

**Corso di Sistemi Operativi**  
**A.A. 2023/2024**  
**Arzigogolo 2 – 14 ottobre 2023**  
**Aurora Lin - Matricola 176191**

**Esercizio 1.** Si individui un meccanismo per registrare e riprodurre sessioni di lavoro al terminale. La soluzione ideale registra su file l'intera sessione di lavoro e consente la riproduzione esatta della stessa (inclusi i tempi di attesa tra un comando ed un altro) a partire dal file memorizzato. Non è consentito effettuare riprese multimediali di alcun tipo.

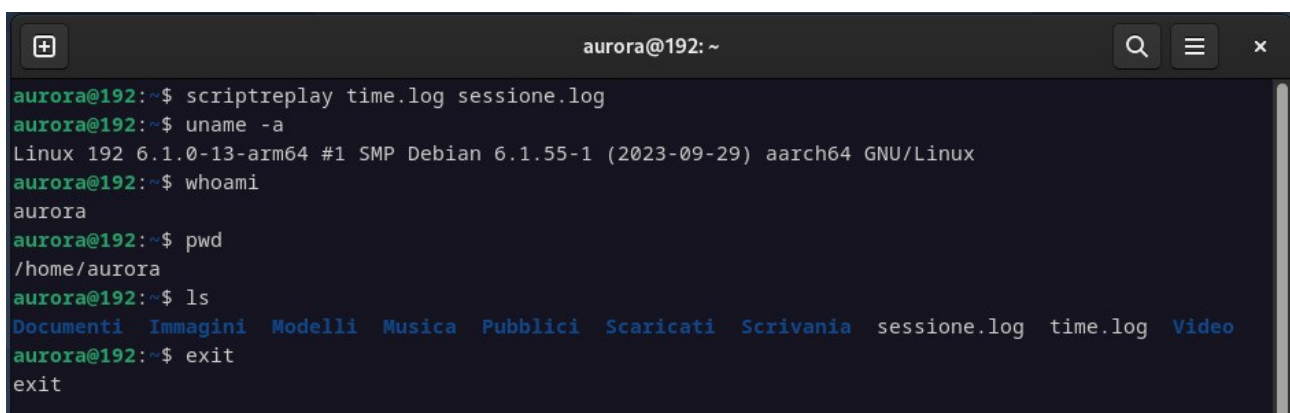
Si individui e si discuta un caso d'uso di un tale meccanismo. Infine, si fornisca al docente l'insieme di file necessario per riprodurre la sessione di comandi seguente:

```
uname -a
whoami
pwd
ls
```

**Soluzione.** Per registrare una sessione di terminale si può utilizzare il comando **script**, che permette di salvare su un file di log tutto ciò che viene scritto e visualizzato sul terminale. È possibile registrare anche le informazioni sui tempi aggiungendo l'opzione **-t**. Una volta avviato il comando verranno catturate tutte le informazioni della sessione, finché non si termina con **exit**. Successivamente sarà possibile riprodurre esattamente la stessa sessione, passando i due file appena generati come argomenti al comando **scriptreplay**, come mostrato in Figura 1.

```
script -t=time.log sessione.log
scriptreplay time.log sessione.log
```

Questo meccanismo, per esempio, potrebbe tornare molto utile quando un utente riscontra alcuni problemi e vuole richiedere aiuto a un team di supporto o su un forum. L'utente in questione può registrare la sequenza di azioni che ha causato il problema e condividere i file di log generati. Chi riceverà tali file avrà modo di analizzare tutti gli input e output del terminale e di risolvere il problema.

A screenshot of a terminal window titled 'aurora@192: ~'. The terminal shows the execution of the 'scriptreplay' command with two arguments: 'time.log' and 'sessione.log'. This triggers a replay of a previous terminal session. The replayed commands and their outputs are: 'uname -a' resulting in 'Linux 192 6.1.0-13-arm64 #1 SMP Debian 6.1.55-1 (2023-09-29) aarch64 GNU/Linux', 'whoami' resulting in 'aurora', 'pwd' resulting in '/home/aurora', and 'ls' resulting in a directory listing including 'Documenti', 'Immagini', 'Modelli', 'Musica', 'Pubblici', 'Scaricati', 'Scrivania', 'sessione.log', 'time.log', and 'Video'. The session concludes with the 'exit' command.

*Figura 1: riproduzione sessione*

**Esercizio 2.** Si illustri la procedura di installazione del software “Cool Retro Term” e lo si descriva brevemente. Si provi ad eseguire simultaneamente i due programmi seguenti (senza terminarli):

- **top**
- **ls -alR /**

Si riesce nell'intento? Se sì, perché? Se no, perché? In caso negativo, si cerchi un software che permetta di risolvere questo problema. Si discutano l'installazione e le principali funzionalità fornite dal software.

**Soluzione.** Si installi il software “Cool Retro Term” e lo si apra con i seguenti comandi:

```
sudo apt-get install cool-retro-term
cool-retro-term
```

**top** è un comando interattivo utilizzato per monitorare in tempo reale i processi in esecuzione. Fornisce una visualizzazione in tempo reale con aggiornamento continuo a intervalli regolari. Non termina finché non si preme la combinazione di tasti Ctrl+C. Il comando **ls -alR /** invece elenca ricorsivamente i tutti contenuti della cartella radice e di tutte le sue sottocartelle e, di conseguenza, richiede un tempo abbastanza lungo per eseguire. Entrambi i comandi bloccano il terminale, impedendo l'inserimento di altri comandi, a meno che non si termini ciascuno dei processi con la combinazione di tasti per l'interruzione. È possibile eseguire simultaneamente i due comandi solo tramite un multiplexer di terminale, come **tmux** o **GNU Screen**, un software che consente all'utente di aprire e gestire più terminali virtuali all'interno di una singola finestra di terminale.

Si installi **tmux** con il seguente comando:

```
sudo apt-get install tmux
```

Per avviare una sessione si esegua **tmux** e si divida in due pannelli con la combinazione di tasti **<Ctrl-b>+%** o **<Ctrl-b>+"**. Si preme **<Ctrl-b>+arrow\_key** per spostarsi da un pannello all'altro. Si potranno ora eseguire entrambi i comandi **top** e **ls -alR /** come mostrato in Figura 2. I due comandi sono a tutti gli effetti eseguiti in simultanea perché i pannelli creati rappresentano finestre diverse della stessa sessione, cioè condividono lo stesso contesto: autorizzazioni, variabili d'ambiente e altre informazioni specifiche che influenzano il comportamento dei comandi.

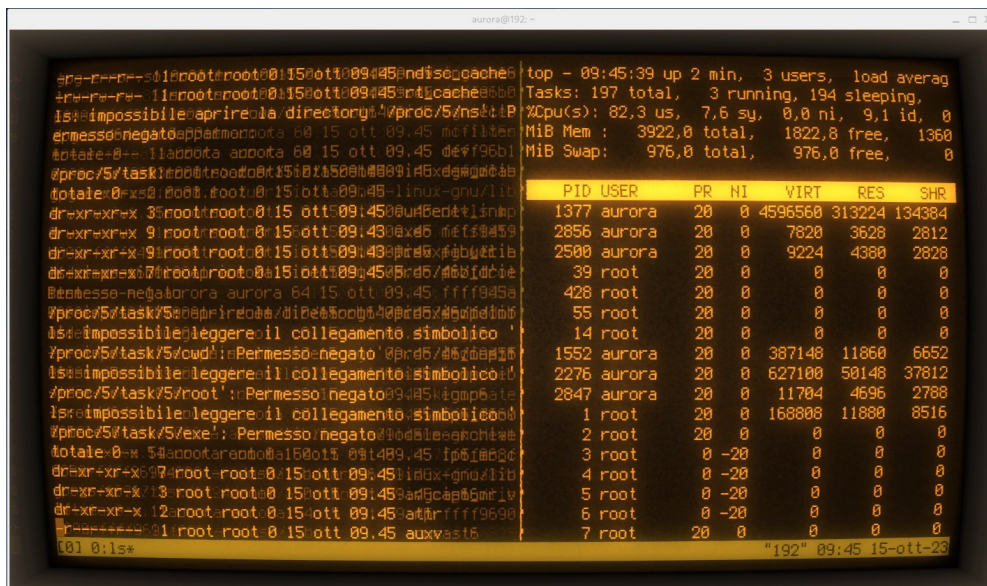
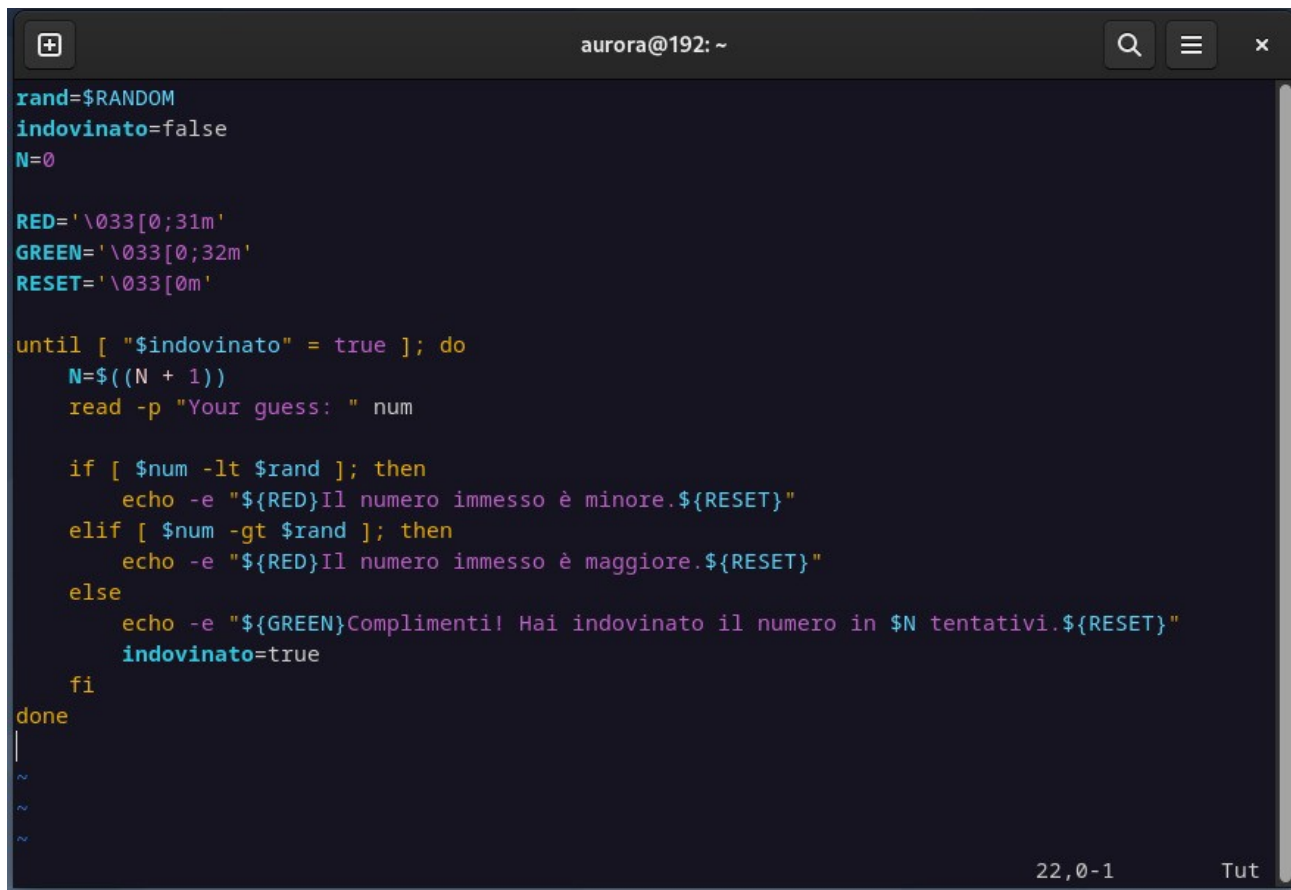


Figura 2: *tmux*

**Esercizio 3.** Si produca uno script shell di nome **indovinello.sh** che implementa il classico gioco dell'indovinello. Viene generato un numero casuale ed eseguito un ciclo infinito nel quale si chiede un numero all'utente. Se il numero è strettamente minore di quello generato, si stampa in colore rosso "Il numero immesso è minore.". Se il numero è strettamente maggiore di quello generato, si stampa in colore rosso "Il numero immesso è maggiore.". Se il numero è uguale a quello generato, si stampa in verde il messaggio "Complimenti! Hai indovinato il numero in N tentativi.", dove N è il numero di tentativi richiesti all'utente per risolvere l'indovinello. Qual è la strategia ottimale per vincere all'indovinello? Qual è la complessità computazionale di tale strategia?

**Soluzione.** **\$RANDOM** (riga 1 Figura 3) è una funzione interna di BASH che ritorna un intero pseudo-casuale nell'intervallo da 0 fino a 32767, ovvero il massimo valore che può essere rappresentato da una variabile intera a 16 bit con segno. Sapendo il numero esatto di elementi nell'intervallo, si può dire con certezza che, se si prova a risolvere il gioco senza utilizzare nessuna strategia e senza ripetere i numeri già provati, si arriverà alla soluzione in al più 32768 tentativi. Procedendo in modo casuale probabilmente non si riuscirà ad ottenere un buon risultato. La strategia migliore è sfruttare al meglio l'output che ci da il gioco dopo l'inserimento di un numero. Si può pensare a un algoritmo del tutto simile al Binary Search, ovvero si parta controllando l'elemento centrale dell'intervallo e, in base all'output, lo si prenda come estremo superiore o inferiore del nuovo intervallo. Si ripeta la stessa tecnica finché non si converge alla soluzione, riuscendo così a finire il gioco in al più  $\log_2(32768) = 15$  tentativi.

The image shows a terminal window titled 'aurora@192: ~'. It contains a shell script named 'indovinello.sh'. The script starts by setting 'rand=\$RANDOM', 'indovinato=false', and 'N=0'. It then defines color codes: 'RED='\033[0;31m'', 'GREEN='\033[0;32m'', and 'RESET='\033[0m''. A 'until' loop continues until 'indovinato' is true. Inside the loop, 'N' is incremented, the user is prompted for a guess, and conditional logic checks if the guess is less than, greater than, or equal to 'rand'. Appropriate color-coded messages are printed, and 'indovinato' is set to true when the guess is correct. The script ends with 'done'. The terminal shows the script being executed, with the user entering '1' as a guess. The output shows the first guess is incorrect (less than the random number). The terminal also shows a status bar at the bottom with '22,0-1' and 'Tut'.

```
rand=$RANDOM
indovinato=false
N=0

RED='\033[0;31m'
GREEN='\033[0;32m'
RESET='\033[0m'

until [ "$indovinato" = true ]; do
    N=$((N + 1))
    read -p "Your guess: " num

    if [ $num -lt $rand ]; then
        echo -e "${RED}Il numero immesso è minore.${RESET}"
    elif [ $num -gt $rand ]; then
        echo -e "${RED}Il numero immesso è maggiore.${RESET}"
    else
        echo -e "${GREEN}Complimenti! Hai indovinato il numero in $N tentativi.${RESET}"
        indovinato=true
    fi
done
|
~
~
~
```

Figura 3: indovinello.sh