SISTEMI OPERATIVI – 26 gennaio 2017
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 15 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(done)	**wait**(done)	**wait**(mutex)
wait(mutex)	**wait**(mutex)	<C>
<A>	<B>	**signal**(mutex)
signal(mutex)	**signal**(mutex)	**signal**(done)
	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)

1. [X] B,A,C,B,C,A,A,C,B …
2. [ ] C,A,B,A,B,C,A,B,C …
3. [X] B,C,C,A,B,A,C,B,C…
4. [ ] A,C,A,C,B,A,B,C,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) Qual è la differenza tra un insieme di peer thread e un insieme di processi? Che vantaggio da l'uso dei thread al posto dei processi?

Un insieme di peer thread condivide lo spazio di indirizzamento. Per questa ragione, il context switch tra peer thread e la creazione di un nuovo peer thread richiedono molto meno tempo delle corrispondenti operazioni sui processi.

ESERCIZIO 2 (5 punti)

In un sistema la memoria fisica è divisa in 2^{16} frame, un indirizzo logico è scritto su 29 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 $2^{(29-10)} = 2^{19}$ entry. Nel sistema vi sono 2^{16} frame, per cui sono necessari 2 byte per scrivere il numero di un frame, e quindi la page table più grande occupa ($2^{19} \cdot 2$) byte = 1 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?

Poniamo 29 = 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: $2^m \cdot 2^1 \leq 2^n$

Da cui: $m + 1 \leq n$. Poiché $m = 29 – n$; risolvendo il semplice sistema si ha $n \geq 15$, ossia le pagine devono almeno essere grandi $2^{15} = 32$ Kbyte.

c) assumendo che nel sistema possano essere presenti al massimo 256 processi, e facendo riferimento ai dati iniziali del problema, quanto sarebbe grande la IPT del sistema? (Motivate numericamente la vostra risposta)

La IPT ha un numero di entry pari al numero di frame del sistema, e ogni entry della IPT deve contenere il PID di processo (un byte) e il numero di una pagina (3 byte).
Dunque la IPT è grande $2^{16} \cdot$ 4 byte = 256 Kbyte.

d) Quali sono gli **svantaggi** della paginazione della memoria?

Maggior lavoro da parte del sistema operativo (gestione dei frame liberi, aumento del tempo di context switch), maggior overhead della memoria principale a causa della presenza delle tabelle delle pagine, maggior temo di traduzione degli indirizzi da logici a fisici, con conseguente necessità di un supporto hardware per limitare i costi (memoria associativa per implementare il TLB)

ESERCIZIO 3 (5 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:   echo “ciao” > pippo        // crea un nuovo file di nome pippo contenente la stringa ciao
5:   ln pippo paperino
6:   ln –s pippo pluto
7:   ln paperino topolino
8:   rm pippo
9:   cat pluto                  // cat stampa il contenuto del file passato come argomento
10:  cd ..
11:  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 *mynewdir*? 2
cosa possiamo dire del link counter dell’index-node associato al link fisico *tmp*? Che è aumentato di 2, a causa delle entry “..” inserite dentro le sottocartelle mynewdir e aseconddir.
Qual è l’output del comando numero 9? “no such file or directory”

b) è più veloce l’accesso ad un file attraverso un link fisico o attraverso un link simbolico? (motivate la vostra risposta)

attraverso un link fisico, perché il link simbolico costringe a leggere un index-node in più.

c) 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.

d) 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.

SOLUZIONI
SISTEMI OPERATIVI – 26 gennaio 2017
Esame di LABORATORIO – corso A

Esercizi relativi alla parte di UNIX (3 punti)

1. Descrivere l'esecuzione del PROGRAMMA 1, indicando quante volte viene stampato il carattere A e quante volte B. Cosa cambia modificando la condizione nel test if() come nel programma PROGRAMMA 2?

```
/************ PROGRAMMA 1 ************/
int main(int argc, char** argv) {
  int i;

  for(i=0; i<2; ++i) {
    if (fork()) {
      printf("A");
      exit(0);
    }
  }

  printf("B");
  exit(0);
}

/************ PROGRAMMA 2 ************/
int main(int argc, char** argv) {
  int i;

  for(i=0; i<2; ++i) {
    if (!fork()) {
      printf("A");
      exit(0);
    }
  }

  printf("B");
  exit(0);
}
```

Il primo processo P1 entra nel ciclo (i=0), esegue la fork stampa A e termina.
Prosegue l'esecuzione del figlio P2 che rientra nel ciclo (i=1), forka stampa A e termina.
Prosegue l'esecuzione del figlio P3 che non entra nel ciclo (i=2), stampa B e termina.

Il programma emette 2 volte il carattere A e 1 volta B.

Nel secondo programma sono i figli a entrare nel ciclo di fork stampando A e ad uscire, mentre il genitore stampa B.

Esercizi relativi alla parte di C

ESERCIZIO 1 (3.5 punti)

Si implementi la funzione con prototipo

```
void find_first_and_last(char *str, char find_char, int *first, int
*last);
```

`find_first_and_last()` è una funzione che cerca il carattere `find_char` nella stringa di caratteri `str` e modifica i parametri `first` e `last` (passati per riferimento) assegnando loro l'indice nell'array della prima e dell'ultima occorrenza di `find_char`. Se il carattere non fosse presente o se `str` fosse uguale a NULL allora si consideri -1 come indice della prima e dell'ultima occorrenza.

Esempi;

- `str = dddghrddq`
- `find_char = d`

allora la funzione assegnerebbe 0 a first e 7 a last

- `str = dddghrddq`
- `find_char = g`

allora la funzione assegnerebbe 3 a first e 3 a last

```
#include <stdio.h>

#include <string.h>


void find_first_and_last(char *str, char find_char, int *first, int
*last){
```

```
  *first = -1, *last = -1;


  if(str != NULL) {
    int i;
    int length = strlen(str);


    for( i = 0; i < length; i++)
      if(*(str+i)==find_char){
      *first = i;
      break;
    }
    for( i = length - 1; i >= 0; i--)
      if(*(str+i)==find_char){
        *last = i;
        break;
      }
    }
}
int main() {
  char str[20]="dddghrddq";
  char find_char = 'd';


  int first, last;


  find_first_and_last(str,find_char,&first,&last);
  printf("String %s char %c first %d last %d
\n",str,find_char,first,last);
```

```
  return 0;
}
```

ESERCIZIO 2 (2 punti)

Qual è l'output del seguente programma C?

Quale sarebbe l'output se nel programma l'istruzione `int x = 10;` nel main fosse commentata?

```
#include <stdio.h>
int x=90;

int g(int x){
  printf("%d\n",-x);
  if(x > 20) {
    int x = -90;
    return x;
  }
  else {
    int x = -60;
    return x;
  }
}

int f(int x){
  if(x > 20) {
    int x = -50;
    printf("%d\n",g(x));
  }
  else {
```

```
    int x = -70;
    printf("%d\n",g(-x));
  }
  return -x;
}
int main() {
  int x = 10; // Da commentare

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

  return 0;
}
```

Risposta primo caso: -70 -90 10 -70 -90 -70 -90

Risposta secondo caso: -70 -90 90 -70 -90 50 -60

ESERCIZIO 3 (3.5 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 check_almost_zero(link head, int max_non_zero);
```

`check_almost_zero()` è una funzione che restituisce TRUE se e solo se tutti gli elementi sono zero tranne, al massimo, `max_non_zero` di questi. La funzione restituisce FALSE altrimenti. Nella soluzione dovete:

- definire opportunamente TRUE e FALSE
- gestire il caso di lista vuota in modo che la funzione, in questo caso, restituisca FALSE
- controllare il valore del parametro `max_non_zero` che deve essere positivo

Ad esempio:

se `head`: 1 → 0 → 0 → 7 → `NULL` e `max_non_zero`=3 allora `check_almost_zero` restituisce TRUE	se `head`: 1 → 0 → 0 → 7 → `NULL` e `max_non_zero`=1 allora `check_almost_zero` restituisce FALSE

```
#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 check_almost_zero(link head, int max_non_zero){
  int ret_value = FALSE;

  if(head != NULL && max_non_zero > 0) {
    int nonzero = 0;
    while(head != NULL) {
      if(head->value != 0)
        nonzero++;

      head = head->next;
    }
    if(nonzero <= max_non_zero)
      ret_value = TRUE;
    }
    return ret_value;
}

int main() {
link prova1 = (link)malloc(sizeof(nodo));
prova1->value=0; prova1->next=NULL;
link prova2 = (link)malloc(sizeof(nodo));
prova2->value=2; prova2->next=prova1;
link prova3 = (link)malloc(sizeof(nodo));
prova3->value=0; prova3->next=prova2;
link prova4 = (link)malloc(sizeof(nodo));
prova4->value=7; prova4->next=prova3;
```

```
link prova5 = (link)malloc(sizeof(nodo));

prova5->value=0; prova5->next=prova4;

link prova6 = (link)malloc(sizeof(nodo));

prova6->value=3; prova6->next=prova5;

link prova7 = (link)malloc(sizeof(nodo));

prova7->value=0; prova7->next=prova6;


int nonzero;

int max_non_zero = 3;


nonzero = check_almost_zero(prova7,max_non_zero);

printf("check_almost_zero computed %s for max_non_zero
%d\n",(nonzero==TRUE)?"TRUE":"FALSE",max_non_zero);

nonzero = check_almost_zero(NULL,max_non_zero);

printf("check_almost_zero computed %s for max_non_zero %d on
NULL\n",(nonzero==TRUE)?"TRUE":"FALSE",max_non_zero);

max_non_zero = 2;

nonzero = check_almost_zero(prova7,max_non_zero);

printf("check_almost_zero computed %s for max_non_zero
%d\n",(nonzero==TRUE)?"TRUE":"FALSE",max_non_zero);

return 0;

}
```