Sistemi operativi

sav

2025

Contents

Chapter 1

ANDROID

1.1 Cos'è android

Android è un sistema operativo propietario di Google Inc. con licenza open source BSD. è nato con lo scopo di essere installato su dispositivi mobile, perciò con memoria e potenza limitate, con l'obiettivo di essere utilizzabile su tante tipologie di hardware differenti, quindi favorendo la portabilità. Inoltre è fondamentale che le funzioni di base del dispositivo, un telefono ad esempio, funzionino e che siano affidabili.

1.1.1 Linguaggi di sviluppo

Le applicazioni android sono sviluppate principalmente in Kotlin e Java.

1.1.2 Architettura a 5 livelli: Apps level, Framework level, Librerie native, HAL

L'architettura di android è a 5 livelli: dal livello più alto accesibile dagli utenti comuni(app level) fino al più basso, il Kernel.

1.2 Kernel

SO android si basa su kernel linux, a cui sono state apportate modifiche affichè risponda alle esigienze dei dispositivi mobile, quindi per risorse di calcolo limitate, energia limitata(alimentazione tramite batteria), portabilità(deve funzionare su telefoni con hardware differenti), affidabile nelle operazioni di base. android è open source ed è proprietà di google. l'architettura android è su 5 livelli: kernel, HAL, libreriie native, framework, app. elementi che si distanziano dal classico kernel linux sono il binder e il low memory killer. il binder è ciò che lega tutti i componenti di android: il binder in Android è il meccanismo centrale di comunicazione interprocesso (IPC) che consente ai vari processi e servizi di scambiarsi dati e comandi in modo efficiente e sicuro. il low memory killer è svolge il seguente ruolo: se un processo in background svolge un operazione a bassa priorità allora il demone lmkd(low memory killer demon) attua per terminare(uccidere) tale processo a bassa priorità.

1.2.1 Risparmio energetico - wakelock

Per il risparmio energetico vengono utilizzati i wakelock. Quando un app viene eseguita, che sia in background o no, richiede dei wakelock, ossia delle richieste di utilizzo di risorse(un wavelock è una richiesta di attivazione/disattivazione risorse). Esempio wakelock cpu,display,tastiera: 3 wakelock, tutti

e tre permettono l'uso della cpu, il primo lascia utilizzare solo la cpu, il secondo aggiunge l'uso del display a pieno regime, il terzo display e tastiera.

1.2.2 Risparmio RAM - lmk

Per un utilizzo più efficiente della memoria ram viene utilizzato il Low Memory Killer. Questo "demone" fa si che quando un processo in back ground non è più necessario allora viene terminato, andando a svincolare la memoria utilizzata dal processo in questione, esplicitando che tale memoria possa essere utilizzata per altre operazioni.

1.3 Binder IPC driver

IPC vuol dire "Inter Process Comunication", si tratta di un insieme di tecnologie software che consentono a diversi processi di scambiarsi informazioni e dati. Nel caso di android un ruolo fondamentale lo svolge il Binder, che permette la comunicazione tra due processi nel modo più sicuro possibile, evitando la comunicazione diretta tra processi, ma bensì tramite il binder. In ambito di sicurezza un ruolo viene svolto dal Binder IPC driver, permettendo ai processi isolati all'interno della propria sandbox di comunicare in maniera sincrona. [Il meccanismo Binder in Android è il sistema principale di comunicazione tra processi (IPC). Funziona come un ponte che permette a un processo (client) di invocare metodi e richiedere servizi da un altro processo (server) come se fossero chiamate locali, utilizzando un modello di Reference Counting(RC). A ciascun processo è associato un pool di threads per la gestione della comunicazione.]

1.3.1 Ruolo del IPC nella sicurezza

permette la comunicazione tra due processi nel modo più sicuro possibile, evitando la comunicazione diretta tra processi, ma bensì tramite il binder.

1.4 Dispositivi di storage

1.4.1 Memorie flash

Vengono usate le memorie flash, per via di alcuni vantaggi come quelli qui citati:

No seek time

No seek time, le memorie flash non hanno latenze causate da operazioni di ricerca.

Block erasing

Viene pulita la cella di memoria prima che vi sia consentita la scrittura, questo richiede molto tempo quindi viene fatto in tempi di idle.

- Il tempo di inattività (o idle time in inglese) si riferisce all'intervallo durante il quale un dispositivo, un componente è operativo ma non viene utilizzato attivamente. In altre parole, è il periodo in cui una risorsa è disponibile e funzionante, ma rimane inattiva perché non impegnata in alcuna attività produttiva.

Wear leveling

consiste nel distribuire su più celle di memorie le operazioni di lettura e scrittura in quanto se fatte sempre sulle stesse celle di memoria ne si provoca il deterioramento. Ogni cella di memoria ha un numero limitato di scritture che si possono fare.

1.5 Hal e librerie native *

1.6 Struttura pacchetto android

1.6.1 apk, file formato .dex e .oat

Un'app android viene distribuita tramite pacchetti apk, che conterranno i codici utili a far funzionare l'applicazione. Distinguiamo due tipi fondamentali di file/codici:

- .dex : è un file tipicamente scritto in linguaggio kotlin/java ed è un codice che viene runnato dalla DalvikVM; è un codice che può essere eseguito da più dispositivi android possibile, non è quindi ottimizzato per singoli device.
- .oat : è un codice binario che viene compilato AOT(Ahead of time) ed è specifico per il device su cui viene compilato.

Un'applicazione android è un pacchetto .apk che contiene il codice .dex. Il codice .dex(Dalvik EXecutable) contiene in bytecode compilato nei linguaggi più comuni kotlin/java, non è direttamente legato alla struttura hardware su cui viene fatta runnare l'applicazione, perciò è un codice che funziona su più dispositivi. Il codice oat invece viene prodotto tramite il processo AOT(ahead of time, cioè prima dell'esecuzione), quindi dex2oat produce il codice nativo .oat , specifico per l'architettura del dispositivo in questione, inoltre .oat è un codice binario direttamente eseguibile e specifico per il dispositivo.

1.6.2 ART, AOT, Dalvik

DalvikVM permette di eseguire just in time il codice .dex, ossia il bytecode compilato dei codici in kotlin/java,Il codice in formato .dex viene runnato in due modi diversi: la parte di codice più utilizzata (Hot Code) viene compilato Just-In-Time; il resto (Cold Code) viene interpretato. la VM di ART invece sostituisce dalvik da android 5.0, si basa su AOT, perciò permette di runnare il codice nativo .oat. l'app viene interamente compilata in codice macchina nativo durante l'installazione sul device, e non durante l'esecuzione(ahead of time, prima dell'esecuzione). il codice compilato è perciò direttamente eseguibile dalla macchina(codice binario) riducendo i tempi di esecuzione rispetto al DalvikVM e rispettivamente al .dex

1.7 ZYGOTE

Zygote è un processo che viene avviato al momento dell'accensione del dispositivo; è un componente fondamentale del sistema operativo Android, fungendo da processo genitore per tutte le applicazioni e i servizi di sistema. La sua funzione principale è ottimizzare l'avvio delle applicazioni e la gestione della memoria. lo scopo è quello di avviare la prima istanza di ART e crearne una nuova ogni qualvolta viene avviato un nuovo processo, eseguendo una fork() (una copia del processo padre, di ART) sulla prima istanza; ogni app nasce già con un ambiente di esecuzione pronto. oltre a questo precarica le librerie java condivise(riducendo i tempi di avvio) ed avvia il system server. copy-on-write() permette di duplicare le risorse solamente dopo una scrittura.

1.8 Ruolo di JNI nello stack android

Il JNI svolge un ruolo cruciale, facilitando l'integrazione tra il codice gestito (Java o Kotlin) e le librerie native. JNI permette alle applicazioni Android di chiamare funzioni native, offrendo accesso a funzionalità specifiche del sistema o a librerie ottimizzate per prestazioni elevate. Questo è particolarmente utile quando si necessita di eseguire operazioni che richiedono un uso intensivo delle risorse o quando si vuole

sfruttare codice legacy già esistente. Attraverso JNI, le applicazioni possono interagire direttamente con componenti hardware del dispositivo, come fotocamere e sensori. L'uso di codice nativo tramite JNI può migliorare le prestazioni in situazioni dove il codice Java potrebbe risultare meno efficiente. [JNI(Java native interface) è una componente dello stack android ed il suo ruolo è quello di mettere in comunicazione il livello più superficiale delle applicazioni con le librerie native.]

1.9 SDK e NDK

SDK(software developement kit) è l'insieme dei software utili allo sviluppo di app android in linguaggio Java o Kotlin e che girano su macchina virtuale ART. Con SDK si favorisce la portabilità, infatti tali codici sono indipendenti dall'hardware. NDK(native development kit) permette di sviluppare applicazioni in linguaggio nativo, usando C/C++, il codice viene ricompilato specificatamente per l'architettura del dispositivo.

1.10 Componenti di un app android

I componenti di un app android sono i seguenti: Activity, Service, Broadcast receaver, contetn provider, Bundle. Fungono da entry point, quindi da punti di accesso per utenti o applicazioni che desiderano utilizzare i dati di un'altra applicazione.

1.10.1 Activity - schermata - user entry point

Una activity è una singola schermata con la quale l'utente può interagire con l'applicazione, quindi funge da entry point(user entry point).

Ciclo di vita

Il ciclo di vita di un'activity è scandito da tre fasi:

- esecuzione: è posta in primo piano
- pausa: oscurata parzialmente
- terminata: quando viene chiusa e una nuova activity viene messa in primo piano

I metodi per gestire un'activity e le sue fasi sono: onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy().

Visible lifetime e foreground lifetime

- visible lifetime: da onStart() a onStop()
- foreground lifetime: da onResume() a onPause()

1.10.2 Service - operazioni in background

Un service è un componente e serve a eseguire operazioni in background, senza un'interfaccia utente

Metodi callback

I metodi callback di un service android sono i seguenti:

1. onCreate(): viene invocato solo al momento della creazione del service e serve per inizializzare le risorse necessarie al service

2. onDestroy(): viene invocato quando il service viene terminato e serve per "svincolare" tutte le risorse utilizzate nel corso dell'esecuzione del service
3. onStartCommand(): permette di richiamare una operazione da eseguire in background
4. onBind(): mette in comunicazione il service e l'app che richiede quel service

1.10.3 Broadcast receaver - ascolto di eventi

Il broadcast receaver è un componente passiva che resta in ascolto per degli eventi. Rimane in ascolto anche se l'app non è in esecuzione ed ha solo un metodo di callback: onReceive().

1.10.4 Content provider - dati app

Il content provider è un componente che gestisce l'accesso ai dati di un'applicazione, l'accesso ai dati avviene come in un database. Esempio di content provider: rubrica telefonica.

Metodi

I metodi di un content provider sono:

1. onCreate(): inizializza il content provider
2. update(): aggiorna un istanza
3. remove(): rimuove un istanza
4. insert(): inserisc una nuova istanza
5. query(): restituisce il risultato di una query

Struttura URI

I content provider sono identificati da URI(Uniform resource identifier) l'indirizzo URI di un content provider è così composto:
content://com.example.transportationprovider/trains/122

- protocollo: content
- package del content provider: com.example.transportationprovider
- tabella: trains
- istanza: 122

Il content resolver consente di accedere ai content provider tramite URI.

1.10.5 Bundle*

Un Bundle è una mappa chiave/valore usata per passare dati tra activity e fragment. serve a passare parametri a un'altra Activity o Fragment, a salvare temporaneamente dati e a restituire risultati.

1.11 Intent - richiesta avvio componenti

Un intent è una richiesta che permette di avviare le componenti un'app.

1.11.1 Implicito e Esplicito

Esistono due tipologie di intent:

- esplicito: specifica destinatario (tramite un id) e la componente richiesta
- implicito: non viene specificato un destinatario e quindi è un intent utile per la richiesta generica di determinate componenti.

1.12 Android manifest

L'androidManifest.xml è un documento che svolge un ruolo chiave all'interno di un'app android. Al suo interno sono presenti le informazioni essenziali dell'applicazione:

- le componenti dell'app(activity, service, Broadcast receiver, content provider)
- intent
- permessi
- informazioni sulla compatibilità(versione minima di android richiesta)
- ID, package dell'app, icone, label *

1.12.1 Tag dei permessi

Al suo intern ìo troviamo i tag dei permessi, che servono a ... . Distinguiamo due tag:

- <permission>: definisce un nuovo permesso personalizzato, usi questo tag quando vuoi che altre app chiedano un permesso per accedere a componenti della tua app
- <uses-permission>: dichiara che la tua app ha bisogno di un permesso già esistente(permessi di sistema o permessi definiti precedentemente), è ciò che serve quando vuoi, ad esempio, accedere alla fotocamera, alla posizione, o all'internet.

1.13 Sandboxing - sicurezza applicazioni

Le app android sfruttano il concetto di sandboxing riguardo la gestione della sicurezza. ogni applicazione è confinata all'interno del proprio sandbox. ciò vuol dire che normalmente un'app in android non interferisce con le risorse esterne del sistema, ma solo con quelle a cui gli è stato dato un permesso. perciò per accedere a dati e risorse vanno accettati determinati permessi, che sono generalmente di due tipologie: predefiniti(che permettono l'accesso a funzionalità standard di android) e custom(sono propri dell'applicazione e vengono definiti all'interno del manifest dell'app).

1.14 Domande Android

1. Descrivere sinteticamente i componenti di architettura S.O Android, specificando quali componenti sono stati modificati o sostituiti rispetto al Kernel Linux e spiegarne il motivo.
2. Descrivere il meccanismo adottato dal S.O Android per gestire le situazioni di carico elevato della memoria RAM del dispositivo.
3. Descrivere i meccanismi per il risparmio energetico adottati dal S.O. Android. Si facciano degli esempi.
4. Spiegare sinteticamente il funzionamento del Power Manager di Android.
5. Spiegare sinteticamente il meccanismo di comunicazione mediante Binder su Android.
6. Spiegare il funzionamento del Binder IPC driver ed il suo ruolo nella gestione della sicurezza nel Sistema Operativo Android.
7. Spiegare sinteticamente i meccanismi di block erasing e memory wearing all'interno del SO Android.
8. Descrivere in dettaglio le differenze tra i due principali formati `.dex` e `.oat` delle applicazioni Android.
9. Descrivere sinteticamente le peculiarità di Dalvik VM. Indicare le differenze con ART.
10. Descrivere ruolo e funzionamento del processo ZYGOTE di Android.
11. Descrivere le differenze tra SDK e NDK.
12. Elencare brevemente le fasi del ciclo di vita di una Activity Android.
13. Cosa si intende per Visible Lifetime e Foreground Lifetime in Android?
14. Descrivere le caratteristiche di un Service Android.
15. Elencare e descrivere i principali metodi callback previsti dall'interfaccia di un Service Android.
16. Descrivere sinteticamente il funzionamento del componente Broadcast Receiver in Android.
17. Descrivere il funzionamento dei Content Provider in Android. Indicare inoltre la struttura del relativo URI pubblico.
18. Descrivere le tipologie di Intent previsti dal S.O Android.
19. Qual è il ruolo di `Bundle`*?
20. Cos'è l'`AndroidManifest.xml`?
21. Descrivere l'utilità dei tag `<permission>` e `<uses-permission>` nella configurazione di un'app Android.
22. Spiegare in dettaglio il meccanismo di sicurezza basato sui permessi del SO Android.
23. Spiegare ruolo e funzione dello strato JNI (Java Native Interface) nello stack Android.

1.15 Caratteristiche di Android

1.15.1 Sistema Operativo Android

Android è un sistema operativo open source per dispositivi mobili, progettato per tenere conto di caratteristiche hardware limitate, come:

- risorse di calcolo ridotte,
- affidabilità delle funzionalità di base,
- autonomia energetica contenuta,
- eterogeneità nelle configurazioni hardware.

Android si basa su un **kernel Linux** opportunamente modificato.

1.15.2 Architettura di Android

L'architettura di Android è composta da:

- **Applicazioni (Apps)**
- **Application Framework**
- **Librerie native e Android Runtime (ART)**
- **Hardware Abstraction Layer (HAL)**
- **Linux Kernel**
- **Secure Element**

Linux Kernel Contiene i driver hardware essenziali (audio, display, fotocamera, ecc.) e componenti personalizzati per Android:

- **Low Memory Killer** ottimizzato per dispositivi con RAM limitata, sostituisce il meccanismo di swap.
- **Power Management** basato sui *wakelocks*, che impediscono lo sleep della CPU o del display durante operazioni critiche.
- **Binder IPC Driver**, meccanismo esclusivo di comunicazione tra processi.
- **Logcat**, sistema di logging centralizzato.

HAL (Hardware Abstraction Layer) È lo strato intermedio tra il kernel e i componenti software di livello superiore.

- Fornisce interfacce standard ai driver hardware.
- Permette l'implementazione di nuove funzionalità senza modificare il framework.
- Rende Android indipendente dall'hardware sottostante.

Librerie Native e Android Runtime (ART)

- Include librerie scritte in C/C++.
- ART esegue il codice in formato `.dex` e `.oat`.
- È una macchina virtuale *register-based* (meno istruzioni rispetto a JVM, ma più uso di memoria).
- Le app vengono distribuite come pacchetti `.apk`, contenenti codice `.dex` e risorse.

Application Framework Fornisce l'ambiente e i servizi per l'esecuzione delle applicazioni create tramite l'SDK Android.

1.16 Gestione della memoria: Low Memory Killer

Android utilizza il **Low Memory Killer (LMK)** per gestire scenari di bassa memoria.

- Basato sul meccanismo *vmpressure*, monitora la pressione sulla memoria virtuale.
- Quando la pressione aumenta, l'*lmkd* termina processi con priorità bassa (background) per liberare RAM.

1.17 Risparmio energetico: Wakelocks

Android utilizza i **wakelocks** per gestire il consumo energetico.

- Le app devono richiederli esplicitamente.
- In loro assenza, il sistema disattiva automaticamente CPU, schermo o altre risorse.

1.18 Comunicazione tra processi: Binder IPC

- Il **Binder IPC driver** è un meccanismo sincrono che permette la comunicazione tra processi.
- Gestisce l'allocazione/deallocazione tramite *Reference Counting*.
- Ogni processo comunica senza condividere direttamente la memoria.

Sicurezza

- Garantisce isolamento tra processi grazie al modello sandbox.
- Evita la condivisione non autorizzata di dati tra app.

1.19 Storage e File System nei dispositivi Android

I dispositivi Android utilizzano memorie flash, che presentano le seguenti caratteristiche:

- **Block Erasing**: per poter scrivere nuovi dati, è necessario prima cancellare esplicitamente un blocco di memoria. Questa operazione è costosa e viene preferibilmente eseguita nei momenti di inattività.
- **No Seek Time**: le memorie flash non hanno parti meccaniche in movimento. I dati vengono letti/scritti elettricamente, quindi l'accesso è immediato, con ottime prestazioni negli accessi casuali.
- **Wear Leveling**: ogni cella ha un numero limitato di scritture. Le scritture e letture devono essere distribuite uniformemente per evitare il deterioramento precoce.

1.20 Formati `.dex` e `.oat`

`.dex` (Dalvik EXecutable)

- Codice scritto in Java/Kotlin, interpretato in modo indipendente dalla macchina.
- Favorisce la portabilità, ma con prestazioni inferiori.
- Viene eseguito tramite ART, che applica:
  - **Just-In-Time (JIT)** sul *hot code*.
  - Interpretazione sul *cold code*.

`.oat` (Optimized Android Executable)

- Codice binario nativo, generato **Ahead-Of-Time (AOT)** dal compilatore `dex2oat`.
- È specifico per l'architettura hardware.
- Occupa più spazio, ma è eseguito direttamente (più veloce).

1.21 Dalvik vs ART

Dalvik VM

- Era la macchina virtuale originaria di Android.
- Supportava solo compilazione JIT sul `.dex`.
- Codice meno ottimizzato, ma più flessibile.

ART (Android Runtime)

- Sostituisce Dalvik dalle versioni più recenti.
- Supporta sia AOT (`.oat`) che JIT + interprete.
- Offre migliori prestazioni e tempi di avvio ridotti.

1.22 Zygote

- Zygote è il processo padre di tutte le istanze ART.
- Viene avviato all'accensione del sistema, e resta in ascolto su un socket.
- Quando necessario, **forka** un nuovo processo ART.
- Precarica le librerie condivise per ridurre i tempi di avvio.
- Usa **Copy-On-Write** per ottimizzare la memoria.

1.23 SDK vs NDK

- **SDK (Software Development Kit)**: permette lo sviluppo in Java/Kotlin su ART, indipendente dall'hardware.
- **NDK (Native Development Kit)**: consente sviluppo in C/C++, richiede ricompilazione per ogni architettura.

1.24 Ciclo di vita di un'Activity

Stati principali

- **onCreate()**: inizializzazione layout e risorse.
- **onStart()**: activity visibile.
- **onResume()**: activity in esecuzione e interagibile.
- **onPause()**: activity parzialmente visibile (focus perso).
- **onStop()**: activity non più visibile.
- **onRestart()**: ritorno da onStop.
- **onDestroy()**: activity distrutta.

Foreground lifetime: da `onResume()` a `onPause()`

Visible lifetime: da `onStart()` a `onStop()`

1.25 Il Service

- Componente senza interfaccia utente, eseguito in background.
- Esempi: riproduzione audio, download, sincronizzazione.
- Metodi principali:
  - `onCreate()`
  - `onDestroy()`
  - `onStartCommand()`
  - `onBind()`

1.26 Broadcast Receiver

- Componente passivo in ascolto di eventi (sistema o altre app).
- Può ricevere eventi anche se l'app è chiusa.
- Metodo principale: `onReceive()`.

1.27 Content Provider

- Permette l'accesso strutturato ai dati di un'app, anche da altre app.
- Funziona come un database relazionale.
- Metodi principali: `onCreate()`, `insert()`, `remove()`, `update()`, `query()`.
- L'identificativo è un URI con struttura definita.

1.28 Intent

- Rappresenta una comunicazione tra componenti Android.
- Può essere:
  - **Esplicito**: destinatario specificato (es. aprire una activity interna).
  - **Implicito**: compito generico (es. aprire browser).

1.29 Bundle e Manifest

Bundle: contenitore per il passaggio di dati tra activity, spesso usato negli intent o per salvare lo stato.

AndroidManifest.xml: file XML che descrive:

- Nome del package e ID dell'applicazione.
- Componenti dell'app: activity, servizi, receiver, provider.
- Requisiti minimi (versione Android compatibile).

Tipo	**Descrizione**
`PARTIAL_WAKE_LOCK`	Mantiene attiva solo la CPU. *È il più usato.*
`FULL_WAKE_LOCK`	Mantiene CPU, schermo e tastiera attivi. *Deprecato.*
`SCREEN_DIM_WAKE_LOCK`	Schermo acceso a bassa luminosità. *Deprecato.*
`SCREEN_BRIGHT_WAKE_LOCK`	Schermo acceso a piena luminosità. *Deprecato.*

Caratteristica	**SDK**	**NDK**
Linguaggio	Java / Kotlin	C / C++
Esecuzione	Android Runtime (ART)	Codice nativo
Portabilità	Alta	Bassa
Prestazioni	Buone	Molto alte

Chapter 2

IOS

2.1 Cos'è IOS

IOS è un sistema operativo di proprietà di Apple ed è nato per soddisfare le richieste dei dispositivi mobili di Apple, come Iphone, Ipad, Ipod e Applewatch, nelle rispettive distribuzioni: IOS e Iwatch. Con IOS sono state implementate per il grande pubblico le tipiche funzionalità degli smartphone moderni, come il pinch o lo swipe. Anche il concetto di applicazioni e di interfaccia utente da smartphone nascono con IOS.

2.1.1 Componenti open e closed source

è un s.o. closed source con delle componenti open source, inoltre il suo kernel si basa su Darwin, un sistema operativo open source su licenza BSD.

2.1.2 Architettura app - modello MVC

La filosofia di base si basa sul concetto si sistema app-centric, cioè tutto è un'applicazione, anche l'interfaccia utente principale SpringBoard. Le interazioni sono tutte dirette con tocchi e gesti. L'interfaccia si basa il più possibile sull'accessibilità. La distribuzione del software è centralizzata(app store).

CocoaMVC vs MVC tradizionali

Le applicazioni IOS seguono il modello CocoaMVC, una rivisitazione di apple del modello più standard MVC(model view control).

- model: contiene i dati dell'app e la loro logica
- view: sono dei rettangoli sullo schermo e servono a gestire le interfacce utente, le gesture e possono essere composte in gerarchie
- controller: è l'intermediario tra model e view, gestisce le interazioni con l'utente aggiornando le gerarchie di view in base alla logica di model

A differenza del modello MVC standard, con CocoaMVC le componenti model e view comunicano tramite il controller, perciò è possibile riutilizzare determinate view, cosa non possibile col modello MVC stardard.

Ciclo di vita di un'app

Il ciclo di vita è il seguente:

- Not running: l'app non è stata lanciata o è stata terminata

- Inactive, l'app è in foreground ma non riceve eventi
- Active, l'app è in foreground e riceve eventi
- Background, l'app è in background e sta eseguendo un'operazione, le applicazioni hanno un timer disponibile per l'esecuzione
- Suspended, l'app è in background ma non sta eseguendo operazioni

2.2 Stack software

2.2.1 CocoaTouch, media layer, core services, core os, kernel e driver*

CocoaTouch

Costituisce il principale framework di sviluppo di software su piattaforma IOS. È orientato principalmente alle interfacce utente ed all'interazione tramite tocco. Scritto in Objective-C e basato su Cocoa per macOS.

Media layer

Si occupa della gestione dei contenuti multimediali con un focus su: immagini, video, audio, rendering 2D/3D.

Per le immagini, c'è il cosiddetto CoreImage, un manipolatore di immagini e task di visione artificiale. Per gli audio e i video, ci sono AVFoundation(registrazione, modifica e riproduzione di audio e video) e i CoreAudio e CoreVideo(si occupano di elaborare stream audio e video di basso livello).

Per il rendering 2D si utilizzano il CoreGraphics e il CoreAnimation. CoreGraphics renderizza tramite CPU mentre CoreAnimation renderizza tramite OpenGL.

Per il rendering 3D si utilizza OpenGL/Metal, OpenGL è cross platform mentre Metal è Apple Only ed offre prestazioni superiori.

Core services

Servizi essenziali per le app, ma non inerenti all'interfaccia utente:

- Foundation, la libreria standard di IOS che fornisce tipi di dato di base, collection, accesso al filesystem, algoritmi di filtraggio e ordinamento.
- CoreData, framework per la gestione e persistenza dei dati, visti come nodi di un grafo. Può essere immagazzinato come un ORM (Object Relational Mapping) con funzionalità avanzata per la gestione del ciclo di vita dei dati.
- CoreLocation, consente di ottenere la posizione GPS e l'orientazione del dispositivo.
- CoreMotion, permette l'elaborazione di eventi generati da accelerometro, giroscopio, magnetometro, ecc
- WebKit, motore di rendering di pagine web

Core os

È il livello più basso sello stack userland, comunicante direttamente con i device driver e con il kernel.

2.3 Kernel ibrido - darwin - XNU

Il kernel di IOS è ibrido e si basa su quello di Darwin. Darwin è un sistema operativo open source con kernel XNU(X is Not Unix); il kernel IOS è composto quindi da microkernel Mach e kernel monolitico BSD, che compongono XNU. (Kernel Ibrido, modularità e sicurezza derivanti dal microkernel con alcuni servizi derivanti dal monolitico)

2.3.1 Mach - microkernel

Mach è il microkernel che compone il kernel ibrido di ios ed offre le seguenti funzionalità:

- CPU scheduling
- supporto soft real time
- IPC basato su message-passing
- RPC basato su Mach IPC
- memoria virtuale paginata
- kernel thread

2.3.2 BSD - monolitico

BSD è un kernel monolitico che compone il kernel ibrido di ios, offre le seguenti funzionalita:

- Protezione della memoria
- VFS(VIrtual FileSystem)
- Networking
- Modello di sicurezza UNIX
- Modello processi
- Thread lato utente

2.4 Apple FileSystem

Il filesystem adottato da Apple è APFS(Apple FIle System) e supporta inode numbers a 64 bit; utilizza il meccanismo del Copy On Write (i file condivisi vengono ripetuti solo se modificati), e grazie a questo meccanismo crea periodicamente una snapshot del FS per ripristinarlo in caso di crash; supporta partizioni a dimensione dinamica e protegge crittograficamente i file con una o più chiavi.

Introduce:

- COW(Copy On Write): quando un file viene copiato, la sua memoria non viene duplicata, quindi grazie a COW gli snapshot(istantanee dei contenuti, in modo da poter ripristinare file o interi volumi ad uno stato noto) non utilizzano spazio addizionale al momento della loro creazione
- crittografia: cifratura a chiave singola univoca per file, o a chiave multipla per cifrare diverse parti dei file con una chiave diversa
- supporto a partizioni di dimensione dinamica tramite condivisione di container di dimensione fissa

2.5 Secure boot chain

La sicurezza nei sistemi operativi apple gioca un ruolo chiave, uno dei meccanismi che garantisce la stabilità in un dispotivo che monta ios è la secure boot chain.

Questa garantisce che l'accensione del dispositivo avvenga nel modo più sicuro possibile.

L'avvio di iOS avviene per diversi stadi, ognuno dei quali è crittograficamente verificato dallo stadio precedente.

Ogni stadio prosegue con l'esecuzione solo se quello precedente ha avuto successo.

Inoltre i diversi stadi sono dislocati in memorie differenti.

All'avvio viene eseguito del codice immutabile BootROM dalla secure ROM (memoria a sola lettura) che contiene il certificato root CA utile a verificare l'integrità degli stadi successivi.

Lo stadio successivo avviene con IBoot, il bootloader di alto livello, che verifica l'integrità del kernel XNU.

Se uno di questi stadi fallisce allora il sistema cerca di aggiornare il firmware o va in recovery mode.

BootROM, LLB, iBoot

2.6 Feature di Sicurezza

Sicurezza dei dati

Per quanto riguarda la sicurezza dei dati, iOS sfrutta un co-processore per la crittografia, cifrando usando una o più chiavi uniche ogni file; inoltre i dati biometrici sono salvati in un sottosistema hardware (SEP, Secure Enclaver Processor) inaccessibile a qualunque processo, kernel incluso. Mentre per quanto riguarda la sicurezza delle applicazioni, in iOS è presente il code signing ovvero ogni app dev'essere firmata crittograficamente da Apple per poter essere lanciate; oltre al sandboxing utilizzato per evitare che ogni app non acceda a file di altre app o a feature alle quali non ha autorizzazione

2.6.1 Sicurezza del kernel

Per la sicurezza del kernel ios sono adottati i seguenti meccanismi:

- KPP (kernel patch protection): un check periodico per verificare l'integrità del kernel
- KASLR (kernel address space layout randomization): il kernel viene caricato in memoria in indirizzi randomici
- PAC (pointer authentication) : autenticazione crittografata dei puntatori alla memoria
- write xor execute: alle pagine di memoria vengono assegnati permessi di scrittura o esecuzione, mai entrambi

2.6.2 Sicurezza applicazioni

Tramite le seguenti feature di sicurezza:

- code signing: le app devono essere firmate crittograficamente da Apple tramite il sistema AMFI (Apple Mobile File Integrity), una componente del Kernel
- sandboxing: ciascuna app funziona in sandbox, che limita l'accesso ai file di altre app e a feature per chi non ha autorizzazioni
- Entitlement: l'accesso ad alcune funzionalità può essere concesso previa richiesta di consenso all'utente

2.7 Domande IOS

1. Si descrivano le peculiarità generali del sistema operativo iOS.
2. Descrivere il modello MVC adottato da iOS per il design delle applicazioni (architettura di un'app in iOS). Chiarire in cosa esso si differenzia rispetto al modello MVC tradizionale.
3. Apple File System: caratteristiche principali, prestazioni e sicurezza anche in rapporto con altri file system noti (descrivere brevemente le caratteristiche del filesystem adottato da iOS).
4. Descrivere il funzionamento della Secure Boot Chain di iOS.
5. Spiegare i meccanismi adottati da iOS per la protezione del Kernel.
6. Schematizzare la struttura del Kernel iOS.
7. Descrivere come iOS attua politiche di sicurezza per i dati e applicazioni.
8. Descrivere sinteticamente le funzionalità gestite dal componente Mach del kernel XNU di iOS. Spiegare i vantaggi e i possibili svantaggi della gestione delle funzionalità rispetto al kernel Linux.
9. Descrivere in modo schematico ma dettagliato le funzionalità gestite dal componente BSD del Kernel XNU di iOS.
10. Indicare le funzionalità gestite da ciascuna componente del kernel ibrido di iOS.

Chapter 3

ROS

3.1 Cos'è ROS - framework

ROS è un framework che fornisce le componenti software per lo sviluppo in ambito robotico. Non è un sistema operativo.

- Permette l'interazione con le componenti fisiche di robot differenti utilizzando uno stesso codice, questo poichè vengono poste delle interfacce tra hardware e software, che risolvono il problema della diversità del hardware(astrazione dell'hardware)
- I robot sono sistemi asincroni perchè devono interaggire in tempo reale con l'esterno(reagendo a possibili imprevisti).
- comunicazione con nodi
  - topic
  - service
- publish/subscribe

3.1.1 Astrazione dell'hardware

3.1.2 Problema asincronicità

3.2 Comunicazione interprocesso - nodi

Alla base della comunicazione tra processi in ROS c'è il concetto di nodi(un nodo è un file eseguibile, un programma che fa una cosa specifica(nodo motore, nodo sensore)), topic e servizi, questa avviene grazie a due meccanismi: topic e service.

3.2.1 Topic - public/subscribe

I topic sono i canali di comunicazione tra i nodi molti a molti; la comunicazione avviene grazie al concetto di public/subscribe, ossia, un nodo può pubblicare dati su un topic e altri nodi possono iscriversi a quel topic per riceverli.(Per tipizzazione dei topic in ROS si intende che ad ogni topic (coda) sia assegnato un solo tipo di messaggio).

3.2.2 Service - request/reply

Con il service la comunicazione tra nodi che usa il concetto di request/reply;

a differenza del topic, il nodo fa una richiesta ad un nodo e questo risponderà alla richiesta(perciò request/reply). I service sono costituiti da una coppia di messaggi (uno per la request e uno per la reply).

3.3 Comando: roscore

3.3.1 Ros master - logging mode, parameter server

Il comando $roscore in ROS è il primo comando eseguibile e serve ad avviare:

- ROS MASTER: Il nodo master in ROS si occupa di stabilire una connessione per la comunicazione tra i nodi, senza di esso i nodi non potrebbero comunicare, fornisce un meccanismo di consultazione per permettere ai processi di identificarsi l'un l'altro a runtime,il comando per avviarlo è roscore
- LOGGING MODE: consente agli utenti di leggere i messaggi che i nodi si scambiano
- PARAMETER SERVER: fornisce parametri di configurazione

3.3.2 Comandi

- rosrun: rosrun package [nomefile]

  permette di avviare il file all'interno del package
- rosparam list: per fornire la lista dei parametri salvati dal server
- rosparam set background_color [valore]: per aggiornare il colore del background
- rostopic type [nome_topic]: mostra il tipo di messaggio di uno specifico topic
- rostopic pub topic/events "alarm"

  rostopic pub è il comando per pubblicare su un topic, topic/events è il percorso e alarm è il messaggio
- rosmsg show [msg] che appunto mostra la struttura con cui è composto il messaggio passato come parametro
- rosnode info [nome_nodo] mostra informazioni sul nodo passato come parametro tra cui i topic su cui pubblica (publisher) e i topic a cui è sottoscritto (subscriber)

3.4 Domande ROS

1. Si descrivano le peculiarità del sistema ROS.
2. Si descrivano sinteticamente i due principali meccanismi di comunicazione interprocesso tra nodi in ROS.
3. Descrivere le caratteristiche dei service ROS, indicando il meccanismo IPC (Inter Process Communication) adottato. Scrivere inoltre il comando da terminale per elencare i service attivi e spiegarne la sintassi.
4. Spiegare il ruolo dei messaggi nella comunicazione publish/subscribe in ROS e indicare il numero massimo di tipi di messaggi che possono essere associati ad un singolo topic.
5. Scrivere:

- Il comando da terminale per pubblicare un messaggio “alarm” di tipo stringa su `/events`.
- Il comando da terminale per mostrare il tipo di messaggio di uno specifico topic.
- I comandi per elencare i topic publisher/subscriber.

6. Spiegare cosa si intende per tipizzazione dei topic in ROS.
7. Spiegare il funzionamento dei comandi `roscore` e `rosrun` su ROS.
8. Spiegare il funzionamento del nodo master in ROS e indicare il comando per avviarlo dal terminale.
9. Descrivere cos’è il Parameter Server di ROS, specificare i comandi per:
   - Elencare tutti i parametri.
   - Aggiornare il parametro `background_color` con valore 130.
10. Quale funzione riveste il ROS Parameter Server? Si indichi almeno un comando per interagire con tale componente.
11. Spiegare il funzionamento delle callback in ROS, aiutandosi con degli esempi.
12. Si spieghi il meccanismo di storage e retrieval dei parametri a runtime previsto da ROS (ossia cos’è il ROS Parameter Server). Aiutarsi con uno schema.

Chapter 4

NUTTX/PX4

4.1 UAV e autopilota PX4

Un UAV è un aereoveicolo che vola grazie a sistemi di autopilota come PX4.

4.1.1 Layers di PX4 - flight stack e middleware

Il flight stack è uno dei due layer del sistema operativo open source PX4.
Consta di tre componenti fondamentali:

- estimator: riceve dati dai sensori e definisce un setpoint
- controller: tramite i dati sullo stato stimato del veicolo effettua delle correzioni, quindi rileva il setpoint riguardante l'assetto attuale e lo modifica affinchè il veivolo si muova verso la posizione desiderata
- mixer: riceve dati dal controller e li traduce in comandi per i motori

Il middleware è un componente chiave che collega tutte le parti del sistema tra loro(sensori, moduli di controllo, attuatori ecc), perciò è fondamentale per la comunicazione tra i vari blocchi del flight stack.
I moduli per la comunicazione sono:

- oURB: è un bus che permette lo scambio tra messaggi
- MAVLink: permette la comunicazione tra il veivolo e la stazione di terra(GCS) e lo scambio di messaggi tra autopilota e altri componenti esterni

4.1.2 Caricamento firmware PX4

Il caricamento del firmware (il processo per cui il PX4 viene installato) PX4 avviene tramite la cross-compilazione dei makefile, che definiscono le regole per la costruzione dei moduli caricabili. per includere tali codici compilati usiamo il tool make.

Integrazione moduli esterni

Inoltre se si desidera integrare moduli esterni bisogna includerli in default.cmake, mentre nel file CMake-List.txt si specificano le istruzioni per la compilazione del codice aggiunto in default.cmake

4.2 Sistemi operativi real-time

4.2.1 Tipologie: hard e soft

Un sistema operativo real-time DEVE essere deterministico(sistema in cui tutte le variabili sono definite, di cui si possono avere informazioni in tempo reale) e la risposta di tali richieste devono essere fornite entro un certo intervallo di tempo, perciò che rispettino una deadline.

Possono essere:

- hard real time: hai sempre delle deadline sui processi e vanno rispettate(esempi: sistemi di controllo centrali nucleari)
- soft real time: ammette violazioni sulle deadline(esempi: sistemi multimediali)

4.2.2 Algoritmi di scheduling CPU

Gli algoritmi di scheduling utilizzati nei RTOS sono:

- Round robin: è un algoritmo con preemption(con preemption un processo può interrompersi e poi riprendere successivamente), in cui viene definito il tempo in cui un processo può usare la CPU, potrebbe violare le deadline
- Priority based: è un algoritmo con preemption che da priorità ai processi real time, è solo per i soft real time e non tiene conto delle deadline
- Rate monotonic: è adatto ai sistemi hard real time(NuttX) e da la priorità ai processi che richiedono la CPU più spesso(con preemption)

Quale usa NattX?

NuttX è Hard real time perchè

4.3 Cos'è NattX

NattX è un so hard real time scritto principalmente in linguaggio C,è composto da due layers:

4.3.1 Architettura - user layer e os layer*(driver)

- user layer: è il livello più vicino all'utente, che permette la comunicazione con l'utente e contiene le librerie che permettono la comunicazione con il layer inferiore
- os layer: contiene i driver e la logica di base di NuttX, questo livello è a sua volta suddiviso in:
  - upper half driver: interagisce con lo user layer e implementa operazioni generiche come read e write
  - lower half driver: interagisce con l'hardware

4.3.2 IPC - signals - POSIX - semafori

I meccanismi di IPC(Si tratta di un insieme di tecnologie software che consentono a diversi processi di scambiarsi dati e informazioni) in NuttX sono:

- signals: permette ai task di inviare segnali tra loro tramite il taskID

- POSIX message queue: i task comunicano tra loro tramite code, ad ogni task viene associata una coda e viene messo in pausa, quando riceve un messaggio il task si sveglia
- semafori: tramite mutual exclusion e s i task vengono inizializzati come acceso e spento, 1 o 0

4.4 Task

Un task in NuttX non è altro che un processo che non gestisce uno spazio di indirizzo privato, ciò poiché molti microcontrollori non possiedono la Memory Management Unit e quindi NuttX opera solo con un flat address space. le proprietà di un task sono descritte in una struttura detta TCB, che specifica risorse associate, stato e ID. I task sono organizzati in relazioni parent-child, il task padre attende la fine del task figlio prima di essere rimossa.

4.5 Domande NUTTX/PX4

1. Descrivere i principali algoritmi di scheduling della CPU adottati dai sistemi operativi real-time. Specificare quale fra essi è impiegato nel sistema operativo NuttX e chiarire in dettaglio le motivazioni della scelta.
2. Descrivere brevemente l'architettura del flight stack PX4.
3. Descrivere gli stati previsti dallo scheduling dei task su NuttX.
4. Descrivere il concetto di task in NuttX, chiarendo le eventuali analogie e differenze con processi e thread di un sistema operativo general purpose.
5. Descrivere i principali meccanismi di IPC di NuttX.
6. Descrivere come avviene il caricamento del firmware di PX4 su una piattaforma.
7. Spiegare le principali differenze tra un sistema operativo hard e soft real-time. Indicare a quale delle due tipologie appartiene il sistema operativo NuttX e motivare la risposta.
8. Spiegare la struttura generale dei driver del sistema operativo NuttX.

Chapter 5

TEORIA GENERALE

5.1 Struttura sistema op e funzionamento pc

5.1.1 Interrupt Handler

5.1.2 Remote Procedure Call

5.1.3 Fork, processo zombie e orfano

5.1.4 Interrupt sincroni e asincroni

5.2 Thread

5.2.1 Modelli di programmazione multithread

Molti a uno

Uno a uno

Molti a molti

5.3 Scheduling

5.3.1 Schedulatore a medio termine

5.3.2 Multilevel queue

5.3.3 Short term e long term scheduler

5.3.4 Schedulazione con coda multilivello retroazionata

5.3.5 Scheduling SJF

serie esponenziale - next CPU burst

5.3.6 Convoy effect

5.3.7 Code di schedulazione

5.4 Sincronizzazione processi

5.4.1 Condizione necessaria semaforo

5.4.2 Busy waiting

5.4.3 Starvation

5.5 DeadLock

5.8 Domande

5.8.1 Struttura sistema op e funzionamento pc

1. Spiegare la struttura e descrivere le funzionalità dell'interrupt Handler. Chiarire in quale momento del processo di gestione di una interruzione esso interviene.
2. Si spieghi brevemente a cosa serve una Remote Procedure Call e qual è la funzione del client-stub all'interno di essa.
3. Spiegare sinteticamente il funzionamento della chiamata di sistema fork e indicare in quali casi può riscontrarsi la presenza di un processo zombie o un processo orfano
4. chiarire le differenze (intrinseche e di gestione da parte del S.O) tra gli interrupt sincroni e asincroni.

5.8.2 Thread

1. Descrivere sinteticamente i tre modelli di programmazione multithread. Per ciascuno di essi chiarire pregi e difetti. Indicare anche quale dei modi produce il maggiore impatto sulle prestazioni della macchina

5.8.3 Scheduling

1. Si chiarisca il ruolo dello schedulatore a medio termine nell'ambito del processo di scheduling della CPU e se ne spieghino inoltre caratteristiche tecnologiche.
2. Si descrivano brevemente le caratteristiche degli algoritmi di scheduling mediante code multithread (multilevel queue). Si chiarisca con precisione di quale meccanismo di schedulazione esse sono una approssimazione
3. Si chiarisca il ruolo dello schedulatore a medio termine spiegandone inoltre la funzionalità
4. Si spieghi brevemente qual è la differenza (funzionale ed architetturale) tra lo short-term scheduler ed il long-term scheduler
5. Rappresentare schematicamente il funzionamento della schedulazione con coda multilivello retroazionata. Spiegare perché tale meccanismo previene la starvation dei processi.
6. Si descriva la serie esponenziale per la stima del " next CPU burst" in un algoritmo di scheduling SJF.
7. Cos'è il Convoy effect?
8. Spiegare l'utilizzo del meccanismo delle code di schedulazione per gestire la distribuzione dei processi nella macchina, per l'accesso alla CPU

5.8.4 Sincronizzazione processi

1. Qual è la condizione indispensabile perché sia possibile realizzare un semaforo?
2. Fenomeno del busy waiting nei semafori. Da dove ha origine? Come è possibile risolverlo
3. Si descriva brevemente il fenomeno della starvation nell'ambito della schedulazione della coda di ready nel ciclo di esecuzione dei processi e si dica, motivando la risposta, con quali approcci si può limitare un effetto del genere.

5.8.5 DeadLock

1. Descrivere brevemente le condizioni necessarie per il verificarsi di un deadlock
2. Si spieghi quale è il significato del permesso di esecuzione assegnato ad un file relativo ad un device
3. Spiegare sinteticamente le problematiche associate al rilascio anticipato delle risorse nel caso di deadlock detection and recovery

5.8.6 Memoria centrale e virtuale

1. Descrivere il meccanismo dell'allocazione collegata (linked) di un file, evidenziando vantaggi e svantaggi rispetto al meccanismo di allocazione contigua
2. Descrivere il problema della frammentazione (interna ed esterna) della memoria, con particolare riferimento allo schema di gestione che fa uso della segmentazione. Quali sono le differenze rispetto all'utilizzo della paginazione?
3. Si illustri l'implementazione di sostituzione delle pagine LRU mediante utilizzo di stack
4. Descrivere sinteticamente le tre principali tecniche di paginazione della memoria per architettura a 64 bit
5. Si illustri il significato e le conseguenze della cosiddetta anomalia di Belady. Spiegare se ed eventualmente in che modo essa è superabile.
6. Si tracci lo schema del contenuto di un Process Control Block evidenziando per ciascuna componente in quale circostanza della vita del processo viene creata/aggiornata

5.8.7 Struttura dischi

1. Quali sono le componenti del tempo di accesso a disco? Quale di esse tende ad essere dominante? È più conveniente effettuare pochi trasferimenti di blocchi di grosse dimensioni oppure molti trasferimenti di blocchi piccoli? Si motivino le risposte

Chapter 6

Lista comandi utili linux

6.1 Comandi

6.1.1 ls, ps, cut, grep, head, sort, sed

6.2 Domande

1. Spiegare l'utilità del comando `kill` sottolineando le differenze nell'utilizzo con i due flag elencati qui sotto:
   - `kill -15 PID`
   - `kill -9 PID`
2. Si spieghino le differenze tra i file `/etc/shadow` e `/etc/passwd`. Si indichi come sono strutturati, qual è la rispettiva funzione e chi può accedervi.
3. Chiarire le differenze esistenti tra i seguenti due comandi:
   - `chmod g+w nome.prova`
   - `chmod 664 nome.prova`
4. Spiegare il significato delle seguenti variabili di ambiente e chiarire preliminarmente cosa permette di fare questo strumento in Linux e in altri sistemi operativi: `PATH`, `HOME`, `USER`, `SHELL`, `PWD`.

5. Sia assegnato il seguente albero di directory e si supponga di avere i privilegi di superutente:

```
/
  var/
    www/
      html/
  etc/
    passwd
    default/
    shadow
```

   Dire come l'albero risulta modificato dopo la seguente sequenza di comandi:

```
cd /etc/default
cp ../pa* ./
cd /var/www
mv ./html /
touch ./html
mkdir prova
cd -
touch ./file
```

6. Spiegare, motivando la risposta, quali sono gli effetti del comando:

```
chmod 744 ./file_A
```

   relativamente al file:

```
-rwxrwxr-x 2 user group_a 55231 Apr 23 20:32 file_A
```

7. Spiegare il significato dei campi del seguente output del comando:

```
$ ls -la
drwxrwxr-x 2 pippo group_mail 52874 Feb 8 16:36 .file
```

8. Si dica con precisione quali informazioni sono contenute all'interno del file `/etc/passwd` e se questo è accessibile da un utente senza privilegi di amministratore.

9. Spiegare il significato dei campi del seguente output del comando `ls -la`:

```
drwxrwxr-x 2 user_group 34234 Feb 9 15:34 .file_name
```

10. Si supponga che la CWD sia `~/dir`. Descrivere la modalità per copiare il file `/dir/file.prova` nella home directory, evitando di sovrascrivere un eventuale file esistente.

11. Indicare quali sono i possibili comandi Linux che possono essere adoperati per la cancellazione di una directory.

12. Disegnare l'albero di directory e file generato dalla seguente sequenza di comandi digitati, avendo i permessi di amministratore di sistema. La directory di partenza è la home directory dell'utente attualmente loggato (nome utente: `user`):

```
root@ubuntu:/home/user# mkdir dir1 dir2
root@ubuntu:/home/user# cd dir1
root@ubuntu:/home/user# touch file1.txt
root@ubuntu:/home/user# mv file1.txt ..
root@ubuntu:/home/user# cd ../dir2
root@ubuntu:/home/user# touch file2.txt
root@ubuntu:/home/user# cp ../file1.txt .
```

13. Si indichi la struttura tipica dell'output del comando `ps aux`. Spiegare il significato delle flag e indicare un possibile comando alternativo. Il comando `ps` permette di visualizzare una schermata statica dei processi attivi.

14. Spiegare a cosa serve il comando `top`, chiarendo le differenze principali con comandi analoghi. Specificare in dettaglio quali informazioni esso produce in output.

15. Si indichi la struttura tipica dell'output del comando `man`.

16. Si chiarisca la differenza di utilizzo tra i comandi `help`, `man` e `h`, riportandone la sintassi. Chiarire la differenza tra i tre diversi tipi di comandi che una shell può eseguire.

17. Qual è la differenza essenziale tra i comandi `jobs`, `top` e `ps`?

18. Si scriva la sinossi del comando UNIX per cambiare il gruppo di un file, specificando i prerequisiti indispensabili.

Esercizi

1. Si scriva il comando complesso che permetta di contare quanti processi all'interno della macchina host hanno un PID caratterizzato da un numero pari.

2. Si scriva un unico comando (pipeline) per fornire la lista dei file semplici all'interno della home directory per cui gli utenti 'altri' del sistema possiedano esclusivamente il permesso di lettura e il cui nome sia una stringa che termina con una vocale seguita da una cifra.

3. Scrivere una sequenza di comandi che consenta di ordinare in ordine alfabetico il contenuto di un file di nome `prova.testo` e di estrarre dal file ordinato le prime 5 righe scrivendole in appendice sul file `output.txt` presente nella home directory dell'utente loggato.

4. Si scriva un unico comando (pipeline) in grado di restituire la lista delle directory all'interno della home per le quali il gruppo proprietario ha (esclusivamente) il permesso di lettura e il cui nome è una stringa che termina con tre cifre.

5. Si scriva il comando per mostrare sullo standard output l'elenco degli studenti che hanno una votazione maggiore o uguale a 22, con ordinamento numerico decrescente eliminando le righe ripetute:

   - mario rossi 22
   - rocco verdi 24
   - marco rossi 23
   - sergio bianchi 26
   - mario rossi 22
   - rosa barbieri 28

6. Contare quanti processi all'interno della macchina host non appartengono all'utente identificato dal nome `UserA`.

7. Elaborando il contenuto del file `/etc/passwd`, si scriva il comando per mostrare sullo standard output l'elenco di tutti gli utenti del sistema diversi da `root` che utilizzano `bash` come shell di default.

8. Scrivere il comando Linux che seleziona dal file `/etc/passwd` e stampa a video tutte le righe che contengono due cifre dispari, separate da un qualsiasi carattere.

9. Si scriva il comando Linux che permette di aggiungere un account utente al sistema con le seguenti caratteristiche:
   Nome: `mario`
   Home directory: `/home/mario`
   Shell: `/bin/bash`
   Gruppo principale: `mario`
   Altri gruppi: `users, mail, printer`

10. Scrivere una pipeline di comandi che fornisca in output il numero di file semplici (includendo anche quelli nascosti), presenti all'interno della directory corrente, per i quali:
    - l'utente proprietario ha almeno i permessi di lettura e scrittura;
    - il nome dell'utente proprietario inizia con la lettera 'a';
    - il gruppo proprietario ha solo il permesso di lettura ed il suo nome termina con 'so'.
11. Utilizzando esclusivamente i comandi `ls` e `sed` si visualizzi l'elenco in formato esteso dei file presenti all'interno della directory corrente, eliminando la colonna relativa all'ACL.
12. Scrivere una pipeline di comandi che fornisca in output il numero di file semplici (anche nascosti), presenti all'interno della directory corrente, di cui l'utente proprietario possiede almeno i permessi di scrittura.
13. Spiegare il comando:
    `cat lista_nomi.txt | head -4 | grep '\(3|[34])$' -v > output.txt`
    e indicare l'effetto che esso produce se il file `lista_nomi.txt` è il seguente:

    ```
    mario rossi 080 541234
    paolo rossi 083 434232
    rocco verdi 093 3424033
    marco rossi 066 091312
    sergio bianchi 083 311234
    fabio giallo 080 123080
    rosa barbieri 081 312313
    ```

14. Spiegare qual è l'effetto del seguente comando: ls -aF1 — grep '^\.'
15. Si scriva il comando per inviare l'output del list della directory `/etc/var/www/html` sul file `result.txt` presente nella home directory dell'utente loggato, filtrando tutte e solo le directory di cui il gruppo proprietario ha il permesso di lettura ed esecuzione.
16. Spiegare il comando:
    `cat lista_nomi.txt | head -3 | grep '\<r.*i\>.*$' -v > output.txt`
    e indicare l'effetto che esso produce se il file `lista_nomi.txt` è il seguente:

    ```
    mario rossi 080 541234
    paolo rossi 081 434234
    luigi verdi 094 3424080
    marco rossi 066 091312
    sergio bianchi 080 311231
    fabio giallo 080 123080
    rosa barbieri 081 312313
    ```

17. Si scriva il comando per inviare l'output del list della directory `/home/utenti` sul file `lista.txt` presente nella home dell'utente attualmente loggato, filtrando solo i file standard su cui gli utenti proprietari hanno almeno i permessi di lettura e scrittura.
18. Si scriva il comando per inviare l'output del list della directory `/etc/var/www/html` sul file `output.txt` presente nella home directory dell'utente loggato, filtrando tutti e soli gli elementi che iniziano con la lettera 'm' o 'M'.

19. Utilizzare i comandi `sed` e `cut` per costruire il file `lista_utente.testo`, nella home directory dell'utente attualmente loggato, in modo che contenga l'elenco di tutti i nomi degli utenti del sistema insieme alle relative home dir, separati dal carattere '-' (esempio: user-/home/user).

20. Come è possibile sapere quali utenti (esclusivamente il nome), il cui account cominci per 'm' o 'M', sono titolari di un processo denominato `gnome-terminal` su terminali diversi? Eliminare eventuali righe duplicate.

21. Scrivere il comando Linux per visualizzare le informazioni estese di tutti i file semplici presenti all'interno della directory corrente, per i quali l'utente proprietario ed i membri del gruppo proprietario hanno almeno i privilegi di lettura e scrittura e tutti gli altri utenti possiedono solo il permesso di lettura. Le informazioni estratte devono essere salvate su un file di nome `output.txt`.

22. Il file `elenco.txt` ha il seguente contenuto:

```
Rossi Mario
Rossi Paolo
Verdi Giuseppe
Bollo Franco
Bianchi Alessandra
Cerri Elena
Dodi Francesco
```

    Si costruisca il file `risultato.txt` che contenga in ordine alfabetico tutti gli elementi con cognome che inizi per R o per B.

23. La directory corrente contiene il file `binary.txt` le cui righe sono costituite da stringhe formate solo da combinazioni di '1' e '0'. Si scriva il comando UNIX per estrarre da questo file solo le righe che contengono almeno tre '0' consecutivi e produrre il risultato nel file `output.txt` nella home directory dell'utente loggato.

24. Si scriva il comando per inviare l'output del list della directory `/var/www/html` sul file `web.list` presente nella home directory dell'utente loggato, filtrando tutti e soli i file o le directory nascoste.

25. Si scriva il comando per inviare l'output del list della directory `/home/utenti` sul file `lista.txt` presente nella home dell'utente attualmente loggato, filtrando solo i file standard su cui gli utenti proprietari hanno almeno i permessi di lettura e scrittura.

26. Scrivere il comando UNIX per visualizzare, dal più piccolo al più grande, i numeri contenuti all'interno del file `numeri` presente nella directory corrente e produrre il risultato nel file `fileA` nella home directory dell'utente attualmente loggato.

27. Scrivere il comando UNIX per visualizzare in ordine alfabetico i file contenuti nella directory corrente e produrre il risultato nel file `fileA` nella directory dell'utente attualmente loggato.

28. Scrivere il comando per leggere dal file ./lista un elenco di nomi, cognomi e codici fiscali mostrando a schermo soltanto le righe relative agli utenti nati nel 1982.

29. Estrarre dalle ultime 6 righe del file matricole.txt Contenuto nella home directory dell'utente loggato tutte le righe che contengono almeno 3 cifre pari consecutive

30. Scrivere il comando linux per montare il dispositivo a blocchi /dev/sdb1 con i permessi di lettura e scrittura in /media

31. Scrivere una sequenza di comandi che consenta di ordinare in ordine alfabetico il contenuto di un file di nome prova.testo e di estrarre dal file ordinato le ultime 3 righe scrivendole in appendice sul file output.txt presente nella home directory dell'utente loggato.

32. Scrivere il comando che consenti di sostituire all'interno del file matricole.txt nelle prime 8 righe tutte le occorrenze che iniziano per 'abb' con la stringa 'cbb'.

33. Scrivere il comando UNIX per creare nella home directory dell'utente attualmente loggato un link simbolico di nome slink_studenti al file studente.testo Esso deve essere presente nella directory corrente

6.3 Risposte domande esame

1. `ps | grep -E '[[:blank:]]*[[:digit:]]*[02468]\>'`

   Non uso flag di ps perché il primo campo dell'output standard ha come primo campo, seguito da un numero non ben definito di spazi o tabulazioni, l'ID del PID. Inoltre specifico che la parola (cioè il codice PID) finisca con un numero pari.

2. `ls -l | grep -E '^-.{6}r--.*[aeiou][[:digit:]]$'`

   Non è necessario specificare che `aeiou` e `digit` siano l'ultima parola perché il dollaro `$` indica proprio la fine della stringa.

3. `sort prova.testo | head -5 >> ~/output.txt`

   `sort` ordina alfabeticamente, `head` prende le prime 5 righe, `>>` scrive in appendice.

4. `ls -l ~/ | grep -E '^d.{3}r--.*[[:digit:]]{3}$'`

5. `grep -E '(2[2-9] | 30)' risultati_esame.txt | sort -nr | uniq`

   `sort -n` ordina numericamente crescente, `r` inverte il sort, `uniq` elimina duplicati.

6. `ps -u ~/ | grep -cv 'UserA'`

   `ps -u` mostra processi con nome utente all'inizio; `grep -cv` conta tutte le righe che NON contengono UserA.

7. `grep -Ev 'root' /etc/passwd | grep -E 'bash$'`

   Struttura del file `/etc/passwd` → `user:password:uid:gid:info:home:shell`

8. `grep -E '[13579].[13579]' /etc/passwd`

9. `sudo useradd -m -d /home/mario -s /bin/bash -g mario -G users,mail,printer mario`

   `sudo` è necessario. Flag usati: `-m` crea home, `-d` la specifica, `-s` specifica la shell, `-g` gruppo principale, `-G` gruppi secondari.

10. `ls -la | grep -cE '^rw.r--.*/<a.*/>.*/<.*so/>.*$'`
    `ls -la | grep -cE '^rw.r--.{3}[[:blank:]]+[[:digit:]]+[[:blank:]]+/<a.*/>[[:blank:]]+/<.*so/>.*$`

11. `ls -l | sed 's/^-.\{9\}/ /'`

    `sed 's/testo da sostituire/nuovo testo/'`

12. `ls -la | grep -cE '^-.w'`

13. `cat lista_nomi.txt | head -4 | grep ' (3|[34])$' -v > output.txt`
    Tramite `cat` e `head` prendo le prime 4 righe. `grep -v` esclude righe che finiscono per 33 o 34. Risultato:

    ```
    paolo rossi 083 434232
    marco rossi 066 091312
    ```

14. `ls -aF1` mostra file e directory (anche nascosti) uno per riga, con simboli di tipo. `grep '^\.'` filtra quelli che iniziano con punto.

15. `grep -E '^d.{3}r.x' /etc/var/www/html > ~/result.txt`

16. `cat lista_nomi.txt | head -3 | grep '\<r.*i\>.*$' -v > output.txt`
    Estrae le prime 3 righe e poi esclude le parole che iniziano con `r` e finiscono con `i`. Risultato:

    ```
    luigi verdi 094 3424080
    ```

17. `ls -l /home/utenti | grep -E '^-rw' > ~/lista.txt`

18. `ls -l /etc/var/www/html | grep -E '/<[mM].*/>' > ~/output.txt`

19. `cut -f1,6 -d':' /etc/passwd | sed 's/\:/\-/' > ~/lista_utente.testo`

20. `ps ax -f | grep -E '^[mM].*gnome-terminal$' | cut -f1 | uniq`

21. `ls -l | grep -E '^-rw.rw.r--' > output.txt`

22. `grep '^[RB]' elenco.txt | sort > risultato.txt`

23. `grep '000' binary.txt > ~/output.txt`

24. `ls -a1 /var/www/html | grep -E '^\.' > ~/web.list`
    Uso `-a1` per mostrare file nascosti uno per riga, senza dettagli.

25. `ls -l /home/utenti | grep -E '^-rw' > ~/lista.txt`

26. `sort -n numeri.txt > ~/fileA.txt`

27. `ls | sort > ~/fileA.txt`

6.4 Domande comandi linux ChatGPT

1. Scrivere un comando per contare i file nella directory corrente che siano leggibili solo dal proprietario e scrivibili da tutti.
2. Trovare i file nascosti nella home directory che terminano con `.conf` e contengono almeno una cifra nel nome.
3. Visualizzare l'elenco degli utenti presenti nel file `/etc/passwd` che utilizzano `/bin/sh` come shell.
4. Visualizzare il PID di tutti i processi che contengono la parola `ssh` nel comando e appartengono all'utente corrente.
5. Estrarre dal file `studenti.txt` tutte le righe contenenti nomi che iniziano per vocale e voti superiori a 25.
6. Scrivere un comando per elencare i file nella directory corrente ordinati per dimensione decrescente.
7. Visualizzare le directory nella home il cui nome termina con `_backup` e che siano scrivibili solo dal proprietario.
8. Estrarre dal file `log.txt` le righe che contengono tre lettere maiuscole consecutive.
9. Cercare i file nella home che abbiano permessi `r--r-----` (solo lettura per owner e gruppo).
10. Elencare tutti i processi avviati da utenti il cui nome inizia con la lettera `s`.
11. Scrivere un comando per sostituire in un file tutte le parole che iniziano per `temp` con `TMP_`.
12. Creare una pipeline per contare le directory il cui nome contiene un numero.
13. Trovare i file che hanno permessi di esecuzione per tutti e dimensione maggiore di 1 MB.
14. Mostrare il contenuto del file `/etc/group` limitandolo ai gruppi che contengono più di due membri.
15. Ordinare alfabeticamente un file chiamato `prodotti.txt` ed estrarre le ultime 5 righe.
16. Cercare tra i file `.log` nella directory corrente quelli che contengono almeno 5 occorrenze della parola `error`.
17. Estrarre solo i nomi utente dal file `/etc/passwd` usando `cut` e salvarli su `utenti.txt`.
18. Stampare a video tutte le righe di `esami.txt` che non terminano con un numero.
19. Contare quanti file nella directory corrente hanno un numero dispari di caratteri nel nome.
20. Sostituire tutte le vocali accentate con quelle non accentate nel file `documento.txt`.
21. Mostrare le righe duplicate presenti in `log_accessi.txt`.
22. Scrivere una pipeline per mostrare le 10 righe più lunghe del file `testo.txt`.
23. Creare nella home un link simbolico a un file chiamato `riservato.dat` presente nella directory corrente.
24. Cercare tra i file `.sh` quelli che iniziano con `#!/bin/bash` e che contengano la funzione `main()`.
25. Ordinare un file contenente voti e nomi degli studenti in base al voto, in ordine crescente.
26. Creare un comando che seleziona gli utenti dal file `/etc/passwd` con ID maggiore di 1000 e shell `/bin/bash`.

27. Scrivere un comando per visualizzare l'elenco dei file presenti nella directory `/home/docenti` che siano directory, il cui nome inizia con una consonante seguita da una cifra, e per i quali il gruppo proprietario abbia solo permessi di lettura.

28. Scrivere un comando per salvare in `output.txt`, nella home directory dell'utente loggato, la lista dei file (anche nascosti) della directory corrente che siano eseguibili dal proprietario, leggibili dal gruppo e non accessibili dagli altri utenti.

29. Utilizzare esclusivamente i comandi `sed` e `cut` per costruire il file `utenti_group.txt` nella home directory dell'utente loggato, contenente le coppie `nomeutente:gruppo` prese dai file `/etc/passwd` e `/etc/group`.

30. Scrivere un comando che restituisca i nomi degli utenti del sistema che non hanno una home directory personale, elaborando il file `/etc/passwd`.

31. Scrivere un comando per contare quanti file semplici nella directory corrente sono scrivibili dal proprietario, non hanno permessi per il gruppo, e il cui nome termina con `.log`.

32. Scrivere un comando per ottenere l'elenco delle directory all'interno di `/var/log` che iniziano con una lettera maiuscola e per le quali il proprietario ha permessi di lettura e scrittura.

33. Scrivere un comando per creare nella home directory dell'utente loggato un file contenente i nomi delle directory presenti in `/tmp` il cui nome inizia con `temp` e che siano leggibili da tutti gli utenti.

34. Scrivere un comando per salvare nel file `accessi.txt`, presente nella home directory dell'utente loggato, tutte le righe del file `auth.log` che contengono almeno tre indirizzi IP.

35. Scrivere un comando per elencare i file nella home directory che sono leggibili dal gruppo e il cui nome contiene esattamente 6 caratteri, dei quali il terzo è una cifra.

36. Scrivere un comando per cercare all'interno della directory `/etc` i file il cui nome termina con `.conf` e che siano scrivibili solo dal proprietario. L'output deve essere salvato nel file `conf_files.txt` nella home directory.