SISTEMI OPERATIVI – 11 giugno 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)

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(mutex)	**wait**(done)	**wait**(done)
<A>	**wait**(mutex)	**wait**(done)
signal(mutex)	<B>	**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,**B**,C,A,A,C,B,A …
2. [X] B,C,A,B,A,B,C,A,A …
3. [X] C,A,A,A,B,C,A,C,A …
4. [ ] B,A,A,C,B,**C**,B,A,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) 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)
All'interno di un sistema operativo, un certo processo P è correntemente in stato di "running", e si sa che, una volta 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).
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, SJF non-preemptive e round robin. Infatti, nel caso di SJF preemptive 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 volontariamente, ossia prima che scadesse il suo quanto di tempo?

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

ESERCIZIO 2 (5 punti)

In un sistema la memoria fisica è divisa in 2^{21} 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: $2^m \cdot 2^2 \leq 2^n$
Da cui: $m + 2 \leq n$. Poiché $m = 32 - n$; risolvendo il semplice sistema si ha $n \geq 17$, 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)

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 pluto
7:   ln –s pippo paperino
8:   ls pluto topolino
9:   rm pippo
10:  cat topolino                 // cat stampa il contenuto del file passato come argomento
11:  mkdir aseconddir
```

Dopo l'esecuzione di tutti i comandi:
qual è il valore del link counter nell'index-node associato al link fisico *pluto*? 1
qual è il valore del link counter nell'index-node associato al link fisico *mynewdir*? 4
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) Tra i diversi livelli di sistema RAID visti a lezione, qual è quello che garantisce in media la maggior velocità di lettura dei dati memorizzati sui dischi del sistema?

Il livello 1, poiché grazie al mirroring, anche la lettura di due strip memorizzate sullo stesso disco può avvenire in parallelo sfruttando il disco di mirroring.

c) 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)

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 strcouple(char * str);
```

`strcouple` è una funzione restituisce 1 se la stringa `str` contiene almeno una coppia di elementi contigui uguali tra loro e 0 altrimenti. Ipotizzate che la funzione sia sempre richiamata con parametro `str` diverso da NULL e mai con la stringa vuota.

```
int strcouple(char * str) {
        int ret_value = 0;

        while (*(str+1) != '\0' && ret_value==0) {
                if((*str) == *(str+1))
                        ret_value = 1;
                str++;
        }
        return ret_value;
}
```

ESERCIZIO 2 (3 punti)

Si implementi la funzione con prototipo

```
int sum_in_interval(int * numbers, int length, int min, int max);
```

`sum_in_interval` 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 compresa nell'intervallo definito da `min` e `max`, e 0 altrimenti. Ipotizzate che la funzione sia sempre richiamata con parametro `numbers` diverso da NULL.

```
int sum_in_interval(int * numbers, int length, int min, int max){
        int index,arraysum=0;

        for(index=0; index < length; index++){
             arraysum = arraysum + numbers[index];
        }
        if(arraysum >= min && arraysum <= max)
                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 verify_extreme_elements(link head, int value);
```

`verify_extreme_elements` è una funzione che restituisce:

1 se l'ultimo elemento della lista è uguale al primo ma entrambi diversi da `value`
2 se l'ultimo elemento della lista è uguale al primo ed entrambi sono uguali a `value`
0 altrimenti.

Se la lista è vuota la funzione restituisce, invece, -1.

```
int verify_extreme_elements(link head, int value){
        int ret_value = -1;
        int first_value = -1;
        int last_value = -1;

        if(head != NULL) {
                first_value = head->value;
                while (head->next != NULL) {
                        head = head->next;
                }
                last_value = head->value;
                ret_value = 0;

                if (first_value == last_value) {
                        ret_value = ret_value + 1;
                        if(first_value == value)
                                ret_value = ret_value + 1;
                }
        }
        return ret_value;
}
```