Caselle riservate

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

Sistemi Operativi

Compito d'esame

21 Giugno 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.**

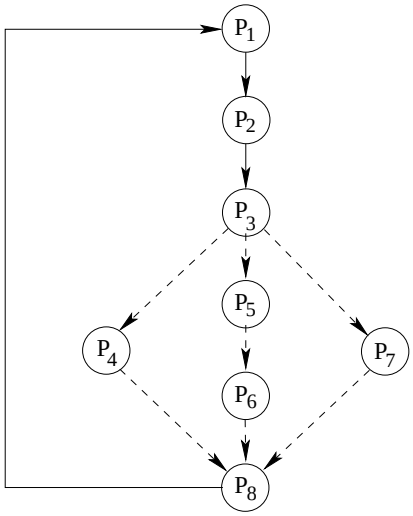
1. Si indichi quali sono le principali differenze tra una codifica ASCII e una UNICODE, quali le differenze tra un file di testo e uno binario e che cosa si intende per serializzazione.

   Utilizzando le funzioni di input/output diretto POSIX

   ```
   int open (const char *path, int flags, mode_t mode);
   int read (int fd, void *buf, size_t nbytes);
   int write (int fd, void *buf, size_t nbytes);
   off_t lseek (int fd, off_t offset, int whence);
   int close (int fd);
   ```

   si scriva un programma che sia in grado di creare una copia di un file binario invertendo l'ordine dei byte in esso contenuti (il primo byte diventa l'ultimo e viceversa). Il programma riceve su linea di comando il nome del file di input (che si suppone esistere) e il nome del file di output. Si osservi che non è consentito memorizzare l'intero file in memoria centrale.

2. Dato il seguente grafo di precedenza, realizzarlo utilizzando il **minimo** numero possibile di semafori. I processi rappresentati devono essere processi ciclici (con corpo del tipo `while(1)`). Gli archi tratteggiati indicano che solo due rami (tra i rami P_4, P_5/P_6, e P_7) devono essere eseguiti (arbitrariamente) per ogni iterazione (ovvero devono essere eseguiti P_4 e P_5/P_6, oppure P_7 e P_5/P_6 oppure ancora P_4 e P_7). Utilizzare le primitive `init`, `signal`, `wait` e `destroy`. Riportare il corpo dei processi ($P_1, \ldots, P_8$) e l'inizializzazione dei semafori.



3. Si descriva l'utilizzo dei segnali nel sistema operativo Unix/Linux con relativi vantaggi e svantaggi. Si descrivano in particolare le system call `signal`, `kill`, `pause` e `alarm`.

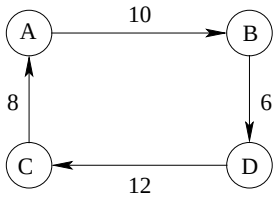
   Un programmatore desidera richiamare la funzione `npCompleteFunction` dall'interno dal programma principale. Tale funzione, pur essendo senza parametri, può richiedere tempi di esecuzione molto elevati. Volendo evitare di rimanere troppo a lungo in attesa della sua terminazione, il programmatore desidera fare in modo che:

   - Prima di chiamare la funzione stessa, il programma si predisponga ad auto-inviarsi e a gestire il segnale `SIG_CHLD` dopo 1000, 5000 e 10000 secondi.
   - Una volta richiamata la funzione, il programma chieda all'utente se si desidera continuare l'esecuzione della funzione oppure se si preferisce terminarla ad ogni ricezione del segnale `SIG_CHLD`.
   - Il programma esegua la scelta dell'utente terminando oppure continuando l'esecuzione della funzione `npCompleteFunction`.

   Si osservi che **non** è possibile utilizzare la system call `alarm` per realizzare il comportamento desiderato.

4. Un grafo diretto pesato è memorizzato in un file di testo con il seguente formato. Ogni riga del file specifica un arco del grafo. Per ogni arco sono indicati l'identificatore del vertice di partenza, quello del vertice di arrivo, e il peso dell'arco (valore intero). Il successivo è un esempio corretto di file e del grafo corrispondente.

```
A B 10
C A  8
B D  6
D C 12
```



Si implementi uno script BASH in grado di:

- Ricevere sulla riga di comando il nome di un file con la struttura precedentemente definita.
- Leggere da tastiera delle sequenze di identificatori di vertici separati da uno spazio.
- Verificare i vertici specificati formino un percorso sul grafo. In caso affermativo visualizzare il peso totale di tale percorso. In caso negativo terminare lo script.

Per il grafo riportato precedentemente, il successivo costituisce un esempio di esecuzione corretto:

```
script.bash nome_file.txt
> A B D C
path corretto, peso complessivo 28.
> C A B
path corretto, peso complessivo 18.
> A B C D
path non valido ... fine script.
```

5. **Per i candidati iscritti al corso nell'anno accademico 2018–2019.**
   Si scriva un programma concorrente multi-thread che riceve sulla riga di comando due valori interi `n` e `p`.

   Il thread principale crea `n` buffer globali (condivisi) di interi e dimensione costante arbitraria, esegue `p` thread produttori e un unico thread consumatore, quindi termina.

   Ogni thread produttore itera all'infinito effettuando le seguenti operazioni: dorme un numero casuale di secondi variabile tra 0 e 3, seleziona in maniera casuale uno degli `n` buffer, memorizza in tale buffer il proprio identificatore (ovvero un numero intero incluso tra 0 e `p-1`) non appena gli è possibile.

   Il consumatore itera all'infinito cercando di leggere valori dai buffer il più rapidamente possibile, ovvero senza rimanere bloccato in attesa indefinita su nessun buffer. I valori letti vengano visualizzati a video con il numero del buffer da cui sono stati letti.

   Suggerimento: Il consumatore può utilizzare la system call `sem_trywait` per evitare di attendere indefinitamente la scrittura di un dato su un buffer vuoto.

   Per i candidati iscritti al corso prima dell'anno accademico 2018–2019.
   Scrivere uno script AWK in grado di risolvere il problema dell'esercizio numero 4.

6. Si descriva il problema dello stallo modellandolo mediante grafo di allocazione delle risorse. Si indichino le tecniche per l'identificazione di uno stallo, mostrando un esempio in cui lo stallo sussiste e uno in cui non sussiste. Si indichi che cosa si intende per grafo di attesa e per arco di reclamo. Si indichi che cosa si intende per grafo di rivendicazione e per arco di richiesta.

   Infine si illustri il meccanismo di prevenzione del deadlock basato sull'utilizzo gerarchico delle risorse. Si dimostri che tale tecnica previene il verificarsi di deadlock.