# Esame 2019/07/15

Blascovich Alessio

## Domanda 1

**MAX PUNTI:** 4
**Domanda:**
Spiega in dettaglio come funziona il meccanismo della Tabella delle Pagine Invertita.
**Soluzione:**
Si basa su una tabella unica per tutto il sistema, quindi non per i singoli processi.
Questa tabella contiene una tupla con ¡pid,numero-pagina¿ con:

- PID: è l'identificativo del processo che detiene la pagina.
- numero-pagina: l'indirizzo logico della pagina

Per cercare un indirizzo bisogna quindi scorrere tutta la tabella per trovare la tupla che contiene la combinazione pid e numero-pagina cercata. Gli indirizzi vengono tradotti cercando la tupla nella tabella e tenendo conto in una variabile `i` di che posizione occupa nella tabella la nostra tupla (simile a dire "a che indice `i` sta la nostra tupla").
Viene poi fatto un append tra `i` e l'offset `d`, l'indirizzo ottenuto è l'indirizzo fisico cercato.

## Domanda 2

**MAX PUNTI:** 4
**Domanda:**
Descrivere le differenze tra semafori normali e spinlock, fornire poi un esempio di utilizzo di semafori spin-lock da parte del kernel.
**Soluzione:**
I semafori implementati con spinlock(busy waiting) delegano la CPU ad un continuo controllo dello stato della sezione critica.
Questo li porta ad essere molto CPU-intensive e adatti a contesti dove c'è bisogno di una risposta immediata, come nel caso di accessi alla memoria.
Sono moto facili da implementare ed anche molto scalabili.
I semafori sono usate dal kernerl per sincronizzare tra di loro processi/thread.

## Domanda 3

**MAX PUNTI:** 5
**Domanda:**
Data la stringa di riferimenti alla memoria 1, 2, 3, 4, 1, 2, 5, 1, 2,3, 4, 5, si determinino il numero di page fault che si avranno usando gli algoritmi FIFO, LRU e ideale.
Si supponga di avere inizialmente 3 frame vuoti.
**Soluzione:**
Si consideri la sigla pf come l'acronimo di page-fault.

- *FIFO*:
  Il metodo *FIFO* gestisce la memeoria come una semplice coda, il primo segmaneto ad essere entrato è il primo ad essere fatto uscire.

| 1 | null | null | $\xrightarrow{pf=1}$ | 1 | 2 | null | $\xrightarrow{pf=2}$ | 1 | 2 | 3 | $\xrightarrow{pf=3}$ | 4 | 2 | 3 | $\xrightarrow{pf=4}$ | 4 | 1 | 3 | $\xrightarrow{pf=5}$ | 4 | 1 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| $\xrightarrow{pf=6}$ | 5 | 1 | 2 | $\xrightarrow{pf=6}$ | 5 | 1 | 2 | $\xrightarrow{pf=6}$ | 5 | 1 | 2 | $\xrightarrow{pf=6}$ | 3 | 1 | 2 | $\xrightarrow{pf=7}$ | 3 | 4 | 2 | $\xrightarrow{pf=8}$ | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

$\xrightarrow{pf=9}$ Termino con un totale di 9 page fault.

- *LRU*:
  L'algoritmo *LRU* tiene associato ad ogni frame il clock della CPU nell'istante in cui quel frame è stato usato per l'ultima volta, verrà tolto il frame che non viene usato da più tempo.
  Nella prima riga sono indicati gli istanti, mentre nella seconda il numero del frame.

| 1 | 0 | 0 |
|---|---|---|
| 1 | null | null |

$\xrightarrow{pf=1}$

| 1 | 2 | 0 |
|---|---|---|
| 1 | 2 | null |

$\xrightarrow{pf=2}$

| 1 | 2 | 3 |
|---|---|---|
| 1 | 2 | 3 |

$\xrightarrow{pf=3}$

| 4 | 2 | 3 |
|---|---|---|
| 4 | 2 | 3 |

$\xrightarrow{pf=4}$

| 4 | 5 | 3 |
|---|---|---|
| 4 | 1 | 3 |

$\xrightarrow{pf=5}$

| 4 | 5 | 6 |
|---|---|---|
| 4 | 1 | 2 |

$\xrightarrow{pf=6}$

| 7 | 5 | 6 |
|---|---|---|
| 5 | 1 | 2 |

$\xrightarrow{pf=7}$

| 7 | 8 | 6 |
|---|---|---|
| 5 | 1 | 2 |

$\xrightarrow{pf=7}$

| 7 | 8 | 9 |
|---|---|---|
| 5 | 1 | 2 |

$\xrightarrow{pf=7}$

| 10 | 8 | 9 |
|---|---|---|
| 3 | 1 | 2 |

$\xrightarrow{pf=8}$

| 10 | 11 | 9 |
|---|---|---|
| 3 | 4 | 2 |

$\xrightarrow{pf=9}$

| 10 | 11 | 12 |
|---|---|---|
| 3 | 4 | 5 |

$\xrightarrow{pf=10}$ Termino con un totale di 10 page fault.

- *Ideale*:
  Come dice il nome questo è un algoritmi perfetto, perchè dovrebbe prevedere il futuro, infatti dovrebbe sapere quali saranno le prossime richieste che un pogramma potrebbe fare.

| 1 | null | null |
|---|---|---|

$\xrightarrow{pf=1}$

| 1 | 2 | null |
|---|---|---|

$\xrightarrow{pf=2}$

| 1 | 2 | 3 |
|---|---|---|

$\xrightarrow{pf=3}$

| 1 | 2 | 4 |
|---|---|---|

$\xrightarrow{pf=4}$

| 1 | 2 | 4 |
|---|---|---|

$\xrightarrow{pf=4}$

| 1 | 2 | 4 |
|---|---|---|

$\xrightarrow{pf=4}$

| 1 | 2 | 5 |
|---|---|---|

$\xrightarrow{pf=5}$

| 1 | 2 | 5 |
|---|---|---|

$\xrightarrow{pf=5}$

| 1 | 2 | 5 |
|---|---|---|

$\xrightarrow{pf=5}$

| 3 | 2 | 5 |
|---|---|---|

$\xrightarrow{pf=6}$

| 3 | 4 | 5 |
|---|---|---|

$\xrightarrow{pf=7}$

| 3 | 4 | 5 |
|---|---|---|

$\xrightarrow{pf=7}$ Termino con un totale di 7 page fault

# Domanda 4

**MAX PUNTI:** 6
**Domanda:**

Data la seguente tabella contenente un insieme di processi:

| *Processo* | *Burst* | *Tempo di arrivo* |
|---|---|---|
| 1 | 3 | 0 |
| 2 | 1 | 1 |
| 3 | 2 | 3 |
| 4 | 4 | 4 |
| 5 | 8 | 1 |

Disegnare lo schema di arrivo dei processi senco gli algoritmi HRRN e RR con quanti di tempo pari a 2.
Per quanto riguarda il RR si immagini che i processi vengano inseriti nella *ready queue* in un tempo tale per cui il tempo di risposta è minimo.
Calcolare tempo di attesa, risposta e turnaround per ogni processo