SISTEMI OPERATIVI – 2 febbraio 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	15
P2	3	8
P3	6	4
P4	11	3

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 …(3) …. P2….(6)….P3….(10)….P2….(11)….P4…(14)….P2…(18)…P1….(30)

Waiting time medio:
P1 = (30 – 0) – 15 = 15;
P2 = (18 – 3) – 8 = 7;
P3 = (10 – 6) – 4 = 0;
P4 = (14 – 11) – 3 = 0;
waiting time medio = 22/4

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 in stato running può passare allo stato waiting

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) richiesta di operazione di I/O; wait su un semaforo; page fault.

c) (1 p.) Siano dati tre programmi concorrenti A, B e C, al cui interno vengono eseguite, rispettivamente, le procedure Pa, Pb e Pc. Si vuole essere certi che Pa verrà eseguita prima di Pb e Pc, mentre Pb e Pc devono poter essere eseguiti in qualsiasi ordine (ossia, in esecuzioni diverse dei tre programmi, può succedere che una volta venga eseguita prima Pb e poi Pc, o viceversa). Fornite una possible soluzione al problema, usando le opportune system call di sincronizzazione con il (o i) relativo semaforo e il suo (o loro) valore di sincronizzazione.

Semaphore sync = 0;

A:
Pa;
signal(sync);
signal(sync);

B:
wait(sync);
Pb;

C:
wait(sync);
Pc;

ESERCIZIO 2 (5 punti)

In un sistema che implementa la paginazione della memoria ma non la memoria virtuale, un indirizzo fisico è scritto su 32 bit, l'offset più grande all'interno di una pagina è pari a FF, e la tabella delle pagine più grande del sistema occupa 24 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 $2^{32}/2^8 = 2^{24}$ byte, dunque ci vogliono 24 bit, ossia 3 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 24 megabyte, ossia $3 * 2^{23}$ byte e quindi ha 2^{23} entry. Lo spazio di indirizzamento logico quindi è grande $2^{23}*2^8= 2^{31}$ byte.

b) (2 p.) Se il sistema ha un tempo di accesso in RAM di 100 ns e adotta un TLB con un tempo di accesso di 10 ns, quale hit rate minimo deve garantire il TLB perché il sistema abbia un medium access time (mat) di 130 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 (100ns + 10ns) + (1 – hit-rate)*(100ns + 100ns) < 130ns ricaviamo:

110hit-rate + 200 – 200hit-rate < 130; 70 < 90hit-rate; da cui si ricava un hit-rate > 77,7%

c) (1 p.) Quali sono i vantaggi nell'adottare una tabella delle pagine invertite, e quali gli svantaggi?

Il vantaggio è un minor consumo di memoria principale (abbiamo una sola tabella per tutti i processi, anziché una tabella delle pagine per ciascun processo), lo svantaggio è dovuto ai maggiori tempi di accesso in RAM, a meno di non usare un TLB sufficientemente ampio.

ESERCIZIO 3 (4 punti)

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

```
 1:   cd /tmp
 2:   mkdir newdir1
 3:   mkdir newdir2
 4:   cd newdir1
 5:   mkdir anotherdir
 6:   echo "ciao" > pippo          // crea un nuovo file di nome pippo contenente la stringa ciao
 7:   ln pippo pluto
 8:   ln /tmp/newdir1 newdir3
 9:   ln –s pippo paperino
10:   ls pluto paperino
11:   rm pippo
12:   cat paperino                 // cat stampa il contenuto del file passato come argomento
13:   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 *newdir1*? 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 newdir1 e newdir2.
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.) Occupa più spazio l'uso di un link fisico o di un link simbolico? Si accede più velocemente ad un file attraverso un link fisico o un link simbolico? (motivate le vostre risposte)

Occupa più spazio un link simbolico perché richiede l'allocazione di un nuovo index-node. È più veloce l'accesso ai file attraverso i link fisici, perché con i link simbolici il SO deve leggere un index-node in più.

c) (1 p.) Elencare vantaggi e svantaggi del sistema RAID di livello 5

Vantaggi: massimizzazione dello spazio disponibile per la memorizzazione dei dati (al contrario ad esempio del livello 1) pur garantendo la possibilità di recuperare i dati in caso di rottura di un disco (al contrario ad esempio del livello 0). Usura omogenea dei dischi che compongono il sistema (al contrario ad esempio del livello 4 in cui il disco di parità è più sollecitato degli altri).

Svantaggi: maggiore complessità dell'algoritmo di gestione del sistema e di recupero dei dati persi in caso di guasto (a causa della distribuzione degli strip di parità fra tutti i dischi del sistema)

ESERCIZI RELATIVI ALLA PARTE DI UNIX (6 punti)

ESERCIZIO 1 (2 punti)

Si scriva lo pseudo-codice necessario generare un albero binario di processi di profondità n per un totale di 2^(n+1)-1 processi generati.

Soluzione

```
int dupl_i(int val) {
  printf("Child at level %d\n",0);
  for (int i=0; i<val ; ++i) {
    if (0 == fork()) { // left child
      printf("Child at level %d\n",i+1);
      break; // This break is key for correctness
    }
    else // parent or error
      if (0 == fork())  // right child
          printf("Child at level %d\n",i+1);
  }
  return 0;
}
```

ESERCIZIO 2 (2 punti)

Illustrare il funzionamento della system call *msgsnd()*.

Soluzione

La syscall msgsnd() invia un messaggio ad 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, e -1 in caso di errore. Il primo argomento è un intero che indica l'identificatore della coda, mentre il secondo è un puntatore ad 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*).

ESERCIZIO 3 (2 punti)

Posto che la struttura sembuf è definita come segue

```
struct sembuf {
      unsigned short sem_num; //
      short sem_op;   //
      short sem_flg; //
};
```

Fornire un'implementazione di una delle due funzioni *reserveSem* o *releaseSem* su un semaforo identificato dalla coppia *semId* e *semNum*. Commentare ogni istruzione di codice.

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

int releaseSem (int semId, int semNum) {
   struct sembuf sops;
   ...
}
```

Soluzione

```
int reserveSem(int semId, int semNum) {
        struct sembuf sops;
        sops.sem_num = semNum;  // indica il semaforo su cui si interviene
        sops.sem_op  = -1;      // operazione wait
        sops.sem_flg = 0;       // nessun flag particolare per l'operazione:
                                // potrebbe essere SEM_UNDO o IPC_NOWAIT
        semop(semId, &sops, 1); // esecuzione atomica delle operazioni indicate
                                // dall'array di operazioni sops sul set di semafori
                                // indicato da semId
}

int releaseSem (int semId, int semNum) {
        struct sembuf sops;
        sops.sem_num = semNum;
        sops.sem_op  = 1;
        sops.sem_flg = 0;
        semop(semId, &sops, 1);
}
```

ESERCIZI RELATIVI ALLA PARTE DI C

ESERCIZIO 1 (3 punti)

Si implementi la funzione con prototipo

```
int check_multiple_positions(char * str, int pos, char check_char);
```

`check_multiple_positions()` è una funzione che restituisce TRUE se gli elementi di `str` in posizioni multiple di `pos` sono uguali a `check_char` e FALSE altrimenti. Definite opportunamente TRUE e FALSE, gestite il caso in cui `str` sia NULL ed il caso in cui `pos` non sia maggiore di 0.
Esempi:

- se la stringa fosse: dfdgdrdgd; `pos` 2 e `check_char` g allora la funzione restituirebbe FALSE
- se la stringa fosse: dfdgdrdgd; `pos` 2 e `check_char` d allora la funzione restituirebbe TRUE
- se la stringa fosse: adfsgdsdgs; `pos` 3 e `check_char` s allora la funzione restituirebbe FALSE
- se la stringa fosse: sdfsgdsdgs; `pos` 3 e `check_char` s allora la funzione restituirebbe TRUE

```
#include <stdio.h>
#include <string.h>
#define TRUE 1
#define FALSE 0

int check_multiple_positions(char * str, int pos, char check_char){
        int ret_value = FALSE;

        if(str != NULL && pos > 0) {
                int i;
                int length = strlen(str);
                ret_value = TRUE;
                for( i = 0; i < length && ret_value==TRUE; i=i+pos)
                        if(*(str+i)!=check_char)
                                ret_value = FALSE;
        }
        return ret_value;
}

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

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

       check_char = 'd';
       printf("String %s pos %d check_char %c results %d\n",
       str,pos,check_char,check_multiple_positions(str,pos,check_char));

       check_char = 's';
       pos = 3;
       strcpy(str,"adfsgdsdgs");
       printf("String %s pos %d check_char %c results %d\n",
       str,pos,check_char,check_multiple_positions(str,pos,check_char));

       strcpy(str,"sdfsgdsdgs");
       printf("String %s pos %d check_char %c results %d\n",
       str,pos,check_char,check_multiple_positions(str,pos,check_char));

       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;` fosse commentata?

```
#include <stdio.h>
int x=30;

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

void f(int x){
        printf("%d\n",-x);
        if(x > 20) {
                int x = 50;
                printf("%d\n",g(x));
        }
        else {
                int x = -70;
                printf("%d\n",g(-x));
        }
}
int main() {
        int x = 10; // Da commentare

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

Risposta primo caso: 10 -10 -70 90 70
Risposta secondo caso: 30 -30 -50 90 50

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 count_elements(link head, int value, int *gte, int *lte);
```

`count_elements()` è una funzione che restituisce il numero di elementi della lista `head` che sono diversi da `value`. Inoltre, nelle variabili passate per riferimento (`gte` ed `lte`) restituisce il numero di elementi maggiori o uguali e minori o uguali di `value`, rispettivamente.
Ad esempio, data la lista `head`: 1 → 3 → 5 → `NULL` e `value` uguale a 3, `count_elements` ritorna 2 e entrambi gli argomenti passati per riferimento assumono il valore 2.

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

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

int count_elements(link head, int value, int *gte, int *lte){
        int noteq = 0;
        *lte = 0;
        *gte = 0;

        while (head != NULL) {
                if(head->value!=value)
                        noteq++;
                if(head->value <= value)
                        (*lte)++;
                if(head->value >= value)
                        (*gte)++;
                head = head->next;
        }
        return noteq;
}

int main() {

link prova = (link)malloc(sizeof(nodo));
prova->value=1; prova->next=NULL;

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

link prova3 = (link)malloc(sizeof(nodo));
prova3->value=5; prova3->next=prova2;

int value = 3;
int lte,gte,eq;

eq = count_elements(prova3,value,&gte,&lte);
printf("value %d : diferent %d lte %d gte %d\n",value,eq,lte,gte);

return 0;
}
```