Caselle riservate

Ex. 1	
Ex. 2	
Ex. 3	
Ex. 4	
Ex. 5	
Ex. 6	
Tot.	

Sistemi Operativi

Compito d'esame

27 Febbraio 2019

Matricola _____ Cognome _____ Nome _____

Docente: ◯ Quer ◯ Sterpone

Non si possono consultare testi, appunti o calcolatrici a parte i formulari distribuiti dal docente. Risolvere gli esercizi negli spazi riservati. Fogli aggiuntivi sono permessi solo quando strettamente necessari. Riportare i passaggi principali.
Durata della prova: 100 **minuti.**

1. Si supponga che dal programma successivo

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
int main (int argc, char *argv[]) {
  int i = 2;
  if (!strcmp (argv[0], "/home/quer/p")) {
    i++;
    fprintf (stdout, "exec 1 %d\n", i);
    execlp ("echo", "./p", "/home/quer/p", "i", NULL);
  }
  if (!strcmp (argv[0], "/home/quer/20190227")) {
    fprintf (stdout, "exec 2 %d\n", i);
    execlp ("./p", "/home/quer/p", "/home/quer/20190227", "++i", NULL);
  }
  fprintf (stdout, "i = %d\n", i);
  return (1);
}
```

sia stato ottenuto l'eseguibile di nome `/home/quer/20190227`.

Si supponga inoltre di eseguire lo script successivo:

```
/home/quer/20190227
ln /home/quer/20190227 p
/home/quer/p
```

Si riporti il grafo di controllo del flusso e l'albero di generazione dei processi a seguito della sua esecuzione. Si indichi inoltre che cosa esso produce su video e per quale motivo.

2. Si illustrino le caratteristiche delle *pipe* per la comunicazione e la sincronizzazione tra processi.

   Se ne illustri inoltre l'utilizzo risolvendo il seguente problema. Due processi P_1 e P_2 desiderano scambiarsi grandi quantità di dati tramite la scrittura e la lettura dello stesso file sincronizzando le proprie operazioni sul file mediante l'utilizzo di una o più pipe. Il file memorizza un insieme indefinito di stringhe in ragione di una stringa su ciascuna riga. Ciascun processo:

   - legge il contenuto del file una stringa alla volta e visualizza le stringhe a video.
   - riscrive completamente il file memorizzandoci un insieme di stringhe lette da tastiera. Le stringhe vanno lette da tastiera una alla volta. La lettura viene terminata dall'introduzione della stringa "`end`".

   Il protocollo deve essere tale per cui quando P_1 è in esecuzione P_2 attende e viceversa. Entrambi i processi devono terminare in maniera ordinata quando da tastiera viene introdotta la stringa "`END`" (maiuscola).

3. Si indichino le principali differenze tra processi e thread. Si descriva la gestione di tali entità da parte del sistema operativo (struttura in memoria, etc.) e se ne riportino le caratteristiche principali con i relativi vantaggi e svantaggi. Si indichino infine le principali differenze tra thread a livello utente e thread a livello kernel. Che implicazioni hanno i due modelli sulle politiche di scheduling?

4. Si implementi uno script BASH di nome `isto.sh` in grado di:
   - Ricevere il nome di un direttorio sulla riga di comando.
   - Controllare il corretto passaggio dei parametri ricevuti, segnalando un errore e terminando in caso di errore.
   - Verificare che il direttorio specificato esista, segnalando un errore e terminando in caso contrario.
   - ¡ Determinare la dimensione di tutti i file regolari contenuti nel direttorio specificato sulla riga di comando e in tutti i suoi sotto-direttori.
   - Visualizzare un istogramma a barre orizzontali composte dal carattere "#" in cui la barra di posizione n rappresenta il numero di file di dimensione compresa tra $(n-1)$KByte e (n)KByte incluso. Ovvero la prima barra ha una lunghezza pari al numero di file di dimensione compresa tra 0KByte e 1KByte incluso, la seconda quelli compresi tra 1KByte e 2KByte incluso, etc.

   Il successivo è un esempio di esecuzione dello script:

```
./isto.sh dir
1 #########
2 ####
5 ##
9 #####
...
```

   Si noti che le barre dell'istogramma di dimensione nulla (e.g., 3, 4, etc.) non vanno visualizzate.

5. **Per i candidati iscritti al corso nell'anno accademico 2018–2019.**
La sequenza di Fibonacci $F(n)$ è tale per cui:

$$\begin{aligned} F(1) &= 1 \\ F(2) &= 2 \\ F(i) &= F(i-1) + F(i-2) \qquad \text{con i} > 2 \end{aligned}$$

Ad esempio, la sequenza di Fibonacci per i primi $n = 7$ numeri è la seguente:

$$1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13$$

Si scriva un programma concorrente multi-thread in grado di:

- ricevere il numero n sulla riga di comando
- generare esattamente n thread, in modo che il thread di posizione i generi il numero di Fibonacci di posizione i (utilizzando il valore calcolato dai thread di posizione $(i-1)$ e $(i-2)$) e lo visualizzi
- sincronizzare i thread in modo da ottenere il comportamento desiderato.

Per i candidati iscritti al corso prima dell'anno accademico 2018–2019.
Un server registra le operazioni di autenticazione da parte dei propri utenti mediante un file di log costituito da messaggi (uno per riga) aventi il seguente formato:

```
Data Ora [Oggetto] Result Username IPaddress
```

dove `Data` e `Ora` sono stringhe contenenti rispettivamente data e ora del messaggio, `Oggetto` è un codice univoco che rappresenta l'operazione oggetto del messaggio (tale codice inizia per "register" nel caso delle registrazioni ed inizia per "login" nel caso dei tentativi di accesso), `Result` è l'esito dell'operazione (`SUCCESS` nel caso di successo, `FAIL` nel caso di errore), `Username` è il nome dell'utente che ha richiesto l'operazione e `IPaddress` è l'indirizzo IP da cui è avvenuta la richiesta.

Un esempio di tale file di log è riportato di seguito. Supporre che il formato del file di log sia sempre corretto.

Scrivere uno script AWK che, ricevuto in input il percorso del file di log, permetta di rilevare tutti quei i casi in cui, per un utente correttamente registrato, siano avvenuti due o più tentativi di login errati nello stesso giorno da parte di indirizzi IP diversi da quello di registrazione. Per ognuno di questi casi lo script deve stampare su output un messaggio di warning contente il numero di tali tentativi di login, il nome dell'utente ed il giorno in cui questi sono stati effettuati.

Ad esempio, dato il seguente input:

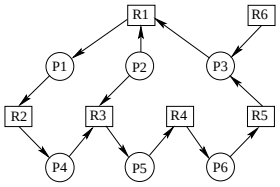
```
27/02/2019 14:30:00 [register.plain] FAIL failguy 55.44.22.3
27/02/2019 14:30:00 [register.oauth] SUCCESS okguy 94.103.22.8
27/02/2019 14:40:00 [login.plain] FAIL okguy 94.103.22.8
27/02/2019 14:40:10 [login.oauth] SUCCESS okguy 94.103.22.8
27/02/2019 16:00:00 [login.fb] FAIL okguy 66.9.233.233
27/02/2019 16:50:00 [login.oauth] SUCCESS okguy 66.9.233.233
28/02/2019 09:00:00 [login.oauth] FAIL okguy 55.169.16.233
28/02/2019 09:00:00 [login.plain] FAIL okguy 55.169.16.233
28/02/2019 21:00:00 [register.oauth] SUCCESS goodguy 179.144.12.12
01/03/2019 16:00:20 [login.oauth] FAIL failguy 94.103.22.8
01/03/2019 14:40:00 [login.fb] FAIL failguy 94.103.22.8
01/03/2019 08:00:00 [login.oauth] SUCCESS goodguy 202.1.1.76
01/03/2019 16:40:00 [login.oauth] FAIL goodguy 202.1.1.76
01/03/2019 16:41:00 [login.plain] FAIL goodguy 202.1.1.76
```

Lo script deve stampare in output:

```
Warning! 2 suspicious logins for user okguy in date 28/02/2019
Warning! 2 suspicious logins for user goodguy in date 01/31/2019
```

6. Si illustri che cosa si intende per “stato sicuro”, “sequenza sicura” e (con un esempio) per “grafico delle traiettorie delle risorse”.

   Per gestire le condizioni di stallo, due sistemi operativi utilizzano rispettivamente il grafo riportato a sinistra e la tabella riportata a destra della seguente figura.



Processo	Fine	Assegnate			Massimo			Necessità			Disponibilità		
		R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	F	1	0	1	4	3	4				3	2	2
P_2	F	0	0	1	4	4	2						
P_3	F	0	1	1	2	3	3						
P_4	F	0	0	0	5	4	5						
P_5	F	0	1	0	1	4	4						

   Considerando i due casi separatamente, si indichi se gli stati rappresentati sono sicuri e in caso lo siano si riporti una possibile sequenza sicura.