Programmazione di Sistema
17 luglio 2017 (teoria)

Si prega di rispondere in maniera leggibile, descrivendo i passaggi e i risultati intermedi. Non è possibile consultare alcun materiale. Durata della prova 70 minuti. Sufficienza con punteggio >= 8. Prima e seconda parte possono essere sostenute in appelli diversi. La presenza a una delle due parti annulla automaticamente un'eventuale sufficienza già ottenuta (per la stessa parte): viene intesa come rifiuto del voto precedente. La risposta alla domanda 1 va scritta su questo foglio.

1. *(4 punti)* Siano dati i frammenti di programma (A e B) rappresentati, che calcolano nel vettore V, rispettivamente, le medie per riga/colonna dei dati in una matrice triangolare inferiore/superiore M *(si ricorda che, in modo informale, una matrice si dice triangolare superiore/inferiore se tutti gli elementi sotto/sopra la diagonale sono nulli):*

A

```
float M[1024][1024],V[1024];
...
for (i=0; i<1024; i++) {
  V[i] = 0.0;
  for (j=0; j<=i; j++) {
    V[i] += M[i][j];
  }
  V[i] = V[i]/(i+1);
}
...
```

B

```
float M[1024][1024],V[1024];
...
for (i=0; i<1024; i++) {
  V[i] = 0.0;
  for (j=0; j<=i; j++) {
    V[i] += M[j][i];
  }
  V[i] = V[i]/(i+1);
}
...
```

I programmi eseguibili generati da tali sorgenti sono eseguiti in un sistema con memoria virtuale gestita mediante paginazione, con **pagine di 4Kbyte**, utilizzando come politica di sostituzione pagine la **LRU** (Least Recently Used). Si sa che un `float` ha dimensione **32 bit** e che le istruzioni in codice macchina, corrispondenti al frammento di programma visualizzato, sono contenute in una sola pagina. Si supponga i che `M` e `V` siano allocati ad indirizzi logici contigui (prima `M`, poi `V`), a partire dall'indirizzo logico 0x8434A000. La matrice `M` è allocata secondo la strategia "row major", cioè per righe (prima riga, seguita da seconda riga, …).

- Quante pagine (e frame) sono necessarie per contenere la matrice e il vettore ?
- Ipotizzando che le variabili i, j siano allocate in registri della CPU, quanti accessi in memoria (in lettura e scrittura) fanno il programma proposto, per accedere a dati (non vanno conteggiati gli accessi a istruzioni) ?
- Detto N_T il numero totale di accessi a dati in memoria, ed N_L il numero di accessi a dati nella stessa pagina di uno dei precedenti 3 accessi, si definisca come **località** del programma per i dati il numero $L = N_L/N_T$. Si calcoli la località del programma proposto.
- Calcolare il numero di page fault generati dal programma proposto, supponendo che siano allocati per esso **10 frame**, di cui uno utilizzato (già all'inizio dell'esecuzione) per le istruzioni. (motivare la risposta)

(I programmi A e B vanno trattati come casi separati, cioè si tratta di due esecuzioni/esperimenti distinte/i)

2. *(3 punti)* Sia dato un file system basato su File Allocation Table (FAT). Si supponga, per semplicità descrittiva, che il file system contenga unicamente 20 blocchi di 2048 Byte (numerati da 0 a 19), nel quale siano immagazzinati 3 file (`"a.exe"`, `"b.jpg"`, `"c.dat"`), allocati nei blocchi (14,2,12,3), (10,1,5,18) e (7,4,17,19,13), rispettivamente. La free list è (15,9,0,8,11,16,6). Si rappresenti la FAT corrispondente a tale allocazione. Supponendo che dapprima il file `"a.exe"` venga esteso mediante aggiunta di un blocco in fondo al file, quindi venga cancellato `"b.jpg"`, si rappresenti la FAT dopo tali operazioni. Supponendo che il file `"c.dat"` contenga una sequenza di record a lunghezza fissa, ognuno di 50 byte, si dica in quale blocco, e a quale offset, si trova il record n. 130 (i record sono numerati a partire da 0). Si calcolino infine la dimensione minima, e la frammentazione interna massima, che può avere il file `"c.dat"`.

3. *(3 punti)* Sia dato un file system in cui è possibile l'accesso concorrente di più processi a uno stesso file.

- Quali operazioni deve svolgere il sistema operativo per realizzare una open()? E una close()?
- A quali strutture dati (tabelle di gestione file) si deve fare accesso per realizzare una read() e/o una write()?
- Che cosa sono la *system-wide open-file table* e la *per-process open-file table*? Perché in un sistema operativo possono essere necessarie entrambe anzichè solo una delle due?

4. *(4 punti)* Sia dato un sistema operativo OS161.

Si spieghi cosa realizza la funzione `syscall()`, e che cosa rappresenta la sua variabile `callno`, a cui si assegna il valore `tf->tf_v0`. Si dica poi che cosa significano le seguenti istruzioni alla fine della funzione `syscall()` (si dica, in particolare, cosa sono i campi `tf_v0` e `tf_a3` del trapframe) ?

```
if (err) {
  tf->tf_v0 = err;
  tf->tf_a3 = 1;
}
else {
  tf->tf_v0 = retval;
  tf->tf_a3 = 0;
}
```

Si dica, volendo realizzare la system call `SYS__exit` (che corrisponde alla `void exit (int status);`) come può essere risolto il problema della chiusura del programma utente che chiama la `exit`, registrando il valore di `status` (ricevuto come parametro) in modo tale che un altro thread possa eventualmente ottenere tale valore.

Si supponga che la funzione `syscall()`, in corrispondenza al valore `SYS__exit` in `callno` chiami la funzione `my_sys_exit()`, si definisca il prototipo di tale funzione e se ne scriva la chiamata (con parametri attuali) in `syscall()`.

Si osservi il seguente frammento di codice (incompleto) della `my_sys_exit()`, si dica da dove può o deve provenire il valore status, e a che variabile o campo di struct va assegnato. Si dica di quale semaforo può essere necessario/opportuno fare V. Si identifichino e si commentino gli errori o problemi contenuti nel codice:

```
my_sys_exit(…) {

  ... = status;
  V(...);
  thread_exit(curthread);
  as_destroy(curproc->p_addrspace);
  proc_destroy(curproc);
}
```