Sesta prova di verifica AESO

The respondent's email (**null**) was recorded on submission of this form.
*Required

1. Email *

2. Numero di Matricola dello studente: *

3. Una chiamata di sistema:

Mark only one oval.

- ◯ tipicamente non restituisce alcun risultato (void) dal momento che le chiamate di sistema lavorano esclusivamente per side effect
- ◯ tipicamente ritorna un puntatore, dove il valore NULL indica un errore
- ⊗ tipicamente ritorna un intero, dove valori negativi indicano che nella chiamata si è verificato un errore
- ◯ tipicamente ritorna un intero, dove il valore 0 indica che nella chiamata si è verificato un errore

4. La chiamata di sistema fork()

Mark only one oval.

- ( ) crea un processo che esegue lo stesso codice del chiamante e va sempre accoppiata ad una chiamata che permette di differenziare il codice
- (X) crea un processo che esegue lo stesso codice del chiamante
- ( ) crea un processo che esegue un codice diverso da quello del chiamante, recuperato da un file su disco che ne rappresenta l'eseguibile
- ( ) non crea alcun processo, ma predispone il sistema all'esecuzione di una chiamata di sistema exec che creerà un processo che esegue il codice contenuto in un file eseguibile

5. Dopo una fork(), seguita da una exec() il processo padre e processo figlio:

Mark only one oval.

- ( ) condividono le sole posizioni della tabella dei file aperti che fanno riferimento a pipe
- ( ) condividono solo le prime tre posizioni della tabella dei file aperti, ovvero quelle che rappresentano lo standard input, lo standard output e lo standard error
- (X) condividono l'intera tabella dei file aperti
- ( ) condividono la tabella dei descrittori dei file

6. I thread realizzati a livello utente:

Mark only one oval.

- ( ) devono avere comunque un thread in kernel space per ognuno dei thread in user space per poter essere utilizzati
- (X) si bloccano in concomitanza di una chiamata di sistema che legge da un dispositivo periferico
- ( ) devono essere necessariamente schedulati utilizzando uno scheduler time sharing

7. Che cosa si intende per "multi-threadeing cooperativo" in un modello di thread realizzati a livello utente:

Mark only one oval.

- ◯ Il thread in esecuzione detiene il processore fino alla sua terminazione senza poter essere interrotto
- ◯ I thread possono cooperare tra di loro secondo il modello ad "ambiente locale"
- ◯ I thread possono cooperare tra di loro secondo il modello ad "ambiente globale"
- ⊗ Il thread in esecuzione detiene il processore fino a quando non invoca la "thread_yield()" o fino a che si sospende su una chiamata bloccante della libreria che implementa i thread
- ◯ Nell'API della libreria che implementa i thread è presente la funzione "thread_yield()" per il rilascio esplicito del processore

8. Quale tra i seguenti dati NON sono contenuti nel TCB (Thread Control Block) di thread realizzati in spazio kernel:

Mark only one oval.

- ◯ Parametri di scheduling
- ◯ Puntatore allo stack
- ⊗ Puntatori alle aree dati
- ◯ Immagine dei registri del processore
- ◯ Stato di esecuzione

9. Quale tra le seguenti affermazioni è vera:

Mark only one oval.

- ( ) I thread implementati a livello kernel non possono usare i semafori per sincronizzarsi
- ( ) I thread implementati a livello kernel non possono invocare la "thread_yield()"
- ( ) I thread implementati a livello utente non possono invocare chiamate di sistema bloccanti
- (X) I thread implementati a livello utente non possono usare i semafori per sincronizzarsi
- ( ) I thread implementati a livello utente non possono utilizzare istruzioni atomiche
- ( ) I thread implementati a livello utente non possono utilizzare spin-lock

10. Cosa sono le "Scheduler Activations" :

Mark only one oval.

- ( ) La politica usata per la gestione delle upcalls
- ( ) Una politica di scheduling dei threads implementati a livello kernel
- (X) Uno schema di notifica di eventi dal kernel allo scheduler user-level di librerie di thread realizzate a livello utente
- ( ) Uno schema di notifica di eventi dallo scheduler user-level di librerie di thread realizzate a livello utente verso il kernel

11. Quanti PID distinti stampa il codice in figura supponendo che entrambe le chiamate fork() siano eseguite con successo?

```
int main(int argc, char ** argv) {

  int pid;
  pid = fork();
  printf("Pid %d\n", getpid());
  pid = fork();
  printf("Pid %d\n", getpid());
  return 0;
}
```

Mark only one oval.

- [ ] 2
- [x] 4
- [ ] 6
- [ ] 8
- [ ] Nessuna delle precedenti

12. Si consideri il programma in figura. Supponendo che il file "doc.txt" non esista e che tutte le chiamate di sistema siano eseguite con successo. Quale tra le seguenti affermazioni è vera:

```
int main() {
  int fd = open("doc.txt",O_CREAT|O_RDWR,0600);
  pid_t pid=fork();
  if (pid>0)  { write(fd, "ciao", 4); close(fd); wait(NULL); }
  if (pid==0) {
    char buf[4];
    int n=read(fd, buf, 4);
    if (n>0) write(1,buf,4);
  }
  return 0;
}
```

Mark only one oval.

- ( ) Il programma è corretto e potrebbe stampare la parola "ciao" o nulla dipendentemente dallo scheduling dei processi.
- ( ) Il programma è corretto e potrebbe stampare la parola "ciao" o qualche suo carattere o nulla.
- ( ) Il programma è errato perché due processi non possono condividere lo stesso file in lettura e scrittura
- ( ) Il programma è corretto e stampa sicuramente la parola "ciao".
- (X) Il programma è corretto e sicuramente non stampa nulla sullo standard output.
- ( ) Nessuna delle precedenti.

13. Quale delle seguenti operazioni NON può essere effettuata in user mode?

Mark only one oval.

- ( ) generare una interruzione software
- ( ) leggere parte del contenuto della parola di stato, anche mediante accesso implicito da parte istruzioni assembler particolari
- ( ) invocare un'operazione di ingresso/uscita
- (X) interagire con un dispositivo di ingresso uscita utilizzando memory-mapped I/O

14. In una upcall, l'handler chiamato

Mark only one oval.

- ( ) esegue in stato kernel ed ha accesso allo stato di tutti i registri generali al momento della upcall
- (X) esegue in stato utente ed ha accesso allo stato di tutti i registri generali al momento della upcall
- ( ) esegue in stato utente e non ha accesso allo stato dei registri generali al momento della upcall
- ( ) esegue in stato utente ed ha accesso allo stato di parte dei registri generali al momento della upcall

15. Il passaggio dal modo operativo "Utente" ad un modo operativo "Supervisore" (o kernel) avviene:

Mark only one oval.

- ( ) con una istruzione di salto, dopo aver settato un flag nella program status word
- (X) utilizzando un'interruzione software
- ( ) chiamando una routine assembler senza utilizzo di istruzioni speciali, dopo aver scritto in un registro generale il numero della syscall
- ( ) utilizzando una istruzione di upcall

16. Durante l'esecuzione dell'handler delle Fast Interrupt (FIQ) nel processore ARM:

Mark only one oval.

- ( ) il codice può utilizzare qualunque registro senza prima salvarne il contenuto, visto che lavora su una copia dedicata dei registri
- (X) il codice può utilizzare un certo numero di registri senza prima salvarne il contenuto, visto che parte dei registri vengono messi a disposizione come copie dedicate
- ( ) il codice deve salvare qualunque registro prima di utilizzarlo, dal momento che i registri contengono dati del programma in esecuzione quando si verifica l'interruzione

17. Si consideri un sistema con solo registri di controllo PC, SP, PSW (parola di stato) e registri generali R1-R13 (non ci sono registri duplicati). All'arrivo di una interruzione dal timer viene eseguito l'handler dell'interruzione

Mark only one oval.

- ad interruzioni disabilitate; l'handler salva tutti i registri sullo stack del kernel e modifica PSW nella fase iniziale.
- solo dopo che è stata cambiata la parola di stato; l'handler disabilita le interruzioni e salva tutti i registri nello stack del kernel
- [x] ad interruzioni disabilitate e solo dopo che sono stati salvati i registri di controllo nello stack del kernel
- ad interruzioni abilitate e solo dopo che sono stati salvati tutti i registri nello stack del kernel. L'handler provvede a disabilitare le interruzioni e modificare i registri di controllo.

18. Si consideri il seguente stato per i processi Pa, Pb, Pc e Pd. Il processo in esecuzione è Pa, la lista dei pronti contiene ->Pb->Pc (Pb è il primo della lista), il processo Pd è in attesa. Si indichi quale delle seguenti affermazione è FALSA:

Mark only one oval.

- se si completa l'operazione di I/O richiesta da Pd, Pd viene aggiunto alla fine della lista dei processi pronti
- se scade il quanto di tempo Pb passa in esecuzione e Pa viene aggiunto in fondo alla lista dei processi pronti
- [x] se Pa richiede una operazione di I/O viene aggiunto alla fine della lista dei processi pronti e Pb passa in esecuzione
- se Pa richiede una operazione di I/O viene messo in attesa; passa in esecuzione il processo Pb e la lista dei processi pronti diventa la lista del solo elemento Pc

19. Si consideri una variabile di lock chiamata "mutex" ed una variabile di condizione chiamata "cond" associata a mutex. L'istruzione "mutex.release()" eseguita alla fine di una sezione critica:

Mark only one oval.

- ( ) risveglia tutti i thread in attesa sulla mutex in modo che possano competere per l'acquisizione della lock
- ( ) non risveglia nessun thread immediatamente; la sveglia di uno dei thread è avvenuta con l'istruzione "cond.signal()" all'interno della sezione critica
- ( ) risveglia un thread tra quelli in attesa sulla mutex passandogli l'accesso alla lock e mettendolo subito in esecuzione
- ( ) risveglia un thread tra quelli in attesa sulla mutex mettendolo in coda pronti senza passargli esplicitamente la lock
- (X) risveglia un thread tra quelli in attesa sulla mutex passandogli l'accesso alla lock e mettendolo in coda pronti

20. La semantica MESA per le variabili di condizione

Mark only one oval.

- ( ) prevede che signal prenda come argomento la lock associata alla variabile e la passi al thread in attesa sulla wait
- ( ) prevede che signal prenda come argomento la lock associata e che la rilasci atomicamente
- (X) non prevede che signal prenda come argomento la variabile di lock associato alla variabile
- ( ) non prevede che ci sia una variabile di lock associata alla variabile di condizione

21. Si consideri una variabile di condizione "cond". Le variabili di condizione sono gestite con semantica MESA. Sulla variabile di condizione è in attesa il thread T1. L'istruzione "cond.signal();" effettuata dal thread T2:

Mark only one oval.

- [x] Comporta la sveglia e la messa in stato di pronto del thread T1
- [ ] Comporta la sveglia del thread T1 con prerilascio del processore a suo favore e la messa di T2 in stato di pronto
- [ ] Comporta la sveglia del thread T1 con prerilascio del processore a suo favore e la messa di T2 in stato di attesa
- [ ] Comporta la sveglia del thread T1 che passa in stato di pronto. Al thread T1 viene passata la lock non appena passa in esecuzione.

22. Si consideri una variabile di condizione "cond" associata con associata la variabile "lock". Le variabili di condizione sono gestite con semantica Hoare. Sulla variabile di condizione è in attesa il thread T1. L'istruzione "cond.signal(&lock);" effettuata dal thread T2:

Mark only one oval.

- [ ] Comporta il passaggio della lock al thread T1 e la sua messa in esecuzione allo scadere del quando di tempo per il thread T2
- [x] Comporta il prerilascio del processore in favore di T1 e la messa in stato di attesa di T2
- [ ] Comporta il prerilascio del processore in favore di T1 e la messa in stato di pronto di T2
- [ ] Comporta la messa del thread T1 in stato di pronto senza prerilascio del processore

23. Considerando il codice in figura eseguito dai thread T1-T4 su un sistema monoprocessore. La coda pronti contiene i thread ->T1 -> T3 -> T4 (T1 è il primo della coda). La coda pronti e gestita in ordine strettamente FIFO. Il thread in esecuzione è T2, mentre T4 possiede la lock. Quando T4 passa in esecuzione esce dalla sezione critica (ed esegue SpinUnlock) e si sospende su una chiamata di sistema. Quale thread sarà il primo ad acquisire la lock ?

```
while(spinLock(&lock) == BUSY)
  thread_yield();

 /* sezione critica */

spinUnlock(&lock);
```

Mark only one oval.

- [ ] T1
- [x] T2
- [ ] T3
- [ ] T4

24. Si consideri il codice in figura eseguito da thread produttori e da thread consumatori in cui si utilizzano tre semafori per realizzare la mutua esclusione e la sincronizzazione: mutex, waitP e waitC.

```
// eseguito dai thread produttori
 while(1) {
1  msg=produci();
2  mutex.P();
3  if (bufferPieno()) waitP.P();
4  inserisci(msg);
5  mutex.V();
6  waitC.V();
 }
```

```
// eseguito dai thread consumatori
 while(1) {
1  mutex.P();
2  if (bufferVuoto()) waitC.P();
3  msg=consuma();
4  mutex.V();
5  waitP.V();
  // lavora su msg
 }
```

Mark only one oval.

- ◯ Il codice è corretto solo se mutex e waitC sono inizializzati a 0 mentre waitP è inizializzato con la dimensione del buffer
- ◯ Il codice è errato perché entrambi gli "if" devono essere sostituiti da un "while" per evitare corse critiche
- ◯ Il codice è errato perché le righe 2 e 3 del codice del produttore e 1 e 2 del codice del consumatore devono essere scambiate
- ◯ Il codice è errato perché le righe 5 e 6 per il produttore e 4 e 5 per il consumatore devono essere scambiate
- ◯ Non posso dire se il codice è corretto o meno senza sapere quali sono i valori di inizializzazione dei semafori
- ⊗ Nessuna delle precedenti

25. Quale delle seguenti affermazioni è FALSA?

Mark only one oval.

- ◯ dopo la creazione di un thread un qualunque altro thread può attenderne la terminazione con una join
- ⊗ una volta creato un thread, è possibile farlo terminare invocando una exit() senza la necessità di eseguire una join
- ◯ l'ordine in cui vengono eseguite le join su un insieme di thread può non rispettare l'ordine con cui sono stati creati i thread

26. Le operazioni che posso fare su una variabile condizione sono:

Mark only one oval.

- ◯ wait e signal
- ⊗ wait, signal e broadcast
- ◯ lock, unlock, wait e signal
- ◯ lock, unlock, wait, signal e broadcast

27. Considerando la variabile di lock "mutex", l'operazione di "mutex.acquire()"

Mark only one oval.

- ◯ comporta necessariamente attesa attiva
- ◯ non comporterà mai attesa attiva
- ⊗ può comportare un'attesa attiva solo nel caso la lock sia già stata acquisita e solo per un breve lasso di tempo
- ◯ può comportare un'attesa attiva qualunque sia lo stato della mutex