Esercizi proposti e svolti su OS161 (tratti da compiti di esame)

NOTA: si sono volutamente lasciati commenti relativi a correzioni di risposte all'esame o indicazioni di errori possibili/frequenti

1. Si riporta una parte della funzione `as_define_region` (file `dumbvm.c`). Si supponga di ricevere, per i parametri `as` e `vaddr` i valori (esadecimali) `0x80048720` e `0x412370`, e per `sz` il valore (decimale) `4128`. Ricordando che PAGE_SIZE è definito come `4096`, e PAGE_FRAME come **`0xfffff000`**, si simulino le istruzioni proposte, indicando, per ognuna, in esadecimale, i valori degli operandi e del risultato (il valore assegnato alla variabile).

```
as_define_region(struct addrspace *as, vaddr_t vaddr, size_t sz,
         int readable, int writeable, int executable) {
    size_t npages;
    /* Align the region. First, the base... */
    sz += vaddr & ~(vaddr_t)PAGE_FRAME;
    vaddr &= PAGE_FRAME;
    /* ...and now the length. */
    sz = (sz + PAGE_SIZE - 1) & PAGE_FRAME;
    npages = sz / PAGE_SIZE;
    ...
}
```

	4096 = 0x1000 4128 = 0x1020
`sz += vaddr & ~(vaddr_t)PAGE_FRAME;`	0x1020 += 0x412370 & 0x000FFF sz <- 0x1020+0x370 = **0x1390**
`vaddr &= PAGE_FRAME;`	0x412370 &= 0xFFF000 vaddr <- **0x412000**
`sz = (sz + PAGE_SIZE - 1) & PAGE_FRAME;`	sz <- (0x1390+0xFFF) & 0xFFF000 = **0x2000**
`npages = sz / PAGE_SIZE;`	npages <- 0x2000 / 0x1000 = 2

Si supponga che il valore di `sz` ricevuto sia inferiore alla dimensione di una pagina (es `4090`). E' possibile che si ottenga per `npages` il valore 2? (motivare la risposta)

Si. E' possibile, in quanto il segmento viene allineato a un multiplo di pagina sia all'inizio che alla fine. In questo caso c'è di fatto una forma di frammentazione interna sia sulla prima che sull'ultima pagina. Con 4090 = 0xFFA, sz avrebbe prima assunto il valore `0xFFA+0x370 = 0x126A`, quindi `0x2000`.

2. 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])
        /* set first free in an interval */
        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;
}
```

/* first-fit */

/* best-fit */

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

Nessuna delle tre.
First-fit NO perché le iterazioni non si fermano alla prima soluzione trovata
Best-fit e Worst fit NO in quanto non c'è una ricerca/gestione di minimo o massimo
La soluzione ritornata è l'ultima tra quelle trovate. La si potrebbe denominare "last-fit".

ERRORE FREQUENTE: *pensare che la soluzione proposta sia* ***first-fit***
COMMENTI (alla parte successiva, il programma da modificare): *quasi tutti gli studenti che hanno riconosciuto la last-fit, hanno proposto la modifica corretta per la first-fit.*
Pochi sono stati in grado di affrontare il problema della ricerca del ***"minimo tra gli intervalli di lunghezza >= np"*** *(o npages).*
Si noti che l'algoritmo proposto è O(nRamFrames) e che la versione iniziale controlla la lunghezza (per semplicità) in tutte le caselle intermedie di un intervallo.
Per realizzare una worst-fit (ricerca di un massimo) sarebbe sufficiente aggiungere la gestione di una variabile max ad ogni iterazione.
La best-fit invece, deve fare in modo di confrontarsi con il minimo provvisorio solo quando si arriva al termine di un intervallo (ad esempio confrontando la cella di indice i+1).
*Ho visto tentativi di soluzioni O(nRamFrames*npages) basati su doppia iterazione; salvo l'efficienza, possono essere ritenuti corretti.*
Non ritengo invece accettabili algoritmi che prevedano un vettore aggiuntivo già precaricato con le lunghezze degli intervalli. In effetti questo vettore risolverebbe in parte il problema, ma (al di la dell'occupazione di memoria), con un costo aggiuntivo per mantenerlo aggiornato ad ogni allocazione/de-allocazione (ciò che si vuole evitare con la bitmap). Oppure il vettore sarebbe precaricato con una passata lineare iniziale nella `getfreeppages`*, ma senza ottenere nulla di meglio rispetto a calcolare al volo le lunghezze degli intervalli (come proposto). In pratica, tutti coloro che hanno utilizzato questa soluzione hanno cercato di spostare/evitare (senza indicare come risolverlo) un pezzo del problema.*
Non sono valutate strategie "a parole": è necessario proporre codice in "C".

Si modifichi la funzione (scrivendo nel riquadro libero le parti modificate, che dovrebbero limitarsi alle istruzioni evidenziate in grigio) in modo tale da realizzare una politica first-fit e una best-fit.

```
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]) {
      /* controlla ogni frame libero (anche
         interno all'intervallo.
         Se è il primo di un intervallo,
         "ricorda" la posizione. */
      if (i==0 || !freeRamFrames[i-1])
        /* set first free in an interval */
        first = i;
      /* ogni intervallo compatibile con np
         viene assegnato. Quindi si ritorna
         l'ultimo */
      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;
}
```

```
/* ATTENZIONE: la soluzione proposta non è
l'unica: ne sono possibili altre (simili) */

/* first-fit: appena trovato esce */
/* variante con uscita strutturata */
...
  for (i=0,first=found=-1;
       i<nRamFrames && found<0; i++) {
...
/* variante con uscita non strutturata */
...
      if (i-first+1 >= np) {
        found = first;
        break;
      }
...

/* best fit: ricerca minimo (solo alla fine di
un intervallo) */
int min;
...
   /* si possono in alternativa mettere tutte
      le condizioni in AND (unico if) */
   /* se è l'ultimo frame di un intervallo */
   if (i==nRamFrames-1 || !freeRamFrames[i+1])
     /* se la dimensione va bene */
     if (i-first+1 >= np)
       /* se batte il minimo provvisorio */
       if (found<0 || i-first+1 < min) {
        found = first; min = i-first+1;
       }
...
```

3. Si spieghi, in relazione alla funzione `syscall()`, che cosa rappresenta la sua variabile `callno`, a cui si assegna il valore `tf->tf_v0`.

Si tratta del selettore della system call da effettuare, utilizzato all'interno della `syscall` per selezionare il relativo `case`. Tale valore viene assegnato al campo `tf->tf_v0` dalla routine che gestisce la trap e chiama `mips_trap->syscall`.

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;
}
```

Le istruzioni, poste alla fine della funzione `syscall`, gestiscono il lo stato e il valore di ritorno:

- In `v0` viene posto il valore di ritorno della system call (che coincide con un codice di errore in caso, appunto di errore)
- In `a3` viene ritornato lo stato successo(0)/errore(1)

4. Si consideri la realizzazione dei lock in un sistema OS161. Quale thread deve essere considerato owner (proprietario) di un lock?
   - Il thread che ha creato il lock?
     NO. Aver creato un lock non significa essere l'owner.
   - l'ultimo thread che ha chiamato la funzione `lock_acquire`?
     Non necessariamente. Solo nel caso in cui la lock_acquire sia stata fatta sul lock in questione (non su un altro lock) e il thread abbia effettivamente acquisito il lock (non sia ancora in attesa)
   - altro…(completare)
     Si veda la risposta precedente: l'owner di un lock è il thread che ha effettuato `lock_acquire` sul lock ed ha superato l'eventuale attesa.

Sono date le funzioni `lock_release` e `lock_do_i_hold` proposte in figura. Nelle funzioni sono presenti errori: li si identifichi e li si corregga (occorre motivare/spiegare)

```
void loc_release(struct lock *lock) {
  KASSERT(lock != NULL);
  spinlock_acquire(&lock->lk_lock);
  KASSERT(lock_do_i_hold(lock));
  lock->lk_owner=NULL;
  wchan_wakeone(lock->lk_wchan, &lock->lk_lock);
  spinlock_release(&lock->lk_lock);
}
```

```
bool
lock_do_i_hold(struct lock *lock) {
  spinlock_acquire(&lock->lk_lock);
  if (lock->lk_owner==curthread)
    return true;
  spinlock_release(&lock->lk_lock);
  return false;
}
```

L'errore nella lock_do_i_hold è l'istruzione return senza rilascio dello spinlock. Una possibile correzione è data dall'uso di una variabile booleana

```
bool lock_do_i_hold(struct lock *lock) {
  bool ret;
  spinlock_acquire(&lock->lk_lock);
  ret = lock->lk_owner==curthread;
  spinlock_release(&lock->lk_lock);
  return ret;
}
```

L'errore nella `lock_release` è dato dall'acquisizione dello spinlock prima della `lock_do_i_hold`, che tenterà di acquisire lo stesso spinlock. Si tratta di un problema di deadlock. Una possibile soluzione consiste nello spostare la `spinlock_acquire` dopo la chiamata a `lock_do_i_hold`.

5. Perché, in un contesto multi-core (sono presenti più CPU) non è possibile realizzare la mutua esclusione semplicemente disabilitando e riabilitando l'interrupt ?

Perché l'interrupt verrebbe disabilitato sulla sola CPU corrente, questo non precluderebbe quindi l'esecuzione di altri thread o processi sulle altre CPU.
ATTENZIONE: *Non sarebbe neppure sufficiente estendere la disabilitazione degli interrupt alle altre CPU, in quanto su queste ci potrebbero già essere altri thread in esecuzione. La disabilitazione degli interrupt serve quindi solo nel caso di un single core, in quanto significa che il thread corrente è l'unico in esecuzione sul sistema.*

Dato il codice (ridotto alle parti essenziali) delle funzioni di semaforo P e V, riportate in seguito

```
void P(struct semaphore *sem) {
   spinlock_acquire(&sem->sem_lock);
   while (sem->sem_count == 0) {
     wchan_sleep(sem->sem_wchan,
                 &sem->sem_lock);
   }
   sem->sem_count--;
   spinlock_release(&sem->sem_lock);
}
```

```
void V(struct semaphore *sem) {
   spinlock_acquire(&sem->sem_lock);
   sem->sem_count++;
   KASSERT(sem->sem_count > 0);
   wchan_wakeone(sem->sem_wchan,
                 &sem->sem_lock);
   spinlock_release(&sem->sem_lock);
}
```

Si risponda alle seguenti domande:

- A cosa serve lo spinlock (in entrambe le funzioni)?

Lo spinlock è necessario per poter utilizzare un wait-channel, quindi per chiamare correttamente `wchan_sleep` e `wchan_wakeone`. Serve a garantire la gestione in mutua esclusione della condizione (`sem->sem_count`)
ATTENZIONE: *Si valuta corretta solo in parte una risposta "generica", in cui si spieghi cosa sia uno spinlock. La domanda è relativa all'utilità dello spinlock in queste due funzioni.*

- Perché la P contiene un ciclo `while` anziché un `if (sem->sem_count == 0)`, mentre il ciclo non è presente nella V?

Perché la sincronizzazione tra `wchan_wakeone` e `wchan_sleep` (il risveglio, realizzato con semantica "Mesa", anziché "Hoare") non garantisce che la condizione, vera alla chiamata di `wchan_wakeone`, lo sia ancora al ritorno da `wchan_sleep`. Altri thread potrebbero modificarla nel frattempo.
ERRORE FREQUENTE: *pensare che esistano* ***risvegli "spuri"****/errati.* ***NON CI SONO RISVEGLI SBAGLIATI****. Il problema è semplicemente che in un contesto di programmazione concorrente ci potrebbero essere altri thread/processi svegliati (****correttamente****, da altri thread, in questo caso da altre chiamate a V), SENZA GARANTIRE (semantica Mesa) LA CRONOLOGIA STRETTA (garantita dalla semantica di Hoare) dell'esecuzione (stato RUN) dei thread, corrispondente ai risvegli (che non mandano in stato di RUN ma solo nella coda READY)*

- Perché la `wchan_sleep` riceve come parametro lo spinlock? Vale lo stesso motivo per la `wchan_wakeone`?

La `wchan_sleep` deve rilasciare lo spinlock, prima di mettere il thread in stato "wait", per poi riprenderlo al risveglio (prima di ritornare al chiamante). La `wchan_wakeone` non deve fare nulla sullo spinlock (le versioni Unix/Linux di tale finzione, ad esempio, NON hanno questo parametro): in OS161 il parametro viene solo verificato (in una KASSERT, per prevenire eventuali errori del programmatore) perché al momento delle chiamata il thread deve essere owner dello spinlock.
ERRORI FREQUENTI: *pensare che lo spinlock serva per garantire la mutua esclusione all'interno di* `wchan_sleep` *o* `wchan_wakeone`*: NO. Le funzioni al loro interno non avrebbero bisogno dello spinlock. Si asserisce l'ownership dello spinlock per verificare che CHI CHIAMA le funzioni stia facendo le cose bene, NON per far funzionare correttamente il wait channel (che, questo si, deve rilasciare e riacquisire lo spinlock in* `wchan_sleep`*).*
Secondo errore (peggiore): pensare che un thread si metta in attesa sullo spinlock. Non ci si mette in attesa su uno spinlock, lo spinlock serve per mutua esclusione, non per sincronizzazione di tipo wait-signal.

- E' possibile che la chiamata alla `wchan_wakeone` svegli più di un ***thread*** in attesa su `wchan_sleep`? Se NO, perché? Se SI, come si fa in modo di rilasciare un solo thread in attesa su `P`?

NO. C'è la garanzia che la `wchan_wakeone` svegli un solo thread in attesa su `wchan_sleep`. La funzione che sveglia più thread in attesa (tutti) è la `wchan_wakeall`.