SISTEMI OPERATIVI
3 febbraio 2015
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)

a) Si consideri il problema dei lettori e scrittori visto a lezione, dove i codici del generico scrittore e del generico lettore sono riportati qui di seguito.

Inserite le istruzioni mancanti necessarie per il funzionamento del sistema secondo la soluzione vista a lezione, , indicando anche il semaforo e i valori di inizializzazione mancanti.

semafori e variabili condivise necessarie con relativo valore di inizializzazione:
semaphore mutex = 1;
semaphore scrivi = 1;
int numlettori = 0;

"scrittore"
{
wait(scrivi);
Esegui la scrittura del file
signal(scrivi)
}

"lettore"
{
wait(mutex);

numlettori++;

if numlettori == 1 **wait(scrivi);**

signal(mutex);

... leggi il file ...

wait(mutex);

numlettori--;

if numlettori == 0 **signal(scrivi);**

signal(mutex);

b) Si consideri l'algoritmo del punto precedente. In un dato istante, mentre un processo scrittore sta aggiornando il file, più processi lettori tentano di accedere in lettura al file. Dove si addormentano i processi lettori?

Il primo lettore si addormenta sul semaforo *scrivi*, gli altri si addormentano sul semaforo *mutex*.

c) Sia dato un sistema operativo time sharing che implementa la memoria virtuale. Indicate separatamente le cause per cui un processo in stato *running* potrebbe abbandonare **volontariamente** e **involontariamente** il processore.

volontariamente	**involontariamente**
termina	quanto di tempo scaduto
operazione di I/O	page fault
chiamata di fork	
chiamata di wait con semaforo < 1	

d) Tre processi P1 P2 e P3 arrivano uno dopo l'altro – nell'ordine indicato ma in tempi diversi – in coda di ready. I tre processi sono caratterizzati da un unico burst di CPU e da nessun burst di I/O. In quale caso il turnaround medio del sistema relativo ai tre processi risulta lo stesso sia adottando un algoritmo di scheduling FCFS che adottando una algoritmo di scheduling SJF preemtpive?

È sufficiente che: Burst_P1 ≤ Burst_P2 ≤ Burst_P3

ESERCIZIO 2 (5 punti)

In un sistema che adotta la memoria paginata le pagine hanno una dimensione tale da produrre una frammentazione interna media per processo di 1 Kbyte. La dimensione della tabella delle pagine più grossa del sistema è di X byte, e si sa che se tale dimensione fosse anche solo X+1 byte, il sistema dovrebbe adottare una paginazione a più livelli. Ogni entry di una tabella delle pagine occupa 2 byte, e tutti i bit vengono usati per scrivere il numero di un frame.

a) Quanto sono grossi lo spazio di indirizzamento logico e fisico del sistema?

Il sistema adotta pagine da 2 Kbyte, ossia 2^{11} byte. La tabella delle pagine più grossa del sistema occupa un intero frame, e quindi ha in tutto $2^{11}/2 = 2^{10}$ entry. Lo spazio logico è quindi grande $2^{10} * 2^{11} = 2^{21}$ byte (2 megabyte) e quello fisico è grande $2^{16} * 2^{11} = 2^{27}$ byte (128 megabyte)

b) Perché un sistema con le caratteristiche del punto a) potrebbe voler implementare la memoria virtuale?

Per poter far girare contemporaneamente un insieme di processi che occupano complessivamente uno spazio logico maggiore di quello fisico.

c) È possibile che un processo P che gira su un sistema A dotato di memoria virtuale venga eseguito più velocemente che su un sistema B del tutto identico ad A ma non dotato di memoria virtuale? (motivate la vostra risposta)

Si, ad esempio se P non genera mai page fault e non è necessario caricare in memoria tutto il codice di P per l'esecuzione.

d) perché l'algoritmo della seconda chance è una approssimazione di LRU?

Perché usando il reference bit riesce a discriminare tra pagine riferite di recente e non riferite di recente

e) Per evitare il problema del thrashing conviene aumentare la dimensione della RAM o la dimensione dell'area di SWAP? (motivate la vostra risposta)

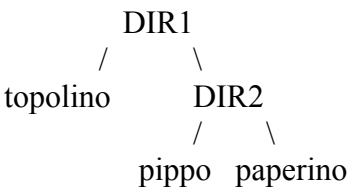
La dimensione della RAM perché così diminuisce la probabilità che un processo generi page fault.

ESERCIZIO 3 (4 punti)

a) Considerate la seguente sequenza di comandi Unix (assumete che l'utente abbia tutti i permessi per eseguire i comandi indicati):

```
1:    cd /tmp
2:    mkdir DIR1
3:    cd DIR1
4:    mkdir DIR2
5:    cd DIR2
6:    echo "ciao" > pippo          // crea un nuovo file di nome pippo
7:    ln /tmp/DIR1  DIR3
8:    ln pippo  ../topolino
9:    ln ../topolino  paperino
```

a) Disegnate la porzione di file system cosi ottenuta a partire da /tmp, inclusi i nomi dei file contenuti nelle varie cartelle



b) quanto vale il link counter di paperino? 3
Quanto vale il link counter di DIR1? 3

c) se nell'elenco di comandi sopra riportato si usassero link simbolici al posto dei link fisici, cambierebbe qualcosa nel disegno che avete riportato al punto a) e/o nelle risposte che avete dato al punto b)? (motivate la vostra risposta).

Si, il comando 7 avrebbe successo, per cui dentro DIR2 comparirebbe anche DIR3. Il link counter di paperino varrebbe 1 (essendo paperino associato ad un nuovo i-node).

d) perché conviene adottare un RAID di livello 5 anziché 4?

Perché così gli strip di parità sono distribuiti su tutti i dischi e non c'è un disco più sollecitato degli altri.

ESERCIZI RELATIVI ALLA PARTE DI UNIX (6 punti)

ESERCIZIO 1

(2 punti)

(1.1) Considerato il prototipo

*pid_t waitpid(pid_t pid, int *status, int options);*

i) illustrare cosa restituisce la system call *waitpid*, e *ii)* spiegare quali valori può assumere il primo argomento, *pid*.

Soluzione [slides 03_controllo_processi.pdf, slide 40 e seguenti]

i) La system call restituisce il process ID di un figlio, *0*, o *–1* in caso di errore. In particolare:
- se non ci sono figli, restituisce *–1* impostando *errno* a *ECHILD*;
- se è specificato il flag *WNOHANG* e non ci sono figli in stop o usciti, restituisce *0*;
- se si verifica un errore o arriva un segnale di terminazione, restituisce -1 e *errno* è assegnato al corrispondente errore.

ii) L'argomento *pid* permette di selezionare il figlio da aspettare, secondo queste regole:
- se *pid > 0*: attendi per il figlio con quel *pid*.
- se *pid == 0*: attendi per qualsiasi figlio nello stesso gruppo di processi del chiamante (padre).
- se *pid < -1*: attendi per qualsiasi figlio il cui process group è uguale al valore assoluto di *pid*.
- se *pid == -1*: attendi per un figlio qualsiasi. La chiamata *waitpid(–1, &status, 0)* è dunque equivalente a *wait(&status)*.

ESERCIZIO 2

(2 punti)

(2.1) Ipotizzando di avere effettuato una chiamata semget

semid = semget(IPC_PRIVATE, 1, S_IRUSR | S_IWUSR);

si scrivano le istruzioni successive (incluse le dichiarazioni di variabili necessarie per la chiamata alla system call) per inizializzare a zero il semaforo identificato da *semid*.

Soluzione [slides 09_semafori.pdf]

```
union semun arg;
arg.val = 0;

if ( semctl(semid, /* semnum= */ 0, SETVAL, arg) == -1 )
     // gestione errore ...
```

ESERCIZIO 3

(2 punti)

Nel contesto dell'uso dei semafori, illustrare le operazioni *IPC_RMID*, *IPC_SET*, e fornire un esempio di invocazione della system call appropriata per ciascuna operazione.

Soluzione [slides 09_semafori.pdf, slides 18 e sgg.]

La system call utilizzata per le operazioni in questione è la *semctl()*, il cui prototipo è

int semctl(int semid, int semnum, int cmd, ...);

La *semctl()* esegue l'operazione di controllo specificata da *cmd* sul set di semafori identificato da *semid*, o sul semaforo di indice *semnum* (i semafori sono numerati a partire da 0).

Le operazioni *IPC_RMID*, *IPC_SET* possono quindi essere specificate tramite *cmd*, e sono le stesse comuni anche agli altri tipi di oggetti IPC di SystemV. In particolare,

- *IPC_RMID*: rimuove immediatamente il set di semafori e l'associata struttura *semid_ds*. Qualsiasi processo bloccato in chiamate *semop()* in attesa su semafori è immediatamente svegliato, e *semop()* riporta l'errore *EIDRM*;
- *IPC_SET*: Aggiorna i membri della struttura *semid_ds* associata al set di semafori utilizzando i valori nel buffer puntato da *arg.buf*.

L'operazione *IPC_SET* fa riferimento alla struttura riferita dal membro *buf* della unione *semun*:

```
union semun {
   int val;
   struct semid_ds *buf; /* Buffer for IPC_STAT, IPC_SET */
   unsigned short *array;
   struct seminfo *__buf; /* (Linux specific) */
};
```

La struttura *buf*, di tipo *semid_ds**, contiene le informazioni relative a ownership e permessi, *timestamp* dell'ultima *semop*, dell'ultima modifica al semaforo, e relative al numero di semafori nel pool.

ESERCIZI RELATIVI ALLA PARTE DI C

ESERCIZIO 1 (2 punti)

Si implementi la funzione con prototipo

```
int str_equal_spacing(char * str, int spacing);
```

`str_equal_spacing()` è una funzione che restituisce TRUE se gli elementi di `str` a partire dal primo che sono separati di `spacing` posizioni sono tutti uguali tra loro e FALSE altrimenti. Ricordate di definire opportunamente TRUE e FALSE e di gestire i casi:
- puntatore NULL
- stringa vuota
- valori particolari di spacing

Esempi:

se la stringa fosse: d f d g d r d g d e `spacing` fosse 2 allora la funzione restituirebbe TRUE
se la stringa fosse: d f d g d r d g d e `spacing` fosse 1 allora la funzione restituirebbe FALSE
se la stringa fosse: d f d g d r d g d e `spacing` fosse 4 allora la funzione restituirebbe TRUE
se la stringa fosse: d f d g d r d g k e `spacing` fosse 4 allora la funzione restituirebbe FALSE
se la stringa fosse: d f d g d r d g k e `spacing` fosse 2 allora la funzione restituirebbe FALSE

```
#define TRUE 1
#define FALSE 0

int str_equal_spacing(char * str, int spacing) {
int ret_value = TRUE;

if(str != NULL && spacing > 0) {

     int length = strlen(str);
     int pos;

     for(pos=0;pos+spacing<length && ret_value==TRUE;pos=pos+spacing)
       if( *(str+pos) != *(str+pos+spacing) )
          ret_value = FALSE;
     return ret_value;
}
}
```

ESERCIZIO 2 (3 punti)

Quale sarebbe l'output di questo programma C? E quale sarebbe se l'istruzione del main() con etichetta decl: fosse commentata?

```
#include <stdio.h>
int x=10;

void f(int x){
        printf("%d\n",x);
        if(x > 20) {
                int x = 20;
                printf("%d\n",x);
        }
        else {
                int x = -20;
                printf("%d\n",x);
        }
}
main() {
decl:   int x = 60;

        printf("%d\n",x);
        f(145);
}
```

Risposta primo caso: 60 145 20
Risposta secondo caso: 10 145 20

ESERCIZIO 3 (2 punti)

Data la struttura `node` definita come segue:

```
typedef struct node {
        int value;
        struct node * next;
} nodo;
typedef nodo* link;
```

implementare la funzione con prototipo

```
int more_than_threshold(link head, int value, int threshold);
```

`more_than_threshold` è una funzione che restituisce:

1 se il numero di elementi della lista uguali a `value` è maggiore del parametro `threshold`
0 altrimenti.

```
int more_than_threshold(link head, int value, int threshold){
        int ret_value = 0;
        int nelem = 0;

        while (head != NULL) {
          if(head->value==value)
               nelem++;
          head = head->next;
        }
        if( nelem > threshold)
            ret_value = 1;
        return ret_value;
}
```