A

Sistemi Operativi L-A Prof. Anna Ciampolini

Seconda prova parziale 17 Giugno 2004

1. [3]

Si consideri la seguente porzione di codice:

```
void h(int s) { write(fd,&c,1); }
int fd,pid,pp[2]; char c='a';
main() {
fd=open("f01",O_RDWR);
pipe(pp);
pid=fork();
if(pid==0) {
  signal(SIGUSR1,h);
  write(pp[1],&c,1);
  close(pp[1]);
  while(read(pp[0]),&c,1)
    write(fd,&c,1);
} else {
  read(pp[0],&c,1);
  while(1) {
    c++; write(pp[1],&c,1);
    if(c>='d') {
      kill(pid,SIGUSR1);
      exit(0); }
} } }
```

In condizioni ideali (senza errori nelle system call):

A [**F**] Uno dei due processi rimarrà bloccato (in deadlock) in attesa di una scrittura su pipe

B [**F**] Al termine dell'esecuzione del processo figlio, `f01` conterrà la sequenza `abcd`

C [**F**] Al termine dell'esecuzione del processo figlio, `f01` conterrà i caratteri `a b c d`, ma non necessariamente in questo ordine

D [**F**] Al termine dell'esecuzione del processo figlio, `f01` conterrà almento il carattere `a`

E [**F**] Dopo l'esecuzione della funzione `h` da parte del figlio, il figlio termina la propria esecuzione

2. [3]

Si consideri la seguente porzione di codice:

```
int n,pp[2]; char s[4];
main() {
pipe(pp1);
fd=open("f01",O_RDWR);
if(!fork()) {
  while(n=read(fd,s,4))
    write(pp[1],s,n);
} else {
  close(pp[1]);
  while(read(pp[0],s,1)) {
    write(1,s,1);
    if(s[0]=='d')
      lseek(fd,-1,1);
} } }
```

Sia dato nel direttorio corrente un file `f01`, contenente la riga `abcdef`. In condizioni ideali (senza errori nelle system call):

A [**V**] Il padre non può terminare (a meno di terminazione involontaria) prima che il processo figlio abbia terminato, chiudendo la pipe in scrittura

B [**F**] A seguito dell'esecuzione del programma, potrebbe venire visualizzata su standard output la sequenza `abcef`

C [**F**] La chiusura della pipe in scrittura da parte del processo padre provoca la perdita dei caratteri eventualmente scritti sulla pipe dal figlio fino a quel momento

D [**F**] A seguito dell'esecuzione del programma, potrebbe venire visualizzata su standard output la sequenza `abcef`

E [**V**] La chiusura della pipe in scrittura da parte del processo padre provoca l'eliminazione di un record dalla tabella dei file aperti di processo

3. [3]

Si consideri un processo P nel sistema operativo Unix, che esegue la seguente istruzione I: `fd=open("filein", O_RDONLY);`
Nell'ipotesi che il file `filein` esista nel direttorio corrente, e che esso sia leggibile per tutti gli utenti:

A [**V**] L'istruzione I (se eseguita con successo) provoca sempre l'inserimento di un nuovo elemento nella *tabella dei file aperti di sistema*.

B [**F**] L'istruzione I provoca sempre l'inserimento di un nuovo elemento nella *tabella dei file attivi*.

C [**F**] Il valore di `fd` e' sempre un intero > 0.

D [**F**] I processi nipoti e pronipoti di P (creati dopo l'esecuzione di I) non possono condividere con P l'I/O pointer relativo a `"filein"`.

E [**F**] Se `filein` e' vuoto (cioe' contiene 0 byte), a `fd` viene assegnato un valore negativo.

4. [3]

Si consideri il file system di Unix e la sua organizzazione fisica.

A [**F**] L'*i-node* di un file F contiene il nome di F.

B [**V**] Se a un direttorio D appartengono N file, il file che rappresenta D contiene esattamente $N + 2$ record.

C [**F**] Il comando `$ rm filetest` (se eseguito con successo) provoca sempre l'eliminazione dell'i-node di `filetest`.

D [**F**] L'allocazione di ogni file e' a lista concatenata.

E [**V**] A piu' nomi di file puo' essere associato lo stesso *i-number*.

5. [3]

Si consideri lo scheduling della *CPU* in un sistema operativo multiprogrammato.

A [**F**] Il valore di *Turnaround* esprime il numero di processi completati nellunità di tempo.

B [**V**] Se la politica di scheduling e' SJF, e' possibile che vi sia starvation.

C [**F**] Se la politica di scheduling e' $FCFS$, e' possibile che vi sia starvation.

D [**F**] Se la politica di scheduling e' *Round Robin*, non c'e' prelazione.

E [**F**] La tecnica di *aging* viene tipicamente applicata al caso di scheduling $FCFS$.

6. [3]

Si consideri la gestione della memoria in un sistema operativo multiprogrammato.

A [**F**] Se nel sistema e' presente uno scheduler di medio termine (*swapper*), il binding degli indirizzi puo' avvenire in modo statico.

B [**V**] La tecnica di allocazione mediante paginazione risolve i problema derivanti dalla frammentazione esterna.

C [**V**] Nel caso di tecnica di allocazione a partizioni variabili: se la quantita' di memoria libera e'320K, l'allocazione di un nuovo processo P che occuperebbe 250K puo' non essere possibile.

D [**F**] Nel caso di tecnica di allocazione mediante segmentazione paginata: se la quantita' di memoria libera e'300K, l'allocazione di un nuovo segmento S che occuperebbe 150K puo' non essere possibile.

E [**F**] Nel caso di tecnica di allocazione mediante paginazione: in assenza di TLB, il binding degli indirizzi generati da un processo P e' mediamente piu' efficiente se la tabella delle pagine e' invertita, rispetto al caso di tabella delle pagine diretta (cioe', non invertita).

7. [3]	Si consideri un sistema operativo con memoria virtuale, con pagine di 4K. A [**F**] Per creare un nuovo processo di dimensioni 235 K, e‘ necessario che in memoria centrale vi siano almeno 59 frame liberi B [**V**] Ogni *page fault* viene notificato al kernel mediante un’interruzione. C [**F**] Se il bit di validita‘ associato ad una pagina logica p vale 0, significa sempre che la pagina p deve essere trasferita dalla backing store alla memoria centrale. D [**V**] In caso di sovrallocazione, la sostituzione di una pagina puo‘ richiedere 1 accesso alla memoria secondaria. E [**F**] Se l’algoritmo di sostituzione e‘ LRU, la sovrallocazione viene gestita sostituendo la pagina allocata da piu‘ tempo in memoria con una nuova pagina.		