Sia dato un sistema di memoria virtuale con paginazione, nel quale vengono indirizzati i Byte. Il sistema dispone di TLB (Translation Look-aside Buffer), su cui si misura sperimentalmente un “hit ratio” del 90%. La tabella delle pagine (“page-table”) viene realizzata con uno schema a due livelli, nel quale un indirizzo logico di 64 bit viene suddiviso (da MSB a LSB) in 3 parti: *p1*, *p2* e *d*, rispettivamente di 40 bit, 12 bit e 12 bit. Non si utilizzano ulteriori strutture dati (quali tabelle di hash o inverted page table) per velocizzare gli accessi. La memoria virtuale viene gestita con paginazione a richiesta.

Si risponda alle seguenti domande:

A) Supponendo che non sia presente una memoria cache, e che la memoria RAM abbia tempo di accesso di 200 ns, si calcoli il tempo effettivo di accesso (EAT) per il caso proposto (TLB hit ratio = 90%), assumendo che il tempo di accesso alla TLB sia trascurabile.

B) Si vuole ora tener conto del fatto che il sistema dispone di memoria cache, per la quale si sa che la probabilità di (cache) miss è del 10% (un accesso non in cache ogni 10) e che la cache ha tempo di accesso 30 ns. Si ricalcoli EAT, con gli stessi dati per quanto riguarda la TLB e ipotizzando, nel caso di cache miss, un tempo di accesso alla RAM di 230 ns.

C) Si consideri ora la frequenza di page fault p. Si faccia riferimento al caso della domanda B (presenza di cache) e si ipotizzi che un page fault sia servito in 5 ms. Quanto deve valere p affinché si possa garantire un degrado massimo del 25% per EAT (causato sia dalla TLB che dai page fault)? Si tratta di un valore massimo o minimo per p?

A) Calcolo EAT_{PT} con TLB hit ratio = 90%:

T_{RAM} = 200 ns (RAM access time)
h_{TLB} = 0.9 (TLB hit ratio)
2 level (hierarchical) PT => 2 reads for PT lookup

$EAT_{PT} = h_{TLB}*T_{RAM} + (1-h_{TLB})*3T_{RAM} = (1 + 2*(1-h_{TLB}))T_{RAM} = 1.2 * T_{RAM} = 240$ ns

B) Calcolo EAT_{PT} con TLB hit ratio = 90% e presenza di cache

L'esercizio è lo stesso già svolto nella domanda A, purché si modifichi T_{RAM} tenendo conto della cache. Si indica con h_{cache} la probabilità di cache hit (90%).
$T_{RAM} = (1-h_{cache})*230$ ns + $h_{cache}*30$ ns = 0.1*230 + 0.9*30 ns = 50 ns.
$EAT_{PT} = 1.2 * T_{RAM} =$ 60 ns

C) Calcolo P per garantire un degrado massimo del 25% per EAT.
Si tratta di un valore massimo o minimo per p?

T_{PF} = 5 ms (PF service time)

$EAT_{PF} = (1-p)*EAT_{PT} + p*T_{PF} = (1-p)*60 + p*5*10^{6}$ ns
$EAT_{PF} \leq 1.25 * T_{RAM} = 75$ ns

$(1-p)*60 + p*5*10^{6} \leq 75$
$(5*10^{6} - 60)*p \leq 75 - 60$
$(5*10^{6})*p \leq 15$ (removed negligible term)
$p \leq 15/5*10^{-6} = 3*10^{-6}$

Valor minimo o massimo (dire perché) ? Valor massimo, vista la disequazione, oppure osservando che al crescere della probabilità di Page Fault aumenta EAT_{PF}, che deve essere inferiore e un limite massimo

Sia dato un file di dimensione 160MB, in un file system di dimensione complessiva 400GB. Si vogliono confrontare le possibili organizzazioni di tale file, su file system con allocazione non contigua, secondo gli schemi:

- *"linked list allocation"*
- "File Allocation Table (FAT)"

Si supponga che i blocchi su disco (sia per i dati che per metadati) abbiano dimensione 8KB, e che i puntatori e gli indici abbiano dimensione 32 bit.
Si dica, per ognuna delle 2 soluzioni:

A) Quanti blocchi (di dato e/o di metadato) sono necessari per il file (attenzione: Si richiede il conteggio ESATTO, eventualmente espresso in termini di potenze di 2, ricordando che $1K = 2^{10}$ e $1M = 2^{20}$!) ?

Nel caso della FAT, si dica quale percentuale di FAT è utilizzata per il file.

B) Quante letture in RAM e quanti accessi a disco (per trasferire un blocco in RAM) sono necessari per leggere il byte n. 42070 all'interno del file? Si consideri un ***accesso diretto a file***. Si supponga di non utilizzare buffer cache, e che un puntatore o indice vanga letto in RAM con una sola lettura. Si supponga poi che il File Control Block (FCB) sia già in memoria RAM, che ogni accesso a disco legga o scriva un blocco di 8KB, e che la FAT (qualora utilizzata) sia già in RAM.

A) Quanti blocchi (di dato e/o di metadato) sono necessari per il file (attenzione: Si richiede il conteggio ESATTO, eventualmente espresso in termini di potenze di 2, ricordando che $1K = 2^{10}$ e $1M = 2^{20}$!) ?

Nel caso della FAT, si dica quale percentuale di FAT è utilizzata per il file.

"linked list allocation": 160MB / (8KB-4B) =20491 blocchi dato, non ci sono blocchi di indice o metadato

"FAT": 160MB / 8KB = 20K = 20480 blocchi dato, non ci sono blocchi di indice. La FAT complessivamente occupa 400GB/8KB*4B = 200MB. La percentuale di FAT riservata al file è 160MB/400GB = 2/5K = 0.04%

B) Quante letture in RAM e quanti accessi a disco (per trasferire un blocco in RAM) sono necessari per leggere il byte n. 42070 all'interno del file?

Il byte 42070 si trova nel blocco 5, cioè il sesto, in entrambi i casi. Con linked list, 42070/8188, con FAT 42070/8192.

"linked list allocation": 1 accesso in RAM per recuperare il primo puntatore da FCB, 6 letture sul disco per recuperare i 6 blocchi. 6 letture in RAM (5 puntatori e il byte desiderato).

"FAT": 1 accesso in RAM per recuperare il primo indice da FCB, 5 letture in RAM per scandire la lista nella FAT. 1 lettura di blocco su disco e 1 in RAM per il byte desiderato.

Si consideri un sistema con paginazione. Si dica, per ognuna delle caratteristiche elencate, se sia preferibile una dimensione di pagina maggiore o minore, motivando la risposta.

- Frammentazione
- Dimensione PT
- Sovraccarico I/O
- Numero di errori di pagina
- Località
- Dimensioni/copertura TLB

Si consideri un sistema con paginazione. Si dica, per ognuna delle caratteristiche elencate, se sia preferibile una dimensione di pagina maggiore o minore, motivando la risposta. *(rispondere, per ogni caratteristica, nella colonna che si ritiene più pertinente)*

	PT più grande (spiega perché)	PT più piccolo (spiegare perché)
Frammentazione		La frammentazione interna massima è di una pagina, quindi è meglio una dimensione della pagina più piccola
Dimensione PT	Le dimensioni della pagina più grandi riducono il numero di pagine, quindi PT più piccole	
Overhead I/O	L'overhead (latenza + tempo di ricerca) è (quasi) costante e dominante. Quindi le pagine più grandi vengono trasferite (quasi) gratuitamente e vengono ridotte di numero.	
Numero di page fault	Le pagine più grandi riducono chiaramente il numero di PF	
Località	Si potrebbe dire quali pagine più grandi aumentano la località (un programma con una singola grande pagina è "locale"), ma questo è a scapito di un maggiore utilizzo della memoria.	Se un programma ha una località elevata, le pagine più piccole riducono la quantità complessiva di memoria utilizzata
Dimensioni/copertura TLB	Dato una TLB con una certa quantità di voci, pagine più grandi aumentano la memoria coperta dal TLB	

Si fornisce una rappresentazione (parziale) della funzione Os161 thread_fork.

```
thread_fork(const char *name, struct proc *proc,
            void (*entrypoint)(void *data1, unsigned long data2),
             void *data1, unsigned long data2) {
      struct thread *newthread;
      int result;
      newthread = thread_create(name);
      /* Allocate a stack */
      newthread->t_stack = kmalloc(STACK_SIZE);
      /* Thread subsystem fields */
      newthread->t_cpu = curthread->t_cpu;
      /* Attach the new thread to its process */
      result = proc_addthread(proc, newthread);
      /* Set up the switchframe so entrypoint() gets called */
      switchframe_init(newthread, entrypoint, data1, data2);
      /* Lock the current cpu's run queue and make the new thread runnable */
      thread_make_runnable(newthread, false);

      return 0;
}
```

Basandosi sul codice riportato e su quanto appreso in lezioni e laboratori, si risponda alle seguenti domande:

A) che cos'è entrypoint e a cosa serve nel contesto della creazione di un thread?

B) Lo stack allocato (di dimensione STACK_SIZE) è stack kernel oppure user? (motivare)

C) Che cosa fa la funzione thread_make_runnable? Manda in esecuzione il thread? (se no, motivare, se si, quando?)

D) E' possibile che l'esecuzione arrivi all'istruzione finale (return 0;), oppure, una volta entrati in thread_make_runnable, non è possibile che il controllo sia ritornato alla thread_fork?

A) che cos'è entrypoint e a cosa serve nel contesto della creazione di un thread?

E' il puntatore alla funzione che deve essere eseguita dal nuovo thread. Viene passato alla funzione switchframe_init affinché il nuovo thread, una volta in esecuzione, inizi da questa funzione.

B) Lo stack allocato (di dimensione STACK_SIZE) è stack kernel oppure user? (motivare)

E' lo stack kernel, in quanto si sta generando un thread a livello kernel. lo stack user viene creato nel contesto della generazione di un processo user.

C) Che cosa fa la funzione thread_make_runnable? Manda in esecuzione il thread? (se no, motivare, se si, quando?)

Il thread viene messo nella coda dei thread ready, quindi la sua esecuzione non è immediata, ma dipende dal thread attualmente in esecuzione e dagli altri già in coda.

D) E' possibile che l'esecuzione arrivi all'istruzione finale (return 0;), oppure, una volta entrati in thread_make_runnable, non è possibile che il controllo sia ritornato alla thread_fork?

L'esecuzione ritorna alla thread_fork e quindi si effettua la return. La funzione non va confusa con la enter_new_process, che invece (chiamata dalla runprogram) non ritorna al programma chiamante, in quanto trasforma il thread kernel corrente in processo user

Si consideri, in OS161, la realizzazione della gestione della memoria virtuale. Si risponda alle seguenti domande
A) Si supponga di voler realizzare in dumbvm la gestione dei frame liberi mediante free-list, invece che con bitmap (come fatto nella soluzione proposta per il lab 2).
A1) si dica se la free list può essere una lista non ordinata oppure se deve essere ordinata. (motivare)

A2) Qualora la lista fosse ordinata, andrebbe ordinata in base a indirizzi fisici oppure logici? (motivare)

B) Si supponga di voler invece realizzare una PAGE TABLE a livello di singolo processo. Viste le dimensioni ridotte della RAM fisica, si ipotizzi che una casella (entry) della PT occupi solo 2 Byte. Si Ipotizzi di NON voler gestire un heap per il processo: il processo contiene quindi unicamente (come fatto in dumbvm) i segmenti di codice, di dati e lo stack. Si sa che:

- La costante PAGE_SIZE è definita come 4096
- Il segmento 1 (codice) inizia all'indirizzo logico 0x400

000 e ha dimensione 10112 Byte.

- Il segmento 2 (dati) inizia all'indirizzo logico 0x412000 e ha dimensione 8028 Byte
- Lo stack ha dimensione 18 pagine, collocate ai massimi indirizzi logici possibili.

B1) Si calcoli la dimensione complessiva dell'address space e, al suo interno, il numero di pagine effettivamente usate (valide).

B2) si calcoli quale sarebbe la dimensione di una page table standard.

B3) si calcoli quale sarebbe la dimensione di una page table gerarchica a due livelli, supponendo di utilizzare 11 bit per p2: si ricorda che l'indirizzo logico è (p1,p2,d).

Può essere sia ordinata che non ordinata ma, per questioni di prestazioni/efficienza, è decisamente opportuno che sia ordinata per indirizzi fisici (crescenti o decrescenti): questo serve a fare in modo che la ricerca di un intervallo di pagine/frame contigui abbia costo lineare (tempo di esecuzione). Con una lista non ordinata, invece, la ricerca sarebbe quadratica.

A2) Qualora la lista fosse ordinata, andrebbe ordinata in base a indirizzi fisici oppure logici? (motivare)

Indirizzi fisici, in quanto si tratta di una free list di frame, e gli intervalli di frame contigui vanno localizzati nello spazio fisico.

B) Si supponga di voler invece realizzare una PAGE TABLE a livello di singolo processo. Viste le dimensioni ridotte della RAM fisica, si ipotizzi che una casella (entry) della PT occupi solo 2 Byte. Si Ipotizzi di NON voler gestire un heap per il processo: il processo contiene quindi unicamente (come fatto in dumbvm) i segmenti di codice, di dati e lo stack. Si sa che:

La costante PAGE_SIZE è definita come 4096

Il segmento 1 (codice) inizia all'indirizzo logico 0x400000 e ha dimensione 10112 Byte.

Il segmento 2 (dati) inizia all'indirizzo logico 0x412000 e ha dimensione 8028 Byte

Lo stack ha dimensione 18 pagine, collocate ai massimi indirizzi logici possibili.

B1) Si calcoli la dimensione complessiva dell'address space e, al suo interno, il numero di pagine effettivamente usate (valide).

La dimensione dell'address space è 2 GB, in quanto gli indirizzi utilizzabili vanno da 0 a 0x7FFFFFFF. Il numero di pagine effettivamente usate è dato dalle pagine dei due segmenti e dallo stack. I due segmenti iniziano a indirizzi multipli di pagina, quindi occupano rispettivamente 3 e 2 pagine (ceil(dimesione/4096)). Lo stack occupa 18 pagine, quinti in totale il numero di pagine valide è 3+2+18 = 23.

B2) si calcoli quale sarebbe la dimensione di una page table standard.

Una PT standard prevede una casella per ogni possibile pagina logica: le pagine logiche (valide e non) sono 2GB/4KB = 512K. La PT occuperà 512K * 2B = 1MB. Si osserverà una PT con quasi tutte le pagine invalid e poche (23) pagine valid.

B3) si calcoli quale sarebbe la dimensione di una page table gerarchica a due livelli, supponendo di utilizzare 11 bit per p2: si ricorda che l'indirizzo logico è (p1,p2,d).

Siccome d ha dimensione 12 bit, restano per p1 9 bit. Quindi la "outer" PT avrà dimensione 2^9*2B = 1KB, mentre una "inner" PT ha dimensione 2^11*2B = 4KB (un frame). Lo stack è coperto da una sola inner PT, per i due segmenti (codice e dati) occorre verificare i valori di p1:
0x00400000 = 00000000 0,1000000 0000,0000 00000000

0x00412000 = 00000000 0,1000001 0010,0000 00000000

Siccome i due segmenti condividono lo stesso valore di p1, è sufficiente per entrambi la stessa inner PT. In conclusione, la PT gerarchica occuperà un frame (usato per un quarto) per la outer PT, e due frame per due inner PT.