Programmazione di Sistema
7 luglio 2017 (teoria)

Nome: …….. Matricola: ……..

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. *(3 punti)* Si consideri la seguente sequenza di riferimenti in memoria nel caso di un programma di 1500 parole, in cui, per ogni accesso, si indica se si tratta di lettura (R) o scrittura (W): W 161, R 311, R 584, W 623, W 570, R 209, R 185, R 1190, R 615, W 946, R 1020, W 1234, R 658, R 1446, W 364.
   Si determini la stringa dei riferimenti a pagine, supponendo che la loro dimensione sia di 150 parole. Si utilizzi un algoritmo di sostituzione pagine di tipo Enhanced Second-Chance, per il quale, al bit di riferimento (da inizializzare a 0 in corrispondenza al primo accesso a una nuova pagina dopo il relativo page fault), si unisce il bit di modifica (modify bit). Si assuma che una pagina venga sempre modificata in corrispondenza a una scrittura (write), che siano disponibili 3 frame e che l'algoritmo operi con il criterio seguente: dato il puntatore alla pagina corrente (secondo la strategia FIFO) si fa un primo giro, senza modificare il reference bit, sulle pagine per localizzare la vittima (l'ordine di priorità è (reference,modify): (0,0), (0,1), (1,0), (1,1)), si fa un secondo giro per raggiungere la vittima ed eventualmente azzerare i reference bit delle pagine "salvate".

   Determinare quali e quanti page fault (accessi a pagine non presenti nel resident set) si verificheranno. Si richiede la visualizzazione (dopo ogni accesso) del resident set, indicando per ogni frame i bit di riferimento e modifica. Si numerino le pagine a partire da 0.

   Utilizzare, per questa domanda, lo schema seguente per svolgere l'esercizio, indicando nella prima riga la stringa dei riferimenti a pagine, nella seconda Read o Write, nelle tre successive (che rappresentano i 3 frame del resident set), le pagine allocate nei corrispondenti frame, indicando per ognuna i bit (reference,modify). Indicare inoltre (sottolineandola, circolettandola o ponendo una freccia), quale pagina si trova in testa al FIFO.
   Nell'ultima riga si indichi la presenza o meno di un Page Fault.

   Head del FIFO indicato con casella grigia

Riferimenti	1	2	3	4	3	1	1	7	4	6	6	8	4	9	2
Read/write	W	R	R	W	W	R	R	R	R	W	R	W	R	R	W
Resident Set	1_{01}	1_{01}	1_{01}	1_{01}	1_{01}	1_{11}	1_{11}	1_{01}	1_{01}	1_{01}	1_{01}	1_{01}	4_{00}	9_{00}	2_{01}
		2_{00}	2_{00}	4_{01}	4_{01}	4_{01}	4_{01}	7_{00}	4_{00}	6_{01}	6_{11}	6_{11}	6_{11}	6_{01}	6_{01}
			3_{00}	3_{00}	3_{11}	3_{11}	3_{11}	3_{01}	3_{01}	3_{01}	3_{01}	8_{01}	8_{01}	8_{01}	8_{01}
Page Fault	X	x	x	x				x	x	x		x	x	x	x

   Numero totale di page fault: …11…….

2. *(4 punti)* Sia dato un file system Unix, basato su inode aventi 13 puntatori (10 diretti, 1 indiretto singolo 1 doppio e 1 triplo). I puntatori hanno dimensione di 32 bit e i blocchi hanno dimensione 1KB. Sia dato un file "d.dat" di dimensione 200 MB. Si rappresenti l'organizzazione del file, calcolando quanti blocchi di dato e di indice sono necessari. Quale è la frammentazione interna di "d.dat"?

R	***Rappresentazione*** *E' sufficiente schematizzare un file che, mediante INode, raggiunga il livello di indirizzamento indiretto triplo.* ***Calcolo numero di blocchi dato*** N_D = 200MB / 1KB = 200K blocchi ***Calcolo numero di blocchi indice*** (1 blocco indice contiene 1KB/4B = 256 puntatori/indici) N bl. dato raggiunti mediante accesso diretto: 10

	N bl. di dato raggiunti mediante indirizzamento ind. singolo: 256 blocchi dato 1 blocco indice N bl. di dato raggiunti mediante indirizzamento ind. doppio: 256*256 = 64K blocchi dato 257 blocchi indice (1 a primo livello e 256 a secondo livello) N bl. di dato raggiunti mediante indirizzamento ind. triplo: 200K – (256*256 + 256 + 10) = 138998 blocchi dato 547 blocchi indice (1 a primo livello, 3 a secondo livello e 543 a terzo livello) ($\lceil 138998/256 \rceil = 543$) **Totale blocchi indice**: 1 + 257 + 547 = 805 **La frammentazione interna** è 0 in quanto la dimensione del file è multipla di 1KB (un blocco)

Si supponga che il file contenga 100000 record di lunghezza variabile (di dimensioni comprese tra 600 e 5000 Byte), e che si sia generato un file di indici "d.ind", contenente, per ogni record di "d.dat", un record a lunghezza fissa di 64 bit (32 per il cognome e 32 per il numero di record logico, cioè un puntatore, in "d.dat"). I record in "d.ind" sono ordinati per cognome. Quanti blocchi di dato e di indice sono necessari per "d.ind" e quale è la sua frammentazione interna?

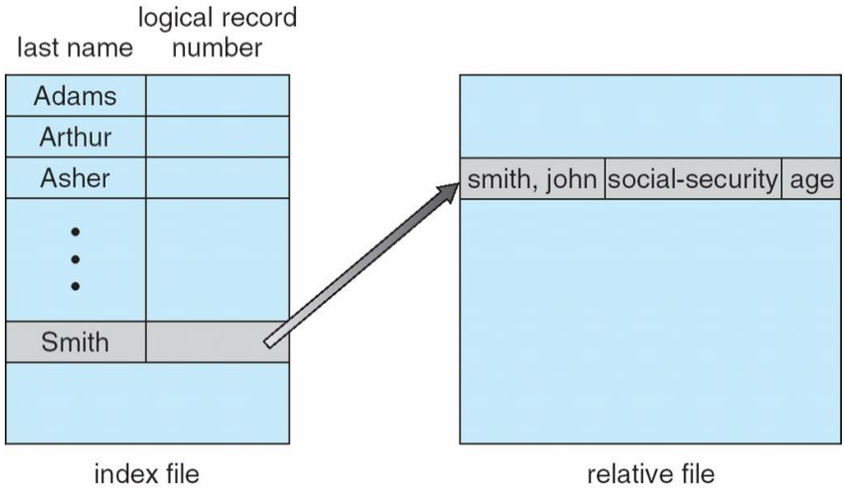
R	***Dimensione file*** |d.ind| = 100000*8B = 800000B ***Calcolo numero di blocchi dato*** $N_D = \lceil 800000B / 1KB \rceil = \lceil 781{,}25 \rceil = 782$ blocchi ***Calcolo numero di blocchi indice*** (1 blocco indice contiene 1KB/4B = 256 puntatori/indici) N bl. dato raggiunti mediante accesso diretto: 10 N bl. di dato raggiunti mediante indirizzamento ind. singolo: 256 blocchi dato 1 blocco indice N bl. di dato raggiunti mediante indirizzamento ind. doppio: 782-(256+1) = 525 blocchi dato 4 blocchi indice (1 a primo livello e 3 a secondo livello) ($\lceil 525/256 \rceil = 3$) **Totale blocchi indice**: 1 + 4 = 5 **Frammentazione interna**: 0,75 blocchi = 768B

Supponendo di utilizzare per la ricerca di un cognome in "d.ind" un algoritmo binario (dicotomico), quanti blocchi di indice e di dato è necessario leggere per portare da disco a memoria i dati su una persona, nel caso peggiore? (NON si tenga conto della gestione di cognomi uguali !)

R	***La ricerca binaria/dicotomico, nel caso peggiore, fa un numero logaritmico (in base 2) di accessi al vettore*** N letture in d.ind = $\log_2 \lceil 100000 \rceil = 17$ Per la ricerca su d.ind si possono dare due interpretazioni (entrambe considerate corrette in fase di correzione) A) Per ogni accesso si rilegge il blocco su disco (anche se è lo stesso di accessi precedenti): si fanno 17 accessi a d.ind B) Quando un blocco ha una copia in RAM si considera che non vada riletto da disco. Si potrebbe tener conto che un blocco dati di d.ind contiene 128 record (2^7). Quindi gli ultimi 7 accessi nella ricerca dicotomica sono nello stesso blocco (al più in due blocchi, considerando che i 128 record su cui converge la ricerca potrebbero essere su due blocchi adiacenti). I primi 10 accessi nella ricerca dicotomica sono invece a blocchi distinti. In totale, la ricerca dicotomica accede al più a 12 blocchi. A questo punto di accede a d.dat, in modo diretto, per leggere il dato. Nel caso peggiore, il record in d.dat occupa 5000 byte, cioè 5 blocchi (caso peggiore 6 se si considera che il record può iniziare a metà di un blocco). Per portare in memoria i dati (il record di d.dat) su una persona, occorre leggere nel caso peggiore A) 17+6 = **23 blocchi dato** (17 in d.ind e 6 in d.dat). B) 12+6 = **18 blocchi dato** (12 in d.ind e 6 in d.dat).

	Per ogni accesso a d.ind occorre, nel caso peggiore, leggere 2 blocchi indice (ind. indiretto doppio). Per ogni accesso a d.dat, nel caso peggiore, si leggono 3 blocchi indice (ind. indiretto triplo). In totale, nel caso peggiore, per effettuare ricerca e caricamento dei dati, si leggono A) 17*2 + 6*3 = **52 blocchi indice**. B) 12*2 + 6*3 = **42 blocchi indice**.

A titolo di esempio, si riporta uno schema (dal Silberschatz) in cui il file a sinistra (index file) può corrispondere al file "d.ind", mentre il file a destra (relative file) a "d.dat" (le dimensioni e i contenuti dei record sono semplificati).



3. *(3 punti)* Si consideri il problema della gestione di una richiesta di IO a blocchi realizzato mediante DMA. Che cosa si intende, in questo contesto, con il termine "*cycle stealing*"? Perché il trasferimento in DMA è vantaggioso rispetto all'IO programmato? Immaginando di dover trasferire 40KB di dati da disco a memoria RAM, quanti Byte transitano sul bus dati della RAM nei due casi (trasferimento in DMA e IO programmato)? (motivare la risposta). Qualora si voglia usare per l'IO un buffer in memoria Kernel, in che cosa il doppio buffer si differenzia dal singolo buffer e per quale motivo può risultare vantaggioso?

R	*(Si mettono in evidenza, in modo sintetico, gli aspetti principali che una risposta corretta dovrebbe contenere)*
	
	• Il "cycle stealing" è la sottrazione (con attesa per la CPU) di cicli di BUS alla CPU, mentre il DMA ha il controllo dei BUS di accesso alla RAM
	• Per due motivi principali:
	  ○ Perché i trasferimenti mediante DMA fanno passare i dati "direttamente" tra IO e RAM (senza passare dalla CPU, con un numero doppio di operazioni)
	  ○ Perché mentre si trasferiscono dati in DMA la CPU può fare altro (aumentando il grado di multiprogrammazione)
	• Transitano 40KB in DMA e 80KB (i 40KB passano due volte, in quanto debbono transitare nella CPU) nel caso di IO programmato.
	• Il doppio buffer è una tecnica nella quale mentre un buffer viene scritto l'altro (riempito in precedenza) può essere letto in parallelo. Permette quindi una forma di pipelining. Con iul buffer singolo uno dei due (chi scrive o chi legge) deve invece attendere il completamento dell'altra operazione. *(ATTENZIONE; si parla qui di doppio buffer di kernel, non di dualità buffer kernel e buffer user)*

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

Come vengono definiti gli indirizzi logici user e kernel? Quali sono gli intervalli di valori ammessi per entrambi?

R	OS161 segue per lo spazio degli indirizzi logici (su 32 bit) lo schema MIPS. Lo spazio logico è unico in quanto non è diviso in uno spazio user e uno kernel, ma comprende una sezione per gli indirizzi user (da `0x00000000` a `0x7fffffff`) e una per gli indirizzi kernel (da `0x80000000` a `0xffffffff`). Gli indirizzi kernel sono a loro volta suddivisi in 3 sezioni: kseg0, 0.5 GB (non mappati in TLB, con utilizzo di cache) a partire da `0x80000000`, kseg1, 0.5 GB (non mappati in TLB, senza utilizzo di cache) a partire da `0xa0000000` (usato per i dispositivi di IO), kseg2, non usato in OS161. L'intervallo degli indirizzi di kernel legali in OS161 è quindi da 0x80000000 a `0xbfffffff` (oppure fino a `0xc0000000` escluso)

Sia dato un processo P, il cui addrspace (nella versione DUMBVM) viene visualizzato (mediante opportune kprintf) nel modo seguente:

```
AS segment 1) L: 0x400000   - P: 0x43000 - size:  2 pages
AS segment 2) L: 0x412000   - P: 0x47000 - size:  2 pages
AS stack    ) L: 0x7ffee000 - P: 0x49000 - size: 12 pages
Page size: 4096 bytes
```

(L e P rappresentano, rispettivamente, indirizzi logici e fisici)

Quanta RAM è stata allocata al processo P?

R	Si sono allocate 16 pagine da 4KB, quindi 64KB

Siano dati gli indirizzi fisici `0x440A0`, `0x45200`, `0x48100` e `0x51018`. Si dica se appartengono o meno al processo P. In caso affermativo, si determinino i corrispondenti indirizzi logici.

R	Per rispondere occorre, di ogni segmento, conoscere l'intervallo degli indirizzi fisici (si noti che 3 cifre esadecimali corrispondono a 12 bit, quelli necessari per l'offset in una pagina da 4KB):
	`segmento 1: 0x43000 – 0x44fff`
	`segmento 2: 0x47000 – 0x48fff`
	`stack     : 0x49000 – 0x54fff`
	Quindi
	`0x440A0` appartiene a segmento 1
	`0x45200` NON appartiene a P
	`0x48100` appartiene a segmento 2
	`0x51018` appartiene allo stack

Si dica poi, per ognuno dei 4 indirizzi (indipendentemente dal fatto che appartengano o meno a P), a quale indirizzo logico corrisponderebbero, se visti come indirizzi di kernel.

R	Si ricorda che il kernel non usa la tlb, ma traduce da indirizzi fisici a logici sommando semplicemente 0x80000000 (`MIPS_KSEG0`), compito svolto in OS161 dalla macro `PADDR_TO_KVADDR`.
	Quindi
	`0x440A0 -> 0x800440A0`
	`0x45200 -> 0x80045200`
	`0x48100 -> 0x80048100`
	`0x51018 -> 0x80051018`

4.a) Commentare il seguente frammento di codice relativo alla funzione cv_wait, spiegando il significato delle singole istruzioni. Correggere eventuali errori presentando, se necessario, l' implementazione corretta (giustificandola).

```
cv_wait(struct cv *cv, struct lock *lock) {
   …
   lockRelease(lock);
   SpinlockAcquire(&cv->cv_lock);
   wchanSleep(cv->cv_wchan,&cv->cv_lock);
   LockAcquire(Lock);
   Spinlock_Release(&cv->cv_lock);
   …
}
```

R	La funzione cv_wait ha come obiettivo mettere il processo/thread in attesa sulla condition variable cv, rilasciando contestualmente il lock. La struct cv contiene un wait channel pr gestire l'attesa e uno spinlock per protezione in mutua esclusione. L'attesa viene realizzata mediante wchanSleep su `cv->cv_wchan` Il lock viene rilasciato prima dell'attesa e ottenuto nuovamente prima di ritornare. La funzione proposta è errata nella gestione dello spinlock, per due motivi: • La `LockAcquire(Lock)` (funzione che può mettere in thread in wait) non può essere chiamata con spinlock acquisito. Si genererebbe una richiesta di ulteriore spinlock interna alla

	`LockAcquire`, non ammessa in OS161. La `Spinlock_Release` va quindi fatta prima di `LockAcquire`.
	• Il rilascio del lock, fatto senza spinlock già acquisito, non garantisce atomicità di wait e rilascio del lock, violando la semantica della condition variable. La `SpinlockAcquire` va fatta prima della `lockRelease`.
	La versione corretta della funzione è quindi:
	
	`SpinlockAcquire(&cv->cv_lock);`
	`lockRelease(lock);`
	`wchanSleep(cv->cv_wchan,&cv->cv_lock);`
	`Spinlock_Release(&cv->cv_lock);`
	`LockAcquire(Lock);`

4.b) SOLO PER STUDENTI CON FREQUENZA SINO AL 2015/2016 compreso: *in sostituzione della domanda 4.a sulle condition variable,* si spieghino il significato e l'utilizzo delle funzioni `splhigh`, `spl0` e `splx`.

R	OS161 usa/gestisce solo 2 deggli 8 livelli di interrupt MIPS. Sono quindi sufficienti le due istruzioni `splhigh`, `spl0` per settare il livello di interruzione ad alto (interrupt disabilitati) o basso (abilitati). Entrambe le istruzioni, oltre a impostare il nuovo valore, ritornano quello vecchio. Qualora, si intenda successivamente ripristinare il precedente livello, la funzione permette di impostare il valore (alto o basso), ricevuto come parametro. Un uso frequente è ad esempio la disabilitazione temporanea degli interrupt con le istruzioni
	`L0 = spl0();`
	`… /* interupt disabilitato */`
	`Splx (L0); /*re-imposta livello alto o basso precedente */`