SISTEMI OPERATIVI – 7 luglio 2014
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 produttori e consumatori, con buffer limitato ad *m* elementi, dove i codici del generico produttore e del generico consumatore sono i seguenti:

semafori necessari con relativo valore di inizializzazione:
semaphore mutex = 1;
semaphore full = 0;
semaphore empty = m;

"consumatore"	*"produttore"*
repeat	**repeat**
	<produci dato>
wait(full)	*wait(empty)*
wait(mutex)	*wait(mutex)*
<preleva dato dal buffer>	
	<inserisci dato nel buffer>
signal(mutex)	*signal(mutex)*
signal(empty)	*signal(full)*
<consuma dato>	
forever	**forever**

Inserite le opportune operazioni di wait e signal necessarie per il corretto funzionamento del sistema, indicando anche i semafori necessari ed il loro valore di inizializzazione.

b) Riportate lo pseudocodice che descrive l'implementazione delle operazioni di Wait e Signal

Si vedano i lucidi della sezione 6.5.2

c) Perché è meglio evitare soluzioni al problema della sezione critica basate sul busy waiting?

Perché fanno sprecare inutilmente tempo di CPU

d) Spiegate perché gli algoritmi di schedulino FCFS e SJF pre-emptive (supponendo sia implementabile) non sono adatti per implementare un sistema time saring

Perché non garantiscono che un processo possa entrare in esecuzione in una quantità di tempo limitata

ESERCIZIO 2 (5 punti)

Un sistema con memoria paginata usa un TLB con un hit-ratio del 80%, e un tempo di accesso di 20 nanosecondi. Un accesso in RAM richiede invece 0,08 microsecondi.

a) Qual è, in nanosecondi, il tempo medio di accesso (Medium Access Time) in RAM (esplicitate i calcoli che fate)?

ma = 0,80 * (80+20) + 0,20 * (2*80 + 20) = 80 + 36 = 116 nanosecondi

b) Il sistema viene ora dotato di memoria virtuale, usando come algoritmo di rimpiazzamento quello della *seconda chance migliorato*. Si supponga pari a 10 millisecondi il tempo medio necessario a gestire un page fault. Se vogliamo una degradazione massima dell'Effective Access Time del 20% rispetto alla versione del sistema senza memoria virtuale, quale deve essere il valore massimo di "p", ossia la probabilità che si verifichi un page fault? (esplicitate il ragionamento e le formule che usate, e arrotondate in modo ragionevole i numeri con virgole decimali. È sufficiente esprimere "p" mediante la formula che permette di calcolarlo)

A partire dalla formula:

eat = (1-p)·ma + p·"tempo di gestione del page fault"

usando i dati del problema, sapendo che vogliamo eat massimo pari a ma + 10% ma, abbiamo:

max eat = 116 + 0,2·116 = 139,2 ≅ 140 ≥ (1-p) ·116 + p · 10.000.000

e risolvendo rispetto a p abbiamo:

140 ≥ 116 + p · (10.000.000 -116);
p < (140 -116) / (10.000.000 -116)
p < 24/10.000.000
(ossia all'incirca meno di un page fault ogni 400.000 riferimenti in memoria)

c. E' possibile che un processo di questo sistema sistema si veda aumentare il numero di frame che usa e contemporaneamente aumenta anche il numero di page fault generati dal processo?

Si, perché nel caso peggiore l'algoritmo si comporta come FIFO, che soffre dell'anomalia di Belady.

d. Quali informazioni conterrà ciascuna entry di una page table di un sistema che usa l'algoritmo della seconda chance migliorato?

Numero di un frame, bit di validità, bit di riferimento, dirty bit.

ESERCIZIO 3 (4 punti)

Un hard disk ha la capienza di 2^{40} byte, ed è formattato in blocchi da 4096 byte.

a) Quanti accessi al disco sono necessari per leggere l'ultimo blocco di un file A della dimensione di 17.000 byte, assumendo che sia già in RAM il numero del primo blocco del file stesso e che venga adottata una allocazione concatenata dello spazio su disco? (motivate la vostra risposta)

5. Ogni blocco infatti memorizza 4092 byte di dati più 4 byte di puntatore al blocco successivo (infatti, $2^{40}/2^{12} = 2^{28}$), per cui sono necessari 5 blocchi per memorizzare l'intero file.

b) Quanto sarebbe grande, in megabyte, la FAT di questo sistema? (motivate numericamente la vostra risposta)

La FAT è un array con una entry per ciascun blocco dell'hard disk e che contiene il numero di un blocco, per cui: 2^{28} x 2^2 byte = 1 gigabyte

c) Quali sono gli svantaggi nell'uso della FAT?

Per garantire un accesso efficiente ai file deve essere sempre tenuta in RAM, se viene persa si perdono tutte le informazioni sul file system, e deve quindi essere periodicamente salvata su disco.

d) Disegnate la FAT di un hard disk formato da 2^4 blocchi e contenente un unico file memorizzato, nell'ordine, nei blocchi 9, 5, 11, 3. Indicate anche dove è memorizzato il numero del primo blocco del file.

Si veda la figura 11.7 della sezione 11.4.2 dei lucidi.

ESERCIZI RELATIVI ALLA PARTE DI UNIX (6 punti)

ESERCIZIO 1 (2 punti)

(1.1) Illustrare il significato delle istruzioni alle line da 4 a 6:

(1 punti)

```
1: #!/bin/bash
2: VAR=PROVA
3:
4: echo "il valore è $VAR"
5: echo `ls -l $VAR`
6: echo '$VAR'
```

Soluzione [slides 14_intro_bash.pdf, pp. 14 e sgg.]

L'istruzione a linea 4 stampa la stringa contenuta fra apici doppi sostituendo a $VAR il suo valore; l'istruzione a linea 5 effettua la sostituzione, interpreta come comando e restituisce il risultato dell'esecuzione del comando (che dipende dalla presenza o meno del file in questione); l'istruzione a linea 6 non effettua alcuna sostituzione, e viene pertanto stampata letteralmente la stringa "$VAR".

Analizzare la seguente definizione di macro; illustrare cosa accade durante l'invocazione sottostante, e quale valore è assegnato alla variabile *result*.

(1 punti)

```
#define MAX(x, y) x > y ? x : y
...
int a=1, b=2;
int result = 5 + MAX(a,b);
```

Soluzione [slides 02_integrazione_linguaggio_e_ripasso.pdf, pp. 41 e sgg.]

Il preprocessore sostituisce le direttive al preprocessore (quindi la macro) prima della compilazione, quindi il codice effettivamente compilato sarà

```
int result = 5 + a > b ? a : b ;
```

La somma *5+a* è eseguita prima del test; poiché *5+a* è maggiore di *2*, la variabile *result* viene assegnata con il valore di *a*, cioè *1*.

ESERCIZIO 2 (2 punti)

(2.1) A cosa serve la system call *waitpid()*? Nel rispondere considerare eventuali argomenti e/o valori restituiti.

(1 punti)

Soluzione [slides 03_creazione_terminazione_processi.pdf, pp. 34 e sgg.]

La *wait* sospende l'esecuzione del processo chiamante finché uno dei figli non termina la propria esecuzione e restituisce il process ID del figlio che ha terminato la propria esecuzione. Permette

semplicemente di attendere che uno dei figli del chiamante termini. La *waitpid* consente di specificare il *pid* del figlio di cui attendere la terminazione. Il prototipo è

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

In particolare,
- 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 *wait(&status)* equivale a *waitpid(–1, &status, 0)*.

(2.2) Illustrare a cosa serve l'utility *make*, come si usa un Makefile, quali sono gli elementi principali del Makefile, e come implementare il comando *make clean*.

Soluzione [slides 02_integrazione_linguaggio.pdf, pp. 18 e 19]

L'utility make è uno strumento che può essere utilizzato per automatizzare il processo di compilazione.

Si basa sull'utilizzo di un file (il *makefile*, appunto) che descrive le dipendenze presenti nel progetto e si avvale delle informazioni relative alla marcatura temporale dei file. In tal modo è possibile ricompilare solo i target file più vecchi dei sorgenti.

Un Makefile contiene un insieme di *target*, le regole per la compilazione e l'istruzione da eseguire per compilare a partire dai sorgenti. In particolare le *righe di dipendenza* illustrano da quali file oggetto dipende il file target; tale riga inizia dalla colonna 1, per esempio

```
nome_target: file_1.o file_2.o file_n.o
```

Le *righe d'azione* o *di comando* indicano come il programma deve essere compilato nel caso sia stato modificato almeno uno dei file *.o*; deve iniziare con una *tabulazione*. Per esempio,

```
	gcc -o nome_target file_1.o file_2.o file_n.o
```

Per implementare il comando *make clean:*

```
clean:
	rm -f *.o
```

ESERCIZIO 3 (2 punti)

Illustrare il funzionamento della system call *msgsnd()* e fornire un esempio di utilizzo.

Soluzione [slides 06_code_messaggi.pdf, pp. 34 e seguenti]

La syscall *msgsnd()* invia un messaggio a una coda di messaggi. Il suo prototipo è

int **msgsnd**(int *msqid*, const void **msgp*, size_t *msgsz*, int *msgflg*);

restituisce 0 in caso di successo, -1 in caso di errore. Il primo argomento è un intero che funge da l'identificatore della coda; il secondo è un puntatore a una struttura definita dal programmatore utilizzata per contenere il messaggio inviato. L'argomento *msgsz* indica la dimensione del messaggio (espresso in byte), mentre l'ultimo argomento è una bit mask di flag che controllano l'operazione di invio. Per esempio utilizzando IPC_NOWAIT, nel caso la coda di messaggi sia piena, invece di bloccarsi in attesa che si renda disponibile dello spazio, la *msgsnd()* restituisce

immediatamente (con errore *EAGAIN*).

Un'invocazione tipica della syscall è quindi
msgsnd(m_id, &q, sizeof(q), IPC_NOWAIT)

ESERCIZI RELATIVI ALLA PARTE DI C

ESERCIZIO 1 (2 punti)

Si implementi la funzione con prototipo

```
int str_all_different(char * str);
```

`str_all_different()` è una funzione restituisce 0 se la stringa `str` contiene almeno una coppia di elementi uguali tra loro ed 1 altrimenti. Ipotizzate che la funzione sia sempre richiamata con parametro `str` diverso da NULL e sempre di lunghezza almeno pari a 2.

```
int str_all_different(char * str) {
int ret_value = 1;
int length = strlen(str);
int first,second;


for(first=0;first<length-1 && ret_value==1;first++)
    for(second=first+1;second<length && ret_value==1;second++)
       if( *(str+first) == *(str+second) )
          ret_value = 0;
return ret_value;
}
```

ESERCIZIO 2 (3 punti)

Si implementi la funzione con prototipo

```
int sum_equal_length(int * numbers, int length);
```

`sum_equal_length` è una funzione che restituisce 1 se la somma degli elementi dell'array di numeri interi `numbers` (la cui lunghezza è data dal parametro `length`) è un multiplo intero di `length` e 0 altrimenti. Ipotizzate che la funzione sia sempre richiamata con parametro `numbers` diverso da NULL.

```
int sum_equal_length(int * numbers, int length){
int index,sum=0;

for(index=0; index < length; index++)
     sum = sum + numbers[index];

if(sum%length == 0)
     return 1;
else
     return 0;
}
```

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 length_equal_value(link head, int value);
```

`length_equal_value` è una funzione che restituisce:

1 se la lista è composta da `value` elementi e se l'ultimo elemento è uguale a `value`
0 altrimenti.

Si ipotizzi che value sia sempre maggiore di 0

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

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