Università di Messina — CdL in Informatica — Reti e Sistemi Distribuiti Mod.B

Tracce di Progetto 2026

Aggiornamento: 21 maggio 2026

Regole per lo svolgimento e la consegna del progetto finale del corso

1. Il progetto deve essere svolto **al massimo da due studenti** (indicare numero progetto, handle github e partner nell'apposito foglio excel sul sito del docente)
2. L'handle github è il nome utente *https://github.com/nomeutente* del vostro profilo
3. Il progetto deve essere consegnato (push su Github) non oltre le **48 ore precedenti** la data d'esame
4. Il progetto deve essere scritto su più file e compilabile tramite **Makefile**
5. Il progetto deve essere accompagnato da un **report descrittivo** in formato PDF
6. Il progetto va presentato appoggiandosi ad una presentazione scritta in LaTeX/beamer
7. Attenzione: le seguenti sono soltanto tracce. Ogni opera di ingegno o di fantasia è graditissima! :D
8. Il docente si riserva, per **insindacabili** motivi didattici, di modificare la traccia scelta dallo studente (è raro, ma succede).
9. E' possibile **proporre** progetti alternativi al docente, che siano coerenti con le finalità didattiche del corso.

#	Progetto	Protocollo	Concetti chiave
1	Secure Chat	TCP	I/O Multiplexing
2	Dropbox Minimale	TCP	chunking, resume
3	Web Server	HTTP/1.0	parsing testuale, MIME, thread
4	Redis Minimale	TCP	threads, rwlock, serializzazione
5	DNS Resolver	UDP	DNS query, caching
6	TCP Proxy Inspector	TCP	proxy, text search
7	Ping & Traceroute	ICMP raw	raw socket
8	`dlqueue`	-	fork + exec, SIGCHLD
9	`procmon`	-	thread per PID, CSV
10	`wordfreq`	-	thread pool, merge

#1 Secure Chat

Una chat multi-utente in cui un server centrale riceve messaggi da N client e li ridistribuisce a tutti gli altri. Il server usa `select() o epoll()` per gestire connessioni multiple in un singolo thread, senza bloccarsi su nessun client. I client possono organizzarsi in stanze indipendenti; il broadcast avviene solo all'interno della stessa stanza.

#2 Dropbox Minimale

Un servizio di file storage in rete che supporta upload, download, listing e cancellazione di file di dimensione arbitraria da più connessioni contemporaneamente. Il trasferimento avviene a *chunk* da 4 KB: il file non viene mai caricato interamente in memoria,

rendendolo scalabile a qualsiasi dimensione. L'header di sessione contiene il digest SHA-256 del file; al termine del download il client verifica l'integrità confrontando il digest calcolato localmente. Una progress bar testuale mostra la percentuale di avanzamento aggiornata ad ogni chunk ricevuto. Se il trasferimento viene interrotto, il client può riprendere dall'offset già scaricato inviando la posizione nell'header di sessione.

#3 Web Server HTTP/1.0

Un server HTTP/1.0 che risponde a richieste reali di browser e `curl`, servendo file statici dalla directory corrente. `select() o epoll()` gestisce più connessioni concorrenti; un thread dedicato riceve le entry di accesso su una coda protetta da mutex e le scrive su file in modo asincrono, senza rallentare il loop principale.

#4 Redis Minimale

Un key-value store in-memory con protocollo testuale ispirato a Redis, accessibile via `telnet`. Il server lancia un thread con `pthread_create` per ogni connessione accettata; ogni thread gestisce l'intera sessione del client e termina alla disconnessione. Il dizionario in RAM è protetto da un rwlock: le operazioni `GET` acquisiscono il lock in lettura e non si bloccano tra loro; `SET` e `DEL` acquisiscono il lock in scrittura. Un thread di background esegue `SAVE` ogni 60 secondi serializzando il dizionario su file; all'avvio il server ricarica automaticamente lo stato.

#5 DNS Resolver

Un resolver DNS (RFC 1035) che costruisce fino ad N query da zero, le invia via UDP alla porta 53 di `8.8.8.8` e parsa la risposta per estrarre gli indirizzi IP. Una cache in-memory con TTL evita di interrogare il server se la risposta è già disponibile e non scaduta

#6 TCP Proxy Inspector

Un intermediario TCP trasparente che si posiziona tra un client e un server, ricevendo tutto il traffico e ritrasmettendolo in entrambe le direzioni. Ogni blocco di dati transitato viene scritto su un file di log con timestamp, direzione (upstream/downstream) e dump esadecimale dei primi 64 byte. Una lista di parole chiave configurabile permette al proxy di bloccare connessioni il cui payload contiene termini proibiti.

#7 Ping & Traceroute

Questo progetto implementa da zero i due strumenti di diagnostica di rete più usati al mondo, usando raw socket ICMP senza librerie. `ping` costruisce un pacchetto ICMP byte per byte, lo invia a un host e misura quanto tempo passa prima che arrivi la risposta; le statistiche finali mostrano RTT minimo, massimo e medio. `traceroute` sfrutta il fatto che ogni pacchetto IP ha un contatore di salti (TTL): inviando pacchetti con TTL 1, 2, 3... ogni router lungo il percorso si rivela rispondendo con un errore. Il calcolo del checksum ICMP viene implementato a mano seguendo la specifica: somma di tutti i byte a coppie con riporto circolare, complemento a uno del risultato. Il task finale implementa un ping sweep: invia richieste in parallelo a tutti i 254 host di una

sottorete e riporta quali sono raggiungibili.

#8 `dlqueue` — download parallelo da lista di URL

Un tool che legge una lista di URL da file (uno per riga) e li scarica in parallelo lanciando al più K processi `wget` simultanei con `fork`/`execvp`. Il processo padre usa `waitpid(WNOHANG)` in un loop guidato da `SIGCHLD` per rilevare i download terminati e lanciare quelli successivi nella lista. Un file di stato (`download.log`) viene aggiornato con esito e timestamp di ogni URL: `OK`, `FAILED` o `SKIP` se il file esiste già. La versione finale aggiunge il retry automatico: gli URL falliti vengono reinseriti in coda con un contatore di tentativi; dopo 3 fallimenti il download viene abbandonato. Il report spiega perché si usa `fork`/`execvp` invece di thread per questo caso d'uso, e qual è il valore ottimale di K su una connessione da 100 Mbit/s.

#9 `procmon` — monitor di risorse per processo

Un tool che campiona ogni secondo CPU e memoria di un insieme di PID passati sulla riga di comando, dedicando un thread a ogni processo monitorato. Ogni thread legge `/proc/PID/stat` e `/proc/PID/status` e calcola la percentuale CPU come delta di `utime+stime` tra due campioni, replicando il calcolo di `top`. I campioni vengono scritti su un file CSV (`timestamp,pid,cpu_pct,rss_kb`) da un thread dedicato che riceve i dati su una coda, mantenendo i thread di lettura liberi dall'I/O su disco. Il segnale `SIGUSR1` stampa su stdout un riassunto (media, min, max, deviazione standard) senza interrompere il monitoraggio in corso.

#10 `wordfreq` — frequenza delle parole su più file

Un tool che conta la frequenza di ogni parola in un insieme di file di testo passati sulla riga di comando, distribuendo i file tra N worker thread. Ogni worker costruisce una tabella hash locale delle parole trovate nel proprio file; al termine una fase di merge (protetta da mutex) fonde tutte le tabelle in quella globale. L'output è la lista delle 20 parole più frequenti in formato `parola: N`, ordinata per frequenza decrescente.