SISTEMI OPERATIVI – 11 giugno 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)

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

Shared **Var** semaphore mutex = 1; (valore iniziale)
semaphore done = 2; (valore iniziale)

P_A:	P_B:	P_C:
repeat forever:	**repeat forever**:	**repeat forever**:
wait(done)	**wait**(done)	**wait**(mutex)
wait(done)	**wait**(mutex)	<C>
wait(mutex)	<B>	**signal**(mutex)
<A>	**signal**(mutex)	**signal**(done)
signal(mutex)	**signal**(done)	

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. [X] B,A,C,B,C,B,A,C,C …
2. [ ] C,A,**B**,A,C,C,A,B,C …

3. [ ] B,C,C,A,B,**A**,B,C,C…
4. [X] A,C,C,C,B,A,C,A,C …

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) La soluzione riportata al punto b) è esente da starvation? E' esente da deadlock? (Motivate le vostre risposte)

La soluzione 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. Di conseguenza non è esente da starvation.

d) Elencate le tecniche che un SO usa per conservare sempre il controllo della macchina anche quando non sta girando.

Uso di istruzioni privilegiate, uso di un timer che restituisce il controllo al SO quando scade, uso di registri speciali per controllare l'accesso ad aree di ram riservate ai vari processi (oppure, paginazione e/o segmentazione)

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 a causa del fatto che 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 fisica è divisa in 2^{19} frame, un indirizzo logico è scritto su 30 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 $2^{(30-10)} = 2^{20}$ entry. Nel sistema vi sono 2^{19} frame, per cui sono necessari tre byte per scrivere il numero di un frame, e quindi la page table più grande occupa ($2^{20} \cdot$ 3) byte = 3 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 30 = 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: $2^m \cdot 2^2 \leq 2^n$
Da cui: $m + 2 \leq n$. Poiché $m = 30 - n$; risolvendo il semplice sistema si ha $n \geq 16$, ossia le pagine devono almeno essere grandi 2^{16} = 64 Kbyte.

c) C'è qualche svantaggio nell'usare, in un sistema, una Inverted Page Table anziché normali tabelle delle pagine?

Si, per assicurare una efficiente traduzione degli indirizzi è necessario tenere la IPT in una memoria associativa.

ESERCIZIO 3 (4 punti)

a) Considerate la seguente sequenza di comandi Unix (assumete che tutti i comandi lanciati possano essere correttamente eseguiti):

```
1:   cd /tmp
2:   mkdir mynewdir
3:   cd mynewdir
4:   mkdir anotherdir
5:   echo "ciao" > pippo          // crea un nuovo file di nome pippo contenente la stringa ciao
6:   ln pippo paperino
7:   ln –s pippo pluto
8:   ln paperino topolino
9:   rm pippo
10:  cat pluto                   // cat stampa il contenuto del file passato come argomento
11:  cd anotherdir
12:  mkdir aseconddir
```

Dopo l'esecuzione di tutti i comandi:
qual è il valore del link counter nell'index-node associato al link fisico *topolino*? 2
qual è il valore del link counter nell'index-node associato al link fisico *anotherdir*? 3
cosa possiamo dire del link counter dell'index-node associato al link fisico *tmp*? Che è aumentato di 1, a causa dell'entry ".." inserita dentro la nuova sottocartella mynewdir.
Qual è l'output del comando numero 10? "no such file or directory"

b) Descrivete brevemente il funzionamento del livello RAID 4 e la variante RAID 5 (se preferite potete rispondere disegnando un esempio di RAID 4 e di RAID 5)

Il RAID 4 suddivide gli strip di dati tra i vari dischi, riservando un disco per gli strip di parità. Nel RAID 5 gli strip di parità sono distribuiti fra tutti i dischi.

c) Quali vantaggi e svantaggi ci sono nell'usare un RAID 5 anziché un RAID 1, a parità di dimensione e numero di dischi usati?

Vantaggi: maggiore spazio di memorizzazione disponibile. Svantaggi: maggiore lentezza media di accesso ai dati, maggiore tempo di recupero dei dati nel caso di guasto di un disco, e in generale minore efficienza nel recupero di un guasto, perché il sistema deve essere fermato, mentre nel RAID 1 può continuare a funzionare.

ESERCIZI RELATIVI ALLA PARTE DI UNIX (6 punti)

ESERCIZIO 1 (2 punti)

(1.1) Illustrare due fra i seguenti comandi, illustrando un esempio di invocazione:

(0.5 punti)

pwd *more* *cat*

Soluzione [slides 01_introduzione_UNIX.pdf]

pwd Scrive il path assoluto della directory di lavoro corrente.
more Visualizza il file che costituisce l'argomento del comando una schermata alla volta. Esempio: *more nomefile*.
cat Legge il file che costituisce il suo argomento scrivendone il contenuto su standard output.

(1.2) Illustrare il significato della riga sottostante, ipotizzando che `a` e `c` siano nomi di file e `b` sia un programma

(0.5 punti)

```
a < b > c
```

Soluzione [slides 01_introduzione_UNIX.pdf, p. 43]

Si tratta di un esempio di combinazione di redirezione dell'input e dell'output: in questo caso il comando `b` prende in input il file `a` e dopo averlo elaborato invia il proprio output in un nuovo file chiamato `c`.

(1.3) Considerare la seguente macro e illustrare se vi sono errori e perché. Nel caso si riscontrino possibili errori, fornire un esempio di invocazione della macro e la corretta formulazione.

(1 punti)

```
#define SQ(x) x * x
```

Soluzione [slides 02_integrazione_linguaggio.pdf, pp. 31 e seguenti]

La definizione in questione dovrebbe creare una macro che restituisce il quadrato di un numero. Poiché manca la parentesizzazione, la seguente istruzione `SQ(7+3)` darebbe per esempio un risultato inatteso (`31`): perché sarebbe calcolata come 7+3*7+3=31. La formulazione corretta è

```
#define SQ(x) ((x) * (x))
```

ESERCIZIO 2 (2 punti)

(2.1) Illustrare *brevemente* cosa sono le direttive al preprocessore, e illustrare il significato della direttiva `#ifdef identifier`.

(1 punto)

Soluzione [slides 02_integrazione_linguaggio.pdf, p. 7]

Si tratta di istruzioni elaborate dal preprocessore: sono contenute in righe che iniziano con il simbolo #. Possono contenere espressioni complesse (come costrutti if-then-else) che permettono di scrivere meta-programmi: le direttive sono interpretate e rimosse –dal preprocessore– prima che il codice arrivi al compilatore.
#ifdef identifier è utilizzata per testare se *identifier* è stato definito, nel qual caso la porzione di codice contenuta fra *#ifdef* e *#endif* sarà compilata; la stessa porzione di codice non sarà compilata in caso contrario.

(2.2) 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)*.

ESERCIZIO 3 (2 punti)

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

Soluzione [slides 07_code_messaggi.pdf, pp. 12 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 check_occurrences(char * str, int n_target, char check_char);
```

`check_occurrences()` è una funzione che restituisce TRUE se gli elementi di `str` uguali a `check_char` sono esattamente in numero `n_target` e FALSE altrimenti. Ricordate di definire opportunamente TRUE e FALSE e di gestire il caso in cui `str` sia NULL.

Esempi:

- se la stringa fosse: d f d g d r d g d, `n_target` fosse 2 e `check_char` fosse g allora la funzione restituirebbe TRUE
- se la stringa fosse: d f d g d r d g d, `n_target` fosse 2 e `check_char` fosse d allora la funzione restituirebbe FALSE
- se la stringa fosse: d f d g d r d g d, `n_target` fosse 1 e `check_char` fosse f allora la funzione restituirebbe TRUE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TRUE 1
#define FALSE 0

int check_occurrences(char * str, int n_target, char check_char){
        int ret_value = FALSE;

        if(str != NULL) {
                int pos;
                int n_occurrences = 0;
                for( pos = 0; pos < strlen(str); pos++)
                        if(*(str+pos)==check_char)
                                n_occurrences++;
                if(n_occurrences == n_target)
                        ret_value = TRUE;

        }
        return ret_value;
}

main() {
       char str[20]="dfdgdrdgd";

       int n_target = 2;
       char check_char = 'g';
       printf("String %s n_target %d check_char %c results
       %d\n",str,n_target,check_char,check_occurrences(str,n_target,check_char));

       n_target = 2;
       check_char = 'd';
       printf("String %s n_target %d check_char %c results
       %d\n",str,n_target,check_char,check_occurrences(str,n_target,check_char));

       n_target = 1;
       check_char = 'f';
       printf("String %s n_target %d check_char %c results
       %d\n",str,n_target,check_char,check_occurrences(str,n_target,check_char));
}
```

ESERCIZIO 2 (2 punti)

Qual è l'output del seguente programma C?
E quale sarebbe l'output se nel programma l'istruzione

```
int x = 60;
```

fosse commentata?

```
#include <stdio.h>

int x=20;

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() {
        int x = 60; // Da commentare

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

Risposta primo caso: 60 60 20
Risposta secondo caso: 20 20 -20

ESERCIZIO 3 (3 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 in_range(link head, int value, int min, int max);
```

`in_range()` è una funzione che restituisce:

TRUE se il numero di elementi della lista uguali a `value` è compreso tra `min` e `max`
FALSE altrimenti.

```c
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0


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

typedef nodo* link;


int in_range(link head, int value, int min, int max){
        int ret_value = FALSE;
        int nelem = 0;

        while (head != NULL) {
                if(head->value==value)
                        nelem++;
                head = head->next;
        }

        if(nelem >= min && nelem <= max)
                ret_value = TRUE;
        return ret_value;
}

main() {

       link prova = (link)malloc(sizeof(nodo));
       prova->value=3;
       prova->next=NULL;
       link prova2 = (link)malloc(sizeof(nodo));
       prova2->value=3;
       prova2->next=prova;

       link prova3 = (link)malloc(sizeof(nodo));
       prova3->value=3;
       prova3->next=prova2;

       int min = 1;
       int max = 2;
       printf("value 3 min %d max %d result %d\n",min,max,in_range(prova3,3,min,max));

}
```