

La concorrenza è la situazione in cui più processi gestiti da un SO vengono eseguiti contemporaneamente, spesso con condivisione di risorse. Questa modalità di esecuzione può portare, se non gestita correttamente, a diversi problemi, come la race condition (situazione in cui la manipolazione delle risorse condivise risente dell'ordine di accesso dei vari processi, col rischio di sovrascrizione o manipolazione errata, non controllata), la starvation (situazione in cui un processo non riesce ad ottenere delle risorse di cui ha bisogno per continuare nella sua esecuzione, nonostante il suo stato di readiness, e rimane quindi bloccato ad un certo punto della sua esecuzione), o il deadlock. Il deadlock è la problematica più complessa: si tratta della situazione in cui ogni processo è bloccato, ed è in attesa di essere sbloccato da un altro processo (dalla sua terminazione o dalla fine del suo turno) che a sua volta è bloccato da un altro processo ancora. Ci sono 3 condizioni necessarie per il deadlock, che sono mutual exclusion, hold & wait, no preemption, ed una condizione necessaria e sufficiente, che è la circular wait (che corrisponde alla presenza di un ciclo nel grafo delle attese dei vari processi).

L'unico modo per evitare il deadlock è avere un SO che "vede nel futuro": dovrebbe infatti sapere, prima dell'inizio dell'esecuzione dei processi, se una delle varie allocazioni delle risorse condivise porta a deadlock. È più facile prevenire un deadlock (tramite una prevenzione indiretta, che evita una delle 3 condizioni necessarie, o diretta, che evita la condizione sufficiente) o trovarlo e risolverlo (uccidendo tutti i processi in deadlock, insieme o uno alla volta, o deallocando tutte le risorse condivise, insieme o una alla volta).

La sincronizzazione è la condizione per cui è garantita la concorrenza con la corretta gestione delle risorse condivise tramite un accesso "protetto" alla sezione critica, cioè ad una porzione di programma accessibile con mutua esclusione forzata: accede un solo processo per volta, tale processo non può bloccarsi in CS, non può andare in deadlock né in starvation, nessun processo da fuori può impedirne l'accesso in CS, e questo accesso avrà un tempo finito ma nessuna condizione sulla durata di esso.

La mutua esclusione forzata si ottiene tramite algoritmi, semafori, block/unblock messages, hardware.

Il message passing è utilizzato sia per la inter process communication (sia tra processi indipendenti che cooperanti) che per la sincronizzazione: usa le primitive send e receive, che possono contenere o meno l'indirizzo del mittente o del destinatario, e possono essere bloccanti o meno. I canali usati per il message passing sono le pipe (dispositivo logico monodirezionale che fornisce un descrittore per la lettura e uno per la scrittura. Questi descrittori vanno sempre chiusi per non rischiare deadlock. I processi che sfruttano le pipe per comunicare devono essere parenti) e le fifo (dispositivi logici bidirezionali, con due descrittori e apertura bloccante. Non richiede la parentela).

IOT è un sistema distribuito che riguarda l'interconnessione di dispositivi smart (qualsiasi dispositivo connesso ad internet con capacità di comunicazione), oggetti embedded, cioè piccoli dispositivi con poca potenza di calcolo, funzionalità di base, capacità di connessione ad internet per lo scambio di informazioni.

Alcune componenti dei dispositivi: sensori, attuatori, microcontrollori, ricetrasmettitori. Queste servono a raccogliere informazioni dallo spazio circostante, misurarle e trasformarle dall'analogico al digitale, per essere in grado poi di comunicarle via cavo o wireless.

I sistemi operativi in funzione su questo tipo di dispositivi devono avere memoria limitata, supportare hardware eterogeneo e applicazioni realtime, ottimizzati dal punto di vista della gestione dell'energia.

Un sistema distribuito è una tipologia di sistema informatico costituito da un insieme di processi interconnessi tra loro, in esecuzione su un insieme di calcolatori autonomi connessi fra loro tramite una rete che gli permette di coordinare le proprie attività e di condividere le risorse del sistema in modo che gli utenti percepiscano il sistema come un unico servizio, come se fosse proveniente dall'esecuzione di un singolo programma su una singola macchina.

La comunicazione avviene tramite RPC (Remote Procedure Call), cioè un'estensione del concetto di chiamata di procedura, che consente cioè l'interazione tra procedure di applicazioni diverse oppure che girano su macchine distinte. Le RPC permettono quindi a programmi che girano su macchine distinte di comunicare tramite semplici procedure di call/return. Le RPC possono essere sincrone o asincrone.

Una di queste procedure è il client/server binding (che può essere persistent o non persistent), che consiste nell'apertura di una socket a cui un processo si connette per scambiare informazioni.

TCP e UDP sono due protocolli dello stato di trasporto e si basano sul protocollo IP.

Protocollo IP: è un protocollo per l'indirizzamento di rete e l'instradamento dei pacchetti, senza connessione, con consegna best effort.

Protocollo TCP: è un protocollo orientato alla connessione, con controllo di congestione e di flusso. È più lento dell'UDP ma garantisce la consegna dei segmenti.

Protocollo UDP: è un protocollo non orientato alla connessione, senza garanzia di consegna (non si occupa della ricostruzione della sequenza dei pacchetti ricevuti né della gestione dei duplicati). È usato quando il focus è l'efficienza, più che l'integrità dei dati.

CONCORRENZA → caratteristica dei SO nei quali può verificarsi che un insieme di processi siano in esecuzione nello stesso istante. L'esecuzione contemporanea può condurre a interazioni fra processi se vengono coinvolte risorse condivise.

PROBLEMI

Legati all'uso di una risorsa condivisa

RACE CONDITIONS

RISOLTA CON MUTUA ESCLUSIONE

DEADLOCK: quando tutti i processi sono in attesa di un evento o sbloccato da un altro processo

CONDIZIONI

STARVATION: quando un processo non riesce ad ottenere le risorse nonostante potrebbe farlo

SOLUZIONI

1. MUTUA ESCUSIONE: una risorsa può essere usata da un proc. alla volta
2. HOLD & WAIT: un proc. può trattenere una risorsa mentre aspetta un'altra
3. NO PREEMPTION: nessuna risorsa può essere rimossa da forza se un proc. la trattiene
4. CIRCULAR WAIT: se il grafo delle attese ha cicli c'è deadlock

NECESSARIE MA NON SUFFICIENTI

NECESS. E SUFF.

PREVENIRE

INDIRETTO: DIRETTO:
no una no due
o tutte 3

EVITARE

il sistema deve sapere prima se l'autorizzazione di una risorsa porta deadlock

TROVARLO

riconoscerlo e risolverlo → uccidendo tutti i processi in deadlock
→ uccidendo uno alla volta per trovare la causa
→ deallocando le risorse (preemption) ad un processo sospetto

CONCORRENZA

SINCRONIZZAZIONE

garantisce ingresso e uscita dalla sez. critica con

MUTUA ESCUSIONE FORZATA

1. un solo proc. alla volta
2. nessuno può bloccarsi in sc
3. no deadlock, no starvation
4. no assunzioni su velocità
5. nessuno fuori dalla sc può impedire el'ingr
6. l'accesso alle sc ha un tempo

ALGORITMI

DIJKSTRA BACKERY
no dead si mutual

SEMAFORI

si ottiene con
HARDWARE
COMPARA & SWAP
if awk. hw non
interrompibile
confronto un valore di
mem. con uno del test e
se è = lo sostituisco con
un nuovo valore

BLOCK / UNBLOCK

MESSAGES

→ MESSAGGIO PASSA → meccanismo con cui comuniciamo tramite primitive di send(dist, msg) e receive(src, msg). Per la sinc. si possono usare send e receive bloccanti diverse combinazioni. Un'indirizzamento può essere diretto o indiretto (coda). Nel secondo caso la coda è gestita all'interno del kernel che dev'essere invocato per tutto.

→ PIPE P → la pipe in apertura fornisce un descrittore in lettura e uno in scrittura. Si può andare in deadlock se non si chiudono i descrittori che noi servono. I processi devono essere parenti.

→ FIFO → sono bidirezionali, non no 2 descr. e non c'è bisogno di parentesi. Aperture bloccante

CONCORRENZA

poiché ogni processo ha la sua area di mem. hanno bisogno di comunicare

INTER PROCESS CONUNICAZIONE

PROCESSI COOPERANTI
• condivisione info
• velocità
• modularità

PROCESSI INDEPENDENTI

Usano SHARED MEMORY

non si invoca il kernel per le operazioni ma solo per creare, assegnarlo e rilasciarlo. Si ottiene un puntatore esterno. Se sbaglio qualcosa non me ne accorgo perché non c'è controllo d'errore

INTERNET OF THINGS → è un sistema distribuito

→ COS'E'? → è l'ultimo sviluppo della rivoluzione dei computer e delle comunicazioni.
È l'interconnessione di dispositivi smart cioè qualsiasi oggetto che è connesso ad internet e comunica (es. cluster). La novità è l'utilizzo di oggetti embedded cioè piccoli dispositivi, con poca potente di calcolo, con funzionalità base ma che sono connessi ad internet e riescono a scambiare informazioni possedute, raccolte e/o elaborate.

→ COMPONENTI → sensori: misura gli eventi, ottiene i dati e li fornisce in forma analogica
→ Attuatori: attiva e/o controlla le azioni
→ Microcontrollore: riceve dati dai sensori e comunica i comandi agli attuatori
→ Ricetrasmittitore: serve per comunicare, connessione via cavo o wireless
→ RFID: identifica il dispositivo

→ SISTEMI OPERATIVI → I dispositivi embedded hanno bisogno del loro SO però sono:
si devono sviluppare su piattaforme limitate
perché so portabili → tanti (non posso spendere troppo se ne ho tanti)
ricavati da quelli esistenti (HCLinux, RIOT) → autonomi (hanno le batterie quindi per resistere a lungo hanno poche esigenze)

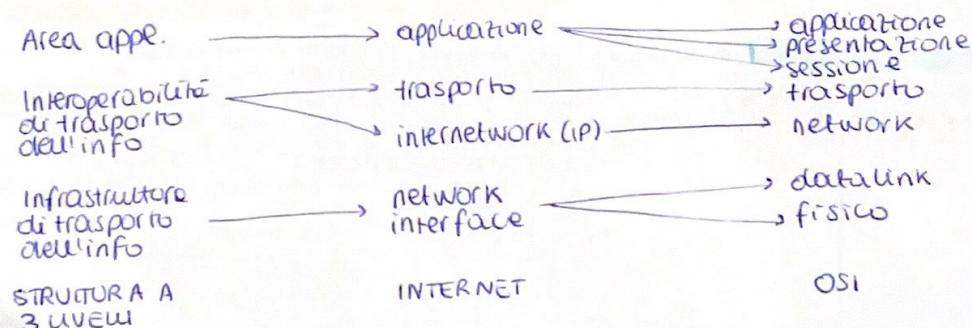
QUA SONO I REQUISITI DEL SO?
- memoria limitata (sia RAM che ROM)
- devono supportare hardwareeterogeneo
- comunicazione
- ottimizzazione attività per gestione energia
- devono supportare app. real time
- devono fornire sicurezza

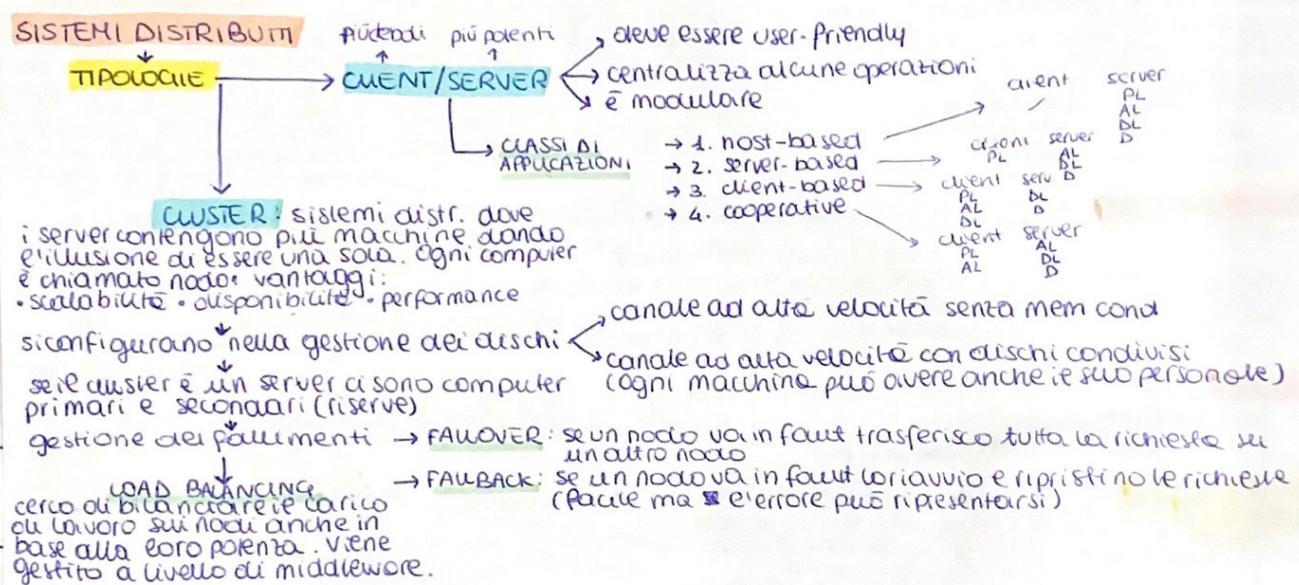
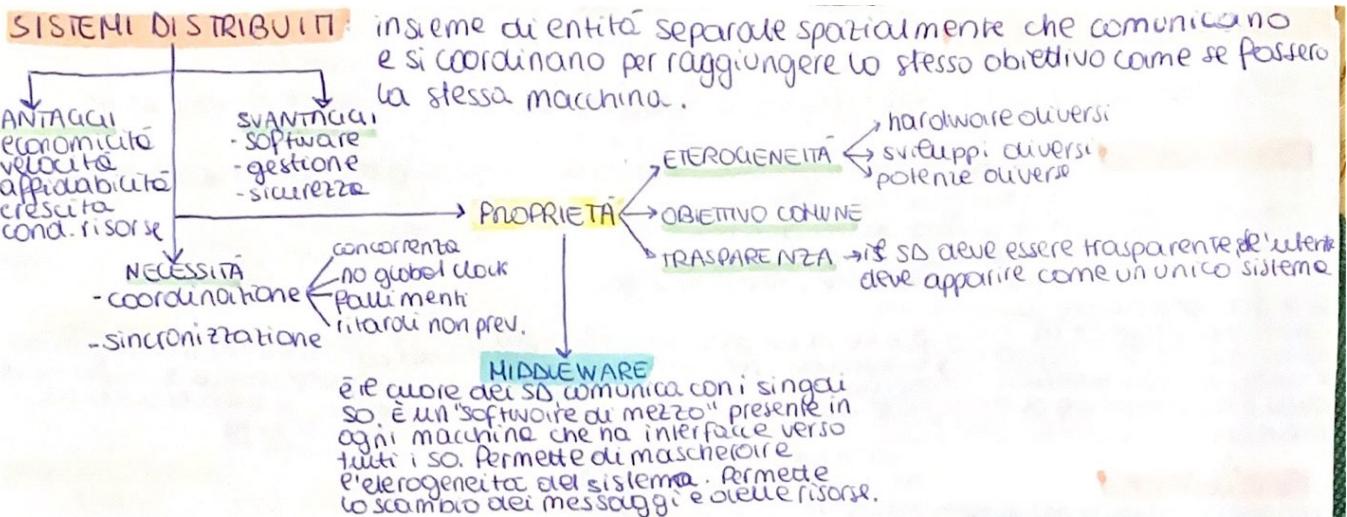
LAN → Local Area Network, copre un'area geografica molto piccola. Ha una velocità elevata. I computer comunicano fra loro tramite access point o switch ethernet.

Più larghe bande.
MAN → Metropolitan Area Network, reti particolari. Hanno un'area più grande delle LAN. Difficili da progettare e mantenere. Costose.

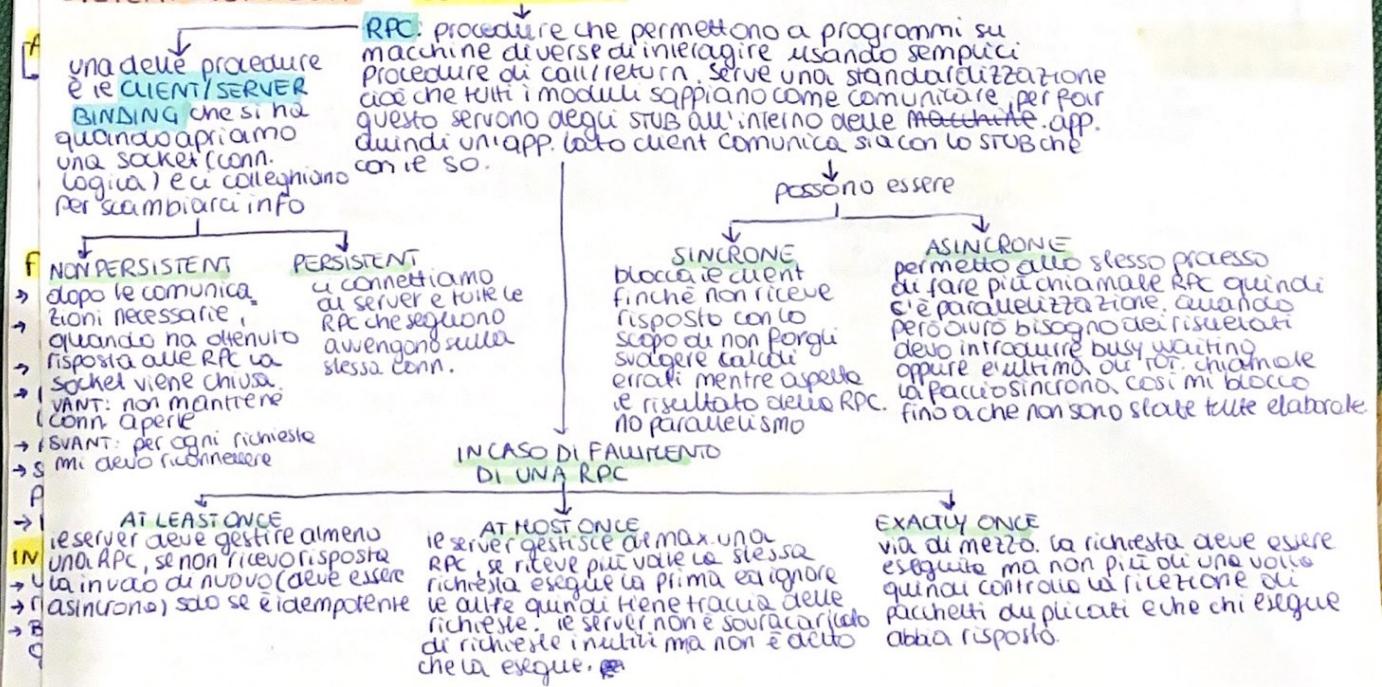
WAN → Wide Area Network, coprono grandi aree. Sembra di avere collegamenti dedicati ma si passa dentro Internet. Può essere di proprietà private. Le grandi aziende affittano i collegamenti usando fibre nere (cioè anche non utilizzate).

Esiste interoperabilità fra diverse reti cioè è possibile far comunicare reti basate su diversi modelli:





SISTEMI DISTRIBUITI → COMUNICAZIONE



IP, TCP, UDP e SOCKET

- TCP e UDP sono 2 servizi dello strato di trasporto e si poggiano sul protocollo IP.

PROTOCOLLO IP

- senza connessione (cioè scelgo volta per volta dove mandare i pacchetti)
- consegna best-effort
- indirizzamento di rete
- insrtadimento pacchetti
- frammentazione e riassiembaggio

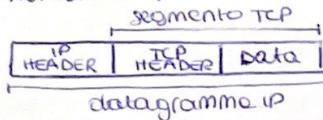
Se il prot. IP si occupa di identificare quale è la macchina che deve ricevere le date, il TCP identifica quale è il processo che deve riceverle

PROTOCOLLO TCP

- orientato alla connessione
- manda segmenti
- controllo di congestione
- controllo di flusso
- più lento dell'UDP
- garantisce che i dati siano ricevuti → si usa se si ha necessità di uno scambio di dati continuo.

Poiché in un processo ci sono più sottoprocessi per capire quale è il destinatario si usano le PORTE cioè punti logici di accesso.

dunque prima di passare al livello IP aggiungo ai dati un header contenente il numero di porta.



Ad ogni servizio si assegnano delle porte su cui si aspetta il riscontro e il server ascolta su alcune porte.

PROTOCOLLO UDP

- non orientato alla connessione (non cerco prima di stabilire una conn. con il server)
- non c'è garanzia di consegna
- non si occupa di riordinare la sequenza
- non sa se ci sono duplicati
- serve quando non necessita di effettuare più che di consegna dei dati

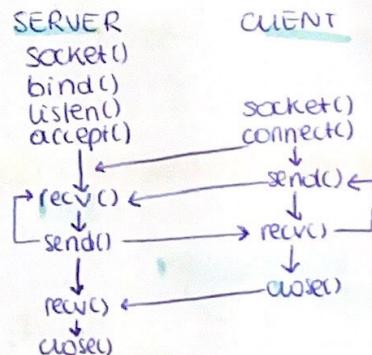
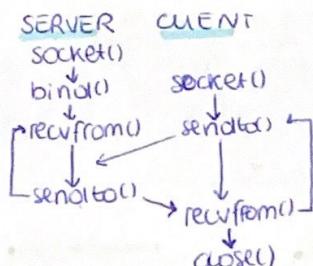
SOCKET

- permette la comunicazione fra host client e server
- l'interfaccia socket si trova fra il livello di app. e il liv. di trasporto
- sono composte da IP e porta

STREAM SOCKET BASATE SU TCP

- il server attende passivamente conn. ed esp.
- il client decide attivamente di stabilire una connessione.
- la recv() è bloccante
- la bind() assegna IP e porta

DATAGRAM SOCKET BASATE SU UDP



SICUREZZA

- Confidentialità: garantire che nessun altro abbia accesso a info conf.
- Integrità: no modifica o distruzione dei dati
- Disponibilità: poter accedere a risorse come dovremmo
- Autenticità: dimostrare identità e paternità dei dati
- Accountability: rintracciare chi è responsabile di cose

TIPI DI ATTACCO

- PASSIVO legge
- ATTIVO
 - Replay
 - Itasquerade
 - modifica
 - distrugge
 - genera

ATTACANTI

- HACKER
 - Black hat
 - white hat
- INTERNO
- GRUPPI CRIMINALI
- ADVANCED PERSISTANT THREAT

DENIAL-OF-SERVICE: si fa consumando risorse. Classico attacco è saturare le reti con richieste TCP, UDP... c'sono 3 linee di difesa: preventione, identificazione e traceback.

- bloccando indirizzi sospetti
- limitando traffico in upstream per alcuni pacch.
- usare cookies
- bloccando i broadcast
- usare puzzle per distinguere umani e non umani
- usare server replicati

MECCANISMI DI ATTACCO

BUFFER OVERFLOW
gestione sbagliata di un buffer che permette di scrivere più bytes del possibile sarà scrivendo a uscire lo stack. L'attaccante sovrascrive il buffer con codice malevolo e modifica la return address. Difesa: compile-time defenses, stack protection mechanism, canarini, DEP/Nx bit

MALWARE: software maligno per compromettere e sfruttare le risorse di un sistema

1. BACKDOOR: punti di accesso segreti
2. CAVALLI DI TROY: codice dannoso in programmi utili
3. PLATFORM INDEPENDENT CODE: malware funzionanti su ogni sistema
4. VIRUS: software che infetta i programmi e si diffondono composto da: meccanismo di infestazione, un trigger e payload
5. MALWARE MULTIPLE-THREAT: diversi veicoli di infestazione
6. ROOTKIT: programmi installati per mantenere l'accesso alle ammin.
7. BOTS: risiede nel nostro PC, comunica con la botnet, è privo di intelligenza ed è telecomandato, composto da: bot, controllo remoto e meccanismo di diffusione. Può fare: attacco distribuito DOS, spamming, sniffing traffic, keylogging, diffusione...

SICUREZZA → TECNICHE DI PROTEZIONE

AUTENTICAZIONE

→ username + pass
(pass = hash + salt)
"qualcosa che so"

→ token: es. smart card
"qualcosa che posso"

→ biometrica: carat. univoca epoca vari. nel tempo
"qualcosa che sono"

CRIPCRAFIA: numeri casuali, 2 meccanis.

ACCESS CONTROL: gestisce cosa possiamo fare una volta entrati nel sistema.

1. DAC: si stabilisce, per ogni utente e ogni oggetto cosa l'utente può fare con quell'oggetto
2. NAC: ogni utente ha i permessi riguardo le label che sono sugli oggetti
3. RBAC: la label è sull'utente.

SYMMETRIC ENCRYPTION: abbiamo input chiave, chi riceve ha la chiave e decifra dato cifrato stessa dimensione di prima addeveranno essere molte chiavi diverse

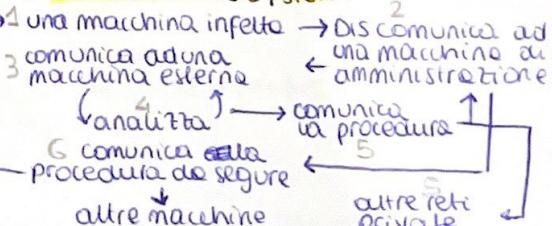
PUBLIC KEY ENCRYPTION: cifratura efficiente decaffatura onerosa. Cifra con chiave pubblica del ricevente che decifra con chiave privata.

Si usa per l'**AUTENTICAZIONE DEL MESSAGGIO**.
cifra con la sua chiave privata e il ricevente capisce che il mess. è chiave di essere se riesce a decaffare con chiave pubblica.

ALGORITMO MAC: calcola il MAC, e lo aggiunge al messaggio e trasmette. Chi riceve separa il MAC e se conosce la chiave verifica se misurie. Un meccanismo famoso è l'**HASH**.

- mess + hash con chiave simm.
- mess + hash con chiave assimm.
- hash (segreto + mess) + mess

DIGITAL IMMUNE SYSTEM:



HONEY POT

- sistema che simula la rete
- gli attacchi finiscono prima qui
- monitorato

ANTIVIRUS → cura → non ottimizzato per i SD

→ detection, identification and removal.
Controlla, se trova virus lo identifica (deve conoscere bene le famiglie) e ottiene la procedura da seguire per rimuoverlo. Contiene al suo interno un emulation control model, un emulatore della CPU e uno scanner delle signature così da eseguire i file nell'emulatore della macchina.

FIREWALL → prevenire

- punto di controllo → + accessi → + firewall
- non fa analisi del dato ma della connessione, il pericolo è l'utente finale
- è messo fra la rete esterna e quella interna
- Packet filtering firewall: scatta tutto - trasmetti tutto. Livello di trasporto
- Application Proxy firewall: a livello applicativo
- svantaggi: no prevenzione attacchi a vulnerabilità, polo logging, vulnerabile ai bug di TCP/IP
- In un SD conviene avere 2 firewall, interno ed esterno.

INTRUSION DETECTION SYSTEM → prevenire

- utente normale diverso da molesto/nostante
- riconosce retrofficio malevolo
- Buono in un SD con agenzie IDS in ogni rete, nodo centrale e gestione decentralizzata

SICUREZZA

①

- La sicurezza è un processo, niente è al 100% sicuro
- La sicurezza di un sistema dipende dalle sue componenti meno sicure (spesso è l'utente finale)
- Non spiegare come funzionano gli algoritmi non le rende più sicuri
- La criptografia è il miglior elemento ma non è sufficiente

Cosa significa sicurezza?

- > È la protezione effettuata ad un sistema di informazione automatizzato per ottenere obiettivi applicabili per preservare integrità, disponibilità e confidenzialità dell'informazione delle risorse del sistema.
- > La sicurezza non riguarda solo i dati ma hardware, software, firmware e telecomunicazioni.
- * TRIADE CIA (confidenzialità, integrità, disponibilità)
 - perdere confidenzialità vuol dire che qualcuno ha avuto un accesso non autorizzato a delle informazioni riservate. GARANTIRE CONFIDENTIALITÀ = garantire che nessun altro abbia accesso a info. confidenziali
 - perdere l'integrità comporta la modifica o la distruzione di operazioni
 - perdere disponibilità vuol dire che si perde la possibilità di accedere a risorse a cui dovremmo poter accedere

CONFIDENTIALITÀ

- dati confidentiali: info private che non devono essere accessibili
- PRIVACY: scegliere l'utente cosa è privato e cosa no

Concetti addizionali:

- autenticità: non solo i dati devono essere integri ma posso dimostrare di averli generati. È anche la possibilità di dimostrare che gli utenti sono chi dicono di essere.
- accountabilità: rintracciare chi è responsabile di cosa. Bisogna tener traccia delle attività fatte perché non me ne accorgo sul momento.

TIPI DI ATTACCO

- > attacco PASSIVO: l'attaccante legge le informazioni, non interagisce
- > attacco ATTIVO: l'attaccante legge, modifica, distrugge e genera info. Gli attacchi passivi sono più difficili da identificare, possiamo fare prevenzione. Gli attacchi attivi si dividono in 4 categorie:
 1. Replay: catturano i dati e li riutilizzano più avanti per produrre un effetto non autorizzato
 2. Masquerade: cercano di prendere il posto di un altro, persone
 3. Modifiche del messaggio: sono sulla linea, catturano messaggi, li modificano e li inoltrano
 4. Negare il servizio

ATTACCANTI:

1. HACKER: in genere lo fa per il gusto di farlo e non per soldi e per prestigio
 - BLACK HAT: malintenzionati
 - WHITE HAT: avvertimento e segnalazione
2. INTERNO: persone che lavorano o lavoravano lì, es. lavoratore che viene assunto da azienda concorrente.
3. GRUPPI CRIMINALI: gruppi organizzati, si scambiano info e si coordinano. Hanno in genere dei target e cercano di essere rapaci.
4. ADVANCED PERSISTENT THREAT: gruppi sponsorizzati dalle nazioni (es. gruppo militare cibernetico) fanno attacchi continui per raggiungere l'obiettivo

COS'È IL MALWARE?

Ha lo scopo di causare il danno o ottenere ~~nuove~~ risorse. Spesso viene nascosto.
Ci sono tanti tipi di malware:

(2)

1. BACKDOOR: non sono necessariamente malware. Sono punti di accesso segreti a programmi, può essere lo stesso programmatore a lasciarli. Sono usati anche per attacchi. È difficile identificare la presenza.
2. CAVALI DI TROIA: programmi o comandoli apparentemente utili che contengono codice che eseguito provoca qualcosa di dannoso. I cavalli di troia possono:
 - continuare a fare quello che faceva il programma originale + altre attività
 - continuare a fare quello che faceva il programma ma in modo diverso per nascondere attività malevole
 - fare qualcosa di diverso dal programma originale
3. PLATFORM INDEPENDENT CODE: malware che funzionano su qualsiasi sistema e quindi sono altamente dannosi
4. VIRUS: software che infettano altri programmi modificandoli. Contengono codice che vengono viene trasferito all'interno del programma. Può diffondersi. Un virus è composto da 3 parti: un meccanismo di infezione (modem ou diffusion), un trigger (quello che l'attiva) e un payload (parte di codice che deve essere esattamente copiate). I virus vanno ad infettare le bootsector (quando in fase di avvio), i file o le macro.
5. MALWARE MULTIPLE-THREAT: usano diversi veicoli di infezione.
6. ROOTKIT: è un insieme di programmi che vengono installati per mantenere l'accesso da amministratore (può essere anche benevolo). I rootkit possono essere persistenti quando vengono riavviati ogni volta che vengono riavviati i sistemi; ~~oppure sono~~ memory based, user mode o kernel mode.

BUFFER OVERFLOW

È un comune meccanismo di attacco. È dovuto ad una gestione sbagliata di un buffer che permette di scrivere più byte del possibile sovrascrivendo celle già usate oppure usando lo stack. L'attaccante deve analizzare il software per trovare la vulnerabilità. Sovrascrivono il buffer con il codice dell'attacco e alterano il return address affinché parta dal punto del codice (spesso apre una shellcode). Più il linguaggio è di basso livello più è facile manipolare la memoria. Abbiamo 2 meccanismi di difesa:

- Compile-time defenses: sistemi introdotti nei compilatori.
- Stack protection mechanisms: utilizzare funzioni che considerino la lunghezza del buffer come strcpy
- Canarini: cercare di identificare un buffer overflow sul return address. La funzione identifica, tramite una variabile, uno bufferoverflow perché quando controllerà la mia variabile la troverà modificata dal momento che ~~non~~ un attaccante non può evitare di modificare anche il canarino.
- DEP/Nx bit: l'idea è quella di tenere separate l'area dati dall'area dell'eseguibile. Dunque anche se inserisco codice in memoria non riesco ad eseguirlo direttamente dall'area dati.

7. BOTS:

È un programma che comunica con altri in internet e risiede nel nostro computer. Viene chiamato anche zombie perché sta fermo, poi agisce ed è privo di intelligenza oppure viene chiamato drone perché è telecomandato. Un bot viene posizionato nel computer ma è utile quando si crea un botnet cioè una rete di bot che agiscono in modo coordinato. Ogni botnet è composto da: bot, controllo remoto e meccanismo di spreading (per la diffusione). L'obiettivo iniziale è estendere la rete infettando più dispositivi possibili. Gli elementi chiave sono: un software che possa effettuare l'attacco, sfruttare una vulnerabilità presente in molti sistemi e avere delle strategie per localizzare altri dispositivi con vari criteri. I bots possono fare tante cose:

- attacco distributed denial-of-service: impedire ad altri utenti di accedere ai servizi
- spamming: invio di mail
- sniffing traffic: ottenere info sui traffico

- keylogging: catturare quello che viene digitato (username, password)
- diffusione di un nuovo malware
- installare add-ons pubblicitari nei browser
- attaccare chat
- manipolare poll e giochi

(3)

Uno degli attacchi principali è il denial-of-service cioè impedire l'uso di reti, sistemi o applicazioni dai utenti autorizzati consumando risorse. Attaccare i computer in questo modo non è molto efficace, lo è più attaccare direttamente i server. Classico attacco è saturare la rete tramite richieste di connessioni TCP, UDP (ICMP, UDP SYN).

Come ci si può difendere?

Cercando di evitare picchi di traffico che però potrebbero essere leggibili dunque ci sono 3 linee di difesa: prevention, identificazione di denial-of-service e filtro del traffico, creare un feedback allo scopo di far partire azioni legate.

Come si fa PREVENZIONE?

- Bloccando indirizzi spoofed e da cui sono già partiti attacchi
- Limitando il traffico in upstream per specifici pacchetti (ICMP, TCP, UDP)
- usare i cookies
- bloccando IP broadcast
- bloccando servizi e combinazioni sospetti
- utilizzanolo puzzle per distinguere umani e non umani
- utilizzando server replicati

Come si risponde agli attacchi?

Servono le coinvolgimento degli ISP che devono impostare filtri in uscita. Si deve catturare e analizzare i pacchetti, identificare i bug usati.

TECNICHE DI PREVENZIONE

La tecnica più usata è la criptografia:

la base di tutto sono i numeri casuali che si ottengono con algoritmi oppure tramite eventi catturabili ad esempio con sensori. Alcuni algoritmi sono: meccanismi sono:

- Symmetric Encryption: abbiamo un input e una chiave, chi riceve ha la chiave e quindi deifra. Il dato cifrato ha la stessa dimensione del dato originale. È veloce. Il problema è che mittente e ricevente devono avere la chiave e deve essere diversa per ogni ricevente. Si può attaccare facendo criptoanalisi per ricostruire la chiave oppure con brute-force cioè provando tutte le chiavi (più grande la chiave più è difficile)
- Public Key Encryption: cifratura efficiente, deifratura onerosa. L'algoritmo cifra con la chiave pubblica del ricevente che deifra con una chiave privata (modalità asimmetrica). Diversi algoritmi usano questo meccanismo

Le chiavi asimmetriche sono usate anche per l'AUTENTICAZIONE: in questo caso quando voglio trasmettere qualcosa non lo voglio proteggere in modo che solo il mittente possa aprirlo ma chiunque nel mezzo potrebbe vederlo. Il mittente cifra con la sua chiave privata e il ricevente capisce che il mittente è chi dice di essere se riesce a decifrarlo con la sua chiave pubblica. L'autenticazione serve per proteggersi dai attaccanti attivi e per verificare che il messaggio è autentico. Ci sono vari metodi:

1. Algoritmo MAC:

Dato un messaggio e una chiave calcola una specie di cifrato con la caratteristica che l'output ha una certa dimensione indipendente dalla quelle del mess. orig. Calcolato il MAC lo aggiungo al messaggio e trasmetto, chi riceve separa il MAC dal resto del messaggio e se conosce la chiave verifica che il messaggio l'ho mandato io. Serve a 2 scopi: autenticare e verificare che non è stato modificato. Il messaggio qui è in chiaro. Un MAC famoso è le SWEET HASH FUNCTIONS che dato un messaggio di dim. M genera dei valori hash h che, qualora il mess. venisse modificato, cambia molti dei bit. Modi per usare l'hash:

- Mess. + hash crittografato con chiave simmetrica
- Mess. + hash crittografato con chiave asimmetrica
- Valore segreto + ~~Mess.~~ Mess., calcolo l'hash del mess+segreto e trasmetto mess+hash, se il ricevente conosce il segreto deifra

Un altro meccanismo MAC sono i certificati. Si prende il messaggio, si fa l'hash che viene cifrata con una chiave privata che nel certificato di autenticità (non generata da me) è trasmessa. Il ricevente può decifrare con una chiave pubblica sempre richiesta dall'autorità certificante. Lo usano ad esempio le grandi aziende per aggiornare i dispositivi.

S'arriva alle digital envelopes che mette insieme le tecniche.

Per autenticare noi stessi (e non il messaggio) ci sono vari metodi:

- **username e password**: la password non deve essere memorizzata in chiaro da un server. Non si memorizza la password ma un suo hash con il SALT cioè un numero casuale memorizzato in chiaro. Per verificare la password confronto l'hash code con l'hash della salt e password.
- **TOKEN**: non mi identifico con qualcosa che so ma con qualcosa che posiedo, cioè una "carta". All'inizio si usava una memory card che non poteva ~~calcolare~~ elaborare dati. Ora si usa una smart card (con processori embedded). Servono per entrambi i lettori appositi.
- **autenticazione biometrica**: mi identifico con qualcosa che sono. Es. impronta digitale. Sono caratteristiche univoci e devono avere poca variabilità nel tempo.

ACCESS CONTROL (controllo dei permessi)

È la parte di sicurezza che gestisce cosa possiamo fare dopo, cioè una volta entrati nel sistema. Abbiamo 3 tipi di controllo dell'accesso:

1. Discretionary Access Control (DAC): si stabilisce, per ogni utente e per ogni oggetto, cosa l'utente può fare con quell'oggetto.
2. Mandatory Access Control (MAC): è un sistema di controllo basato sui label, cioè ogni utente ha i permessi riguardo alle label. (la label è sull'oggetto)
3. Role-based access control (RBAC): non si mette la label all'oggetto ma alla persona.

Ogni sistema deve avere almeno uno dei tre ma può usarne più di uno. L'access control è una matrice in cui abbiamo una riga per l'utente e una per l'oggetto in cui negli oggetti ci sono disci, processi, file ecc...

Nel RBAC l'idea è catalogare gli utenti e poi dare loro i permessi ~~riguardo~~ tramite i gruppi. Nella matrice non abbiamo più gli utenti ma i ruoli e abbiamo un'altra matrice ~~che dice~~ che dice qual è il ruolo di ogni utente.

ANTIVIRUS

L'antivirus fa detection, identification e removal. Controlla se vi è una compromissione e se riconosce qualcosa che non va cerca di identificare il tipo di virus per poi rimuoverlo insieme a tutte le sue attività ma per fare questo deve conoscere bene il virus.

Fortunatamente i virus appartengono a delle famiglie quindi una volta riconosciuta la famiglia ~~l'~~ antivirus (che ormai è online) ottiene le info sul virus e sui procedimenti da prendere. Il virus però potrebbe essere voluto dall'utente (un crack) quindi, una volta che l'antivirus lo ha messo in quarantena, deve abilitarlo. L'antivirus, per non eseguire i file, contiene al suo interno un emulatore semplice della macchina stessa dove viene decifrato ed eseguito il file. Quindi un antivirus contiene un emulation control model, un emulatore della CPU e uno scanner della signatur (per riconoscere le caratteristiche delle famiglie di virus).

L'antivirus non è ottimizzato per i sistemi distribuiti per i quali si usano i Digital Immune System. Nel momento in cui si infetta una macchina del sistema avrà una componente del DIS che comunica ad una macchina di amministrazione la presenza del virus e questa lo comunica ad una macchina esterna che fa l'analisi completa del virus. E seguendo l'analisi, la macchia esterna comunica a quelle di amministrazione le procedure da seguire e questa lo comunica alla macchina infettata e anche alle altre macchine che potrebbero già essere infette. La macchina esterna comunica ~~le~~ de farsi anche alle altre reti private (che potrebbero essere di altre aziende) in modo da ricevere a sua volta segnalazioni su altri virus da queste in futuro. Dunque c'è ~~cooperazione~~ cooperazione fra aziende.

FIREWALLS

L'antivirus è la cura ma il firewall è ~~la cura~~ il vaccino. Il firewall protegge la rete bloccando dei dati, è un singolo punto di controllo (un muro) infatti se abbiamo più accessi dobbiamo mettere più firewalls. fa monitoring ma a sono comunque attacchi che riescono ad aggirarlo ad esempio quelli all'utente finale (es. virus per email) perché il firewall non fa analisi del dato ma della connessione, controlla chi manda a chi, blocca connessioni non autorizzate ecc...

Dunque l'idea è quella di piazzare un firewall fra la rete esterna e quella interna. Ci sono vari tipi di firewall, quello più semplice è il packet filtering firewall che lavora al livello di trasporto e si basa sui coppie TCP-IP. Ci sono firewall più avanzati come l'Application proxy firewall che lavora al livello applicativo.

Come opera il packet filtering firewall? Controlla ~~che~~ l'IP di sorgente e destinazione della porta, ~~e~~ il protocollo e l'interfaccia e in base a delle regole decide se il pacchetto deve essere scartato. Ci sono 2 politiche di default per decidere come operi il firewall:

- scarica tutto: devo specificare cosa per cosa se può passare uno
- trasmetti tutto: a meno che non sia esplicitamente proibito (più facile)

Qui svantaggi dei firewall sono:

- non possono prevenire attacchi basati su vulnerabilità
- fanno poco logging (quindi non è facile poi fare l'analisi)
- sono vulnerabili agli attacchi sui bug dei protocolli TCP/IP

Quando siamo in un sistema distribuito conviene avere un firewall che separa le reti interne da tutto ciò che è esterno, ~~e~~ ~~è~~ una ~~a~~ zona DMZ network dove ci sono cose accessibili sia dall'interno che dall'esterno (quindi protetta da un firewall esterno) e una zona accessibile solo dall'interno (protetta da un altro firewall interno).

INTRUSION DETECTION SYSTEM

L'idea è quella di riuscire a riconoscere traffico malevolo, bloccare logici non autorizzati, bloccare l'abuso di privilegi. Questi sistemi hanno tecniche avanzate e si basano sul fatto che un intruso si comporta diversamente rispetto ad un utente normale.

Un sistema funziona bene quando è distribuito dove abbiamo ogni rete con il suo ~~agente~~ agente dell'IDS, un nodo centrale che fa da monitor e colleziona i dati per l'intera rete e qualcuno che lo gestirà a livello decentralizzato.

HONEY POTS

È un sistema finto che simula la mia rete. Se arriva un attacco prima di riuscire a passare il firewall finisce nell'honeypot e lì si diffondono però compromessi sistemi che sono finti. L'esperto di sicurezza monitora l'attività sulle honeypots e riesce ad anticipare quello che potrebbe succedere.

MULTI THREADS

- Thread → semafori unnamed
- Se so quanti thread devo creare:

```
pthread_t threads[num_thread];
thread_args_t args[num_thread]; → se lo metto nel for devo fare:
for(i=0; i<num_thread; i++) {
    args[i].campo = valore campo;
    ...
    ret = pthread_create(&threads[i], NULL, funzione, &args[i]);
    if(ret) handler_error("errore");
}
} → args
se devo attendere:
for(i=0; i<num_thread; i++) {
    ret = pthread_join(threads[i], NULL);
    if(ret) handler_error("errore");
}
```

PIPE MULTIPROCESS

- metto la variabile globale pipefd[2]; - fd[0] lettura, fd[1] scrittura
- scrittura e lettura come al solito
- processi → semafori named

ret = close(pipefd[n]) → chiudo quello che non ti serve
- fai cose -
ret = close(pipefd[n]) → chiudi quello che hai usato

creare processo:

```
1) pid_t pid = fork();
if(pid<0) handler_error(..);
if(pid==0){ // figlio
    ...
    _exit(0);
} else{ // padre
    ...
}
ret = wait(0);
```

```
n) pid_t pid;
for(i=0; i<num_proc; i++){
    pid = fork();
    if(pid<0) handler_error(..);
    if(pid==0){ // figlio
        ...
        _exit(0);
    } else{ // padre
        ...
    }
    if(int status;
    for(i=0; i<num_proc; i++){
        ret = wait(&status);
    }
```

LIFO e FIFO

LIFO : inserimento

read_id = write_id

write_id = (write_id + 1) * BUFFER_SIZE

LIFO : estrazione

vegli[read_id]

write_id = read_id

read_id --;

SOCKET

SERVER

```
socket_desc = socket(AF_INET, SOCK_STREAM, 0)
if (socket_desc == -1) handle_error("...");

ret = bind(socket_desc, (struct sockaddr*)&server_addr, sockaddr_len);
if (ret == -1) handle_error("...");

ret = listen(socket_desc, MAX_CONN_QUEUE);
if (ret == -1) handle_error("...");
```

```
client_desc = accept(socket_desc, (struct sockaddr*)&client_addr, (socklen_t*)&socket);
if (client_desc == -1) handle_error("...");
```

CIENT

```
socket_desc = socket(AF_INET, SOCK_STREAM, 0);
if (socket_desc == -1) handle_error("...");

ret = connect(socket_desc, (struct sockaddr*)&server_addr, sockaddr_len);
if (ret == -1) handle_error("...");
```

PRODUTTORE-CONSUMATORE

semafori : e, n, prod, cons.

1) CASO N-M

// produttore	// consumatore
wait(&e)	wait(&n)
wait(&prod)	wait(&cons)
...	...
post(&prod)	post(&cons)
post(&n)	post(&e)

2) CASO N-1

// produttore	// consumatore
wait(&e)	wait(&n)
wait(&prod)	...
...	...
post(&prod)	post(&e)

3) CASO 1-M

// produttore	// consumatore
wait(&e)	wait(&n)
...	wait(&cons)
...	...
post(&n)	post(&cons)
...	post(&e)

4) CASO 1-1

// produttore	// consumatore
wait(&e)	wait(&n)
...	...
...	...
post(&n)	post(&e)

SHARED MEMORY

CIENT

```
fd_shm = shm_open(NAME, O_RDWR | O_EXCL, 0666);
if (fd_shm == -1) handle_error("...");

struct shared_memory* ptr;
ptr = (struct shared_memory*) mmap(0, sizeof(struct shared_memory), PROT_READ |
PROT_WRITE, MAP_SHARED, fd_shm, 0);
if (ptr == MAP_FAILED) handle_error("...");

ret = unmap(ptr, sizeof(struct shared_memory));
if (ret == -1) handle_error("...");

ret = close(fd_shm);
if (ret < 0) handle_error("...");
```

SERVER

```
shm_unlink(NAME)
```

```
fd_shm = shm_open(NAME, O_RDWR | O_CREAT | O_EXCL, 0666),
ret = ftruncate(fd_shm, sizeof(struct shared_memory));
memset(ptr, 0, sizeof(struct shared_memory));
```

```
shm_unlink(NAME)
```

BACKERY

```

// dichiarazione delle variabili globali comuni
bool sceglie[N] = {false}; // N costante numero di processi
int numero[N] = {0};      ↳ SE È NELLA FASE DI SCELTA (FASE 1)
                           ↳ REGISTRA I NUMERI PRESI DA OGNI PROCESSO

int i; // indice del thread in esecuzione
// ...
while (numero[i] == 0) { → SE NON HO SCELTO IL NUMERO ENTRA NELLA FASE DI SCELTA
    //prendo il numero
    sceglie[i] = true; → SEGNALE DI ESSERE NELLA FASE DI SCELTA
    numero[i] = 1 + max(numero[0], numero[1], ..., numero[N - 1]); //il più grande tra tutti i numeri in possesso
    sceglie[i] = false; //ho preso il numero → NUMERO SCELTO!
    for j in 1 .. N except i { ↳ SE I PASSI LE ATTESI PRECEDENTI: ALLEDE A SEZIONE CRITICA
        while (sceglie[j] == true); //aspetto la scelta di altri
        if (numero[j] != 0 && (numero[j], j) < (numero[i], i)); //controllo di avere il numero più basso, dopo i vari zeri
    } → VERRIFICO DI ESISTERE IL PROSSIMO
    ↳ SE I PASSI LE ATTESI PRECEDENTI: ALLEDE A SEZIONE CRITICA
    // <sezione critica>
    numero[i] = 0; → RISETTA IL NUMERO A ZERO, COSÌ PUÒ RISLEGGERE E RIMETTERSI IN ATTESA
    // <sezione non critica>
}

```

SELEZIA DEL NUMERO

FASE DI ATTESA

IL TUO TURNO SEZIONE CRITICA

- a) SCELTA DEL NUMERO:
 1. SEGNALE INIZIO
 2. SCELTA
 3. SEGNALE FINE
- b) ATTESA:
 1. ATTENDO LA SCELTA DI OGNI PROCESSO
 2. ATTENDO DI AVERIA IL NUMERO PIÙ PICCOLO
- c) SEZIONE CRITICA:
 1. ACCESSO A RISORSE CONDIVISE IN MUTUA ESCLUSIONE
 2. RESET DELLA SCELTA DEL NUMERO

DJIKSTRA X n processi:

```

/* global storage */
boolean interested[N] = {false, ..., false} → REGISTRA L'INTERESSE DI OGNI ALGORITMO
boolean passed[N] = {false, ..., false} → SE HA PASSATO LA FASE 1 AD OLTRENAZIONE BAVAIA CANTINA
int k = <any> // k ∈ {0, 1, ..., N-1} ↳ PROBLEMA IN ESECUZIONE
/* local info */
int i = <entity ID> // i ∈ {0, 1, ..., N-1} → IL MIO PROBLEMO
1. interested[i] = true → HAPOSTO IL MIO PROBLEMO LORO INTERESSANDO AD OLTRENAZIONE IN CS
2. while (k != i) { → SE K MIO PROCESSO NON È IL PROSSIMO, ATTESA
    3. passed[i] = false → NON HA PASSATO LA FASE 1
    4. if (!interested[k]) then k = i → SE K NON È PIÙ INTERESSATO, TOGLIE IL SUO INTERESSE, DIRETTA K, IL PROBLEMO IN ATTO
    } ↳ QUANDO LA 4. È VERIFICATA PASSA ALLA FASE 2.
    5. passed[i] = true ↳ SEGNALE DI AVER PASSATO FASE 1
    6. for j in 1 ... N except i do → EXCLUDE I: ASTERISCHI COKK (VERIFICA CHE SIA L'UNICO AD ESSERE IN FASE 2)
    7. if (passed[j]) then goto 2 → SE UNO DI QUESTI PROCESSI HA PASSATO LA FASE 1 (QUANDO SI TROVA CONC.)
    8. <critical section> ↳ VERRIFICO DELLE RISORSE
    9. passed[i] = false; interested[i] = false → SEGNALE DI NON ESSERE PIÙ INTERESSATO,
                                               → SEGNALE DI DOVER RITORNARE IN ATTESA.

```

ATTESA

VERIFICA DI ESISTERE CANTINA
IL TUO TURNO SEZIONE CRITICA

- a) ATTESA:
 1. SEGNALE LA NECESSITÀ DELLE RISORSE
 2. ATTENDO CHE K ABbia FINITO
 3. I DIVENTA K, PROBLEMO CORRIBILE
- b) VERIFICA:
 1. SE 3 Oltre A ME UN PROBLEMO CHE HA FINITO IN FASE DI ATTESA, QUINDI SE K IDENTIFICA PIÙ PROCESSI
 - Si ↳ TURNA
 - No
- c) SEZIONE CRITICA:
 1. ACCESSO A RISORSE CONDIVISE IN MUTUA ESCLUSIONE
 2. SEGNALE LA NON NECESSITÀ DI RISORSE
 3. RESET STATO DI ATTESA

DEKKER X 2 processi

```

int me = 0, other = 1; // P0 (flip for P1)

while (true) {
    /*NCS*/
    flag[me] = true; → DEVO PRENDERE LE RISORSE CONDIVISE
    while (flag[other]) { → FINCHE' 'OTHER' VIENE LE RISORSE VERIFICO IL SUO TURNO
        if (turn == other) { → SE È IL TURNO DI OTHER
            flag[me] = false; ↳ NON POSSO ACCEDERE
            ATTESA [2] → while (turn == other) /* busy wait */ ; → TURNO DI OTHER FINISCA
            flag[me] = true; → FINITO IL TURNO DI OTHER
        }
    }
    /* CS */
    turn = other; → DICHIARO IL MIO TURNO FINITO
    flag[me] = false; → NON DEVO USARE PIÙ LE RISORSE
}

```

- PRE: DICHIARO DI VOLERE LE RISORSE
- a) SE OTHER VIENE LE RISORSE ALLORA b) SENZA c)
 - b) ATTENDO CHE IL TURNO DI OTHER TERMINI
 - c) SEZIONE CRITICA