Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria

prof. Luca Breveglieri **prof.ssa Donatella Sciuto**
prof. Gerardo Pelosi **prof.ssa Cristina Silvano**

AXO – Architettura dei Calcolatori e Sistemi Operativi

SECONDA PARTE – giovedì 1 luglio 2021

Cognome_____ **Nome** _____

Matricola _____**Firma** _____

Istruzioni

- Si scriva solo negli spazi previsti nel testo della prova e non si separino i fogli.
- Per la minuta si utilizzino le pagine bianche inserite in fondo al fascicolo distribuito con il testo della prova. I fogli di minuta se staccati vanno consegnati intestandoli con nome e cognome.
- È vietato portare con sé libri, eserciziari e appunti, nonché cellulari e altri dispositivi mobili di calcolo o comunicazione. Chiunque fosse trovato in possesso di documentazione relativa al corso – anche se non strettamente attinente alle domande proposte – vedrà annullata la propria prova.
- Non è possibile lasciare l'aula conservando il tema della prova in corso.
- Tempo a disposizione 1 h : 30 m

Valore indicativo di domande ed esercizi, voti parziali e voto finale:

esercizio 1 (4 punti) _____

esercizio 2 (4 punti) _____

esercizio 3 (6 punti) _____

esercizio 4 (2 punti) _____

voto finale: (16 punti) _____

esercizio n. 1 – programmazione concorrente

Si consideri il programma C seguente (gli "`#include`" e le inizializzazioni dei *mutex* sono omessi, come anche il prefisso `pthread` delle funzioni di libreria NPTL):

```
pthread_mutex_t surface
sem_t flat, steep
int global = 0
```

```
void * walk (void * arg) {
   mutex_lock (&surface)
   sem_post (&flat)
   global = 1                                   /* statement A */
   mutex_unlock (&surface)
   global = 2
   sem_post (&steep)
   sem_wait(&flat)
   return (void *) 3
} /* end walk */
```

```
void * rest (void * arg) {
   mutex_lock (&surface)
   sem_wait (&steep)
   global = 4                                   /* statement B */
   mutex_unlock (&surface)
   sem_wait (&flat)
   global = 5                                   /* statement C */
   sem_post (&flat)
   return NULL
} /* end rest */
```

```
void main ( ) {
   pthread_t th_1, th_2
   sem_init (&flat, 0, 0)
   sem_init (&steep, 0, 0)
   create (&th_1, NULL, walk, NULL)
   create (&th_2, NULL, rest, NULL)
   join (th_1, &global)                         /* statement D */
   join (th_2, NULL)
   return
} /* end main */
```

Si completi la tabella qui sotto **indicando lo stato di esistenza del** *thread* nell'istante di tempo specificato da ciascuna condizione, così: se il *thread* **esiste**, si scriva ESISTE; se **non esiste**, si scriva NON ESISTE; e se può essere **esistente** o **inesistente**, si scriva PUÒ ESISTERE. Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).

Si badi bene alla colonna "condizione": con "subito dopo statement X" si chiede lo stato che il *thread* assume tra lo statement X e lo statement immediatamente successivo del *thread* indicato.

condizione	*thread*	
	th_1 – *walk*	**th_2 – *rest***
subito dopo stat. **A**	Esiste	Può esistere
subito dopo stat. **B**	Può esistere	Esiste
subito dopo stat. **C**	Esiste	Esiste
subito dopo stat. **D**	Non esiste	Può esistere

Si completi la tabella qui sotto, **indicando i valori delle variabili globali** (sempre esistenti) nell'istante di tempo specificato da ciascuna condizione. Il **valore** della variabile va indicato così:

- intero, carattere, stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quanti
- il semaforo può avere valore positivo o nullo (non valore negativo)
- si supponga che il mutex valga 1 se occupato, e valga 0 se libero

Si badi bene alla colonna "condizione": con "subito dopo statement X" si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del *thread* indicato.

condizione	**variabili globali**			
	surface	*flat*	*steep*	*global*
subito dopo stat. **A**	1	1	0	1
subito dopo stat. **B**	1	1 - 0	0	4 - 3
subito dopo stat. **C**	0	0	0	5
subito dopo stat. **D**	1 - 0	0	1 - 0	3 - 4

Il sistema può andare in stallo (*deadlock*), con uno o più *thread* che si bloccano, in (almeno) **due casi diversi**. Si chiede di precisare il comportamento dei thread in **due casi**, indicando gli statement dove avvengono i blocchi e i possibili valori della variabile *global*:

caso	**th_1 – *walk***	**th_2 – *rest***	*global*
1	mutex_lock(&surface)	sem_wait(&steep)	0
2	-	sem_wait(&flat)	3 - 4
3			

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

```
// programma esercizio.c
int main ( ) {
   pid1 = fork ( )
   if (pid1 == 0) {            // codice eseguito solo da Q
      read (stdin, msg, 3)
      execl ("/acso/nuovo", "nuovo", NULL)
   } else {                    // codice eseguito solo da P
      pid1 = wait (&status)
      write (stdout, msg, 25)
   } /* if */
   exit (0)
} /* esercizio */
```

```
// programma nuovo.c
sem_t pass
mutex_t lock
int glob = 1
```

```
void * first (void * arg) {
   if (glob == 1) {
      mutex_lock (&lock)
      sem_post (&pass)
      mutex_unlock (&lock)
      sem_post (&pass)
   } /* if */
   return NULL
} /* first */
```

```
void * second (void * arg) {
   glob = 2
   sem_wait (&pass)
   mutex_lock (&lock)
   sem_wait (&pass)
   mutex_unlock (&lock)
   sem_wait (&pass)
   return NULL
} /* second */
```

```
int main ( ) {
   pthread_t TH_1, TH_2
   sem_init (&pass, 0, 0)
   pthread_create (&TH_2, NULL, second, NULL)
   sem_post (&pass)
   pthread_create (&TH_1, NULL, first, NULL)
   if (glob == 1) {
      pthread_join (TH_2, NULL)
      pthread_join (TH_1, NULL)
   } else exit (-1) /* if */
} /* nuovo */
```

Un processo **P** esegue il programma **`esercizio`** e crea un figlio **Q** che esegue una mutazione di codice (programma **`nuovo`**). La mutazione di codice va a buon fine e vengono creati i thread **TH_1** e **TH_2**.

Si simuli l'esecuzione dei vari processi completando tutte le righe presenti nella tabella così come risulta dal codice dato, dallo stato iniziale e dagli eventi indicati. Si completi la tabella riportando quanto segue:

- ⟨ PID, TGID ⟩ di ciascun processo (normale o thread) che viene creato
- ⟨ evento oppure identificativo del processo-chiamata di sistema / libreria ⟩ nella prima colonna, dove necessario e in funzione del codice proposto (le istruzioni da considerare sono evidenziate in grassetto)
- in ciascuna riga lo stato dei processi **al termine dell'evento o della chiamata associata alla riga stessa**; si noti che la prima riga della tabella **potrebbe essere solo parzialmente completata**

TABELLA DA COMPILARE

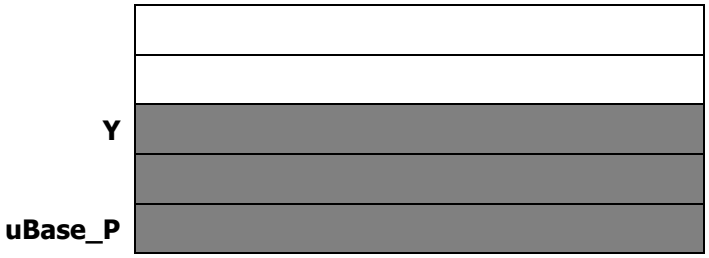
identificativo simbolico del processo		**IDLE**	**P**	**Q**	**TH_1**	**TH_2**
evento oppure processo-chiamata	**PID**	1	2	3	5	4
	TGID	1	2	3	3	3
P – pid = fork ( )	**0**	**pronto**	**esec**	**pronto**	**NE**	**NE**
P - wait	**1**	pronto	A wait	ESEC	NE	NE
Q - read	**2**	ESEC	A wait	A read	NE	NE
interrupt da stdin tutti i caratteri trasferiti	**3**	pronto	A wait	ESEC	NE	NE
Q - execl	**4**	pronto	A wait	ESEC	NE	NE
Q - pthread_create(TH2)	**5**	pronto	A wait	ESEC	NE	pronto
interrupt da RT_clock scadenza quanto di tempo	**6**	pronto	A wait	pronto	NE	ESEC
TH2 - sem_wait(&pass)	**7**	pronto	A wait	ESEC	NE	A sem
Q - sem_post(&pass)	**8**	pronto	A wait	pronto	NE	ESEC
Interrupt da RT_CLOCK e scadenza quanto di tempo	**9**	**pronto**	**A wait**	**esec**	**NE**	**pronto**
Q - pthread_create(TH1)	**10**	pronto	A wait	ESEC	pronto	pronto
Q - exit	**11**	pronto	ESEC	NE	NE	NE
P - write	**12**	ESEC	A write	NE	NE	NE
interrupt da stdout tutti i caratteri trasferiti	**13**	pronto	ESEC	NE	NE	NE
P - exit	**14**	ESEC	NE	NE	NE	NE

seconda parte – moduli del kernel

Sono dati due processi normali **P** e **Q**. Nel sistema c'è un altro processo **R**, sospeso su un **timer non ancora scaduto**. Nel sistema non ci sono altri processi oltre a **P**, **Q** e **R**. Lo stato iniziale delle pile di sistema e utente di **P** e **Q** è parzialmente riportato qui sotto, mentre le pile di sistema e utente di **R** non hanno rilevanza e qui non sono mostrate.

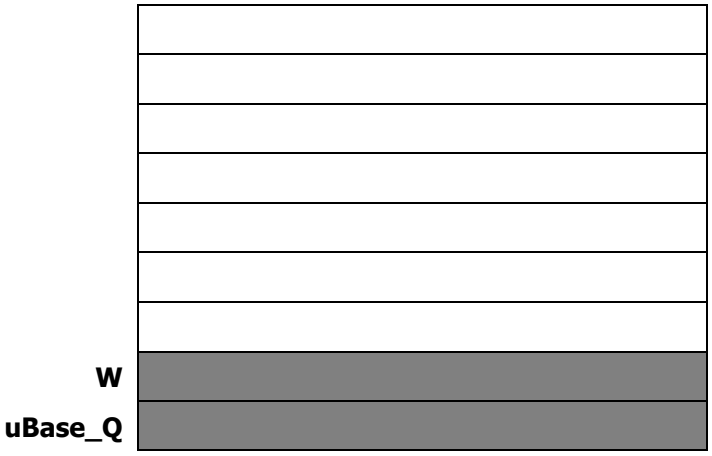
P è in esecuzione in modo U. **Q** è *sospeso per scadenza del suo quanto di tempo*.

1) Si completi la pila di sistema di Q:

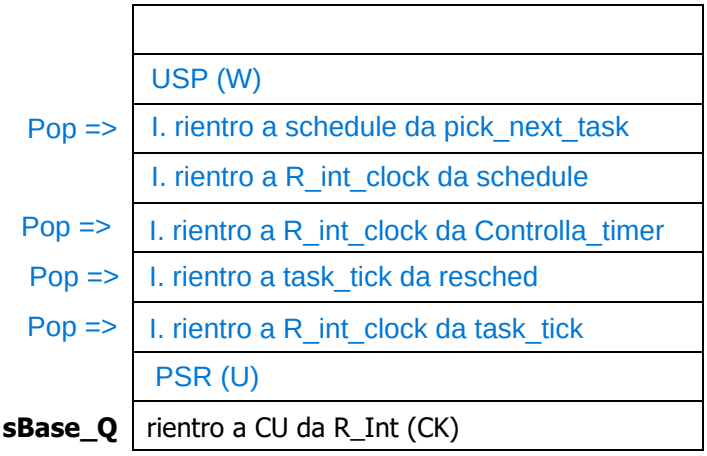


uStack_P – iniziale

sStack_P – iniziale



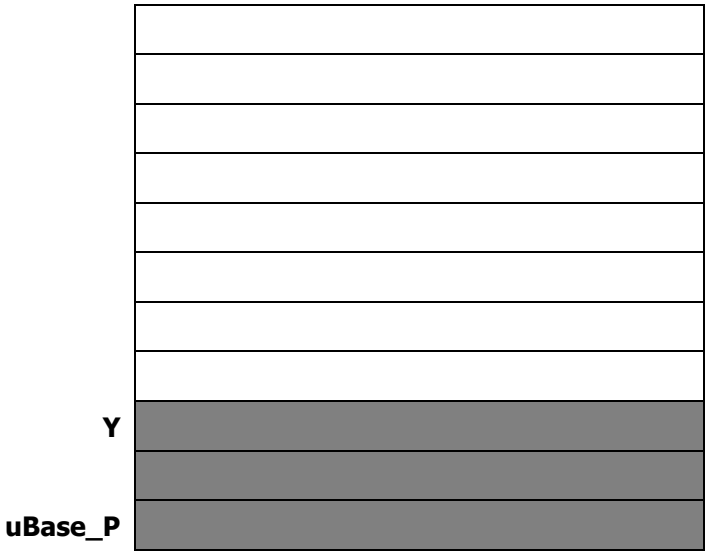
uStack_Q – iniziale



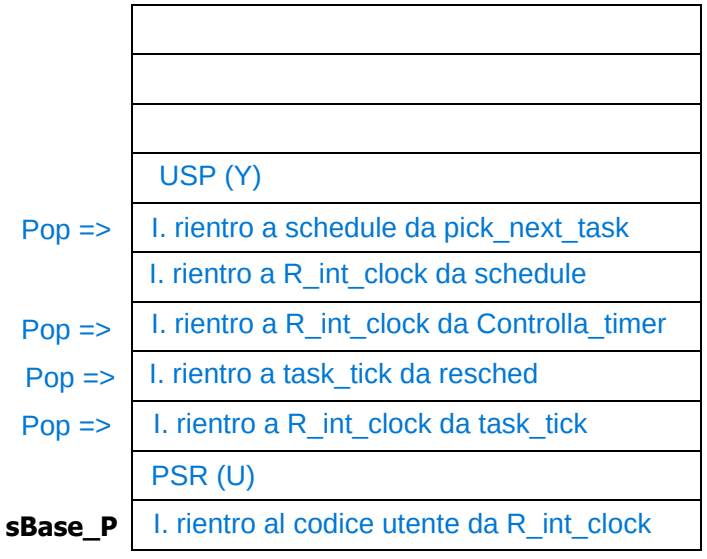
sStack_Q – iniziale (da completare)

Ora si consideri l'**evento** seguente: il *timer* di **R** scade, poi il *quanto di tempo* di **P** scade e **Q** torna in esecuzione riportandosi in modo U.

2) Si mostri lo stato delle pile del processo **P** fino al **momento** in cui il processo **Q** è tornato in esecuzione **in modo U**:



uStack_P – finale



sStack_P – finale

3) Si risponda alle seguenti domande:

a) Indicare il **modulo** di SO dove il processo **Q** era stato sospeso: R_int_clock

b) Indicare il **modulo** di SO dove il processo **Q** si trova nel **momento** preciso del suo ritorno in esecuzione: schedule

c) Indicare il **valore di USP** nel momento in cui **Q** è tornato in esecuzione: W

4) Si mostrino le invocazioni di **tutti i moduli** (ed eventuali relativi ritorni) fino al **momento** in cui il processo **Q** è tornato in esecuzione **in modo U** (numero di righe vuote non significativo):

tabella di invocazione dei moduli

processo	modo	modulo
P	U	codice utente
P	U -> S	> R_int_clock
P	S	> task_tick <
P	S	> Controlla_timer
P	S	> wakeup_process
P	S	> enqeue_task <
P	S	> check_preempt_curr <
P	S	wakeup_process <
P	S	Controlla_timer <
P	S -> U	IRET (R_int_clock <)
P	U	Codice utente di U
P	U -> S	> R_int_clock
P	S	> task_tick
P	S	> resched <
P	S	task_tick <
P	S	> Controlla_timer <
P	S	> schedule
P	S	> pick_next_task <
P -> Q	S	CONTEXT_SWITCH (schedule)
Q	S	schedule <
Q	S -> U	IRET (R_int_clock <)
Q	U	Codice utente di Q

esercizio n. 3 – memoria e file system

prima parte – memoria virtuale

È dato un sistema di memoria caratterizzato dai seguenti parametri generali:

MAXFREE = 3 **MINFREE = 1**

situazione iniziale: esistono due processi **P** (padre) e **Q** (figlio), e **Q** è in esecuzione

```
PROCESSO: Q  ************************************************
    VMA : C  000000400,  2 , R  ,  P  ,  M  , <X, 0>
          S  000000600,  2 , W  ,  P  ,  M  , <X, 2>
          P  7FFFFFFFC,  3 , W  ,  P  ,  A  , <-1,0>

    PT: <c0 :1 R>  <c1 :-  ->  <s0 :- ->  <s1 :- ->  <p0 :2 D W>
        <p1 :- ->  <p2 :-  ->

    processo Q - NPV di PC e SP:  c0, p0


____MEMORIA FISICA____(pagine libere: 4)_____________________________
      00 : <ZP>                ||  01 : Pc0 / Qc0 / <X, 0>  ||
      02 : Qp0   D             ||  03 : Pp0   D             ||
      04 : ----                ||  05 : ----                ||
      06 : ----                ||  07 : ----                ||


SWAP FILE:     ----, ----, ----, ----,
LRU ACTIVE:    QP0, QC0, PP0, PC0
LRU INACTIVE:
```

evento 1: *read* **(Qc0) –** *write* **(Qp0, Qp1) – 4** *kswapd*

PT del processo: **Q**				
c0: :1 R	c1: :- -	s0:	s1:	p0: :2 W D
p1: :4 W	p2: :- -			

process Q	NPV of **PC**: c0	NPV of **SP**: p1

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 <X, 0>
02: Qp0 (D)	03: Pp0 (D)
04: Qp1	05:
06:	07:

LRU ACTIVE: QP1, QP0, QC0

LRU INACTIVE: pp0, pc0

evento 2: *sbrk* **(8192)** // argomento dato in numero di byte

VMA del processo **Q** (è da compilare solo la riga relativa alla VMA **D**)							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
D	0x 0000 0060 2	2	W	P	A	-1	0

evento 3: *read* **(Qc0, Qd0) – *write* (Qp1, Qd1) – 4 *kswapd***

PT del processo: **Q**				
c0: :1 R	c1: :- -	s0:	s1:	d0: :5 W
d1: :6 W	p0: :s1 W	p1: :4 W	p2: :- -	

MEMORIA FISICA	
00: <ZP> / Qd0	01: Pc0 / Qc0 / <X, 0>
02: Qp0 (D)	03: - - - -
04: Qp1	05: Qd1
06:	07:

SWAP FILE	
s0: Pp0	s1: Qp0

LRU ACTIVE: QD1, QD0, QP1 QC0

LRU INACTIVE: qp0, pc0

evento 4: *context switch* **(P)**

MEMORIA FISICA	
00: <ZP> / Qd0	01: Pc0 / Qc0 / <X, 0>
02: Qp0 (D)	03: Pp0
04: Qp1 (D)	05: Qd1 (D)
06:	07:

SWAP FILE	
s0:	s1:

LRU ACTIVE: QD1, QD0, QP1 QC0

LRU INACTIVE: qp0, pc0

evento 5: *read* **(Pc0, Pp0) – 4 *kswapd***

MEMORIA FISICA	
00: <ZP> / Qd0	01: Pc0 / Qc0 / <X, 0>
02:	03: Pp0
04: Qp1 (D)	05: Qd1 (D)
06:	07:

SWAP FILE	
s0: Qp0	s1:

LRU ACTIVE: PP0, PC0

LRU INACTIVE: qd1, qd0, qp1, qc0

seconda parte – file system

È dato un sistema di memoria caratterizzato dai seguenti parametri generali:

MAXFREE = 2 MINFREE = 1

Si consideri la seguente **situazione iniziale**:

```
____MEMORIA FISICA____(pagine libere: 2)_____________________________
      00 : <ZP>                  ||  01 : Pc0 / <Y, 0>          ||
      02 : Pp0                   ||  03 : Pp1                   ||
      04 : ----                  ||  05 : <F, 1>                ||
      06 : <F, 2>                ||  07 : ----                  ||

Processo P - NPV di PC e SP:  c0, p1
File Aperto F in proc P   f_pos: 12000 -- f_count: 1
Accessi a pagine del DISCO per file F: Lettura 3, Scrittura 0
LRU ACTIVE: PC0
LRU INACTIVE: pp1, pp0,
```

Il processo **P** ha aperto il file **F** con descrittore *fd*, ha letto il file **F** ed è in esecuzione.

Per ciascuno dei seguenti eventi compilare le tabelle richieste con i dati relativi al contenuto della memoria fisica, relativi al valore delle variabili del FS per il file **F**, e relativi al numero di accessi a disco effettuati in lettura e in scrittura.

ATTENZIONE: il numero di pagine di file lette o scritte è cumulativo, ossia è la somma delle pagine lette o scritte su quel file da parte tutti gli eventi precedenti oltre a quello considerato. Si ricorda che la primitiva *close* scrive le pagine dirty di un file solo se *f_count* diventa = 0.

evento 1: *write* (fd, 4000)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <Y, 0>
02: Pp0	03: Pp1
04: <F, 3> (D)	05: <F, 1>
06: <F, 2> (D)	07:

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	16000	1	4	0

eventi 2 e 3: *lseek* (fd, −16000), *write* (fd, 4000) // offset lseek negativo

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <Y, 0>
02: Pp0	03: Pp1
04: <F, 0> (D)	05: ----
06: <F, 2> (D)	07:

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	4000	1	5	1

eventi 4 e 5: *fork* (Q), *context switch* (Q)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <Y, 0>
02: Pp0	03: Qp1 (D)
04: <F, 0> (D)	05: Pp1 (D)
06: <F, 2> (D)	07:

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	4000	2	5	1

evento 6: *write* (Qp2, Qp3)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <Y, 0>
02: Pp0 / Qp0 (D)	03: Qp1 (D)
04: Qp2	05: Pp1 (D)
06: Qp3	07:

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	4000	2	5	3

LRU ACTIVE: QP3, QP2, QC0, PC0

LRU INACTIVE: qp1, qp0, pp1, pp0

evento 7: *read* (fd, 4000)

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 / <Y, 0>
02: <F, 0>	03: Qp1 (D)
04: Qp2	05: <F, 1>
06: Qp3	07:

nome file	f_pos	f_count	numero pag. lette	numero pag. scritte
F	8000	2	7	3

SWAP FILE	
s0: Pp1	s1: Pp0 / Qp0

0 ---- 4096 ---- 8192 ---- 12288 ---- 16384 ---- 20480 ---- 24576
0 1 2 3 4 5

esercizio n. 4 – domande su argomenti vari

tabella delle pagine

Date le VMA di un processo riportate sotto, **si definisca**:

1. la decomposizione degli indirizzi virtuali dell'NPV iniziale di ogni area secondo la notazione:

   PGD : PUD : PMD : PT

2. il numero di pagine necessarie in ciascun livello della gerarchia e il numero totale di pagine necessarie per rappresentare la Tabella delle Pagine (TP) del processo
3. il numero di pagine virtuali occupate dal processo
4. il rapporto tra l'occupazione della TP e la dimensione virtuale del processo in pagine
5. la dimensione virtuale massima del processo in pagine, senza dovere modificare la dimensione della TP
6. il rapporto relativo

VMA del processo **P**							
AREA	NPV iniziale	dimensione	Read/ Write	Private/ Shared	Mapped/ Anonymous	nome file	offset (pagine)
C	0000 0040 0	4	R	P	M	X	0
K	0000 0060 0	2	R	P	M	X	6
S	0000 0060 2	2	R	P	M	X	8
D	0000 0060 4	2	W	P	A	–1	0
M0	0000 CC00 B	513	W	S	M	F	4
T1	7FFF F6FF E	5	W	P	A	–1	0
P	7FFF FFFF E	5	W	P	A	–1	0

Decomposizione degli indirizzi virtuali:

		PGD	**PUD**	**PMD**	**PT**
C	0000 0040 0	0	0	2	0
K	0000 0060 0	0	0	3	0
S	0000 0060 2	0	0	3	2
D	0000 0060 4	0	0	3	4
M0	0000 CC00 B	0	2	96	10
T1	7FFF F6FF D	255	511	439	509
P	7FFF FFFF E	255	511	511	510

Numero di pagine necessarie:

#pag. PGD	1
#pag. PUD	2
#pag. PMD	3
#pag. PT	6
#pag. totali	12

Numero di pagine virtuali del processo:	533
Rapporto di occupazione:	2.25%
Dimensione massima del processo in pagine virtuali:	6 * 512 = 3072
Rapporto di occupazione con dimensione massima:	0.39%

spazio libero per brutta copia o continuazione

0x 0000 CC00 B => 0000 0000 0000 0000 1100 1100 0000 0000 1010 => 0 2 96 10

0x 7FFF F6FF D => 0111 1111 1111 1111 1111 0110 1111 1111 1101 => 255 511 439

spazio libero per brutta copia o continuazione