

# Pandos+ Operating System

Specifiche di Progetto

**FASE 3**

**v.0.1**

Anno Accademico 2021-2022

# Pandos+

- Sistema Operativo in 6 **livelli** di astrazione.

**Livello 6:** Shell interattiva

**Livello 5:** File-system

**Livello 4:** Livello di supporto

**Livello 3:** Kernel del S.O.

FASE1!

**Livello 2:** Gestione delle Code

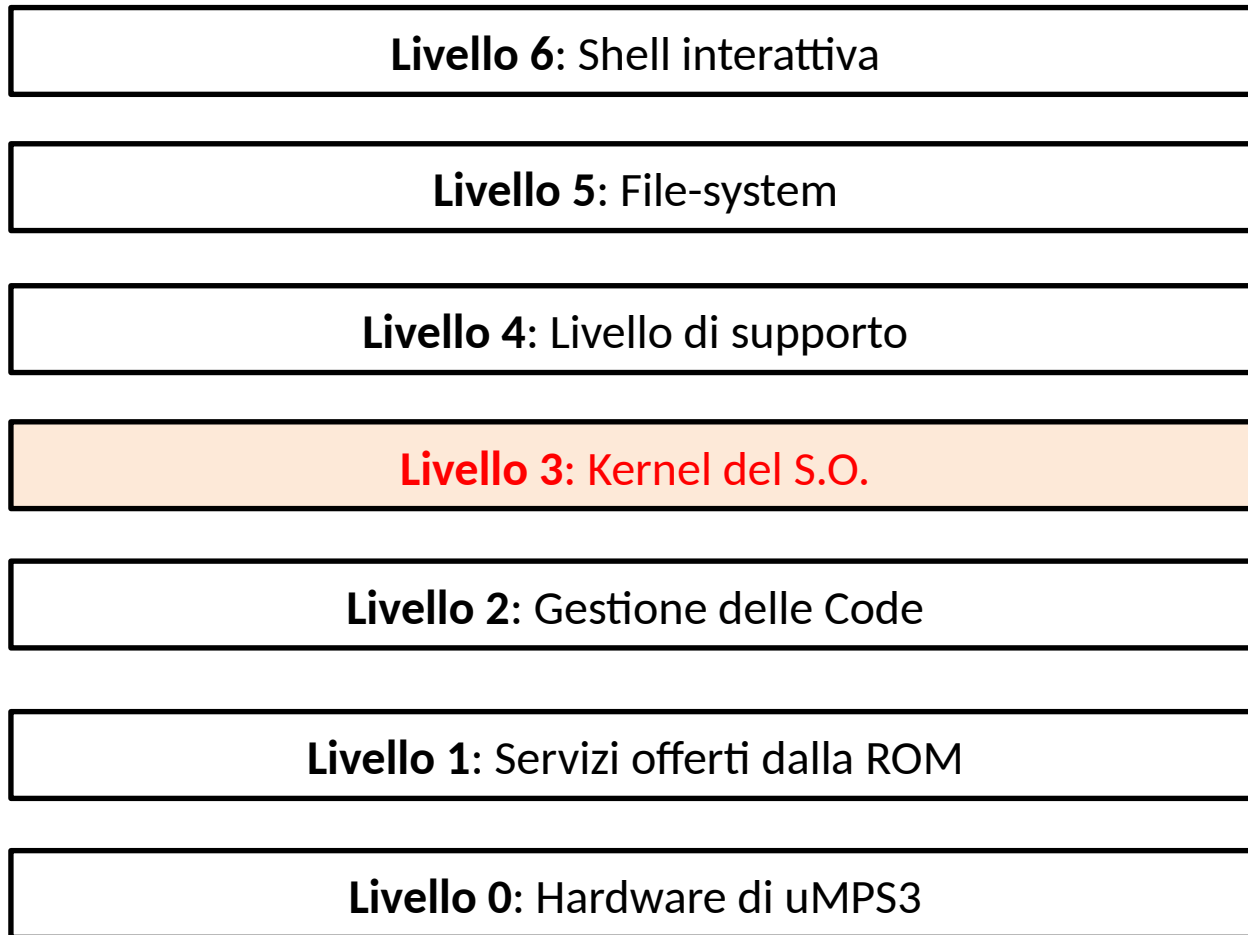
**Livello 1:** Servizi offerti dalla ROM

NOTI

**Livello 0:** Hardware di uMPS3

# Pandos+

- Sistema Operativo in 6 **livelli** di astrazione.



FASE2!

NOTI



# Pandos+

- Sistema Operativo in 6 **livelli** di astrazione.

Livello 6: Shell interattiva

Livello 5: File-system

FASE3!

Livello 4: Livello di supporto

Livello 3: Kernel del S.O.

Livello 2: Gestione delle Code

Livello 1: Servizi offerti dalla ROM

Livello 0: Hardware di uMPS3

NOTI



# Struttura di supporto

```
typedef struct support_t {  
    int          sup_asid;  
    state_t      sup_exceptState[2];  
    context_t    sup_exceptContext[2];  
    pteEntry_t  
        sup_privatePgTbl[USERPGTBLSIZE];  
} support_t;
```

# Livello 4 del S.O.

- **Funzionalità** che il livello di supporto deve gestire:
  - **Memoria virtuale**
  - **Mediazione semplificata con il livello 3**

**Quella che segue e' una visione panoramica delle specifiche; per le informazioni dettagliate e' indispensabile fare riferimento ai manuali di PandOS+ (capitolo 4) e uMPS3.**

# Risultato finale

- Vi verranno forniti 8 processi compilati come eseguibili a parte e impacchettati in un flash device.
- Voi dovrete caricarli dal loro dispositivo e fare in modo che eseguano con successo.
- Oltre allo scheduling che avete già implementato entrano in gioco memoria virtuale e delle system call per una interazione più semplice con i dispositivi.

# Risultato finale

Non e' necessario modificare il lavoro svolto in fase 2: la fase 3 si costruisce sopra di esso.

Interrupt, System Call < 0, scheduler e strutture dati continuano a funzionare in maniera identica.

(cio' non significa che non e' permesso fare correzioni alle fasi precedenti)



# Inizializzazione

La parte di inizializzazione di fase 2 (funzione main) rimane uguale.

Il processo di test non e' piu' fornito da noi ma costruito da voi.

La funzione test dovra':

- Inizializzare le strutture dati di fase 3
- Caricare e far partire l'esecuzione dei processi che vi forniamo
- Mettersi in attesa della fine di questi ultimi (e' opportuno che il sistema si fermi una volta terminati tutti)

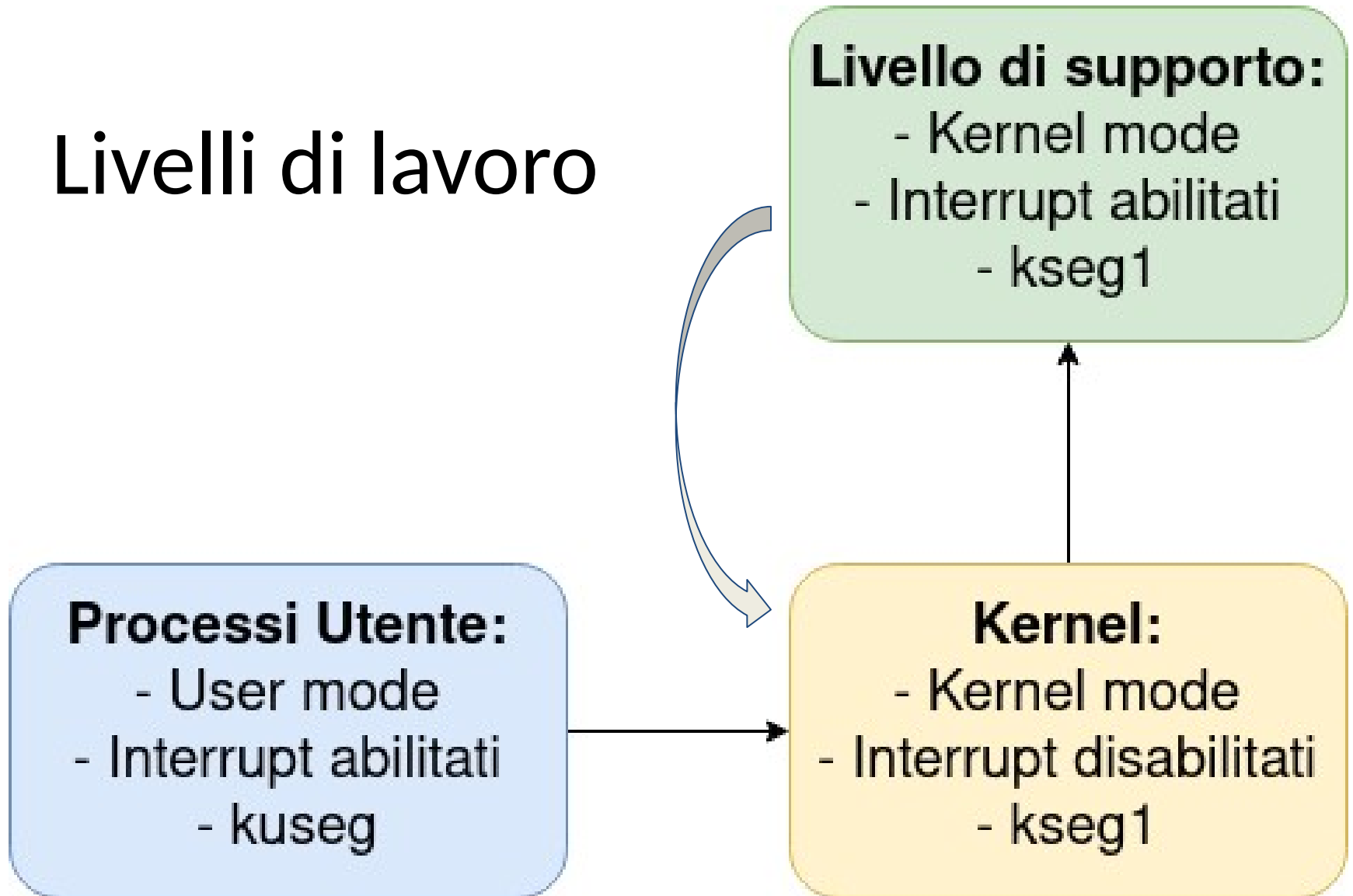
# Esecuzione

Una volta fatti partire i processi di test l'esecuzione e' in mano a loro.

Oltre a intervallare la loro attivita' con lo scheduling portato avanti dal Kernel il livello di supporto dovra' gestire adeguatamente le eccezioni relative alla memoria virtuale (TLB).

Infine vanno aggiunte alcune System Call (sempre al livello di supporto).

# Livelli di lavoro



# Livelli di lavoro

Nuove operazioni di fase 3:

- Le system call lavorano a livello di supporto e del kernel
- Gli eventi TLB-Refill lavorano a livello del kernel
- I page fault lavorano al livello di supporto

# Memoria Virtuale

I processi utenti eseguono in uno spazio di memoria virtuale (kuseg) univoco e contiguo a partire dall'indirizzo 0x8000.0000.

Quando avviene un accesso in memoria uMPS3 cerca l'indirizzo fisico corrispondente nel suo Translation Lookaside Buffer (TLB).

# Memoria Virtuale



Figure 6.6: **EntryHi**

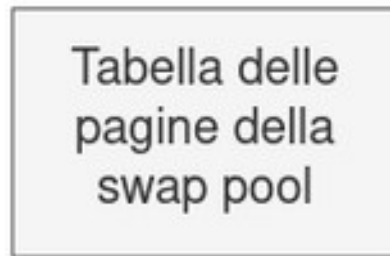
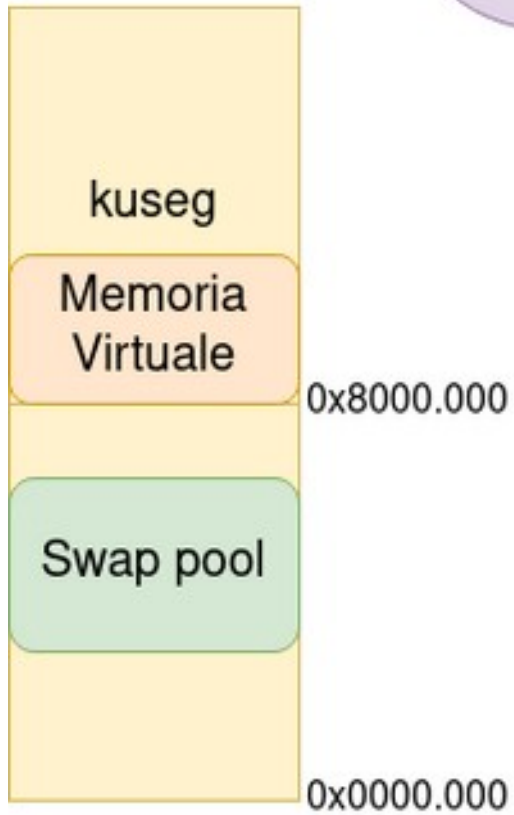
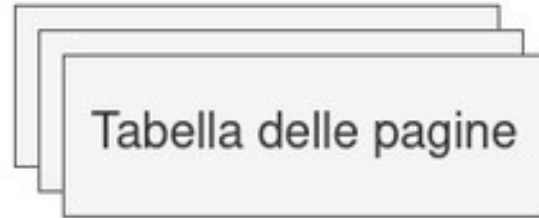
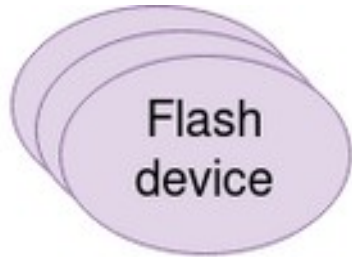


Figure 6.7: **EntryLo**

# I Processi

- Vi verranno forniti 8 processi sotto forma di dispositivi flash precompilati.
- Ogni processo avra' assegnato un id (ASID) da 1 a 8 (diverso dal process ID).
- Ogni processo ha assegnato un terminale, una stampante e un dispositivo flash (corrispondente al proprio ASID)
- Ogni processo lavora sulla stessa area di memoria virtuale
- Il livello di supporto gestisce la paginazione
- La memoria dedicata a un processo e' fissa: 32 \* 4 KB

# I Processi





# Tabella delle pagine del processo

	EntryHI			EntryLo				
	VPN	ASID		PEN	N	D	V	G
0	0x80000	$i$			1	0		
1	0x80001	$i$			1	0		
•								
•								
•								
30	0x8001E	$i$			1	0		
31	0xBFFFF	$i$			1	0		

Figure 4.2: Layout of U-proc  $i$ 's Page Table

# Tabella delle pagine del processo

- Collega l'indirizzo virtuale (VPN) a quello fisico nella swap pool (PFN)
- Gli indirizzi virtuali sono fissi (da 0x80000 a 0x8001E + 0xBFFFF per lo stack)
- Gli indirizzi fisici vanno aggiornati di volta in volta
- ASID e' l'id del processo
- Il bit N non e' usato
- I bit D, V e G devono avere valori fissi 1, 0 e 1

# Swap Pool

- Area di memoria ram (fisica) dedicata ad ospitare le aree di lavoro fisiche dei processi
- I frame dei processi che vi risiedono non sono necessariamente ordinati o contigui
- Deve contenere un numero di frame pari al numero di processi per due.
- Può stare in un punto qualunque della memoria fisica; il manuale di PandOS suggerisce 0x20020000

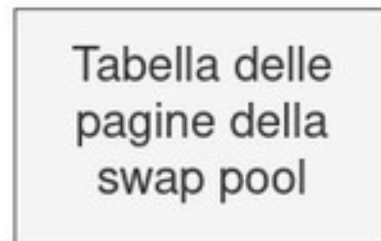
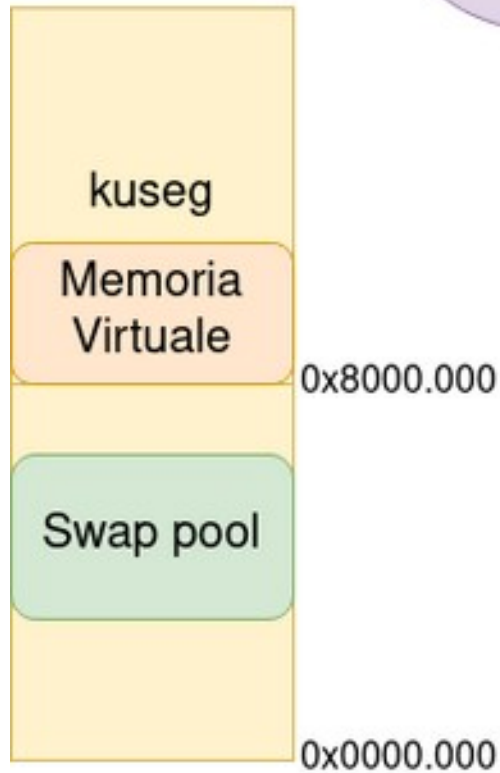
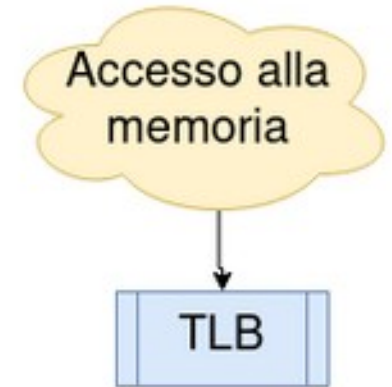
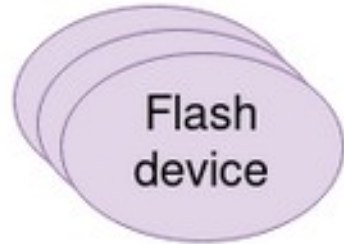
# Tabella delle pagine della Swap Pool

- Struttura dati che tiene traccia, per ogni frame, di quale processo lo stia usando
- Una entry per frame
- Contiene ASID del processo occupante, VPN e puntatore alla entry corrispondente nella tabella delle pagine del processo
- Un frame non utilizzato viene indicato con ASID -1
- E' una struttura dati condivisa, e come tale deve essere protetta da un semaforo

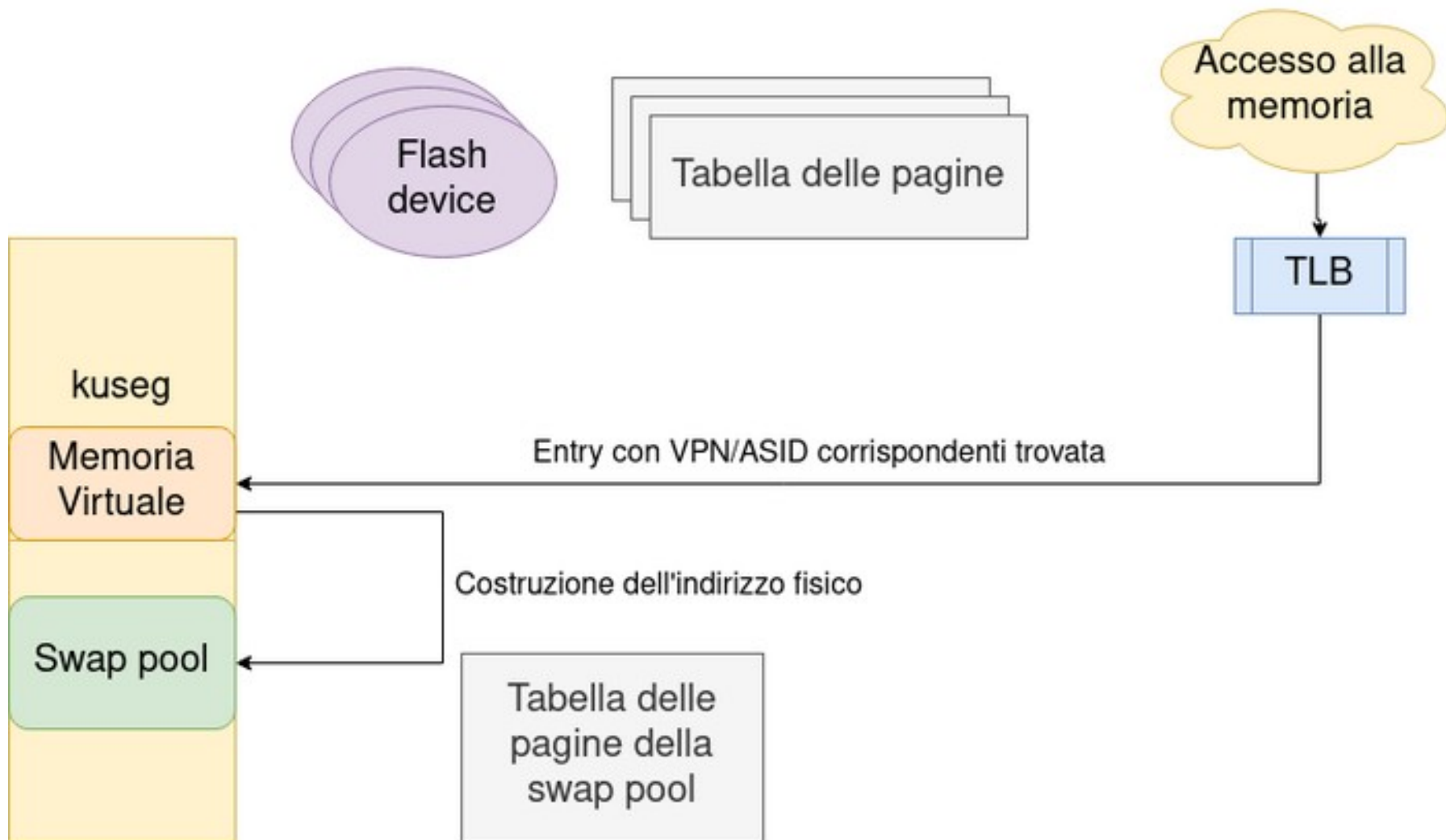
# Flash Storage

- I dispositivi flash che uMPS3 caricherà sono già preparati per voi
- Funzionano a blocchi di 4KB come tutto il resto
- Del formato è sufficiente sapere il punto di partenza: 0x800000B0
- Man mano che il processo richiede indirizzi di memoria il dispositivo verrà letto e il contenuto caricato nella swap pool
- Se un blocco deve essere rimosso per fare spazio ad altri processi il contenuto del dispositivo va aggiornato per non perderlo

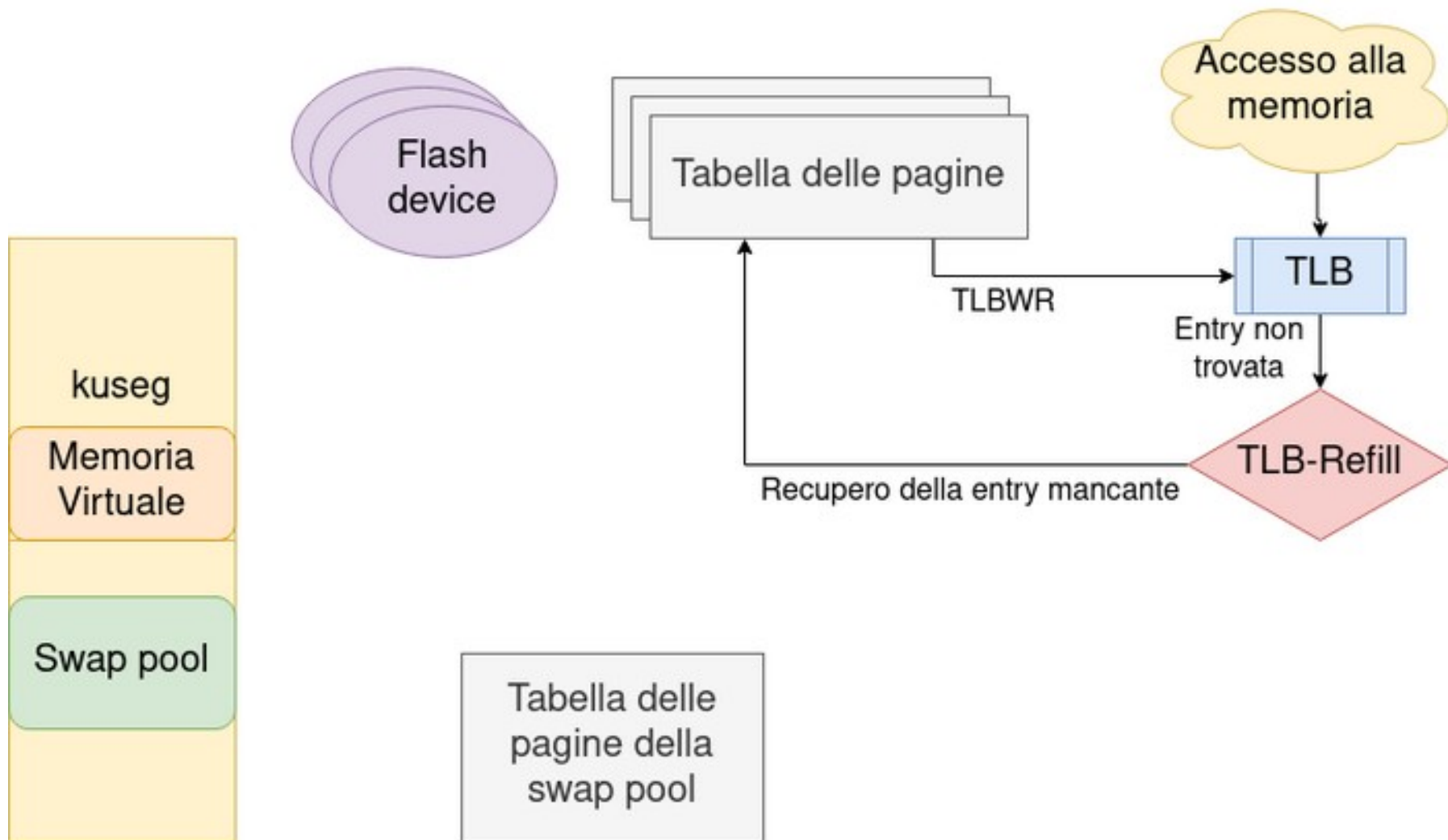
# I Processi



# I Processi

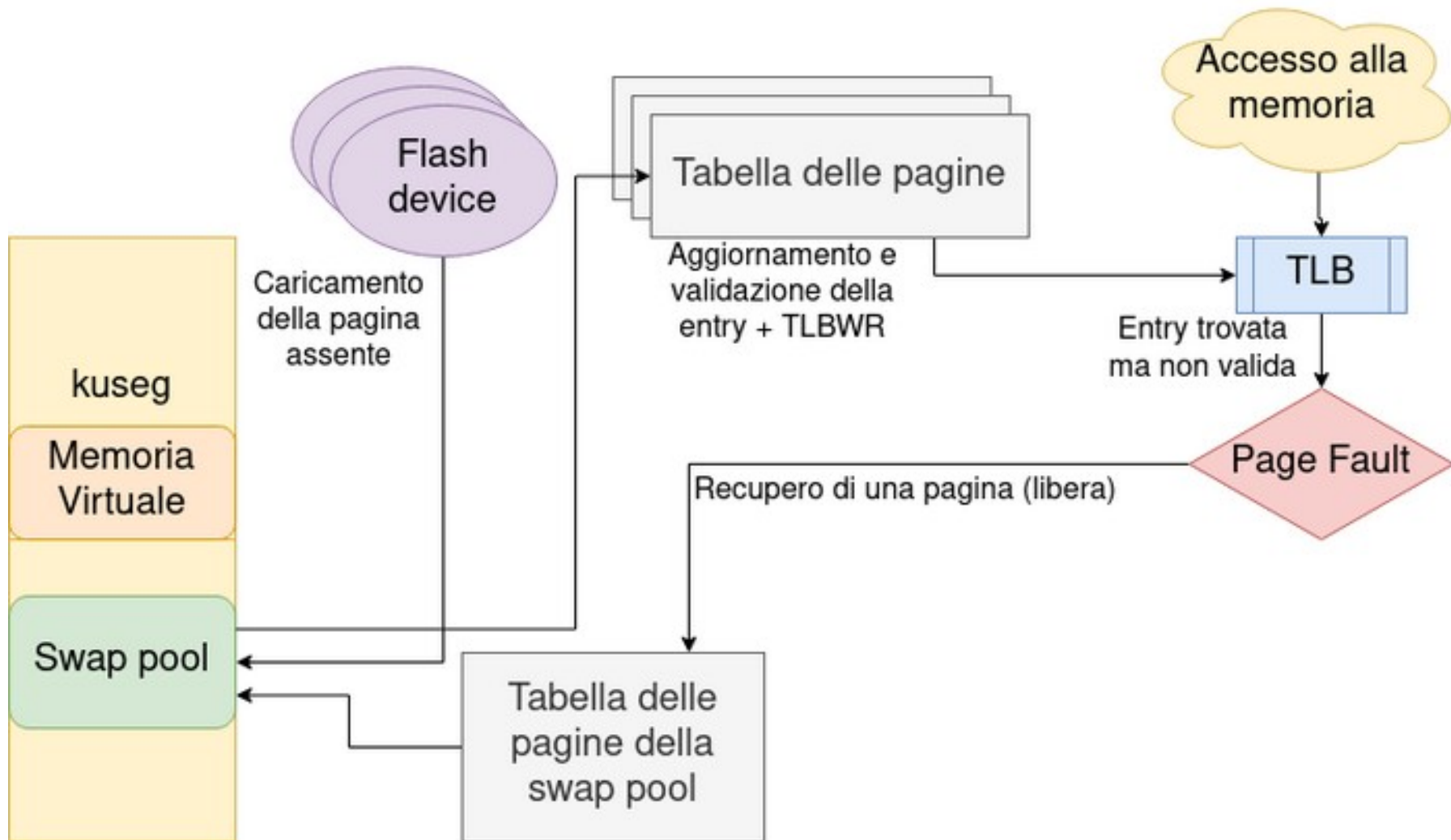


# I Processi

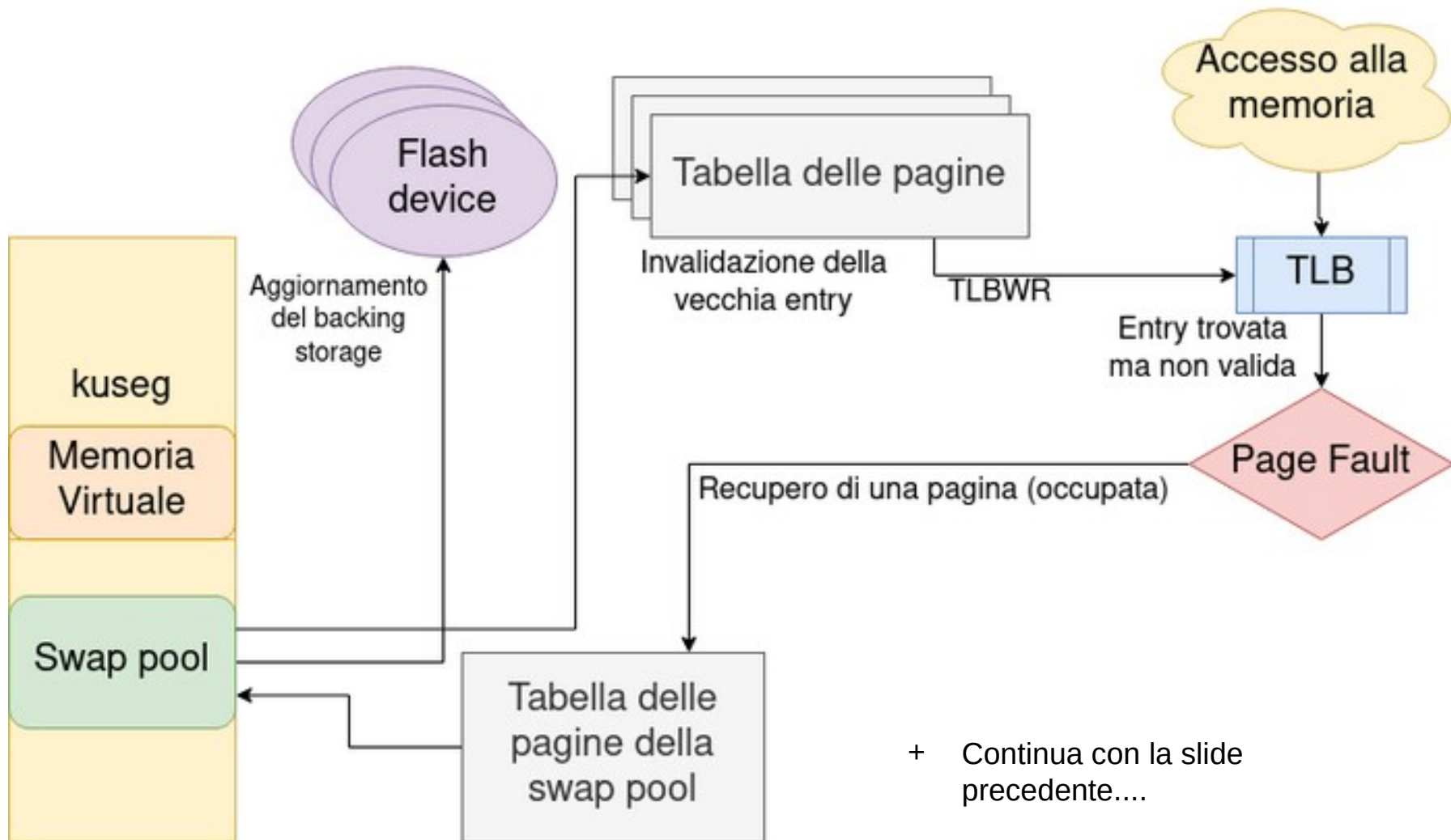




# I Processi



# I Processi



# Eccezioni

- I processi utente devono essere inizializzati con due nuovi gestori di eccezioni nella struttura di supporto
- Questi vengono invocati dal kernel nell'eventualità di un Page Fault, una system call > 0 o un Program Trap
- Al livello del kernel bisogna aggiungere un gestore degli eventi TLB-Refill
- Tutta la restante gestione di fase 2 rimane (potenzialmente) intoccata

# TLB-Refill

- Viene fatto un accesso a memoria virtuale, ma il VPN richiesto non e' nel TLB
- Il vostro gestore deve cercare il VPN (trovato nel campo entry\_hi dello stato salvato) nella tabella delle pagine del processo corrente e caricare la entry corrispondente.
- Fatto questo si ricarica lo stato salvato; verra' fatto un nuovo tentativo di accesso

# Page Fault

- Diverso da TLB-Refill: il VPN e' stato trovato ma il bit V non e' attivo
- Significa che la pagina fisica non e' effettivamente caricata in memoria
- E' necessario trovare un frame (libero o meno) nella swap pool e "validare" la entry nella TLB aggiungendolo come PFN

# System Call

- Le system call negative continuano ad essere gestite normalmente dal Kernel di fase 2.
- A queste se ne affiancano altre 5 che si piazzano al livello di supporto
- Le system call da 1 a 5 vengono passate dal Kernel al gestore che dovrete inserire nella struttura di supporto
- L'esecuzione del livello di supporto puo' comunque chiamare system call del kernel

# Gestione delle SYSCALL

- SYSCALL 1 (**SYS1**) Get\_TOD

```
void SYSCALL(GET_TOD, 0, 0, 0)
```

- Restituisce il numero di microsecondi passati dall'accensione del sistema

# Gestione delle SYSCALL

- SYSCALL 2 (**SYS2**) Terminate

```
void SYSCALL(TERMINATE, 0, 0, 0)
```

- Un semplice wrapper per la system call corrispondente del kernel.



# Gestione delle SYSCALL

- SYSCALL 3 (**SYS3**) Write\_To\_Printer

```
void SYSCALL(WRITEPRINTER, char *str, int len, 0)
```

- Richiede una stampa ininterrotta della stringa richiesta sulla stampante associata al processo
- Fa sostanzialmente il lavoro della funzione print di p2test.c (ma per una stampante)
- se l'indirizzo e' fuori dalla memoria virtuale del processo o la lunghezza richiesta e' zero deve risultare nella sua terminazione
- Restituisce il numero di caratteri stampati in caso di successo, l'opposto dello stato del dispositivo in caso contrario

# Gestione delle SYSCALL

- SYSCALL 4 (**SYS4**) Write\_To\_Terminal

```
void SYSCALL(WRITETERMINAL, char *str, int len, 0)
```

- Richiede una stampa ininterrotta della stringa richiesta sul terminale associato al processo
- Fa sostanzialmente il lavoro della funzione print di p2test.c
- se l'indirizzo e' fuori dalla memoria virtuale del processo o la lunghezza richiesta e' zero deve risultare nella sua terminazione
- Restituisce il numero di caratteri stampati in caso di successo, l'opposto dello stato del dispositivo in caso contrario

# Gestione delle SYSCALL

- SYSCALL 5 (**SYS5**) Read\_From\_Terminal

```
void SYSCALL(READTERMINAL, char *str, int len, 0)
```

- Legge una riga (fino al newline) dal terminale associato al processo
- Mentre l'input viene letto il processo e' sospeso
- se l'indirizzo e' fuori dalla memoria virtuale del processo o la lunghezza richiesta e' zero deve risultare nella sua terminazione
- Restituisce il numero di caratteri letti in caso di successo, l'opposto dello stato del dispositivo in caso contrario

# Et Cetera

- Syscall > 5
- Program trap
- Eccezioni relative alla memoria virtuale che non siano Page Fault o TLB-Refill

...devono risultare nella terminazione del processo corrente

# Inizializzazione dei processi

Stato del processo utente:

- il registro entry\_hi deve contenere l'ASID
- stack pointer = 0xC000.0000
- program counter = 0x8000.00B0
- User mode e interrupt abilitati

# Inizializzazione dei processi

Struttura di supporto:

- Kernel mode e interrupt abilitati
- `sup_exceptContext[0].pc` = gestore dei trap TLB (page fault)
- `sup_exceptContext[1].pc` = gestore delle eccezioni generiche
- `sup_exceptContext[0].stackPtr` =  $\text{ramTop} - (\text{ASID} * 4\text{K} * 2) + 4\text{K}$
- `sup_exceptContext[1].stackPtr` =  $\text{ramTop} - (\text{ASID} * 4\text{K} * 2)$

# Inizializzazione dei processi

Page table:

31 entry con Virtual Page Number a partire da 0x80000 crescendo di un frame alla volta + uno dedicato allo stack con valore 0xBFFFF

# Note generali

- Siccome il livello di supporto lavora con interrupt abilitati non e' garantito l'accesso esclusivo alle strutture dati condivise: serve un semaforo dedicato.



# Riassumendo

Vi viene fornita un archivio tar.gz con i processi da compilare e inserire nei dispositivi flash + la configurazione uMPS3 da usare

L'esecuzione del test e' corretta se tutti i processi arrivano correttamente alla fine.

Nota: il processo 1 richiede un input dal terminale; dovete fornirlo voi a mano.

# Riassumendo

Tutti i dettagli sono spiegati in profondita' nel capitolo 4 del libro

<http://www.cs.unibo.it/~renzo/so/pandos/docs/pandos.pdf>

E nel manuale di uMPS3

<http://www.cs.unibo.it/~renzo/so/pandos/docs/uMPS3princOfOperations.pdf>

# Gestione del progetto

- Cosa consegnare:
  - Sorgenti (al completo)
  - Makefile o build tool analogo
  - Documentazione (.pdf o .txt, evitate i .docx)
  - file AUTHORS.txt, README.txt, etc
- Nella documentazione indicate scelte progettuali ed eventuali difficoltà/errori presenti.

# Gestione del progetto

- **DATE** di consegna
  - 19 giugno
  - 17 luglio
  - 18 settembre
- La consegna deve essere effettuata come per Fase2 spostando l'archivio contenente il progetto nella directory di consegna di Fase3 (submit\_phase3) associata al gruppo.