SISTEMI OPERATIVI – 19 febbraio 2013

corso A nuovo ordinamento e parte di teoria del vecchio ordinamento indirizzo SR

Cognome:	Nome:
Matricola:	

- 1. Ricordate che non potete usare calcolatrici o materiale didattico, e che potete consegnare al massimo tre prove scritte per anno accademico.
- 2. Gli studenti a cui sono stati riconosciuti i 3 cfu di "linguaggio C" devono rispondere solo alle domande delle parti di teoria e di laboratorio Unix, e consegnare entro 1 ora e trenta minuti.
- 3. Gli studenti del vecchio ordinamento, indirizzo SR, devono rispondere solo alle domande della parte di teoria, e devono consegnare entro 1 ora.

ESERCIZI RELATIVI ALLA PARTE DI TEORIA DEL CORSO

ESERCIZIO 1 (5 punti)

Tre processi P_A, P_B e P_C eseguono il seguente codice:

Shared **Var** semaphore mutex = 1; (valore iniziale) semaphore done = 1; (valore iniziale)

 P_A : P_B : P_C :

repeat forever: repeat forever: repeat forever: wait(mutex) wait(done) wait(done) <A> wait(mutex) wait(mutex) wait(done) signal(mutex) wait(mutex)

signal(done) signal(mutex) <C>

signal(done) signal(mutex)

a1)

L'esecuzione concorrente di P_A, P_B e P_C produce una sequenza (di lunghezza indefinita) di chiamate alle procedure A, B e C. Quali delle sequenze qui sotto riportate possono essere la porzione iniziale di sequenze prodotte dall'esecuzione concorrente di P_A, P_B e P_C? (marcate le sequenze che scegliete con una croce nello spazio apposito)

A2)

In ciascuna delle sequenze restanti, che non possono essere prodotte dall'esecuzione concorrente dei tre processi, cerchiate/sottolineate la lettera che rappresenta l'ultima procedura che può effettivamente essere eseguita nell'esecuzione concorrente di P_A , P_B e P_C

- 1. [] $A,C,A,B,B,A,\underline{C},B,A...$
- 2. [X] A,C,A,B,A,B,C,A,B ...
- 3. [] B,A,C,B,A,A,C,A,B ...
- 4. [X] B,A,A,C,B,B,A,C,A...

b)
Riportate lo pseudocodice della prima "soluzione" al "problema dei 5 filosofi" vista a lezione.

filosofo *i*:

```
do{
    wait(bacchetta[i])
    wait(bacchetta[i+1 mod 5])
    ...
    mangia
    ...
    signal(bacchetta[i]);
    signal(bacchetta[i+1 mod 5]);
    ...
    pensa
    ...
} while (true)
```

semaphore bacchetta[i] = 1;

c) Perché la soluzione riportata al punto b) non è esente da deadlock?

Perché se tutti i filosofi riescono a prendere ciascuno una sola delle due bacchette (bacchetta[i]) si crea un'attesa circolare.

d)
All'interno di un sistema operativo, un certo processo P è correntemente in stato di "Ready to Run", e si sa che, una acquisita la CPU non dovrà più rilasciarla volontariamente prima di aver terminato la propria esecuzione (in altre parole, non deve più eseguire operazioni di I/O, di sincronizzazione o di comunicazione con altri processi. Assumete anche che non sia presente la memoria virtuale).

Ouale/quali tra gli algoritmi di scheduling ECES. SIE preemptive. SIE pop-preemptive round robin.

Quale/quali, tra gli algoritmi di scheduling FCFS, SJF preemptive, SJF non-preemptive, round robin garantisce/garantiscono che il processo P riuscirà a portare a termine la propria computazione? (motivate la vostra risposta, assumendo che SJF possa effettivamente essere implementato)

FCFS, e round robin. Infatti, nel caso di SJF preemptive e non, potrebbe sempre arrivare in coda di ready un processo che deve usare la CPU per un tempo minore di quanto rimane da eseguire a P.

e) Si consideri un processo P nella classe *time sharing* (e con priorità maggiore di 0) nello scheduling del sistema Solaris. Come sarà gestito lo scheduling di P nel futuro, se P ha appena rilasciato la CPU perché è scaduto il suo quanto di tempo?

P viene rimesso in coda di ready e gli viene data una priorità più bassa di prima. Quando verrà selezionato per tornare in esecuzione gli verrà assegnato un quanto di CPU più lungo del precedente.

ESERCIZIO 2 (5 punti)

In un sistema la memoria física è divisa in 2²¹ frame, un indirizzo logico è scritto su 32 bit, e all'interno di una pagina, l'offset massimo è 3FF.

a) Quanti byte occupa la la page table più grande del sistema? (motivate numericamente la vostra risposta)

Un frame/pagina è grande $2^{10} = 1024$ byte, e quindi la page table più grande può avere $2^{(32-10)} = 2^{22}$ entry. Nel sistema vi sono 2^{21} frame, per cui sono necessari tre byte per scrivere il numero di un frame, e quindi la page table più grande occupa $(2^{22} \cdot 3)$ byte = 12 Mbyte

b) Quale dimensione minima dovrebbero avere le pagine di questo sistema per essere certi di non dover ricorrere ad una paginazione a più livelli? (motivate la vostra risposta, e per questa domanda assumete di usare 4 byte per scrivere il numero di un frame all'interno di una page table)

Poniamo 32 = m + n (m = bit usati per scrivere un numero di pagina, n = bit usati per scrivere l'offset). Allora il numero di entry della PT più grande, moltiplicato per la dimensione di una entry deve poter essere contenuto in una pagina/frame, ossia: $2^m \cdot 2^2 \le 2^n$

Da cui: $m + 2 \le n$. Poiché m = 32 - n; risolvendo il semplice sistema si ha $n \ge 17$, ossia le pagine devono almeno essere grandi $2^{17} = 128$ Kbyte.

c) Quale vantaggio si ha nell'usare, in un sistema, una Inverted Page Table anziché normali tabelle delle pagine?

Che si risparmia spazio in RAM, in quanto si deve allocare un'unica IPT per tutto il sistema anziché una page table per ciascun processo.

ESERCIZIO 3 (4 punti)

Un hard disk ha la capacità di 32 gigabyte, è formattato in blocchi da 200 (esadecimale) byte, e usa una qualche forma di allocazione indicizzata per memorizzare i file su disco. Sull'hard disk è memorizzato un file A grande 150 Kbyte. Nel rispondere alle domande sottostanti, specificate sempre le assunzioni che fate.

- a) Quante operazioni di I/O su disco sono necessarie per portare in RAM l'ultimo blocco del file A, assumendo che inizialmente sia presente in RAM solo la copia del file directory che "contiene" il file?
 - L'hard disk contiene $2^{35}/2^9 = 2^{26}$ blocchi, e sono quindi necessari 4 byte per scrivere in numero di un blocco. Un blocco indice può quindi contenere al massimo 512/4 = 128 numeri di blocco. La risposta dipende poi dal tipo di allocazione indicizzata assunta:
 - 1) Allocazione indicizzata a schema concatenato: sono necessari 3 blocchi indice per tenere traccia di tutti i blocchi del file. Se assumiamo che sia già in RAM il numero del primo blocco indice, sono necessarie 4 operazioni di I/O: lettura dei tre blocchi indice più lettura del blocco del file.
 - 2) Allocazione indicizzata a più livelli. È sufficiente usare uno schema a due livelli. Se assumiamo che sia già in RAM il numero del blocco indice esterno, sono necessarie 3 operazioni di I/O: lettura del blocco indice esterno, lettura di un blocco indice interno, lettura del blocco del file.
 - 3) Allocazione indicizzata Unix: Assumendo già in RAM il numero dell'index-node, sono necessarie 4 operazioni di I/O: le tre del punto 2) precedute dalla lettura dell'index-node.
- b) Nel caso di accesso ai dati di file molto piccoli, è più efficiente l'implementazione scelta in Windows con NTFS o quella scelta da Unix con gli index-node? (motivate la vostra risposta)
 - NTFS. Infatti per file molto piccoli, l'elemento che contiene gli attributi del file può contenere anche i dati del file stesso.

c) Che cosa indica il campo *link counter* di un index-node?

Il numero di hard link ad un file in un file system Unix.

d) Che differenza c'è tra un sistema RAID di livello 4 ed uno di livello 5?

Nel livello 5 gli strip di parità sono distribuiti omogeneamente tra tutti i dischi, che quindi vengono sollecitati in modo uniforme. Nel livello 4 gli strip di parità risiedono su un unico disco, che viene quindi sollecitato mediamente più di tutti gli altri, dovendo essere aggiornato ad ogni modifica, aggiunta o cancellazione di un qualsiasi file

ESERCIZI RELATIVI ALLA PARTE DI UNIX (6 punti)

|--|

(1.1) Supponendo date			ectory corren		myprog.f	myprog.o	
1 1 1 0	isultato di du	_					
ls *p*							
ls help[23]							
Soluzione [slides (01_introduzio	one_UNIX.po	df, slide 16]			((1 punti)
$\frac{ls\ help?}{ls\ help*}$ help1 $\frac{ls\ *p*}{ls\ help[23]}$ help1 $\frac{ls\ help[23]}{ls\ help}$	help2 help2 help2 2 help3	help23	help3 help3	myprog.	f myprog	.0	
(1.2) Cosa sono gl spiegare a cosa co			comando <i>ar</i> {	gc e argv? Ill	ustrare tipo e fi	anzione di cias	scuno, e
						([1 punti]
Soluzione [02_inte	egrazione_lin	guaggio_e_r	ipasso, slide	24]			

I due argomenti della funzione *main()* rendono disponibili gli argomenti della linea di comando all'interno del programma.

Il primo argomento è l'intero <u>argv</u>: indica quanti argomenti sono presenti; il secondo argomento, *char** argv[] è un array di puntatori agli argomenti della linea di comando, ciascuno dei quali è una stringa terminata dal carattere '\0'.

La prima di queste stringhe, *argv[0]*, è convenzionalmente associata al nome del programma stesso.

ESERCIZIO 2 (2 punti)

La syscall *wait()* serve a informare il processo padre del fatto che uno dei figli ha cambiato stato, è terminato o è stato bloccato da un segnale.

La system call *wait()* attende che uno dei processi figli del chiamante termini, e scrive lo stato di terminazione di quel figlio nel buffer puntato dall'intero status.

La *wait()* restituisce il process ID del figlio che ha terminato la propria esecuzione. Se nessun figlio del processo chiamante ha già terminato, la chiamata si blocca finché uno dei figli termina. Se un figlio ha già terminato al momento della chiamata, *wait()* restituisce immediatamente.

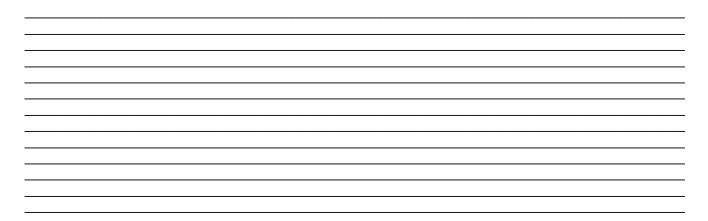
Se status non è *NULL*, l'informazione sulla terminazione del figlio è assegnata all'intero cui punta *status*.

In caso di errore, *wait()* restituisce -1. Un possibile errore è che il processo chiamante potrebbe non avere figli: in questo caso la *wait()* restituisce -1, e *errno* è settato al valore *ECHILD*.

ESERCIZIO 3 (2 punti)

Illustrare la system call *signal()* a partire dal prototipo sottostante, e spiegare in breve cos'è un handler di segnali.

```
void (*signal(int sig, void (*handler)(int))) (int);
```



Soluzione [slides 09 syscall kill signal.pdf, slides 9 e 10]

I sistemi UNIX forniscono due modi per cambiare la disposizione di un segnale: *signal()* e *sigaction()*. La system call *signal()* è l'API originale per assegnare la disposizione di un segnale.

Il primo argomento della *signal()*, *sig*, identifica il segnale di cui vogliamo modificare la disposizione. Il secondo argomento, *handler*, è l'indirizzo della funzione che dovrebbe essere chiamata quando questo segnale è inviato.

Questa funzione non restituisce (è *void*) e prende un intero. Quindi un handler di segnali ha la seguente forma generale:

```
void handler(int sig) {
   /* codice dell'handler */
}
```

Il valore di ritorno di *signal()* è la precedente disposizione del segnale. Come l'argomento dell'handler, questo è un puntatore a funzione che non restituisce nulla e che prende un intero come argomento.

ESERCIZI RELATIVI ALLA PARTE DI C

ESERCIZIO 1 (2 punti)

Si implementi la funzione con prototipo

```
int find (char * str, char c, int n);
```

find è una funzione che cerca nella stringa str la n-esima occorrenza del carattere c e ne restituisce la posizione. Restituisce -1 se non lo trova.

```
int find (char * str, char c, int n) {
    int pos = -1;
    while (*str != '\0' && n>0) {
        if (*str == c) n--;
            str++; i++;
    }
    if (n == 0) return i; else return -1;
}
```

ESERCIZIO 2 (3 punti)

Si implementi la funzione con prototipo

```
node * inserisci(node * head, int value)
```

che inserisce in modo ordinato un elemento con valore value nella lista puntata da head. Si supponga che la lista sia ordinata per valori crescenti di value. Si noti che la lista puo' essere inizialmente vuota.

La struttura node è definita come segue:

```
typedef node* link;
typedef struct node {
     int value;
     link next;
} node;
node * inserisci(node * head, int value)
     node * newnode = (node*)malloc(sizeof(node));
     newnode->value = value;
     newnode->next = NULL;
     node * prec = NULL;
     node * first = head;
     while (head != NULL && value > head->value) {
          prec = head; head = head->next;
     }
     if (head != NULL)
                        newnode->next = head;
     if (prec == NULL) first = newnode;
     else prec->next = newnode;
     return first;
}
```

ESERCIZIO 3 (2 punti)

Data la lista dell'esercizio precedente, implementare la funzione con prototipo

```
link find Min inList(link head);
```

find_Min_inList è una funzione che trova l'elemento con valore piu` piccolo nel campo value e restituisce l'elemento della lista corrispondente. Se la lista e` vuota, la funzione restituisce NULL.

Funzione da implementare

```
link find_Min_inList(link head) {
    return head;
}
```