplikasi dapat mengurangi risiko kebocoran memori.
  - Kecepatan Respons: Dengan menyimpan dan memulihkan status aplikasi dengan baik, pengguna tidak perlu menunggu lama saat beralih kembali ke aplikasi.

- Data yang Terjaga
  - Penyimpanan Status yang Efektif: Ketika pengguna keluar dari aplikasi, jika data penting disimpan dengan benar di onPause() atau onStop(), pengguna dapat melanjutkan dari titik terakhir tanpa kehilangan progres.
  - Konsistensi Data: Data yang ditampilkan dalam aplikasi selalu konsisten dan tidak ada kehilangan informasi saat aplikasi berpindah status.

- Pengalaman Pengguna yang Positif
  - Antarmuka Pengguna yang Intuitif: Saat pengguna kembali ke aplikasi, tampilan dan interaksi tidak terganggu. Ini meningkatkan kepuasan pengguna.
  - Responsif terhadap Aksi Pengguna: Pengguna merasa bahwa aplikasi responsif terhadap perintah mereka, misalnya, tombol yang ditekan segera memberi respons visual.

- Contoh Situasi Keberhasilan
  - Skenario Penggunaan: Seorang pengguna membuka aplikasi, melakukan beberapa pencarian, dan kemudian berpindah ke aplikasi lain. Ketika ia kembali, aplikasi tetap dalam kondisi siap, dengan semua data dan status sebelumnya dipertahankan.
  - Penanganan Situasi Tak Terduga: Jika pengguna menerima panggilan telepon saat menggunakan aplikasi, aplikasi dapat masuk ke status onPause(), dan saat panggilan selesai, pengguna dapat kembali dengan mulus.

3. Status tidak berhasil dikelola

Jika status dalam siklus hidup aplikasi Android tidak berhasil dikelola dengan baik, berbagai masalah dapat muncul yang berdampak negatif pada pengalaman pengguna dan kinerja aplikasi. Berikut adalah beberapa dampak negatif yang mungkin terjadi:

- Keterlambatan dan Pembekuan
  - UI Tidak Responsif: Jika operasi yang berat dijalankan di thread utama (UI

thread), aplikasi dapat menjadi tidak responsif, menyebabkan lag atau bahkan pembekuan yang mengganggu interaksi pengguna.
- Crash Aplikasi: Kegagalan dalam mengelola transisi antar status dapat menyebabkan aplikasi crash, misalnya, jika data yang diperlukan tidak disimpan atau diinisialisasi dengan benar.

➢ Kebocoran Memori
- Penggunaan Memori Berlebihan: Jika onStop() dan onDestroy() tidak diimplementasikan dengan baik, aplikasi dapat mengakibatkan kebocoran memori, di mana memori tidak dibebaskan setelah aktivitas tidak lagi diperlukan. Hal ini dapat menyebabkan perangkat menjadi lambat seiring waktu.
- Penggunaan Sumber Daya yang Tidak Efisien: Aplikasi yang tidak menghentikan thread atau proses latar belakang yang tidak perlu dapat terus mengkonsumsi sumber daya, berpotensi mempengaruhi aplikasi lain yang berjalan di perangkat.

➢ Kehilangan Data
- Data yang Hilang: Jika data penting tidak disimpan dalam metode onPause() atau onStop(), pengguna mungkin kehilangan progres atau informasi saat berpindah keluar dari aplikasi.
- Konsistensi Data yang Buruk: Aplikasi mungkin menampilkan data yang tidak akurat jika status tidak dikelola dengan baik, seperti hasil pencarian yang hilang atau informasi pengguna yang tidak tersimpan.

➢ Pengalaman Pengguna yang Buruk
- Antarmuka Pengguna yang Membingungkan: Ketika pengguna kembali ke aplikasi dan melihat bahwa statusnya tidak sama dengan saat mereka meninggalkan, ini dapat menyebabkan kebingungan dan frustrasi.
- Reputasi Buruk: Aplikasi yang sering crash atau tidak responsif dapat merusak reputasi pengembang dan mengurangi kepercayaan pengguna terhadap aplikasi tersebut.

➢ Contoh Situasi Ketidakberhasilan
- Skenario Pembekuan Aplikasi: Seorang pengguna membuka aplikasi, melakukan beberapa pencarian, dan aplikasi mulai membeku saat pengguna mencoba beralih ke menu lain karena proses berat dijalankan di thread utama.
- Kehilangan Informasi: Pengguna mengisi formulir dan beralih ke aplikasi lain untuk referensi, tetapi saat kembali, semua data yang diisi hilang karena tidak disimpan saat aplikasi memasuki onPause().

➢ Dampak Jangka Panjang
- Pengurangan Pengguna: Jika masalah ini terjadi secara konsisten, pengguna mungkin akan berhenti menggunakan aplikasi tersebut dan beralih ke aplikasi pesaing yang lebih stabil dan responsif.
- Kritik dan Ulasan Negatif: Aplikasi dengan pengalaman buruk dapat mendapatkan ulasan negatif di toko aplikasi, yang dapat memengaruhi pemasaran dan unduhan di masa depan.

E. Register yang digunakan android 7 (NOUGAT) dengan processor ARM 32(ARMV7-A)

1. Jenis register dan fungsinya

- Register CPU (R0 - R15):

  R0-R7: Semua mode berbagi register ini, jadi tidak peduli di mode apa, yang dipakai tetapregister yang sama.

  R8-R12: Di mode FIQ, register ini punya versi khusus (banked), jadi beda dengan register R8-R12 di mode lain. Tujuannya biar interrupt yang cepat bisa ditangani lebih efisien.

  R13 (SP - Stack Pointer): Setiap mode punya register R13 sendiriyang di-banked buat menangani tumpukan (stack).

  R14 (LR - Link Register): Mirip dengan R13, tiap mode punya R14 sendiri buat menyimpan alamat pengembalian kalau ada pemanggilanfungsi atau interrupt.

  R15 (PC - Program Counter): Ini register yang selalu dipakai di semua mode buat nyimpen instruksi yang bakal dieksekusi selanjutnya.

Registers across CPU modes

usr	sys	svc	abt	und	irq	fiq
R0						
R1						
R2						
R3						
R4						
R5						
R6						
R7						
R8						R8_fiq
R9						R9_fiq
R10						R10_fiq
R11						R11_fiq
R12						R12_fiq
R13		R13_svc	R13_abt	R13_und	R13_irq	R13_fiq
R14		R14_svc	R14_abt	R14_und	R14_irq	R14_fiq
R15						
CPSR						
		SPSR_svc	SPSR_abt	SPSR_und	SPSR_irq	SPSR_fiq

- CPSR (Current Program Status Register):
  Register ini nyimpen informasi tentang status prosesor saat ini,seperti:
  - Flags: Zero, Carry, Negative, Overflow.
  - Mode: Menunjukkan mode CPU yang lagi aktif (usr, sys, svc, abt, und, irq, fiq).
  - Interrupt Masks: Menandakan apakah interrupt lagi diizinkan atau tidak.

- SPSR (Saved Program Status Register):
  Setiap mode dengan hak istimewa (seperti svc, abt, und, irq, fiq) punya SPSR-nya sendiri buat nyimpen status lama dari CPSR sebelum pindah mode, jadi setelah penanganan selesai, CPU bisabalik lagi ke status sebelumnya.
  - SPSR_svc: untuk mode supervisor.
  - SPSR_abt: untuk mode abort.
  - SPSR_und: untuk mode undefined.
  - SPSR_irq: untuk mode interrupt request.

- SPSR_fiq: untuk mode fast interrupt request.

2. **Contoh proses register pada sistem operasi android 7 pada arsitektur ARM 32-bit (ARMv7-A)**

```cpp
#include <iostream>
using namespace std;

int main() {
    int a, b, hasil;

    cout << "Masukkan angka pertama: ";
    cin >> a;

    cout << "Masukkan angka kedua: ";
    cin >> b;

    hasil = a + b;

    cout << "Hasil penjumlahan: " << hasil << endl;

    return 0;
}
```

Berikut adalah contoh debugging register untuk program sederhana penjumlahan denganmenggunakan c++ :

Output yang dihasilkan menggunakan gdb:

Sebelum eksekusi penjumlahan akan menghsilkan ouput :

r0	0x5	# Nilai register R0, yang berisi angka pertama
r1	0x30	# Nilai register R1, yang berisi angka kedua
r2	0x0	# Register untuk hasil penjumlahan nanti
r3	0x0	
r4	0x0	
...		
cpsr	0x60000010	# Status CPU

Setelah eksekusi penjumlahan :

r0	0x5	# Angka pertama
r1	0x3	# Angka kedua
r2	0x8	# Hasil penjumlahan (5 + 3 = 8)