SISTEMI OPERATIVI – 12 settembre 2016
corso A nuovo ordinamento
e parte di teoria del vecchio ordinamento

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 e di "*Istituzioni di Sistemi Operativi*" 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)

In un sistema operativo che adotta uno scheduling con diritto di prelazione, quattro processi arrivano al tempo indicato e consumano la quantità di CPU indicata nella tabella sottostante)

Processo	T. di arrivo	Burst
P1	0	18
P2	4	6
P3	6	5
P4	8	1

a) (2 p.)
Qual è il waiting time medio migliore (ossia ottimale) che potrebbe essere ottenuto per lo scheduling dei quattro processi della tabella? RIPORTATE IL DIAGRAMMA DI GANTT USATO PER IL CALCOLO. (lasciate pure i risultati numerici sotto forma di frazione, e indicate quali assunzioni fate)

Diagramma di GANT, assumendo come algoritmo di scheduling SJF preemptive:

(0)….P1 …(4) …. P2….(8)….P4….(9)….P2….(11)….P3…(16)….P1…(30)

Waiting time medio:
P1 = (30 – 0) – 18 = 12;
P2 = (11 – 4) – 6 = 1;
P3 = (16 – 6) – 5 = 5;
P4 = (9 – 8) – 1 = 0;
waiting time medio = 18/4

b1) (1 p.) Riportate, a vostra scelta, il diagramma di stato o il diagramma di accodamento di un sistema time sharing che implementa la memoria virtuale;
b2) (1 p.) elencate le ragioni per cui un processo può passare dallo stato *waiting* allo stato *ready to run*

b1) Per i diagrammi si vedano i lucidi della sezione 3.1.2 (p. 6), 3.2.1 (p. 18) e il lucido a p. 9 del cap. 9.

b2) è stata completata una operazione di I/O; signal sul semaforo su cui il processo era addormentato; è arrivata in ram la pagina mancante che aveva causato un page fault del processo.

c) (1 p.) riportate lo pseudocodice relativo alla corretta implementazione dell'operazione di *wait*.

Si vedano i lucidi della sezione 6.5.2.

ESERCIZIO 2 (5 punti)

In un sistema che implementa la paginazione della memoria ma non la memoria virtuale, un indirizzo fisico è scritto su 41 bit, l'offset più grande all'interno di una pagina è pari a 1FF, e la tabella delle pagine più grande del sistema occupa 64 megabyte.

a) (2 p.) Quanto è grande lo spazio di indirizzamento logico del sistema (esplicitate i calcoli che fate)?

Lo spazio di indirizzamento fisico del sistema è diviso in $2^{41}/2^9 = 2^{32}$ frame, dunque ci vogliono 32 bit, ossia 4 byte per scrivere il numero di un frame, e questa è la dimensione di una entry della tabella delle pagine del sistema. La tabella delle pagine più grande del sistema è grande 64 megabyte, ossia 2^{26} byte e quindi ha 2^{24} entry. Lo spazio di indirizzamento logico quindi è grande $2^{24}*2^9= 2^{33}$ byte.

b) (2 p.) Se il sistema ha un tempo di accesso in RAM di 200 ns e adotta un TLB con un tempo di accesso di 50 ns, quale hit rate minimo deve garantire il TLB perché il sistema abbia un medium access time (mat) di 250 ns? (esplicitate i calcoli che fate, e assumete per semplicità che l'accesso al TLB e alla tabella delle pagine in RAM avvengano in parallelo)

a partire dalla formula: mat = hit-rate (250ns) + (1 – hit-rate)*(200ns + 200ns) < 250ns ricaviamo:

250hit-rate + 400 – 400hit-rate =< 250; 100 =< 150hit-rate; da cui si ricava un hit-rate >= 100%

(nota: dati i valori numerici della domanda, la risposta poteva essere ricavata semplicemente osservando che 200ns +50ns = 250ns = mat richiesto, e dunque l'unico hit rate che permette di raggiungere questo valore è ovviamente un hit rate del 100%)

c) (1 p.) Quanto sarebbe grande la IPT del sistema sopra descritto? (motivate la vostra risposta esplicitando eventuali assunzioni che fate)

La IPT ha 2^{32} entry. Se assumiamo che nel sistema possano essere presenti al massimo 256 processi, basta un byte per numerarli tutti, e poiché lo spazio di indirizzamento logico del sistema è diviso in 2^{24} pagine, ogni entry della IPT sarà grande 1+3 byte, e la IPT ha una dimensione di 2^{34} byte.

ESERCIZIO 3 (4 punti)

a) (2 p.) Considerate la seguente sequenza di comandi Unix:

```
1:  cd /tmp
2:  mkdir MYDIR1
3:  mkdir MYDIR2
4:  cd MYDIR1
```

```
 5:   mkdir SUBDIR
 6:   echo "HELLO" > GOOFY          // crea un file di nome GOOFY contenente la stringa HELLO
 7:   ln GOOFY MICKY
 8:   ln /tmp/MYDIR1 MYDIR3
 9:   ln –s GOOFY DONALD
10:   ls MICKY DONALD
11:   rm GOOFY
12:   cat DONALD                   // cat stampa il contenuto del file passato come argomento
13:   mkdir ONEMOREDIR
```

Dopo l'esecuzione di tutti i comandi:
qual è il valore del link counter nell'index-node associato al link fisico *MICKY*? 1
qual è il valore del link counter nell'index-node associato al link fisico *MYDIR1*? 4
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 *MYDIR1* e *MYDIR2*.
Qual è l'output del comando numero 12? "no such file or directory"
Che effetto ha il commando della linea 8? Nessuno, poiché non sono ammessi link fisici tra directory.

b) (1 p.) Quale tipo di link è proibito fra directory, e perché?

Il link fisico, in quanto potrebbe portare alla creazione di un loop tra directory, la cui presenza rendebbe inutilizzabili comandi e programmi che visitano ricorsivamente una porzione del file system.

c) (1 p.) Quali sono i vantaggi e gli svantaggi di un sistema RAID configurato al livello 1?

Vantaggi: massima affidabilità: il sistema può continuare a funzionare anche se si rompe un disco; velocità di accesso ai dati, poiché si possono leggere in parallelo anche dati appartenenti a strip memorizzati sullo stesso disco (sfruttando il disco di mirroring).

Svantaggi: è la configurazione che richiede il maggior numero di dischi, quindi la più costosa e la meno efficiente in termini di spazio di memorizzazione disponibile.

SOLUZIONI UNIX

(1) 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*.

(1 punti)

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*; tali righe devono 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
```

(1.2) Descrivere tre fra i seguenti comandi fornendo per ciacuno un esempio di invocazione:
cp *mv* *less* *tail*

(0.5 punti)

cp copia un file o directory nella locazione specificata
mv sposta la locazione specificata (sia un file o una directory) o rinomina il file/directory.
less mostra il contenuto di file di testo sul terminale, visualizzandolo una pagina per volta, permettendo di scorrerlo in avanti e indietro, e di effettuare ricerche tramite espressioni regolari.
tail mostra la fine di un file.

(1.3) Supponendo che i privilegi dell'utente *paperinik* sul file *file1* siano i seguenti,

```
	-rw-r--r--  1 paperinik  staff   0 Sep  7 09:08 file1
```

illustrare l'effetto del comando

```
	chmod 622 file1
```

e spiegare se e come tale comando differisce da

```
	chmod go+w file1
```

(0.5 punti)

l'effetto del commando `chmod 622 file1` è il seguente:

```
	-rw--w--w-+ 1 paperinik  staff   0 Sep  7 09:08 file1
```

l'effetto del commando `chmod go+w file1` è invece il seguente:

```
	-rw-rw-rw-+ 1 paperinik  staff   0 Sep  7 09:09 file1
```

(2) Descrivere il significato del seguente codice, e commentarlo riga per riga:

```
int reserveSem(int semId, int semNum) {
  struct sembuf sops;
```

```
    sops.sem_num = semNum;
    sops.sem_op = -1;
    sops.sem_flg = 0;

    return semop(semId, &sops, 1);
  }
```

(2 punti)

Il codice si riferisce ad una parte dell'implementazione di un semaforo binario. Un semaforo binario ha due valori: available (libero) e reserved (in uso). Questo codice rappresenta una funzione che permette di effettuare l'operazione di reserve (o wait, o P) su un semaforo (cioè tenta di riservarlo per uso esclusivo). Se il semaforo è già stato riservato da un altro processo, l'operazione si blocca fino a quando il semaforo è rilasciato.

I parametri semId e semNum identificano il set di semafori e il semaforo all'interno del set rispettivamente. La prima riga di codice definisce la struttura che servirà ad indicare l'operazione sul semaforo. Dalla seconda alla quarta riga vengono inizializzati i valori di sem_num, sem_op e sem_flg della struttura sops per: 1) identificare il semaforo all'interno del set, impostare l'operazione di decremento (reserve), e il flag. L'ultima riga effettua di fatto l'operazione sul semaforo (il terzo parametro "1" rappresenta il numero di operazioni).

(3) Scrivere un programma in cui un padre crea tre processi figli che tentano di accedere a una sezione critica utilizzando le primitive del binary semaphores protocol, che include le primitive initSemAvailable(…), initSemInUse(…), reserveSem(…), releaseSem(…). La sezione critica consiste nella stampa della stringa "Sezione Critica". Tutte le risorse condivise devono essere allocate e deallocate.

(2 punti)

Si tratta di un esercizio discusso a lezione. Le operazioni fondamentali si possono distinguere in tre classi

1. Creazione e la distruzione dei processi (fork, join)
2. Allocazione, inizializzazione, deallocazione di un semaforo (semget**,** initSemWithValue, semctl)
3. Lock e unlock del semaforo binario (reserveSem, releaseSem)

```
int main(int argc, char ** argv) {
  int i,pid,semid,k; union semun arg;
  int n_secs = 5;
  setbuf(stdout, NULL);
  if((semid = semget(IPC_PRIVATE, 1, 0666)) < 0) {...}
  if((initSemWithValue(semid, 0, 1)) == -1) {...}
  for (k=0; k<3; k++) {
    switch(fork()) {
      case -1: {...}
      case 0: // ------- figlio -------
      printf("\nfiglio [PID: %d]: tento di accedere alla s. critica\n",
        getpid());
      // ====================================
      reserveSem(semid,0);
      printf("Sezione Critica [PID: %d]\n", getpid());
      releaseSem(semid,0);
      // ====================================
      exit(EXIT_SUCCESS);
    }
  }
  printf("[padre - PID: %d] attendo terminazione dei figli\n",getpid());
  for(i=0; i<3; ++i) { wait(NULL); }
  printf("[padre - PID: %d] dealloco il semaforo\n", getpid());
  if ((semctl(semid, 0, IPC_RMID, arg)) == -1) {...}
```

```
  exit(EXIT_SUCCESS);
}
```

ESERCIZI RELATIVI ALLA PARTE DI C

ESERCIZIO 1 (3 punti)

Si implementi la funzione con prototipo

```
int find_and_replace_conditional(char * str, int min_count, char
find_char, char replace_char);
```

`find_and_replace_conditional()` è una funzione che sostituisce nella stringa di caratteri `str` ogni occorrenza del carattere `find_char` con il carattere `replace_char` ma solo se il numero di occorrenze di `find_char` è maggiore del valore `min_count`. La funzione restituisce il numero di sostituzioni effettuate.

Esempi;

```
str = dddghrddq
min_count = 2
find_char = d
replace_char = z
```

allora la funzione restituirebbe 5 e la stringa sarebbe `zzzghrzzq`

```
str = dddghrddq
min_count = 5
find_char = d
replace_char = z
```

allora la funzione restituirebbe 0 e la stringa sarebbe `dddghrddq`

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

int  find_and_replace_conditional(char  *  str,  int  min_count,  char
find_char, char replace_char){
     int i, length, n_find_char = 0;

     if(str != NULL) {
          length = strlen(str);
          for( i = 0; i < length; i++)
               if(*(str+i)==find_char)
                    n_find_char++;
     }
     if(str != NULL && n_find_char > min_count) {
          for( i = 0; i < length; i++)
               if(*(str+i)==find_char){
                    *(str+i)=replace_char;
               }
     }
     else n_find_char = 0;
     return n_find_char;
}
int main() {
     char str[20]="dddghrddq";
     int min_count=2;
```

```
    char find_char = 'd';
    char replace_char = 'z';
    int n_replaces;

    printf("Old string %s min_count %d find_char %c replace_char
%c\n",str,min_count,find_char,replace_char);
    n_replaces                                                    =
find_and_replace_conditional(str,min_count,find_char,replace_char);
    printf("New string %s n_replaces %d\n",str,n_replaces);

    min_count=5;
    find_char = 'z';
    replace_char = 'h';

    printf("Old string %s min_count %d find_char %c replace_char
%c\n",str,min_count,find_char,replace_char);
    n_replaces                                                    =
find_and_replace_conditional(str,min_count,find_char,replace_char);
    printf("New string %s n_replaces %d\n",str,n_replaces);

    return 0;
}
```

ESERCIZIO 2 (1 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: 50 -60 90 -70 -90 -70 -90

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 check_alternate(link head, int *odd_value);
```

`check_alternate()` è una funzione che restituisce TRUE se e solo se gli elementi in posizione dispari (il primo, il terzo, il quinto, ecc) sono uguali tra loro, altrimenti restituisce FALSE. Inoltre, nella variabile passata per riferimento (`odd_value`) si deve memorizzare il valore degli elementi in posizione dispari in caso di condizione verificata ed il valore -1 altrimenti. Definite opportunamente TRUE e FALSE e gestite il caso di lista vuota in modo che la funzione, in questo caso, restituisca FALSE.

Ad esempio:

se `head`: 1 → 3 → 1 → 7 → `NULL` allora `check_alternate` restituisce TRUE con `odd_value` che varrà 1.	se `head`: 1 → 3 → 5 → 7 → `NULL` allora `check_alternate` restituisce FALSE con `odd_value` che varrà -1.

```
#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_alternate(link head, int *odd_value){
     int equal = TRUE;

     *odd_value = -1;
     if(head != NULL) {
          *odd_value = head->value;
          while(equal == TRUE && head->next != NULL && head->next-
>next!=NULL) {
               if(head->value !=  (head->next->next)->value) {
                    equal = FALSE;
                    *odd_value = -1;
               }
               head = head->next->next;
          }
     }
```

```
     return equal;
}

int check_alternate2(link head, int *odd_value){
  int equal = TRUE;
  int position = 1;
  *odd_value = -1;
  if (head != NULL){
    *odd_value = head-> value;

    while (head!= NULL && equal){
       if (position %2 != 0 && *odd_value != head->value){
           equal = FALSE;
            *odd_value = -1;
        }
       position ++;
       head = head->next;
    }
  }
  return equal;
}

int main() {
link prova1 = (link)malloc(sizeof(nodo));
prova1->value=1; prova1->next=NULL;
link prova2 = (link)malloc(sizeof(nodo));
prova2->value=2; prova2->next=prova1;
link prova3 = (link)malloc(sizeof(nodo));
prova3->value=1; prova3->next=prova2;
link prova4 = (link)malloc(sizeof(nodo));
prova4->value=7; prova4->next=prova3;
link prova5 = (link)malloc(sizeof(nodo));
prova5->value=1; prova5->next=prova4;
link prova6 = (link)malloc(sizeof(nodo));
prova6->value=3; prova6->next=prova5;
link prova7 = (link)malloc(sizeof(nodo));
prova7->value=1; prova7->next=prova6;
int odd_value,equal;
equal = check_alternate(prova7,&odd_value);
equal = check_alternate2(NULL,&odd_value);
printf("equal condition %s odd_value %d
\n",(equal==TRUE)?"TRUE":"FALSE",odd_value);
return 0;
}
```