Programmazione di Sistema
20 gennaio 2020 (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. Le due parti 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. ***LE RISPOSTE SI/NO VANNO MOTIVATE***

Soluzione proposta: *si intenda la soluzione come indicativa, in particolare nel caso di domande a risposta aperta.*

1. *(4 punti)* Si consideri la seguente sequenza di riferimenti in memoria nel caso di un programma in cui, per ogni accesso (indirizzi in esadecimale, si indirizza il Byte), si indica se si tratta di lettura (R) o scrittura (W): **R 315**, R 3A4, R 465, **W 343**, W 241, W 3CE, R 3C8, W 2A0, **R 42A**, W 3E6, R 270, R 354, R 2B8, R 44E.
   Supponendo che sia indirizzi fisici che logici siano su 12 bit, che si usi paginazione con pagine di dimensione 128 Byte e che il massimo indirizzo utilizzabile dal programma sia D20, si dica quante pagine sono presenti nello spazio di indirizzamento del programma e se ne calcoli la frammentazione interna.

   Numero totale di pagine: …27……. ($D20_{Hex}$ = **11010**0100000_{bin}, 26,25 pagine)

   Frammentazione interna: …96B…… (128 -32)B

   Si determini la stringa dei riferimenti a pagine (Si consiglia di passare da esadecimale a binario, per determinare correttamente il numero di pagina e, se necessario, il displacement/offset). Si utilizzi un algoritmo di sostituzione pagine di tipo LRU (Least Recently Used). Si assuma che siano disponibili 3 frame, agli indirizzi fisici (espressi in esadecimale) 580, 900, A80. Si richiede la visualizzazione (dopo ogni accesso) del resident set (i frame fisici contenenti pagine logiche). Indicare inoltre i page fault (accessi a pagine non presenti nel resident set).

Riferimenti		**6**	7	8	**6**	4	7	7	5	**8**	7	4	6	5	8
Resident Set	580	6	6	6	6	6	6	6	5	5	5	4	4	4	8
	900		7	7	7	4	4	4	4	8	8	8	6	6	6
	A80			8	8	8	7	7	7	7	7	7	7	5	5
Page Fault		x	x	x		x	x		x	X		X	x	x	X

Indicare nella prima riga la stringa dei riferimenti a pagine (rappresentate a scelta in esadecimale o decimale), nelle tre successive (che rappresentano i 3 frame del resident set), le pagine allocate nei corrispondenti frame. La free frame list sia inizialmente (580, 900, A80). Nell'ultima riga si indichi la presenza o meno di un Page Fault.

Numero totale di page fault: ……11……

Si dica infine a quali indirizzi fisici vengono effettuati gli accessi (tra quelli sopra elencati) R 315, W 343, R 42A

Indirizzi logici → fisici: R 315: …595 (580+15).. W 343: …5C3 (580+43)…. R 42A: …92A (900+2A)……

(esprimere gli indirizzi a scelta in esadecimale o decimale)

2. *(3 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/indici hanno dimensione 32 bit e i blocchi su disco hanno dimensione 1KB. Si sa che il file system contiene N file per i quali è necessario indice indiretto triplo, 20*N file con indice indiretto doppio (non si sa nulla delle altre dimensioni). La partizione di disco riservata ai blocchi di dato ha dimensione 1 TB. Si calcoli il valore limite per N, tale da garantire che tale partizione sia occupata al massimo per il 50%. Si tratta di un limite superiore o inferiore *(motivare)*?

Siccome si lavora a livello di partizione e blocchi liberi/occupati, conviene usare come unità di misura il blocco. Si considerano solo i blocchi di dato, in quanto si sa che la partizione è riservata a questi, quindi non si conteggiano i blocchi quelli di indice (in ogni caso di impatto trascurabile)
(NOTA: se se si misurassero le dimensioni effettive dei file (anziché i blocchi occupati) e/o si approssimassero K,M,G,T con le potenze di 10 vicine, si otterrebbero risultati leggermente diversi rispetto a quelli della soluzione proposta (ma comunque valutati positivamente)

Un blocco contiene 256 indici (1KB/4B)

Per **garantire** che la partizione sia occupata al massimo al 50% è necessario porsi nel **caso peggiore**, cioè la dimensione **massima**, per i file che usino indici indiretti tripli (F_3), come pure doppi (F_2).
ATTENZIONE: Non conoscendo nulla dei file piccoli (indice diretto e indiretto singolo), che potrebbero essere in numero molto elevato, a rigore non si può garantire occupazione sotto il 50%. Supponendo tuttavia che questi ultimi siano trascurabili, o intendendo che si possa garantire la condizione in assenza di file "piccoli", si può calcolare un ***limite superiore (massimo)*** per N, come segue.

$|F_2|_{max} = 10+256+256^2$ blocchi
$|F_3|_{max} = |F_2|_{max} +256^3$ blocchi $= 10+256+256^2+256^3$ blocchi

MAX = 0.5 * 1TB/1KB = 512 M blocchi

$20N*|F_2|_{max} + N*|F_3|_{max} < MAX$
$21*(266+256^2)*N + 256^3*N < 512M$
$N < 29,56$
Essendo N intero: $N \leq 29$

Si calcoli poi, assumendo che la partizione sia piena al 50% (e che N e 20*N siano, rispettivamente, i file con indirizzamento triplo e doppio), quale è il massimo numero di file possibili, nel file system, con indice indiretto triplo e doppio

In questo caso non occorre garantire lo riempimento massimo, in quanto si sa già che la partizione è occupata al massimo al 50%. Per calcolare il numero massimo di file, occorre utilizzare l'occupazione minima:

$|F_2|_{min} = 10+256+1$ blocchi
$|F_3|_{min} = |F_2|_{min} +256^2$ blocchi $= 10+256+256^2+1$ blocchi

$10N*|F_2|_{min} + N*|F_3|_{min} < MAX$
$21*267*N + 256^2*N < 512M$
$N < 7546,36$
Essendo N intero: $N \leq 7546$
Numero massimo file con indice indiretto triplo: 7546
Numero massimo file con indice indiretto triplo: 7546 * 20 = 150927

Si calcoli infine, assumendo che la partizione sia piena al 50% (e che N e 20*N siano, rispettivamente) i file con indirizzamento triplo e doppio), quale è il massimo numeri di file possibili, nel file system, con indice indiretto **singolo**

La situazione è simile alla domanda precedente. Detto N1 il numero massimo di file richiesto, lo si raggiunge nel caso di occupazione minima per i file a indice indiretto doppio e triplo. Ipotizzando N>0 (detto in aula), si assuma quindi N=1 e dimensioni $|F_1|_{min,}$ $|F_2|_{min}$ e $|F_3|_{min}$, con $|F_1|_{min} = 10+1$ blocchi = 11 blocchi

$(21*267 + 256^2) + 11*N1 < 512M$
$N1 < (512M - 21*267 - 256^2) / 11 = 48799979 \approx 48,8*10^6 \approx 46,5M$

Nome: …….. Matricola: ……..

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()?

open()
la open riceve come parametro il nome del file, deve aprirlo nella modalità richiesta e ritornare il corrispondente file descriptor (o handle/puntatore, a seconda del sistema operativo)

1) il nome del file viene cercato utilizzando la struttura dati (in memoria e/o disco) basata sui direttori
2) se ha successo, la ricerca ritorna il File Control Block (copia nella system-wide open-file table, già presente in precedenza oppure generate ora).
3) viene creata una nuova entry nella per-process open-file table (che fa riferimento al FCB nella system-wide), di cui si ritorna il puntatore/handle o indice (file descriptor)

close()
la close deve chiudere il file, eliminando le relative entry nelle tabelle dei file aperti, qualora non più utili

1) la corrispondente entry nella per-process open-file table viene cancellata
2) Si decrementa il contatore dei riferimenti nella tabella system-wide, se questo diventa 0, si cancella l'entry anche da questa

   A quali strutture dati (tabelle di gestione file) si deve fare accesso per realizzare una read() e/o una write()?

Si fa accesso a

1) per-process open-file table, dove, nella entry relativa, si trovano il puntatore alla posizione di lettura/scrittura nel file e la modalità di accesso
2) system-wide open-file table. Vi si deve accedere, tra l'altro, per fare la conversione da indirizzo logico nel file a fisico (numero di blocco e offset nel blocco)

   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?

Le tabelle contengono tutte le informazioni necessarie in memoria per gestire correttamente i file. Sono necessarie entrambe perché un file può essere aperto contemporaneamente più volte (in modalità diverse), sia nel contesto di un singolo processo (eventualmente con più thread) che da parte di più processi.
Le informazioni comuni, nella system-wide, includono una copia del FCB ed eventuali primitive per gestire/sincronizzare accessi condivisi e altro.
Le informazioni nella per-process includono il puntatore al file e la modalità di accesso (es. lettura/scrittura)

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

   a. 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)

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 supponga che nel sistema si sia realizzata la `sys_waitpid()`, e che quest'ultima utilizzi come primitiva di sincronizzazione una condition variable. Si completi la `my_sys_exit` riportata nel seguito, fornendone una breve spiegazione:

```
my_sys_exit(int status) {  // lo stato viene ricevuto come parametro
  struct proc *p = curproc;  // serve per poter accedere al processo dopo averlo
                             // staccato da curthread (curproc non più valido)

  p->p_status = status;  // salva lo stato di ritorno per la waitpid

  proc_remthread(curthread);  // stacca il processo dal thread

  // segnala al processo che fa la waitpid (per usare cv_signal è
  // necessario possedere il relativo lock)
  lock_acquire(p->p_lock);
  cv_signal(p->p_cv,p->p_lock);
  lock_release(p->p_lock);

  // meglio NON fare as_destroy qui (lo farà la proc_destroy)
  // il thread finisce qui (diventa zombie)
  thread_exit();
}
```

La chiamata dovrà essere del tipo:

```
my_sys_exit((int)tf->tf_a0);
```

b. Perché in OS161, per gestire gli argomenti al main, è necessario creare una copia di `argv` e `argc`?

E' necessario in quanto gli argomenti al main debbono essere in memoria user, affinchè il programma utente possa farvi accesso. Siccome gli argomenti al main sono in origine in memoria kernel, si rende necessario farne una copia

Dove va creata tale copia?

Va fatta in memoria user, In particolare, in una parte accessibile dell'address space: la soluzione più semplice è l'inizio (indirizzi alti) dello (user) stack.

Perché non sono sufficienti i valori originali nargs e args, passate alla cmd_prog (menu.c: avente prototipo `static int cmd_prog(int nargs, char **args)`), a partire da una stringa di comando?

Il motivo principale è già stato citato: i valori originali sono in memoria kernel, quindi non accessibili al processo user.
Si potrebbe tuttavia aggiungere che, quand'anche il processo potesse accedervi, si tratterebbe di dati nello stack di un altro (kernel) thread, quello del menu, quindi da duplicare in ogni caso, a meno di garantirne la consistenza e accessibilità da un altro thread.

(Per completezza, si veda ad esempio la funzione `cmd_dispatch`, il cui vettore (locale) `args` sarà quello che viene ricevuto come `argv` da `cmd_prog`)