



Politecnico di Milano

Dipartimento di Elettronica, Informazione e Bioingegneria

prof. Luca Breveglieri
prof. Gerardo Pelosi

prof.ssa Donatella Sciuto
prof.ssa Cristina Silvano

AXO – Architettura dei Calcolatori e Sistemi Operativi

SECONDA PARTE – venerdì 15 luglio 2022

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 (5 punti) _____

esercizio 3 (5 punti) _____

esercizio 4 (2 punti) _____

voto finale: (16 punti) _____

CON SOLUZIONI (in corsivo)

esercizio n. 1 – programmazione concorrente

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

```
pthread_mutex_t accept, reject
sem_t comm
int global = 0
```

```
void * send (void * arg) {
    mutex_lock (&accept)
    sem_post (&comm)
    global = 1
    mutex_unlock (&accept)
    global = 2
    mutex_lock (&reject)
    sem_wait (&comm)
    mutex_unlock (&reject)
    return NULL
} /* end send */
```

global = 1	/* statement A */
------------	--------------------------

```
void * receive (void * arg) {
    mutex_lock (&accept)
    global = 3
    sem_wait (&comm)
    mutex_unlock (&accept)
    sem_wait (&comm)
    mutex_lock (&reject)
    sem_post (&comm)
    global = 4
    mutex_unlock (&reject)
    return NULL
} /* end receive */
```

global = 3	/* statement B */
------------	--------------------------

global = 4	/* statement C */
------------	--------------------------

```
void main ( ) {
    pthread_t th_1, th_2
    sem_init (&comm, 0, 1)
    create (&th_1, NULL, send, NULL)
    create (&th_2, NULL, receive, NULL)
    join (th_1, NULL)
    join (th_2, NULL)
    return
} /* end main */
```

join (th_1, NULL)	/* statement D */
-------------------	--------------------------

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	<i>thread</i>	
	th_1 – send	th_2 – receive
subito dopo stat. A	<i>ESISTE</i>	<i>PUÒ ESISTERE</i>
subito dopo stat. B	<i>PUÒ ESISTERE</i>	<i>ESISTE</i>
subito dopo stat. C	<i>ESISTE</i>	<i>ESISTE</i>
subito dopo stat. D	<i>NON ESISTE</i>	<i>PUÒ ESISTERE</i>

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		
	<i>accept</i>	<i>reject</i>	<i>comm</i>
subito dopo stat. A	<i>1</i>	<i>0 / 1</i>	<i>0 / 1 / 2</i>
subito dopo stat. B	<i>1</i>	<i>0 / 1</i>	<i>1 / 2</i>
subito dopo stat. C	<i>0 / 1</i>	<i>1</i>	<i>1</i>
subito dopo stat. D	<i>0 / 1</i>	<i>0</i>	<i>0 / 1</i>

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 – send	th_2 – receive	<i>global</i>
1	<i>wait comm</i>	<i>lock reject</i>	<i>2 / 3</i>
2	–	<i>2a wait comm</i>	<i>2 / 3</i>
3			

esercizio n. 2 – processi e nucleo

prima parte – gestione dei processi

// programma ring_b.c	
sem_t vuoto, pieno	
int anello [3]	
int write_idx = 0, read_idx = 0, cont = 0, out	
pthread_mutex_t mux = PTHREAD_MUTEX_INITIALIZER	
void * produttore (void * arg) {	void * consumatore (void * arg) {
sem_wait (&vuoto)	sem_wait (&pieno)
mutex_lock (&mux)	mutex_lock (&mux)
anello [write_idx] = cont	out = anello [read_idx]
write_idx++	read_idx++
cont++	mutex_unlock (&mux)
mutex_unlock (&mux)	sem_post (&vuoto)
sem_post (&pieno)	sem_wait (&pieno)
sem_wait (&vuoto)	out = out + anello [read_idx]
anello [write_idx] = cont	sem_post (&vuoto)
sem_post (&pieno)	return NULL
return NULL	} // consumatore
} // produttore	
void * help (void * arg) {	
char msg [16] = "Lung. attuale: "	
nanosleep (10)	
for (int i = 0; i < 3; i++) {	
mutex_lock (&mux)	
write (stdout, msg, 15)	
printf ("%d", write_idx - read_idx)	
mutex_unlock (&mux)	
}	
return NULL	
} // help	
main () { // codice eseguito da P	
pthread_t th_1, th_2, th_3	
sem_init (&vuoto, 0, 1)	
sem_init (&pieno, 0, 0)	
create (&th_1, NULL, help , NULL)	
create (&th_2, NULL, consumatore , NULL)	
create (&th_3, NULL, produttore , NULL)	
join (th_1, NULL)	
join (th_2, NULL)	
join (th_3, NULL)	
exit (1)	
} // main	

Un processo **P** crea i tre thread **TH_1**, **TH_2** e **TH_3**. Si simuli l'esecuzione dei processi completando tutte le righe presenti nella tabella così come risulta dal codice dato, dallo stato iniziale e dagli eventi indicati, e tenendo conto che il processo **P non ha ancora creato nessun thread**. Si completi la tabella riportando quanto segue:

- I valori < *PID*, *TGID* > di ciascun processo che viene creato.
- I valori < *identificativo del processo-chiamata di sistema / libreria* > nella prima colonna, dove necessario e in funzione del codice proposto.
- 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 (numero di colonne non significativo)

identificativo simbolico del processo		IDLE	P	TH_1	TH_2	TH_3
	PID	1	2	3	4	5
evento oppure processo-chiamata	TGID	1	2	2	2	2
P – create TH_1	1	pronto	esec	pronto	NE	NE
P – create TH_2	2	pronto	esec	pronto	pronto	NE
interrupt da RT_clock e scadenza del quanto di tempo	3	pronto	pronto	esec	pronto	NE
TH_1 – nanosleep	4	pronto	pronto	attesa nanosleep	esec	NE
TH_2 – sem_wait (pieno)	5	pronto	esec	attesa nanosleep	attesa sem_wait	NE
P – create TH_3	6	pronto	esec	attesa nanosleep	attesa sem_wait	pronto
P – join TH_1	7	pronto	attesa join TH_1	attesa nanosleep	attesa sem_wait	esec
TH_3 – sem_wait (vuoto)	8	pronto	attesa join TH_1	attesa nanosleep	attesa sem_wait	esec
TH_3 – mutex_lock (mux)	9	pronto	attesa join TH_1	attesa nanosleep	attesa sem_wait	esec
interrupt da RT_clock e scadenza del timer	10	pronto	attesa join TH_1	esec	attesa sem_wait	pronto
TH_1 – mutex_lock (mux)	11	pronto	attesa join TH_1	attesa lock	attesa sem_wait	esec
TH_3 – mutex_unlock (mux)	12	pronto	attesa join TH_1	esec	attesa sem_wait	pronto
TH_1 – write	13	pronto	attesa join TH_1	attesa write	attesa sem_wait	esec
TH_3 – sem_post(pieno)	14	pronto	attesa join TH_1	attesa write	esec	pronto
TH_2 – sem_wait (pieno)	15	pronto	attesa join TH_1	attesa write	esec	pronto

seconda parte – moduli del SO

stato iniziale: CURR = P, Q = ATTESA (E) della scadenza di un timer

Si consideri il seguente evento: il processo **P** è in esecuzione in **modo U** e si verifica un'interruzione da **R_int_clock** per scadenza del quanto di tempo, il quale risveglia anche il processo **Q**. Non ci sono altri processi attivi.

domanda

- mostrare le **invocazioni** di tutti i moduli (ed eventuali relativi ritorni) eseguiti nel contesto del processo **P** per gestire l'evento indicato
- mostrare (in modo simbolico) il contenuto della **pila di sistema** del processo **P** al termine della gestione dell'evento considerato

invocazione moduli

processo	modo	modulo
P	U – S	> R_int_clock
P	S	> task_tick
P	S	> resched <
P	S	task_tick <
P	S	> controlla_timer
P	S	> wakeup_process
P	S	> check_preempt_curr
P	S	> resched < TIF_NEED = 1
P	S	check_preempt_curr <
P	S	wakeup_process <
P	S	controlla_timer <
P	S	> schedule
P	S	> pick_next_task <
P – Q	S	context_switch

USP
rientro a schedule da pick_next_task
rientro a R_int_clock da schedule
rientro a check_preempt_curr da resched
rientro a wup_process da check_preempt_curr
rientro a timer.function da wakeup_process
rientro a R_int_clock da controlla_timer
rientro a task_tick da resched
rientro a R_int_clock da task_tick
PSR (u)
rientro a codice utente da R_int_clock

esercizio n. 3 – memoria virtuale e file system

prima parte – gestione dello spazio di memoria

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

MAXFREE = 3

MINFREE = 2

situazione iniziale (esiste un processo P)

processo: P *****

VMA : C 000000400, 2, R, P, M, <YY, 0>
K 000000600, 1, R, P, M, <YY, 2>
S 000000601, 1, W, P, M, <YY, 3>
D 000000602, 3, W, P, A, <-1, 0>
P 7FFFFFFFC, 3, W, P, A, <-1, 0>

PT: <c0 :1 R> <c1 :- -> <k0 :- -> <s0 :- -> <d0 :3 W>
<d1 :4 W> <d2 :5 W> <p0 :2 W> <p1 :6 W> <p2 :- ->

process P - NPV of PC and SP: c0, p1

MEMORIA FISICA (pagine libere: 3)

00 : <ZP>	01 : Pc0 / <YY, 0>
02 : Pp0	03 : Pd0
04 : Pd1	05 : Pd2
06 : Pp1	07 : ----
08 : ----	09 : ----

SWAP FILE: ----, ----, ----, ----, ----, ----

LRU ACTIVE: PP1, PD2, PC0

LRU INACTIVE: pd1, pd0, pp0

evento 1: *read (Pc0) – write (Pp1, Pp2, Pd2) – 4 kswapd*

Legge pagina Pc0 (in memoria); scrive pagina Pp1 (in memoria), alloca pagina Pp2 (pagina di growdown) in pagina fisica 7, la scrive (in memoria) e la mette in testa ad active (restano due pagine libere), scrive pagina Pd2 (in memoria), kswapd per riportare pagine libere a tre, scarica in swap pagina pp0 da inactive; aggiorna liste LRU (restano tre pagine libere).

PT del processo: P				
p0: s0 W	p1: 6 W	p2: 7 W	p3: - -	

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <YY, 0>
02: Pp0	03: Pd0
04: Pd1	05: Pd2
06: Pp1	07: Pp2
08:	09:

SWAP FILE	
s0: Pp0	s1:
s2:	s3:

LRU ACTIVE: PP2, PP1, PD2, PC0 _____

LRU INACTIVE: pd1, pd0 _____

evento 2: read (Pc0) – write (Pp3)

Legge pagina Pc0 (in memoria), alloca pagina Pp3 (pagina di growdown) in pagina fisica 2 (restano due pagine libere), la scrive (in memoria) e la mette in testa ad active.

PT del processo: P				
p0: s0 W	p1: 6 W	p2: 7 W	p3: 2 W	p4: - -

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <YY, 0>
02: Pp3	03: Pd0
04: Pd1	05: Pd2
06: Pp1	07: Pp2
08:	09:

LRU ACTIVE: PP3, PP2, PP1, PD2, PC0 _____

LRU INACTIVE: pd1, pd0 _____

eventi 3: fork (Q) – context switch (Q) – read (Qc0, Qp3, Qd2) – 4 kswapd

Processo P crea processo figlio Q, tutte le pagine di P vengono condivise separatamente con Q e inserite in liste LRU, separa subito pagina di cima pila Pp3, ci sono due pagine libere; invoca PFRA e scarica pagine pd0 / qd0 e pd1 / qd1 da inactive in swap; alloca pagina Pp3 in pagina fisica 3; context switch (svuotamento TLB), marca pagine p1, p2 e d2 condivise dirty in quanto prima scritte da P e dunque dirty nel TLB.

Processo Q legge pagine Qc0, Qp3 e Qd2 (in memoria); aggiorna liste LRU (pagine di P vanno in coda inactive); pagine d0, d1 (condivise in swap) e d2 (condivisa in memoria) predisposte per COW (marcate R in PT di Q) (restano 3 pagine libere).

PT del processo: Q				
d0: s1 R	d1: s2 R	d2: 5 D R		

MEMORIA FISICA	
00: <ZP>	01: Pc0 / Qc0 <YY, 0>
02: Pp3 Qp3 D	03: Pd0 / Qd0 Pp3 D
04: Pd1 / Qd1	05: Pd2 / Qd2 D
06: Pp1 / Qp1 D	07: Pp2 / Qp2 D
08:	09:

SWAP FILE	
s0: Pp0 / Qp0	s1: Pd0 / Qd0
s2: Pd1 / Qd1	s3:

LRU ACTIVE: QP3, QD2, QC0 _____

LRU INACTIVE: qp2, qp1, pp3, pp2, pp1, pd2, pc0 _____

evento 4: read (Qc0) – write (Qd1)

Legge pagina Qc0 (in memoria) e scrive pagina Qd1 (in swap condivisa con Pd1); swap_in di Pd1 / Qd1 in pagina fisica 4, scrive pagina Qd1; COW di Qd1, due pagine libere; invoca PFRA e scarica in swap da inactive pagina pp3 (pag fisica 3 non condivisa) e pagina qp1 (condivisa con pp1 pag fisica 6); alloca pagina Qd1 in pagina fisica 3; rende scrivibile pagina d1 (in PT di Q) (restano tre pagine libere) (Pd1 resta anche in swap in quanto non modificata).

PT del processo: Q				
d0: s1 R	d1: 3 W	d2: 5 D R		

MEMORIA FISICA	
00: <ZP>	01: $Pc0 / Qc0 <YY, 0>$
02: $Qp3 \quad D$	03: $Pp3$ $DQd1$
04: $Pd1$	05: $Pd2 / Qd2 \quad D$
06: $Pp1 / Qp1$ D	07: $Pp2 / Qp2 \quad D$
08:	09:

SWAP FILE	
s0: $Pp0 / Qp0$	s1: $Pd0 / Qd0$
s2: $Pd1$	s3: $Pp3$
s4: $Pp1 / Qp1$	s5:

seconda parte – file system

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

MAXFREE = 3 **MINFREE = 1**

Si consideri la seguente **situazione iniziale**.

PROCESSO: P *****

VMA : C 000000400, 3, R, P, M, <XX, 0>

S 000000600, 2, W, P, M, <XX, 3>

D 000000602, 2, W, P, A, <-1, 0>

P 7FFFFFFFC, 3, W, P, A, <-1, 0>

PT: <c0 :1 R> <c1 :3 R> <c2 :4 R> <s0 :- -> <s1 :- ->

<d0 :- -> <d1 :- -> <p0 :2 W> <p1 :- -> <p2 :- ->

process P - NPV of PC and SP: c2, p0

MEMORIA FISICA (pagine libere: 3)

00 : <ZP>

02 : Pp0

04 : Pc2 / <XX, 2>

06 : ----

01 : Pc0 / <XX, 0>

03 : Pc1 / <XX, 1>

05 : ----

07 : ----

STATO del TLB

Pc0 : 01 - 0: 0:

Pc1 : 03 - 0: 0:

Pp0 : 02 - 1: 1:

Pc2 : 04 - 0: 1:

SWAP FILE: ----, ----, ----, ----

LRU ACTIVE: PC2, PP0

LRU INACTIVE: pc1, pc0

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	F	0	1	0	0

ATTENZIONE: è presente la colonna "processo" dove va specificato il nome/i del/i processo/i a cui si riferiscono le informazioni "f_pos" e "f_count" (campi di struct file) relative al file indicato.

Il processo **P** è in esecuzione. Il file **F** è stato aperto da **P** tramite chiamata **fd1 = open (F)**.

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

Per ciascuno degli eventi seguenti, compilare le tabelle richieste con i dati relativi al contenuto della memoria fisica, delle variabili del FS relative ai file aperti e al numero di accessi a disco effettuati in lettura e in scrittura.

evento 1: read (fd1, 8000)

Bisogna caricare due pagine di file F; carica pagina file <F, 0> in pagina fisica 5 e pagina file <F, 1> in pagina fisica 6; resta una sola pagina libera. Due letture e zero scritture su disco.

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <XX, 0>
02: Pp0	03: Pc1 / <XX, 1>
04: Pc2 / <XX, 2>	05: <F, 0>
06: <F, 1>	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	F	8000	1	2	0

evento 2: read (fd1, 1000)

Legge una pagina di file F; carica pagina file <F, 2> ; c'è una sola pagina libera, invoca PFRA per liberare tre pagine; libera pagine di page cache <F, 0> e <F, 1> senza scaricarle su disco (non sono dirty), e libera pagina di processo pc0 da inactive (non scarica in swap perché è pagina di codice); carica pagina file <F, 2> da disco in pagina fisica 1; aggiorna lista inactive (toglie pagina pc0); aggiorna posizione di file F. Tre letture e zero scritture su disco.

MEMORIA FISICA	
00: <ZP>	01: Pc0 / <XX, 0> <F, 2>
02: Pp0	03: Pc1 / <XX, 1>
04: Pc2 / <XX, 2>	05: <F, 0>
06: <F, 1>	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
P	F	9000	1	3	0

LRU ACTIVE: PC2, PP0, _____

LRU INACTIVE: pc1, _____

eventi 3: fork (Q) – context switch (Q)

Processo P crea processo figlio Q; condivide tutte pagine (di processo) separatamente tra P e Q; separa pagina di pila p0 (cima di pila corrente); pagina Qp0 resta in pagina fisica 2 e pagina Pp0 viene allocata in pagina fisica libera 5; restano due pagine libere; context-switch; TLB svuotato e riempito con PTE di Qc2 e Qp0 (codice corrente e cima pila), vedi anche TLB iniziale; pagina codice corrente e cima pila di Q sono c2 e p0 (come P).

MEMORIA FISICA	
00: <ZP>	01: <F, 2>
02: Pp0 Qp0	03: Pc1 / Qc1 <XX, 1>
04: Pc2 / Qc2 <XX, 2>	05: Pp0 D
06:	07:

processo Q	NPV of PC: c2	NPV of SP: p0
-------------------	----------------------	----------------------

TLB

NPV	NPF	D	A	NPV	NPF	D	A				
<i>Qc2</i>	<i>:</i>	<i>04</i>	<i>-</i>	<i>0:</i>	<i>1:</i>	<i>Qp0</i>	<i>:</i>	<i>02</i>	<i>-</i>	<i>1:</i>	<i>1:</i>
-----				-----							

eventi 4: fd2 = open (G) – write (fd2, 4000)

Processo Q apre file G e scrive una pagina di G; carica pagina di file <G, 0> in pagina libera 6, marcata dirty (viene scritta); aggiorna posizioni e count di file F e G (F è condiviso tra P e Q). Tre letture e zero scritture su disco per F, e una lettura e zero scritture su disco per G.

MEMORIA FISICA	
00: <ZP>	01: <F, 2>
02: Qp0	03: Pc1 / Qc1 <XX, 1>
04: Pc2 / Qc2 <XX, 2>	05: Pp0 D
06: <G, 0> F	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
PQ	F	9000	2	3	0
Q	G	4000	1	1	0

eventi 5: context switch (P) – write (fd1, 4000)

Context switch; processo P scrive in pagina di file <F, 3>; una pagina libera; invoca PFRA per liberare tre pagine; libera pagine di page cache <F, 2> (fis 1) e <G, 0> (fis 6); scarica pagina file <G, 0> di G su disco (è dirty); libera anche pagina di processo qc1 (fis 3) da inactive (pc1 era in inactive dove era stata condivisa separatamente con qc1) senza scaricarla in swap (pagina di codice); aggiorna posizione di file F; carica pagina file <F, 3> in pagina fisica 1, marcandola dirty (viene scritta). Quattro letture e zero scritture su disco per F, e una lettura e una scrittura su disco per G.

MEMORIA FISICA	
00: <ZP>	01: <F, 2>=><F, 3> D
02: Qp0	03: Pc1 / Qc1 <XX, 1>
04: Pc2 / Qc2 <XX, 2>	05: Pp0 D
06: <G, 0>=D	07:

processo/i	file	f_pos	f_count	numero pag. lette	numero pag. scritte
PQ	F	13000	2	4	0
Q	G	4000	1	1	1

esercizio n. 4 – domande su argomenti vari

tabella delle pagine

Date le VMA di un processo P sotto riportate, definire:

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 ogni livello della gerarchia e il numero totale di pagine necessarie a 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

VMA del processo P							
AREA	NPV iniziale	dimensione	R/W	P/S	M/A	nome file	offset
C	0000 0040 0	3	R	P	M	F	0
K	0000 0060 0	1	R	P	M	F	3
S	0000 0060 1	5	W	P	M	F	4
D	0000 0060 6	256	W	P	A	-1	0
M1	0000 1000 0	1	W	P	M	G	4
T0	7FFF F77F E	2	W	P	A	-1	0
P	7FFF FFFF 5	10	W	P	A	-1	0

1. 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 1	0	0	3	1
D	0000 0060 6	0	0	3	6
M1	0000 1000 0	0	0	128	0
T0	7FFF F77F E	255	511	443	510
P	7FFF FFFF 5	255	511	511	501

2. Numero di pagine necessarie

pag PGD: 1

pag PUD: 2

pag PMD: 2

pag PT: 5

pag totali: 10

3. Numero di pagine virtuali occupate dal processo: 278

4. Rapporto di occupazione: $10/278 = 0.035 = 3,5 \%$

5. Dimensione massima del processo in pagine virtuali:

Con la stessa dimensione di TP il processo può crescere fino a $5 \times 512 = 2560$ pagine virtuali

spazio libero per brutta copia o continuazione