

# Progetto di Laboratorio 2

AA 2021-2022

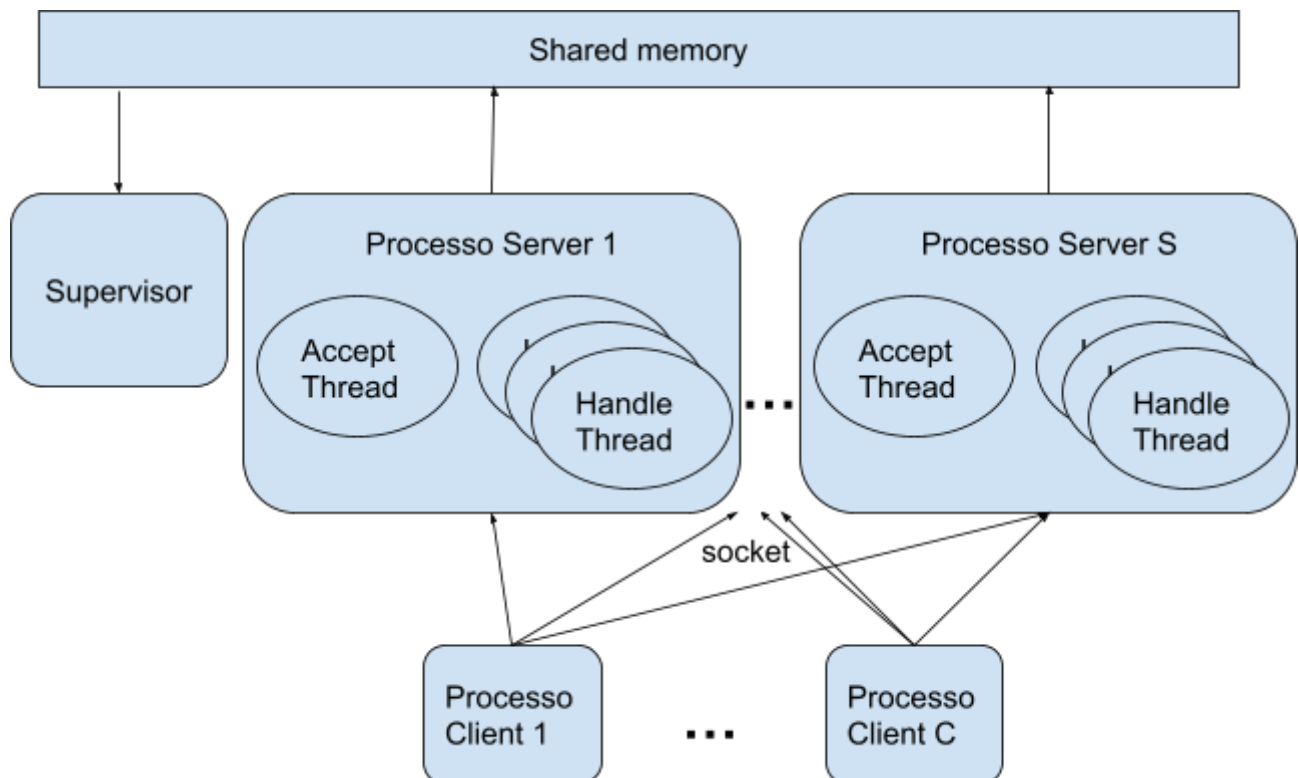
Corso A

## Introduzione

Lo studente dovrà realizzare un sistema per l'*out-of-band signaling*. Si tratta di una tecnica di comunicazione in cui due entità si scambiano informazioni senza trasmettersi direttamente, ma utilizzando segnalazione collaterale: per esempio, il numero di errori "artificiali", o la lunghezza di un pacchetto pieno di dati inutili, o anche il momento esatto delle comunicazioni.

Nel nostro caso, vogliamo realizzare un sistema client-server, in cui i client possiedono un codice segreto (che chiameremo *secret*) e vogliono comunicarlo a un server centrale, senza però trasmetterlo. Lo scopo è rendere difficile intercettare il *secret* a chi stia catturando i dati in transito.

Nel nostro progetto avremo **C** client, **S** server, e 1 supervisor. Viene lanciato per primo il supervisor, con un parametro **S** che indica quanti server vogliamo attivare; il supervisor provvederà dunque a creare i **S** server (che dovranno essere processi distinti). Gli **C** client vengono invece lanciati indipendentemente, ciascuno in tempi diversi. Il funzionamento di ciascuno dei componenti è descritto nel seguito, mentre la figura mostra la struttura generale.



## Il client

All'avvio, il client genera il suo *secret*, che è costituito da un intero su 32 bit fra 1 e 3000, e il suo

ID unico, che è un intero a 64 bit; entrambi devono essere generati pseudo-casualmente (si ricordi di inizializzare il generatore casuale in maniera appropriata). Un client riceve sulla riga di comando tre parametri interi: **S** il numero totale di server, **k** il numero di server a cui questo client si deve connettere, con  $1 \leq k < S$ , e **m** il numero di messaggi da inviare, con  $m > 3k$ . Il parametro **S** dovrà essere lo stesso S passato al supervisor, che controlla quanti server sono attivi sul sistema.

All'inizio, il client deve stampare sul proprio *stdout* un messaggio nel formato "CLIENT *id* SECRET *secret*", in cui *id* è rappresentato da un numero esadecimale (senza prefissi particolari), e *secret* da un numero decimale.

Il client dovrà quindi scegliere casualmente **k** interi distinti compresi fra **1** e **S**, e si collegherà (tramite socket, vedi avanti) ai **k** server corrispondenti agli interi scelti. Inizierà un ciclo in cui, ad ogni iterazione, sceglierà casualmente un server fra i **k** a cui è collegato, gli invierà sulla socket corrispondente un messaggio contenente il proprio ID unico (il formato sarà un buffer di 8 byte contenente l'ID in network byte order), e attenderà *secret* millisecondi prima di ripetere (scegliere un altro server a cui mandare il proprio ID). Si può usare la system call **nanosleep(2)** per l'attesa.

Dovranno essere inviati complessivamente **m** messaggi (in totale, non per ciascun server). Una volta completato il proprio compito, il client termina, dopo aver stampato un messaggio "CLIENT *id* DONE".

## Il server

Ogni server (che viene lanciato dal supervisor) ha assegnato un numero progressivo **i** tra **1** e **S**. Per prima cosa il server apre una socket. Si può usare il dominio AF\_UNIX (si usi come indirizzo la stringa "OOB-server-*i*", in cui *i* è il suo numero progressivo). Alternativamente si può usare il dominio AF\_INET con il numero di porta uguale a **9000+i** dove *i* è il numero progressivo del server. Il server procede quindi ad attendere connessioni da parte dei client, che come detto possono arrivare in qualunque momento. Per ogni connessione, il server dovrà osservare il momento di arrivo dei messaggi da parte del client, e in particolare il tempo trascorso fra messaggi consecutivi. Per come è strutturato il client, il tempo fra due messaggi consecutivi allo stesso server sarà un multiplo del suo *secret*. Compito del server è di stimare il *secret* di ogni client, e – nel momento in cui la connessione viene chiusa da quel client – informare il supervisor di qual'è la sua migliore stima, tramite un segmento di memoria condivisa, passando l'ID del client e la sua stima del *secret*.

Ogni server deve utilizzare 4 thread diversi per gestire le richieste dei client. Un thread accetta le connessioni da client nuovi, e gli altri 3 gestiscono ognuno un sottoinsieme di client.

All'avvio il server deve stampare sul suo *stdout* un messaggio nel formato "SERVER *i* ACTIVE". Per ogni nuova connessione ricevuta, deve stampare un messaggio "SERVER *i* CONNECT FROM CLIENT". Per ogni messaggio ricevuto sulla socket, deve stampare su *stdout* un messaggio "SERVER *i* INCOMING FROM *id* @ *t*", in cui *id* è l'ID del client che ha inviato il messaggio, e *t* è il tempo di arrivo (espresso come un numero, con unità di misura a discrezione). Per ogni chiusura della socket, deve stampare su *stdout* un messaggio "SERVER *i* CLOSING *id* ESTIMATE *s*", in cui *s* è il valore stimato da questo client per il *secret* del client *id*.

Al momento della chiusura di un client, il server (il thread che ha gestito il client) deve comunicare al supervisor la sua migliore stima per il valore del *secret* di *id* (la stessa stampata nel messaggio). La comunicazione fra server e supervisor è realizzata tramite un segmento di memoria condivisa come descritto sotto.

## Il supervisor

All'avvio, il supervisor dovrà stampare sul suo *stdout* un messaggio "SUPERVISOR STARTING S SERVER", in cui **S** è il numero di server da lanciare (che il supervisor riceve come argomento sulla riga di comando). A questo punto, dovrà creare un segmento di memoria condivisa di dimensione almeno 64 + 32 bit (un long + un int) e lanciare (come processi distinti) i **S** server.

Il supervisor utilizza il segmento di memoria condivisa per ricevere le stime dei *secret* dei client da parte dei server. Per ogni nuova stima ricevuta, il supervisor dovrà stampare su *stdout* un messaggio "SUPERVISOR ESTIMATE  $x$  FOR  $id$  FROM  $i$ ", in cui  $x$  è il valore della stima del *secret*,  $id$  è l'ID del client a cui si riferisce la stima, e  $i$  è l'indice del server che ha fornito la stima. La sincronizzazione degli accessi alla memoria condivisa si deve fare tramite semafori POSIX.

Il supervisor riceverà ovviamente stime da più server per lo stesso client. Deve quindi utilizzare le diverse stime arrivate fino a un dato momento per determinare il valore più probabile del *secret* "vero"; chiamiamo questo valore  $X_{id}$ . Quando il supervisor riceve un segnale SIGINT, deve stampare sul suo *stderr* una tabella che mostri le sue stime correnti, in cui ciascuna riga ha il formato "SUPERVISOR ESTIMATE  $X_{id}$  FOR  $id$  BASED ON  $n$ " – in cui  $n$  rappresenta il numero di server che hanno fornito stime per  $id$  fino a quel momento. Se invece riceve un "doppio Ctrl-C", ovvero un secondo segnale SIGINT a distanza di meno di un secondo dal precedente, il supervisor deve terminare, stampando prima la stessa tabella su *stdout* seguita dal messaggio "SUPERVISOR EXITING". All'uscita del supervisor tutti i server devono uscire con un messaggio "SERVER  $i$  EXITING"

## Misurazione

Si dovrà realizzare uno script bash di nome *misura* che, ricevuti come argomenti i nomi di un insieme di file contenenti l'output di supervisor, server e client, ne legga e analizzi i contenuti, e stampi delle statistiche su quanti *secret* sono stati correttamente stimati dal supervisor (intendendo per stima corretta un *secret* stimato con errore entro 25 unità rispetto al valore del *secret* vero), e sull'errore medio di stima.

## Makefile

Il progetto dovrà includere un makefile avente, fra gli altri, i target *all* (per generare tutti gli eseguibili), *clean* (per ripulire la directory di lavoro dai file generati), e *test*. Quest'ultimo deve eseguire un ciclo completo di test, lanciando il supervisor con 8 server, e dopo un'attesa di 2 secondi dovrà lanciare un totale di 20 client, ciascuno con parametri 8 5 20. I client andranno lanciati a coppie, e con un attesa di 1 secondo fra ogni coppia.

Dopo aver lanciato l'ultima coppia, il test dovrà attendere 60 secondi, inviando nel frattempo un SIGINT al supervisor ogni 10 secondi; al termine dovrà inviare un doppio SIGINT, e lanciare lo script *misura* sui risultati raccolti (tramite redirectione dello stdout dei vari componenti su appositi file).

## Note finali

Nel realizzare il sistema oggetto del progetto, si tenga presente che il protocollo di comunicazione fra supervisor e server può essere privato, mentre quello fra server e client è pubblico. In altre parole, il vostro sistema supervisor+server deve funzionare anche se a connettersi sono client diversi da quelli sviluppati da voi. Analogamente, i vostri client dovranno essere in grado di interoperare con supervisor+server sviluppati da altri.

Ci si attende che tutto il codice sviluppato sia conforme POSIX e funzioni sulla macchina di laboratorio laboratorio2.di.unipi.it.

La consegna dovrà avvenire, entro la data e ora dell'appello, attraverso classroom. Per ogni appello ci sarà un assignment nuovo aperto dove potete fare la consegna. Va consegnato un archivio con nome *nome\_cognome*.tar.gz contenente tutto il necessario. Scompattando l'archivio in una directory vuota, dovrà essere possibile eseguire i comandi *make* (con target di default) per costruire tutti gli eseguibili, e *make test* per eseguire il test e vederne i risultati. L'archivio dovrà anche contenere una breve relazione (3-5 pagine) in formato PDF in cui illustrate il vostro progetto.