Esercizi su OS161 (tratti da compiti di esame)

1. 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 anziché solo una delle due?

2. Si consideri un buffer di kernel utilizzato come passaggio per i blocchi di un file in transito tra disco e memoria user: i dati che debbono essere trasferiti (ad esempio mediante una `read(fd,addr,size)`, con `addr` e `size` che determinano la destinazione in memoria user) fanno un passaggio in più: da disco a buffer kernel (per una dimensione size) e successivamente da buffer kernel alla vera destinazione `addr`.
   - Perché può essere vantaggioso il buffer kernel, pur costringendo a un passaggio in più in RAM?
   - In un sistema con paginazione, il parametro sizepuò essere arbitrario, oppure deve essere un multiplo della dimensione di blocco o di una pagina?

3. Si considerino i due tipi di sincronizzazione, relativi ad operazioni di I/O: sincrono e asincrono. Si spieghino le principali differenze tra gli I/O dei due tipi. Si definisca poi I/O bloccante e non bloccante: si tratta di sinonimi di asincrono e sincrono? Ci sono differenze (tra bloccante/non-bloccante e sincrono/asincrono)?

4. Si riporta una possibile realizzazione della funzione `getfreeppages`, che alloca un intervallo di `npages` pagine contigue di memoria fisica libera.

```
static paddr_t getfreeppages(unsigned long npages) {
    paddr_t addr = 0;
    long i, first, found, np = (long)npages;
    if (!isTableActive())
          return 0;
    spinlock_acquire(&freemem_lock);
    for (i=0, first=found=-1; i<nRamFrames; i++) {
          if (freeRamFrames[i]) {
               if (i==0 || !freeRamFrames[i-1])
                    first = i;
               if(i-first+1 >= np)
                   found = first;
          }
    }
    if (found >= 0) {
         for (i=found; i<found+np; i++) {
              freeRamFrames[i] = (unsigned char)0;
         }
```

```
        allocSize[found] = np;
        addr = (paddr_t) found*PAGE_SIZE;
    }
    spinlock_release(&freemem_lock);
    return addr;
}
```

La funzione realizza una politica di allocazione best-fit, worst-fit, first-fit o altro (motivare la risposta) ?