Caselle riservate

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

Sistemi Operativi

Compito d'esame

17 Settembre 2018

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. Un programma concorrente utilizza n thread aciclici rappresentati dai seguenti tratti di pseudo-codice:

```
Thread 1                Thread 2                        Thread n

...                     ...                             ...
printf ("A");           printf ("A");          ...      printf ("A");
...                     ...                             ...
printf ("B");           printf ("B");                   printf ("B");
...                     ...                             ...
```

Si completino tali pseudo-codici in modo che tutti gli n caratteri `A` siano stampati prima di qualsiasi carattere `B`.

Suggerimento: ci si riferisca all'epilogo della soluzione per il problema dei "Readers e Writers" utilizzando una variabile di conteggio e gli opportuni semafori per sincronizzare i thread.

2. Si indichi che cosa succede a un processo quando il suo processo padre termina prima di esso oppure quando esso termina prima del processo padre.

   Si scriva quindi un programma C (non pseudo-codice) in grado di ricevere un valore intero `n` sulla riga di comando e di generare esattamente `n` processi zombie.

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.). Se ne indichino le caratteristiche principali e i relativi vantaggi e svantaggi.

   Si riporti almeno un esempio di programma concorrente (utilizzando dello pseudo-codice) che evidenzi i vantaggi di utilizzare i thread invece dei processi.

   Si indichino inoltre le principali differenze tra i thread a livello utente, quelli a livello kernel, e l'implementazione ibrida.

4. Si scriva uno script (esclusivamente) BASH in grado di:
   - Ricevere un parametro sulla riga di comando, controllandone l'esistenza e terminando in caso di errore.
   - Cercare, nel direttorio il cui path corrisponde al parametro ricevuto, tutti i file di estensione `.c` con uno o più caratteri "–" inclusi nel nome.
   - Generare una stringa rimpiazzando tutti i caratteri "–" del nome del file con il carattere "/" e trascurando l'ultima sotto-stringa contenente il nome e l'estensione del file. La stringa generata corrisponde al path di un direttorio.
   - Se non esiste, creare nel direttorio fornito in ingresso un nuovo direttorio il cui nome è stato generato al passo precedente.
   - Copiare il file nel direttorio destinazione utilizzando la stessa estensione e lo stesso nome presenti nell'ultima sotto-stringa del nome originario.

   Per esempio, se in input viene fornito il direttorio `/home/user` e in tale direttorio è presente un file di nome `a-b-c-foo.c`, occorre generare il direttorio `/home/user/a/b/c` (solo se non esiste) e in esso occorre copiare il file `foo.c`.

   Si osservi che si può supporre non esistano file con un nome che termina con il carattere "–", e.g., `a-b-c-.c`.

5. Scrivere uno script AWK che riceva sulla riga di comando il nome di un file di estensione `.c` e generi un file di estensione `.x` modificando il file `.c` come segue:
   - Tutte le linee che incominciano con la coppia di caratteri "`/*`" oppure finiscono con la coppia "`*/`" devono essere eliminate.
   - Tutte le linee del tipo

     ```
     #define stringa1 stringa2
     ```

     in cui `stringa1` e `stringa2` sono sequenze qualunque di caratteri alfanumerici, devono essere eliminate. Inoltre, tutte le stringhe identiche a `stringa1` presenti nel file (da quel punto in poi) devono essere sostituite con la stringa `stringa2`.
   - Tutte le righe del file non modificate seguendo le due regole precedenti, devono essere ricopiate senza alcuna modifica.

6. Con riferimento all'attività di schedulazione dei task (processi o thread) all'interno di un sistema operativo, si elenchino quali sono le funzioni di costo più utilizzate e quali i loro scopi.

   Si faccia quindi riferimento agli algoritmi di schedulazione senza prelazione FCFS (First Come First Served) e PS (Priority Scheduling). Supponendo che lo scheduler debba gestire 4 processi $\{P_0, P_1, P_2, P_3\}$ con tempi di arrivo, rispettivamente, uguali a $\{0, 1, 2, 3\}$ unità di tempo, si indichi quali priorità e quali richieste di tempo i vari processi dovrebbero avere per rendere uno dei due algoritmi precedenti estremamente vantaggioso rispetto all'altro.

   Si motivino nei due casi le conclusioni ottenute e si indichi come si comporterebbe in tali due casi l'algoritmo SJF (Shortest Job First).