SISTEMI OPERATIVI
11 settembre 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)

a) Si consideri il problema dei lettori e scrittori visto a lezione, dove i codici del generico scrittore e del generico lettore sono riportati qui di seguito.

Inserite le istruzioni mancanti necessarie per il funzionamento del sistema secondo la soluzione vista a lezione, indicando anche il semaforo mancante ed il suo valore di inizializzazione.

semafori e variabili condivise necessarie con relativo valore di inizializzazione:
semaphore mutex = 1;
semaphore scrivi = 1;
int numlettori = 0;

"scrittore"
{
wait(scrivi);
Esegui la scrittura del file
signal(scrivi)
}

"lettore"
{
wait(mutex);

numlettori++;

if numlettori == 1 **wait(scrivi);**

signal(mutex);

... leggi il file ...

wait(mutex);

numlettori--;

if numlettori == 0 **signal(scrivi);**

signal(mutex);

b) Se esistono, indicate un caso in cui a) un processo lettore può andare in starvation; b) un processo scrittore può andare in starvation?

Processo lettore: se il processo che sta scrivendo non termina mai; processo scrittore: se è in attesa di scrivere perché ci sono processi in lettura, e continuano ad arrivare nuovi lettori.

c) Sia dato un sistema operativo multi-tasking (ma non time sharing) che implementa la memoria virtuale. Riportate il diagramma di accodamento che mostra come i processi si muovono fra le varie code di scheduling di questo sistema.

Si vedano la figura del lucido 18 del capitolo 3 sulla gestione dei processi e il lucido 9 del capitolo 9 sulla memoria virtuale, in cui è assente il ramo "time slice expired".

d) Riportate lo pseudocodice che descrive l'implementazione dell'operazione di wait, spiegate perché questa implementazione non fa sprecare inutilmente tempo di cpu se il processo che la chiama trova il valore della variabile semaforica a 0.

Per lo pseudocodice si vedano i lucidi della sezione 6.5.2

Perché il pcb del processo che si addormenta sul semaforo in attesa di entrare in sezione critica viene messo nella coda di waiting del semaforo e non può più essere selezionato dallo scheduler fino a che la sezione critica non si libera.

ESERCIZIO 2 (5 punti)

Si consideri un sistema in cui in una tabella delle pagine di un processo l'indice più grande usabile nella tabella delle pagine di quel processo può essere 1FFF. Un indirizzo fisico del sistema è scritto su 24 bit, e la RAM è suddivisa in 4000 (esadecimale) frame.

a)
Quanto è grande, in megabyte, lo spazio di indirizzamento logico del sistema (esplicitate i calcoli che fate)?

4000(esadecimale) = 2^{14}, per cui un numero di frame è scritto su 14 bit, e la dimensione di un frame, e quindi di una pagina, è di $2^{\wedge 10}$ byte (24 - 14 = 10). Poiché il numero più grande di una pagina è 1FFF, ci possono essere al massimo $2^{\wedge 13}$ pagine, e lo spazio di indirizzamento logico è di $2^{\wedge 13}$ x $2^{\wedge 10}$ byte (pari a circa 8 megabyte).

b)
Se il sistema adotta una paginazione a due livelli, quanto spazio occupa, in byte, la tabella delle pagine *esterna* più grande di questo sistema? (motivate numericamente la vostra risposta).

La tabella delle pagine interna più grande del sistema ha $2^{\wedge 13}$ pagine, e ogni entry contiene il numero di un frame, per cui sono necessari almeno due byte per ogni entry. La dimensione di quella tabella sarà quindi di $2^{\wedge 13} \times 2 = 2^{\wedge 14}$ byte (ossia maggiore della dimensione di un frame, il che implica appunto la necessità di paginare la tabella delle pagine interna). $2^{\wedge 14}$ byte = 16 Kbyte, e quindi questa tabella delle pagine interna occupa 16 frame. Di conseguenza la tabella esterna sarà composta da 16 entry di due byte ciascuno, ossia 32 byte.

c)
In quale caso il sistema descritto qui sopra deve implementare la memoria virtuale?

> Lo spazio di indirizzamento logico è minore di quello fisico, per cui l'unico caso è quello in cui si vogliano avere in esecuzione contemporaneamente processi che, insieme, occupano uno spazio maggiore dello spazio di indirizzamento fisico.

d)
Un sistema (hardware + sistema operativo) soffre spesso del problema del thrashing. Indicate due modifiche al sistema, una hardware e una software, che potrebbero migliorare la situazione. Motivate le indicazioni che date.

Modifica Hardware: aggiungere più ram, cosicché ogni processo del sistema avrà mediamente più frame a disposizione e meno probabilmente genererà un page fault.
Modifica software: diminuire il grado di multiprogrammazione, in modo che meno processi hanno a disposizione ciascuno più frame.

ESERCIZIO 3 (4 punti)

Sia dato un sistema operativo Unix, in cui viene eseguito correttamente il comando:
"mkdir /users/gunetti/myfiles/pippo" (dove "pippo" non esisteva prima dell'esecuzione del comando)

a) cosa succede dal punto di vista degli hard link coinvolti nel comando?

L'hard link di "myfiles" viene incrementato di uno, e l'hard link di "pippo" viene creato e inizializzato al valore 2.

b) vengono ora dati i comandi:
cd /users/gunetti/myfiles
ln pippo pluto

cosa succede? (motivate la vostra risposta)

Il secondo comando fallisce, dato che gli hard link a directory non sono permessi.

c) Si assuma ora che nella cartella pippo sia presente il file di testo A, e si vuole creare un collegamento veloce ad A nella radice del file system. Conviene usare un hard link o un symbolic link? (motivate la vostra risposta)

Un hard link, che permette di raggiungere più velocemente i dati di A. Col symbolic link infatti, si crea un nuovo i-node che rimanda all'i-node di A.

d) occupa più spazio un hard link o un symbolic link? (motivate la vostra risposta)

Un symbolic link, perché alloca un nuovo i-node.

ESERCIZI RELATIVI ALLA PARTE DI UNIX (6 punti)

ESERCIZIO 1

(1.1) Cos'è il *wait status value*? Come è assegnato dalle system call *wait* e *waitpid*? Quali sono gli strumenti per analizzare il *wait status value*?

(2 punti)

Soluzione [slides 03_controllo_processi.pdf, slide 42 e seguenti]

Il valore dello *status* è passato da *wait()* e *waitpid()* per riferimento, come mostra il prototipo della *wait*:

pid_t wait(int **status*);

Lo status consente al genitore di distinguere fra i seguenti eventi per il figlio:
- il figlio ha terminato l'esecuzione chiamando *_exit()* (o *exit()*), specificando un codice d'uscita (*exit status*) intero.
- il figlio ha terminato l'esecuzione per la ricezione di un segnale non gestito.
- il figlio è stato bloccato da un segnale, e *waitpid()* è stata chiamata con il flag *WUNTRACED*.
- Il figlio ha ripreso l'esecuzione per un segnale *SIGCONT*, e *waitpid()* è stata chiamata con il flag *WCONTINUED*.

Il *wait status value* può essere analizzato tramite un insieme standard di macro definite nell'header file <sys/wait.h>. Applicate allo *status* restituito da *wait()* o *waitpid()*, solo una delle seguenti macro restituirà *true*.
- *WIFEXITED(status)*. Restituisce true se il processo figlio è terminato normalmente. In questo caso la macro *WEXITSTATUS(status)* restituisce l'exit status del processo figlio.
- *WIFSIGNALED(status)*. Restituisce true se il figlio è stato ucciso da un segnale. In questo caso, la macro *WTERMSIG(status)* restituisce il numero del segnale che ha causato la terminazione del processo.
- *WIFSTOPPED(status)*. Restituisce true se il figlio è stato bloccato da un segnale. In questo caso, la macro *WSTOPSIG(status)* restituisce il numero del segnale che ha bloccato il processo.
- *WIFCONTINUED(status)*. Restituisce true se il figlio è stato risvegliato da un segnale *SIGCONT*.

ESERCIZIO 2 (3 punti)

(2.1) Esercizio sull'utilizzo dei pipe. Si scriva un programma in cui due processi in relazione padre-figlio condividono un pipe. Il figlio è lettore, mentre il genitore è scrittore.

(3 punti)

Soluzione [slides 05_pipes_FIFOs.pdf, pp. 29 e sgg.]

```
int filedes[2];

if (pipe(filedes) == -1)
     // esegui qualche operazione

switch (fork()) {
     case -1:
     // esegui qualche operazione
     case 0:
     if (close(filedes[1]) == -1)
     // esegui qualche operazione
     ...
     break;
     default:
```

```
      if (close(filedes[0]) == -1)
           // esegui qualche operazione
      ...
      break;
}
```

ESERCIZIO 3 (1 punto)

Data la definizione seguente, scrivere una invocazione della system call *shmctl()* mirante a reperire le informazioni relative alla struttura di tipo *shmid_ds*.

```
struct shmid_ds {
  struct ipc_perm shm_perm;
  size_t shm_segsz;
  time_t shm_atime;
  time_t shm_dtime;
  time_t shm_ctime;
  pid_t shm_cpid;
  pid_t shm_lpid;
  shmatt_t shm_nattch;
}
```

Soluzione [slides 11_memoria_condivisa.pdf, slides 25 e sgg.]

```
struct shmid_ds {
  struct ipc_perm shm_perm;
  size_t shm_segsz;
  time_t shm_atime;
  time_t shm_dtime;
  time_t shm_ctime;
  pid_t shm_cpid;
  pid_t shm_lpid;
  shmatt_t shm_nattch;
}

struct shmid_ds shm_ds;

...

if( shmctl( shm_id, IPC_STAT, &shm_ds ) == -1 ) {
  printf("Failure in shmctl()\n");
}
```

ESERCIZI RELATIVI ALLA PARTE DI C

ESERCIZIO 1 (2 punti)

Si implementi la funzione con prototipo

```
int strlen_is_even(char * str);
```

`strlen_is_even` è una funzione che restituisce 1 se il numero di caratteri di cui è composta la stringa `str` è pari e 0 altrimenti. Nel caso la stringa `str` sia NULL la funzione restituisce -1.

```
int strlen_is_even(char * str) {
        int length=-1;
        if(str != NULL) {
                length=0;
                while (*str != '\0') {
                        length++;
                        str++;
                }
               length=1-(length%2);
        }
        return length;
}
```

ESERCIZIO 2 (3 punti)

Si implementi la funzione con prototipo

```
int even_more_than_odd(int * numbers, int length);
```

`even_more_than_odd` è una funzione che restituisce 1 se l'array di numeri interi `numbers` (la cui lunghezza è data dal parametro `length`) contiene più numeri pari che dispari, 0 se numeri pari e numeri dispari compaiono in quantità uguali e -1 altrimenti.

```
int even_more_than_odd(int * numbers, int length){
        int index,nodd=0,neven=0;
        for(index=0; index < length; index++){
                if(numbers[index]%2==1)
                        nodd++;
                else
                        neven++;
        }
        if(neven > nodd)
                return 1;
        else if(nodd==neven)
                return 0;
        else
                return -1;
}
```

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 count_element_multiples(link head, int value);
```

`count_element_multiples` è una funzione che restituisce il numero di occorrenze di elementi che sono multipli interi di `value` nella lista `head`. Se la lista è vuota, la funzione restituisce, invece, -1.

Funzione da implementare

```
int count_element_multiples(link head, int value) {
     int retvalue = -1;

     if(head != NULL) {
          retvalue = 0;
          while (head != NULL) {
               if ((head->value)%value==0)
                    retvalue++;
               head = head->next;
          }
     }
     return retvalue;
}
```