UNIVERSITÀ DEGLI STUDI DI BERGAMO

Corso di laurea in Ingegneria Informatica - Sistemi Operativi – 26 giugno 2007

Cognome_____Nome_____Matricola_____

1. Un sistema dotato di memoria virtuale con paginazione e segmentazione tipo UNIX è caratterizzato dai seguenti parametri: l'indirizzo logico è di 18 bit; l'indirizzo fisico è di 18 bit. La dimensione delle pagine è di 16K.

1a. Definire la struttura dell'indirizzo logico e di quello fisico indicando la lunghezza dei campi che li costituiscono:

NPV: *4* _____ Spiazzamento logico: *14* _____

NPF: *4* _____ Spiazzamento fisico: *14*_____

Nel sistema all'istante t_0 sono attivi i processi P, Q e R, che eseguono i programmi PGP, PGQ e PGR e condividono un segmento dati. La dimensione iniziale dei segmenti dei tre programmi è la seguente:

CP:	16K;	DP: 16K;	PP: 16K;	COND: 32K;
CQ:	30K;	DQ: 16K;	PQ: 16K;	COND: 32K;
CR:	45K;	DR: 16K;	PR: 16K;	COND: 16K (condivide solo la prima pagina del segmento);

La dimensione complessiva di ognuno dei 3 processi è di 256K e quindi il segmento pila di ogni processo inizia all'indirizzo corrispondente ai 256K; nel processo P il segmento COND è allocato lasciando 2 pagine libere dopo il segmento dati (per permettere una crescita dello Heap, servizio BRK) nel processo Q COND è allocato lasciando 2 pagine libere dopo DQ mentre, nel processo R, COND è allocato lasciando 0 pagine libere dopo DR .

1b. Inserire in tabella 1a il significato delle varie pagine di memoria logica (notazione: CP0, CP1, DP0, PP0,...CQ0, ..., COND0,...).

Indirizzo di pagina virtuale	**Processo P**	**Processo Q**	**Processo R**
0	*CP0*	*CQ0*	*CR0*
1	*DP0*	*CQ1*	*CR1*
2		*DQ0*	*CR2*
3		*DQ1*	*DR0*
4	*COND0*		*COND0*
5	*COND1*	*COND0*	
6		*COND1*	
7			
8			
9			
A			
B			
C			
D			*PR2*
E	*PP1*		*PR1*
F	*PP0*	*PQ0*	*PR0*

1A) Struttura della Memoria Logica

Indirizzo fisico	**Pagine allocate t_0**	**Indirizzo fisico**	**Pagine allocate t_1**
0	*CP0*	0	*CP0*
1	*DP0*	1	*DP0*
2	*COND0*	2	*PR2*
3	*COND1*	3	*COND1*
4	*PP0*	4	*PP0*
5	*CQ0*	5	*DQ1*
6	*CQ1*	6	*CQ1*
7	*DQ0*	7	*DQ0*
8	*PQ0*	8	*PQ1*
9	*CR0*	9	*PR1*
A	*CR1*	A	*CR1*
B	*CR2*	B	*CR2*
C	*DR0*	C	*DR0*
D	*PR0*	D	*PR0*
E		E	*PP1*
F		F	

1B) Memoria Fisica agli istanti t_0 e t_1

1c. Indicare quanto spazio (in pagine) è disponibile per l'allocazione dinamica nei processi

Q: *2* _____ R: *0* _____

Ad un certo istante t_0 sono terminati i seguenti eventi:

1. lancio di P (fork di P ed exec di PGP)
2. lancio di Q (fork di Q ed exec di PGQ)
3. lancio di R (fork di R ed exec di PGR)

1d. Sapendo che il numero di pagine residenti **R** è 6, che viene utilizzato l'algoritmo LRU e che le pagine meno utilizzate in ogni processo sono la prima pagina del segmento codice e quindi la prima pagina del segmento condiviso (le prime caricate sono le meno utilizzate) e ipotizzando che l'allocazione delle pagine virtuali nelle pagine fisiche sia avvenuta in sequenza senza buchi a partire dalla pagina fisica 0, indicare, completando tabella 1B (sinistra), l'allocazione fisica delle pagine dei tre processi all'istante t_0.

Ad un certo istante $t_1 > t_0$ sono terminati i seguenti eventi:

4. crescita della pila di P di 1 pagina
5. allocazione di 1 pagina di memoria per Q (servizio BRK)
6. crescita della pila di R di 2 pagine

1e. Completare la tabella 1A e 1B (destra) nelle medesime ipotesi delineate nel precedente punto (1d).

1f. Indicare il contenuto della tabella delle pagine della MMU all'istante t_1 per i **soli** processi P e R completando la seguente tabella (usare P e R come pid dei corrispondenti processi, oppure NS se nella corrispondente riga della tabella non è allocato alcun processo e quindi è non significativa)); ipotizzare che le righe della tabella siano state allocate ordinatamente man mano che venivano allocate le pagine di memoria virtuale. Indicare anche il valore assunto dal bit di validità di pagina BV (1 significa che la pagina è caricata).

PID	NPV	NPF	BV
P	*0*	*0*	*1*
P	*1*	*1*	*1*
R	*D*	*2*	*1*
P	*5*	*3*	*1*
P	*F*	*4*	*1*
R	*E*	*9*	*1*
R	*1*	*A*	*1*
R	*2*	*B*	*1*
R	*3*	*C*	*1*
R	*4*	*2*	*0*
R	*F*	*D*	*1*
P	*E*	*E*	*1*
NS			*0*
NS			*0*

2. Illustrare con precisione almeno quattro algoritmi di schedulazione della CPU.

Cfr. libro di testo, capitolo 6

3.
Si considerino i seguenti programmi:

```
/* programma main1.c */

main()
{  int pid;
   char c[40];
   ...
   pid=fork();
   if (pid == 0)
     {  /* codice eseguito da Q figlio di P */
      …
      execl("/home/info2/prog1", "prog1", NULL);
      exit(1); /*eseguita solo se la execl fallisce
*/
     }

   else
      { /* codice eseguito dal padre P  */
        ….
        write(stdout, c, 40);
        pid = wait(&status);
        exit(0);
       }

} /* end main1.c */

_____
____

/* programma prog1.c */

main()
{  int pid, d;
   ...
   pid=fork();
   if (pid == 0)
     {  /* codice eseguito da R */
      …
      read(stdin, &d, 1);
      exit(1);
     }

   else
      { ……
        pid = wait(&status);
        exit(0);
       }

} /* end prog1.c */
```

Un processo P esegue il programma **main1**, creando un processo figlio Q che esegue una mutazione di codice lanciando il programma **prog1**, nel codice mutato viene creato un processo figlio R. Si ipotizzi che la execl vada sempre a buon fine.

Nella tabella a pagina seguente sono indicati (nella prima colonna) alcuni eventi che si sono verificati durante l'esecuzione dei programmi da parte di P, Q e R; nella seconda colonna è aggiunta un'indicazione supplementare relativa a tali eventi. Si deve completare tale tabella indicando ordinatamente nella terza colonna tutti i moduli del S.O. che vengono eseguiti (completamente o in parte) in seguito all'evento, nella quarta colonna il contesto nel quale ciascun

modulo è eseguito e, nelle ultime tre colonne, lo stato dei processi P, Q e R dopo che tutti i moduli hanno svolto la funzione e si ritorna al funzionamento in modo U.

Avvertenze per il riempimento della tabella:

indicare i moduli utilizzando la notazione seguente:

Notazione abbreviata	Modulo (o frammento di modulo) di sistema
G_SVC	Gestore SVC
R_Int(Disp)	Routine di interrupt; Disp può valere CK=orologio, Sout=Standard output, Sin=Standard input
<nome routine di sistema>	puo' essere: fork, write, read, wait, exit, preempt, change
Sleep_on(E_n)	Sleep_on. Per indicare l'evento su cui viene sospeso il processo usare convenzionalmente E_1, E_2, E_3, ...
Wakeup(E_n)	Wakeup. E_n indica l'evento, come in Sleep_on

1. non esistono altri processi nel sistema
2. il buffer del driver di standard output ha dimensione di 50 caratteri
3. la notazione 25 interrupt (15 interrupt) indica che si sono verificati 25 (15) interrupt. Nella risposta fare riferimento all'ultimo di tali interrupt
4. il modulo wait invoca sleep_on su un evento opportuno
5. la terminazione di un processo (exit) invoca wakeup per risvegliare il processo padre eventualmente in attesa.
6. se un processo viene risvegliato da wakeup e non ci sono altri processi pronti o in esecuzione, il processo viene immediatamente lanciato in esecuzione (in questo caso wakeup invoca change)

Evento (preceduto dal processo nel cui contesto l'evento si verifica)	**Informazioni aggiuntive**	**Moduli eseguiti per gestire l'evento**	**Processo/i nel cui contesto è eseguito ogni modulo**	**Stato dei processi dopo la gestione dell'evento**		
				P	**Q**	**R**
P: fork	P non ha esaurito il suo quanto di tempo	G_SVC fork	P P	Esec U	Pronto	non esiste
P: interrupt da orologio	P ha esaurito il suo quanto di tempo	*R_int(CK)* *Preempt* *Change* *fork*	*P* *P* *P-Q* *Q*	*Pronto*	*Esec U*	*non esiste*
Q: fork	Q ha esaurito il suo quanto di tempo (n.b. la fork è stata eseguita) P è più prioritario di R	*G_SVC* *fork* *Preempt* *Change*	*Q* *Q* *Q* *Q-P*	*Esec U*	*Pronto*	*Pronto*
P: write	write ha trasferito i 40 caratteri nel buffer R è più prioritario di Q	*G_SVC* *write* *Sleep_on(E1)* *Change* *fork*	*P* *P* *P* *P-R* *R*	*Attesa*	*Pronto*	*Esec U*
R: read	Lo standard input non è pronto	*G_SVC* *read* *Sleep_on(E2)* *Change* *fork*	*R* *R* *R* *R-Q* *Q*	*Attesa*	*Esec U*	*Attesa*
Q: 40 interrupt da standard output	l'ultimo è relativo all'ultimo carattere da trasferire; Q non ha esaurito il suo quanto di tempo	*R_int(Sout)* *Wake_up(E1)*	*Q* *Q*	*Pronto*	*Esec U*	*attesa*
Q: interrupt da standard input	Q ha esaurito il suo quanto di tempo P è più prioritario di R	*R_int(Sin)* *Wake_up(E2)* *Preempt* *Change* *Sleep_on(E1)* *write*	*Q* *Q* *Q* *Q-P* *P* *P*	*Esec U*	*Pronto*	*Pronto*
P: wait	R è più prioritario di Q	*G_SVC* *wait* *Sleep_on(E3)* *Change* *Sleep_on(E2)* *read*	*P* *P* *P* *P-R* *R* *R*	*Attesa*	*pronto*	*Esec U*
R: exit		*G_SVC* *exit* *Change*	*R* *R* *R-Q*	*Attesa*	*Esec U*	*non esiste*
Q: wait		*G_SVC* *wait*	*Q* *Q*	*Attesa*	*Esec U*	*non esiste*
Q: exit		*G_SVC* *exit* *Wake_up(E3)* *Change* *Sleep_on(E3)* *wait*	*Q* *Q* *Q* *Q-P* *P* *P*	*Esec U*	*non esiste*	*non esiste*
P: exit		*G_SVC* *exit*	*P* *P*	*non esiste*	*non esiste*	*non esiste*