

ISTRUZIONI PRATICHE

Esame del modulo di laboratorio di "Sistemi Operativi"

Durata: 120' (2 ore)

- Creare una cartella principale denominata con il proprio numero di matricola e inserire tutti i contenuti richiesti dal compito.
- Questa cartella andrà consegnata "zippandola" (compressione formato "zip") in modo da creare un file avente per nome il proprio numero di matricola più l'estensione ".zip". Deve essere compressa l'intera cartella e non solo il suo contenuto.

Se il proprio numero di matricola fosse 123456 questo deve essere anche il nome della cartella e l'archivio compresso da consegnare deve chiamarsi 123456.zip. All'estrazione dovrà essere presente la cartella 123456.

- Consegna:
 - Dopo 50' ed entro 60' dall'inizio della prova si deve fare una prima consegna (lavoro parziale) con il lavoro compiuto complessivo fino a tale momento ANCHE SE NON FUNZIONANTE utilizzando il modulo nell'homepage del browser. Nominare l'archivio parziale con "<matricola>_parziale.zip"
 - Dopo 90' ed entro 120' dall'inizio della prova si deve fare una seconda consegna (lavoro finale) utilizzando il modulo nell'homepage del browser: questa consegna è l'unica considerata per la valutazione finale.
 - Per consegnare, effettuare il login con il proprio account universitario e selezionare il file "zip" da allegare.
 - I punteggi [x] sono indicativi dato che la valutazione tiene conto anche di dettagli "trasversali" che non sono riferibili a singoli punti. Un punteggio complessivo maggiore o uguale a 31 porterà alla lode.

NOTA: parte delle verifiche può avvenire con procedure completamente o parzialmente automatizzate per cui le denominazioni e gli output devono essere rigorosamente aderenti alle indicazioni.

Eventuali irregolarità comportano l'esclusione dalla prova oltre a possibili sanzioni disciplinari.

Consegna

1. [4] L'applicazione deve leggere il file `"/tmp/credentials.txt"` contenente due stringhe (su righe diverse): l'username e la password (in quest'ordine). Una volta letti, il programma deve riscrivere la password su un nuovo file `"/tmp/secret.txt"`
...
2. [1] ...ed eliminare il file appena letto `"/tmp/credentials.txt"`.
3. [3] Alla pressione di **CTRL+Z** (segnale SIGTSTP) il programma deve terminare invece che sospendersi.
4. [5] L'applicazione deve creare una fifo (pipe con nome) chiamata `"/tmp/authenticator.fifo"` con almeno i permessi di lettura e scrittura e scriverci dentro una stringa contenente il proprio PID (esempio "454"). **Hint:** potete usare `sprintf()` per convertire un intero in stringa.
5. [5] Una volta comunicato il proprio pid, il processo dovrà mettersi in attesa di ricevere messaggi (multipli) sulla fifo `"/tmp/clients.fifo"` (in fase di valutazione sarà già esistente). Su questa fifo verranno inviati (in fase di valutazione) messaggi contenenti il PID di vari processi. Una volta ricevuto un PID, il programma dovrà inviare un segnale SIGUSR1 al processo con quel PID, e poi rimettersi in ascolto per altri messaggi con altri PID (non c'è un limite al numero di messaggi che possono arrivare).
6. [6] L'applicazione deve creare un **nuovo thread** nel quale mettersi in attesa di ricevere un messaggio sulla fifo `"/tmp/login.fifo"` (in fase di valutazione sarà già esistente).
Su questa fifo verranno inviate (durante la valutazione), una alla volta, diverse passwords (stringhe). Il thread deve stampare su **stdout** `"OK\n"` ogni volta che riceve la password corretta (quella letta nel punto 1), `"NO\n"` altrimenti.
Se, per esempio, la password letta al punto 1 è "ciao" e si ricevono 3 password ("come","ciao","stai"), l'output dovrà essere:
NO
OK
NO
7. [3] Dopo aver stampato `"OK\n"` alla ricezione della password corretta (punto 6), il thread deve stampare su **stderr** la stringa `"<username> logged in\n"` con l'username letto al punto 1. **NB:** l'username deve essere passato al thread come argomento (non può essere una variabile globale).
Se, per esempio, l'username letto al punto 1 è "sistemi", l'output d'esempio precedente diventerà:
NO
OK
sistemi logged in
NO

8. **[3]** Il programma deve creare una nuova coda, con una chiave data dalla tupla (`"/tmp/login.fifo", 51`) **Hint:** se necessario per testare, assicuratevi che il file (fifo) `"/tmp/login.fifo"` esista!
9. **[3]** ...successivamente, il programma deve inviare un messaggio per ogni tentativo di login (ovvero per ogni password ricevuta al punto 6). Il messaggio deve avere come payload una stringa (null-terminated) contenente la password appena ricevuta (punto 6), e come tipo l'ultimo PID ricevuto su `"/tmp/clients"` (punto 5). **NB:** si può usare un thread qualunque per queste operazioni.
- Riprendendo l'esempio precedente, il programma dovrà inviare tre messaggi diversi con payload `"come"`, `"ciao"`, `"stai"` rispettivamente, e con tipo uguale all'ultimo PID ricevuto sulla fifo del punto 5.

NB:

- **Utilizzate `fflush(stdout)` e `fflush(stderr)` dopo ogni scrittura sui quei canali.**
- **Non stampate a video nessun messaggio oltre a quelli espressamente richiesti**
- Per vedere le code presenti sul sistema potete usare da terminale `"ipcs -q"` e, per rimuoverle, `"ipcrm -q <queue_id>"`.
- Ogni stringa letta, messaggio ricevuto o messaggio inviato dovrà essere al più 100 bytes.
- Per testare potete creare voi le fifo ed i file necessari. Per testare le fifo potete creare programmi di test oppure usare `"echo"` e `"cat"` da terminale.
- Non consegnate codice che legge i messaggi mandati dalla vostra applicazione!

Riepilogo risorse:

- File `"/tmp/credentials.txt"` già esistente (al momento della valutazione) con delle credenziali
- File `"/tmp/secret.txt"` da creare e sul quale scrivere la password
- Fifo `"/tmp/authenticator.fifo"` da creare con e sul quale inviare proprio pid
- Fifo `"/tmp/clients.fifo"` già esistente (al momento della valutazione) sulla quale ascoltare messaggi contenti PIDs
- Fifo `"/tmp/login.fifo"` già esistente (al momento della valutazione) sulla quale ricevere passwords
- Coda `<"/tmp/login.fifo", 51>` da creare sulla quale mandare le passwords (tipo uguale ai PID)

Si ricorda inoltre che:

- La mancata presenza di opportuni commenti nel codice può portare a delle penalizzazioni.
- La generazione di warnings nella compilazione porterà alla perdita di punti (ad eccezione dei warnings relativi ai parametri del thread).
- Errori di compilazione porteranno alla completa invalidazione della prova