PKaya Operating System

Specifiche di Progetto

FASE 2

Anno Accademico 2011-2012

pKaya OS

- **P-Kaya**: Evoluzione di Kaya O.S., a sua volta evoluzione di una lunga lista di S.O. proposti a scopo didattico (HOCA, TINA, ICARO, etc).
- Caratteristiche del progetto P-Kaya:
  - Basato **esclusivamente** su architettura **uMPS2**
  - Disegnato per sistema **Multi-Processore (MP)** (Da qui il nome di Parallel Kaya)
- Architettura basata su **sei** *livelli di astrazione*, sul modello del S.O. THE proposto da Dijkstra in un suo articolo del 1968 …

pKaya OS

- 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.*

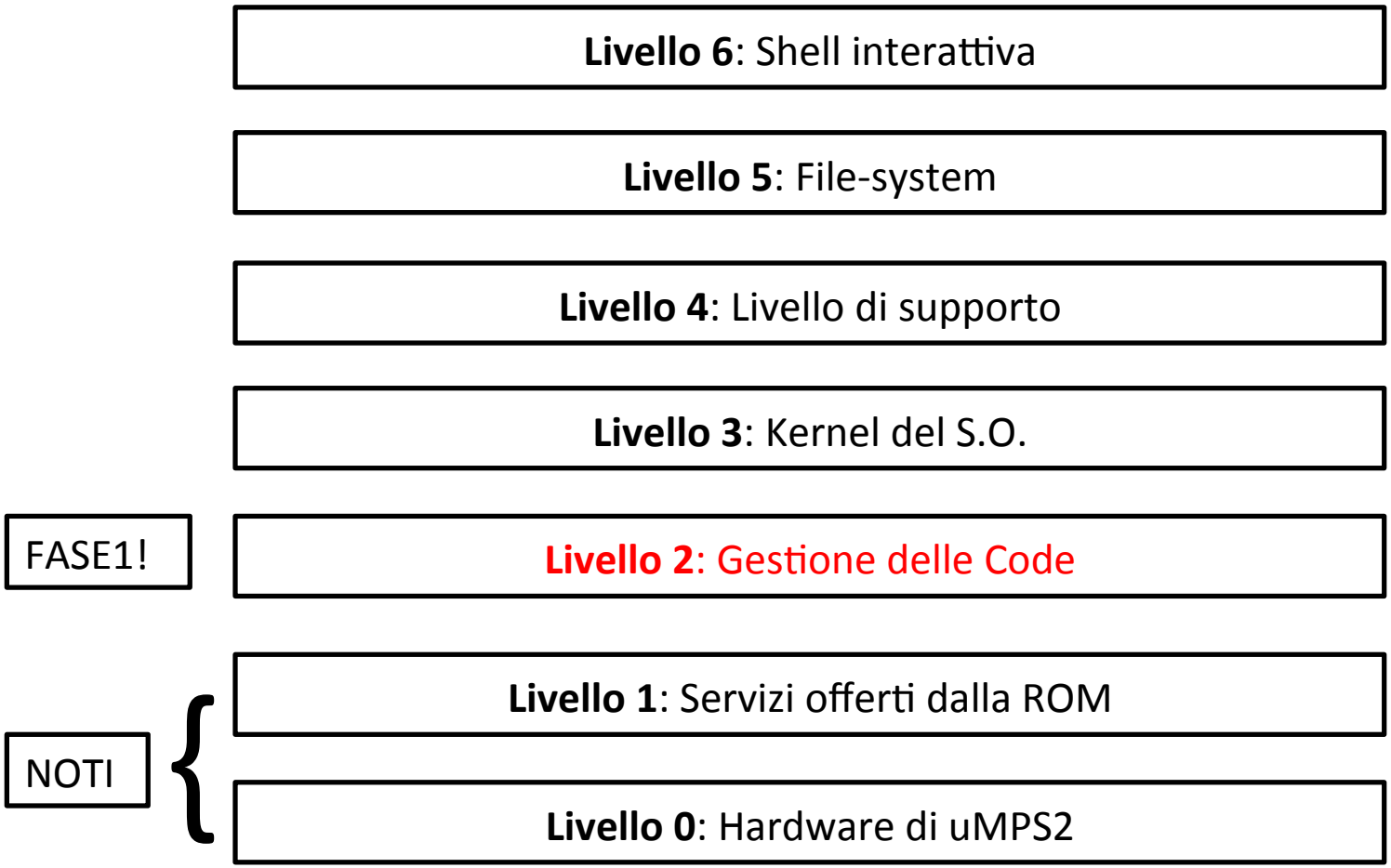
Livello 2: *Gestione delle Code*

Livello 1: *Servizi offerti dalla ROM*

Livello 0: *Hardware di uMPS2*

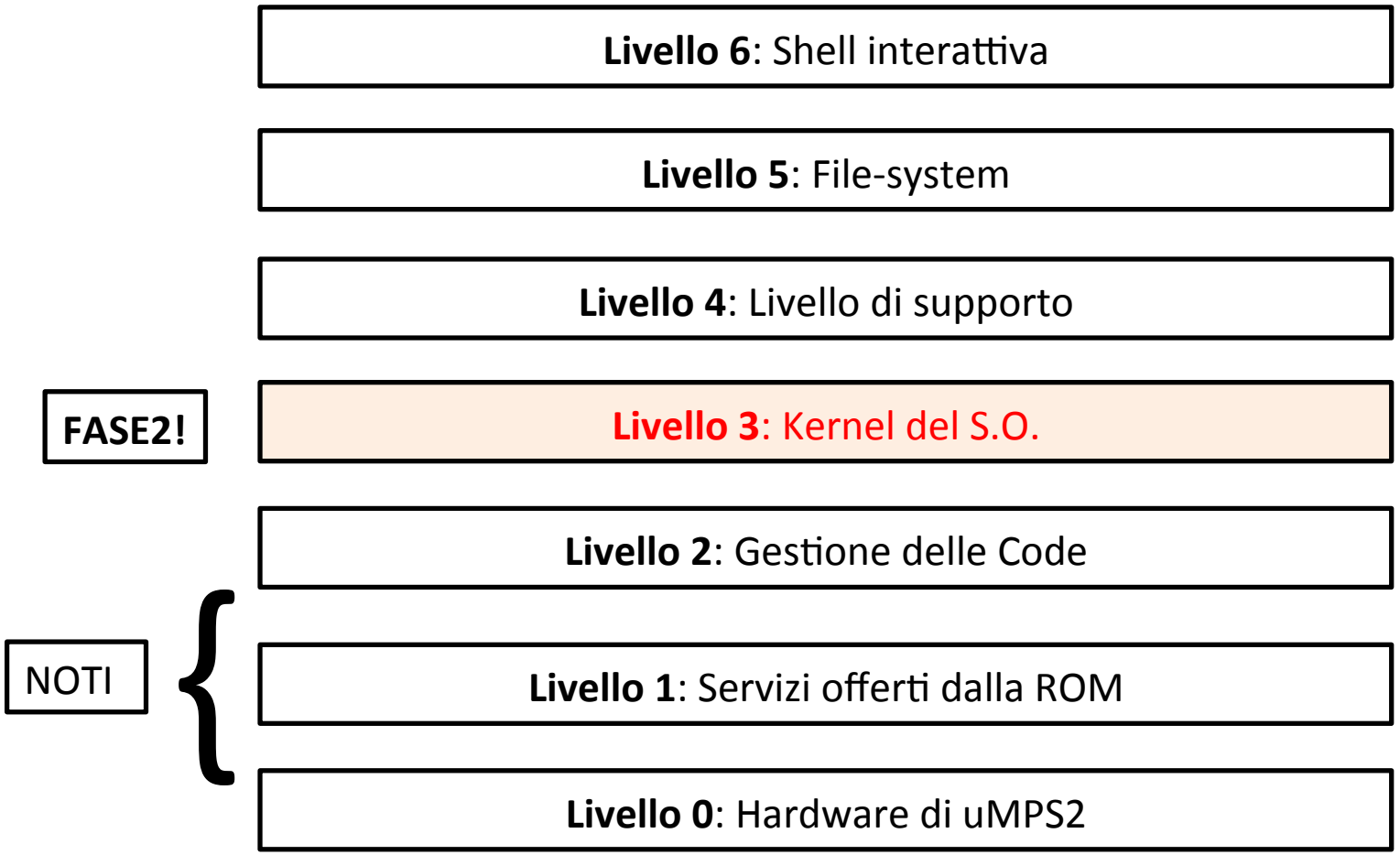
pKaya OS

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



pKaya OS

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



Livello 3 del S.O.

- **Funzionalita'** che il nucleo deve gestire:
  - **Inizializzazione** del sistema
  - **Scheduling** dei processi
  - Gestione delle **syscall**
  - Gestione degli **interrupt**
  - Gestione delle **eccezioni** (BreakPoints, PgmTrap, TLB Exceptions)
- **Novita'** rispetto a versioni precedenti di Kaya → Funzionalita' del nucleo da gestire in **ambiente multi-processore**!!

Livello 3 del S.O.

- **Specifiche di sistema:**
  - **MAX_PROC**: numero massimo di processi attivi nel sistema.
  - **NUM_CPU**: numero di CPU a disposizione sulla macchina (1<=NUM_CPU<=**MAX_CPU**).
    - In uMPS2, **MAX_CPU**=16.
    - In fase di testing → creare una configurazione di uMPS2, settando un valore di riferimento per NUM_CPU (es. 4).
    - Ma … Il nucleo deve funzionare correttamente qualsiasi sia la configurazione di NUM_CPU!

Livello 3 del S.O.

- **Strutture Dati di Fase1:**

  - Liste di **PCB** → es. lista dei processi attivi.
  - Active Semaphore List (**ASL**) → es. liste dei processi bloccati sui semafori dei device.

  Utilizzare le strutture dati di Fase1 ma **tenere conto dell'ambiente multi-processore**:

  - Evitare **race-conditions**
  - Massimizzare **re-entrancy** del codice

Livello 3 del S.O.

- **Strutture Dati di Fase1:**

  - Liste di **PCB** → es. lista dei processi attivi.
  - Active Semaphore List (**ASL**) → es. liste dei processi bloccati sui semafori dei device.

  Utilizzare le strutture dati di Fase1 ma **tenere conto dell'ambiente multi-processore**:

  - Evitare **race-conditions**
  - Massimizzare **re-entrancy** del codice

Livello 3 del S.O.

- Implementazione di **sezioni critiche**:

```
int CAS(int *p, int expected_val, int new_val)
```

- Istruzione **atomica**
  - Se *p==expected_val → allora aggiorna *p con new_val e restituisce 1
  - … altrimenti restituisce 0.

- Gestire l'accesso a *strutture dati condivise*, cercando di massimizzare l'efficienza del nucleo!

Livello 3 del S.O.

- Esempio di utilizzo

```
int p=FREE;
… … …
… … …
while (!CAS(&p,FREE,BUSY));
insertBlocked(SEM_CLOCK,pcb);
CAS(&p,BUSY,FREE);
```

Livello 3 del S.O.

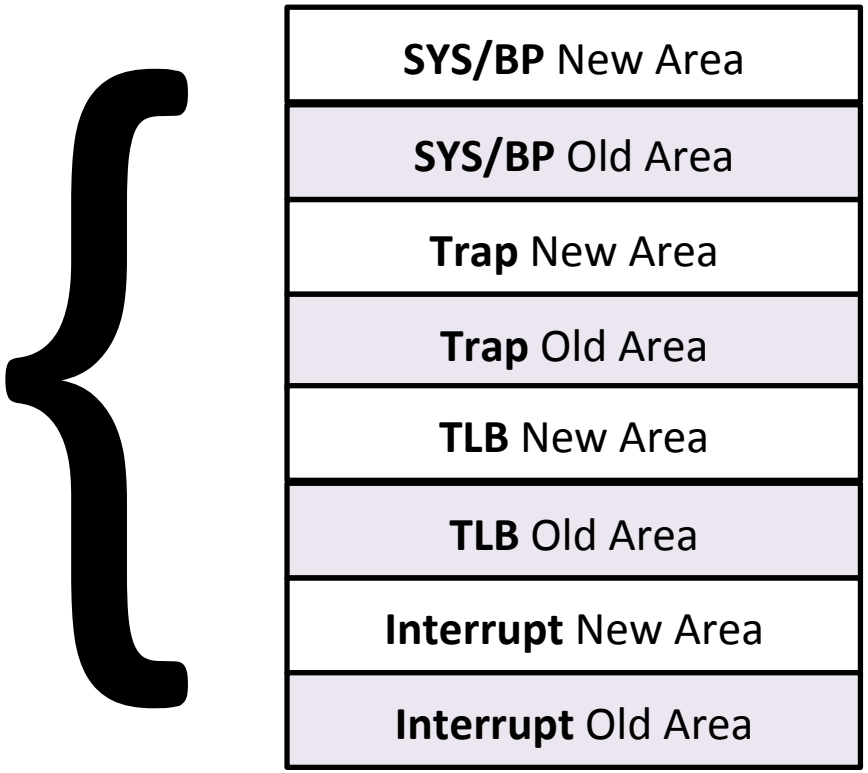
- **Funzionalita'** che il nucleo deve gestire:
  - **Inizializzazione** del sistema
  - **Scheduling** dei processi
  - Gestione delle **syscall**
  - Gestione degli **interrupt**
  - Gestione delle **eccezioni** (BreakPoints, PgmTrap, TLB Exceptions)
- **Novita'** rispetto a versioni precedenti di Kaya → Funzionalita' del nucleo da gestire in **ambiente multi-processore**!!

Inizializzazione del sistema

- Entry-point di Kaya: void **main**()
- Populare le **New Areas** nel ROM Reserved Frame

4 Aree New/Old per ciascuna CPU …

Totale New Area da popolare: **NUM_CPU*4**



Inizializzazione del sistema

- Per ogni **New Area**:

1. Inizializzare il PC all'indirizzo dell'**handler** del nucleo che gestisce quell'eccezione.
2. Inizializzare **$SP** a **RAMPTOP**
3. Inizializzare il registro di **status**:
   - mascherare interrupt
   - disabilitare virtual memory
   - settare kernel mode ON
   - abilitare Processor Local Timer (**PLT**)

Inizializzazione del sistema

- Inizializzare **strutture dati** di Phase1:

`initPcbs()`
`initASL()`

- Inizializzare **variabili** del kernel:

Process Counter → contatore processi attivi

Soft-block Counter → contatore processi bloccati per I/O

…. ….

- Creare **semafori** di sistema:
  - Uno per ogni device o sub-device (es. terminali)
  - Uno per lo **pseudo-clock timer**
  - Settare valore iniziale del semaforo (0)

Inizializzazione del sistema

- **Instanziare** il PCB e lo stato del processo **test**
  - Interrupt abilitati
  - Virtual Memory OFF
  - Processor Local Timer abilitato
  - Kernel-Mode ON
  - $SP=RAMTOP-FRAMESIZE
  - Settare PC all'entry-point del test

    pstate.pc_epc=(memaddr) test
- **Inseririre** il processo nella Ready Queue

Inizializzazione del sistema

- **Inizializzare** lo stato delle **CPU**
  - All'avvio, uMPS2 avvia solo il **processore 0**
  - Per avviare le altre CPU e' necessario farlo esplicitamente tramite la funzione:

```
void INITCPU(uint32_t cpuid, state_t *start_state, state_t *state_areas)
```

- Invia un comando **RESET** al processore **cpuid**
- Salva nella ROM l'indirizzo della New/Old Areas puntate da **state_areas**
- Carica lo stato del processore da **start_state**

Livello 3 del S.O.

- **Funzionalita'** che il nucleo deve gestire:
  - **Inizializzazione** del sistema
  - **Scheduling** dei processi
  - Gestione delle **syscall**
  - Gestione degli **interrupt**
  - Gestione delle **eccezioni** (BreakPoints, PgmTrap, TLB Exceptions)
- **Novita'** rispetto a versioni precedenti di Kaya → Funzionalita' del nucleo da gestire in **ambiente multi-processore**!!

Scheduler di Sistema

- Funzionalita' dello scheduler:
  - **Context-switch** tra processi. Ad ogni processo deve essere assegnato un time-slice di 5 millisecondi (**TIME_SLICE**).
  - **Deadlock** detection:

1. Se Process Count ==0 → **HALT**
2. Se Process Count > 0 && Soft Block Count ==0 → Deadlock, **PANIC**
3. Se Process Count ==0 && Soft Block Count >0 → Stato di Attesa, **WAIT**

Scheduler di Sistema

- Funzionalita' dello scheduler:
  - Gestione della **priorita'** dei processi

    MAX_PRIORITY: livelli di priorita' (es. 5)

    Scelte progettuali:

    1. Priorita' **statiche**

    2. Priorita' **statiche** con meccanismi di **aging** (in modo da evitare rischi di starvation)

    3. Code multiple per differenti priorita' …

Scheduler di Sistema

- Funzionalita' dello scheduler:
  - Supporto per **esecuzione** in ambiente **MP**

  Scelte progettuali:

  1. Singola coda di processi ready

     (NON molto efficiente …)
  2. Code multiple (una per ogni processore)
  3. Meccanismi di load-balancing tra processori
  4. ….

  Scelte progettuali completamente libere … MA l'efficienza del SO costituisce un parametro importante di valutazione!

Livello 3 del S.O.

- **Funzionalita'** che il nucleo deve gestire:
  - **Inizializzazione** del sistema
  - **Scheduling** dei processi
  - Gestione delle **syscall**
  - Gestione degli **interrupt**
  - Gestione delle **eccezioni** (BreakPoints, PgmTrap, TLB Exceptions)
- **Novita'** rispetto a versioni precedenti di Kaya → Funzionalita' del nucleo da gestire in **ambiente multi-processore**!!

Gestione delle SYSCALL

- Gestione delle SYSCALL e BREAKpoint
  - Una SYSCALL si distingue da un BREAKpoint attraverso il contenuto del registro Cause.ExcCode (SYS=8, BP=9)
  - I parametri della SYSCALL/BP si trovano nei registri a0-a3
  - Nel caso delle SYSCALL, il registro a0 identifica la SYSCALL specifica richiesta …
  - 11 possibili SYSCALL, con codici [1…11]

Gestione delle SYSCALL

- Numero della SYS specificata nel registro **a0** …

Routine del nucleo di gestione delle SYS/BP

(l'indirizzo della NewArea deve essere settato opportunamente in fase di system setup)

Gestione delle SYSCALL

- SYSCALL 1 (**SYS1**) **Create_Process**

```
int SYSCALL(CREATEPROCESS, state_t *statep, int priority)
```

  - Quando invocata, la **SYS1** determina la creazione di un processo figlio del processo chiamante
  - Registro **a1**: Indirizzo fisico dello state_t del nuovo processo
  - Registro **a2**: Priorita' del nuovo processo
  - Valori di ritorno nel registro **v0**:
    - 0 nel caso di creazione corretta
    - -1 nel caso di errore

Gestione delle SYSCALL

- SYSCALL 2 (**SYS2**) **Create_Brother**

```
int SYSCALL(CREATEBROTHER, state_t *statep, int priority)
```

  - Quando invocata, la **SYS2** determina la creazione di un processo fratello del processo chiamante
  - Registro **a1**: Indirizzo fisico dello state_t del nuovo processo
  - Registro **a2**: Priorita' del nuovo processo
  - Valori di ritorno nel registro **v0**:
    - 0 nel caso di creazione corretta
    - -1 nel caso di errore

Gestione delle SYSCALL

- SYSCALL 3 (**SYS3**) **Terminate_Process**

```
void SYSCALL(TERMINATEPROCESS)
```

  - Quando invocata, la **SYS3** termina il processo corrente e tutta la sua progenie.

  - ATTENZIONE: Ricordarsi di liberare le risorse utilizzate da ogni processo che si intende terminare …

Gestione delle SYSCALL

- SYSCALL 4 (**SYS4**) **Verhogen**

```
void SYSCALL(VERHOGEN, int semKey)
```

  - Quando invocata, la **SYS4** esegue una V sul semaforo con chiave semKey
  - Registro **a1**: chiave del semaforo su cui effettuare la V.

Gestione delle SYSCALL

- SYSCALL 5 (**SYS5**) **Passeren**

```
void SYSCALL(PASSEREN, int semKey)
```

  - Quando invocata, la **SYS5** esegue una P sul semaforo con chiave semKey
  - Registro **a1**: chiave del semaforo su cui effettuare la P.

Gestione delle SYSCALL

- SYSCALL 6 (**SYS6**) **Get_CPU_Time**

```
cpu_t SYSCALL(GETCPUTIME)
```

  - Quando invocata, la **SYS6** restituisce il tempo di CPU (in microsecondi) usato dal processo corrente.
  - Registro **v0**: Valore di ritorno

Gestione delle SYSCALL

- SYSCALL 7 (**SYS7**) **Wait_For_Clock**

```
void SYSCALL(WAITCLOCK)
```

  - Quando invocata, la **SYS7** esegue una P sul semaforo associato allo **pseudo-clock timer**. La V su tale semaforo deve essere eseguito dal nucleo ogni 100 millisecondi (tutti i processi in coda su tale semaforo devono essere sbloccati).

Gestione delle SYSCALL

- SYSCALL 8 (**SYS8**) **Wait_For_IO_Device**

```
int SYSCALL(WAITIO, int intNo, int dnum, int waitForTermRead)
```

  - Quando invocata, la **SYS8** esegue una P sul semaforo associato al device identificato da intNo, dnume e waitForTermRead
  - Registro **a1**: linea di interrupt
  - Registro **a2**: device number
  - Registro **a3**: operazione di terminal read/write
  - Registro **v0**: status del device.

Gestione delle SYSCALL

- SYSCALL 9 (**SYS9**) **Specify_PRG_State_Vector**

```
void SYSCALL(SPECPRGVEC, state_t* oldp, state_t *newp)
```

  - Quando invocata, la **SYS9** consente di definire gestori di **PgmTrap** per il processo corrente.
  - Registro **a1**: Indirizzo della OLDArea in cui salvare lo stato corrente del processore
  - Registro **a2**: Indirizzo della NEWArea del processore (da utilizzare nel caso si verifichi un PgmTrap)

Gestione delle SYSCALL

- SYSCALL 10 (**SYS10**) **Specify_TLB_State_Vector**

```
void SYSCALL(SPECTLBVEC, state_t* oldp, state_t *newp)
```

  - Quando invocata, la **SYS10** consente di definire gestori di TLB Exception per il processo corrente.
  - Registro **a1**: Indirizzo della OLDArea in cui salvare lo stato corrente del processore
  - Registro **a2**: Indirizzo della NEWArea del processore (da utilizzare nel caso si verifichi una TLB Exception)

Gestione delle SYSCALL

- SYSCALL 11 (**SYS11**) **Specify_SYS_State_Vector**

```
void SYSCALL(SPECSYSVEC, state_t* oldp, state_t *newp)
```

  - Quando invocata, la **SYS11** consente di definire gestori di SYS/BP Exception per il processo corrente.
  - Registro **a1**: Indirizzo della OLDArea in cui salvare lo stato corrente del processore
  - Registro **a2**: Indirizzo della NEWArea del processore (da utilizzare nel caso si verifichi una SYS/BP Exception)

Livello 3 del S.O.

- **Funzionalita'** che il nucleo deve gestire:
  - **Inizializzazione** del sistema
  - **Scheduling** dei processi
  - Gestione delle **syscall**
  - Gestione degli **interrupt**
  - Gestione delle **eccezioni** (BreakPoints, PgmTrap, TLB Exceptions)
- **Novita'** rispetto a versioni precedenti di Kaya → Funzionalita' del nucleo da gestire in **ambiente multi-processore**!!

Gestione degli interrupt

- Interrupt=eventi asincroni legati ad IO/Timers

Routine del nucleo di gestione degli Interrupt

(l'indirizzo della NewArea deve essere settato opportunamente in fase di system setup)

Gestione degli interrupt

- Tabella degli interrupt …

Interrupt Line	Device Class
0	Inter-processor interrupts
1	Processor Local Timer
2	Bus (Interval Timer)
3	Disk Devices
4	Tape Devices
5	Network (Ethernet) Devices
6	Printer Devices
7	Terminal Devices

Interrupt che il nucleo deve essere in grado di gestire

Gestione degli interrupt

- Tabella degli interrupt …

Interrupt Line	Device Class
0	Inter-processor interrupts
1	Processor Local Timer
2	Bus (Interval Timer)
3	Disk Devices
4	Tape Devices
5	Network (Ethernet) Devices
6	Printer Devices
7	Terminal Devices

Un solo dispositivo

Otto dispositivi per Ciascuna linea

Distinguere tra sub-device in ricezione o trasmissione

Gestione degli Interrupt

- Il nucleo deve gestire interrupts causati da dispositivi **I/O**, **Processor Local Timer**(s) ed **Interval Timer**.

- **Azioni** che il nucleo deve svolgere:
  1. **Identificare** la sorgente dell'interrupt
     - **Linea**: registro Cause.IP
     - **Device** sulla linea (>3): Interrupting Device Bit Map
  2. **Acknowledgment** dell'interrupt

     - Scrivere un comando di ack (linea >3) o un nuovo comando nel registro del device.

Gestione degli Interrupt

- **Azioni** che il nucleo deve svolgere in caso di interrupts :

  3. **Effettuare** una V sul semaforo associato al device, e restuire lo stato del device al processo (in v0).

  4. Nel caso la V non sblocchi alcun processo, **salvare** lo stato del dispositivo (nel caso di interrupt anticipato, cioe' **SYS8** ritardata rispetto al completamento dell'interrupt)

Gestione degli Interrupt

- Due tipi di **Timer**:
  - **Processor Local Timer (PLT)**: timer locale ad ogni processore (uno per ogni processore, linea interrupt 1, gestito sempre dal processore di appartenenza)
  - **Interval Timer (IT)**: timer del BUS di sistema, linea interrupt 2

- Quale Timer usare …
  - TIME-SLICE *Scheduling* → Utilizzare PLT
  - Gestione dello *Pseudo-Clock* → Utilizzare IT

Gestione degli Interrupt

- Due tipi di **Timer**:
  - **Processor Local Timer(PLT)**: timer locale ad ogni processore (uno per ogni processore, linea interrupt 1, gestito sempre dal processore di appartenenza)

```
unsigned int setTIMER(unsigned it)
```

```
unsigned int getTIMER()
```

Gestione degli Interrupt

- Due tipi di **Timer**:
  - **Interval Timer (IT)**: timer del BUS di sistema, linea interrupt 2

```
unsigned int setIT(unsigned it)
```

Gestione degli Interrupt

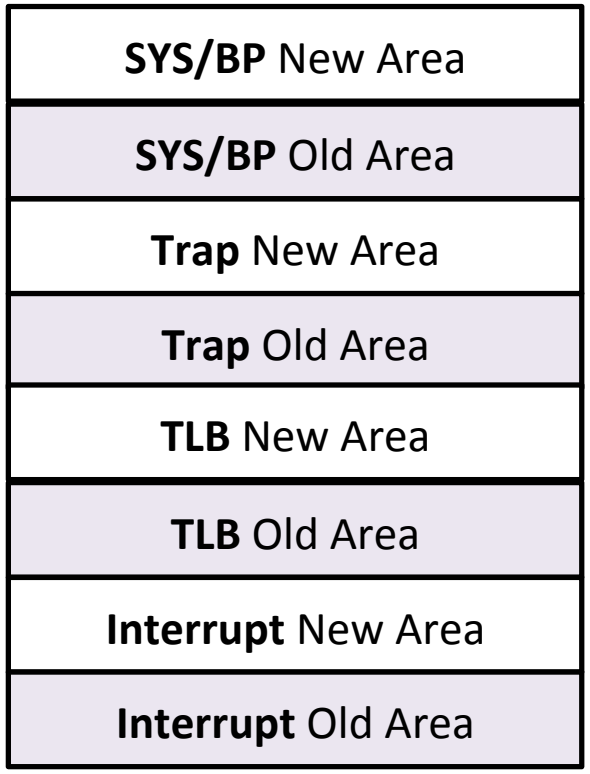
- In caso di interrupt in uMPS2, l'**Interrupt Router** determina il processore che deve gestire l'interrupt stesso.
- Linee di interrupt <2 (IPI e PLT) NON routabili!
- Comportamento di default:
  - Tutti gli interrupt sono gestiti dal processore 0
- **OPZIONALE**: Implementare politiche di scheduling degli interrupt tra processori, in modo da ottimizzare le prestazioni del sistema….

Livello 3 del S.O.

- **Funzionalita'** che il nucleo deve gestire:
  - **Inizializzazione** del sistema
  - **Scheduling** dei processi
  - Gestione delle **syscall**
  - Gestione degli **interrupt**
  - Gestione delle **eccezioni** (BreakPoints, PgmTrap, TLB Exceptions)
- **Novita'** rispetto a versioni precedenti di Kaya → Funzionalita' del nucleo da gestire in **ambiente multi-processore**!!

Gestione delle Eccezioni

- Un'eccezione di tipo **PgmTrap** viene sollevata quando un programma utente prova ad eseguire un'azione illegale o non definita.



Routine del nucleo di gestione dei PgmTrap

(l'indirizzo della NewArea deve essere settato opportunamente in fase di system setup)

Gestione delle Eccezioni

- Il nucleo gestisce le eccezioni PgmTrap effettuando un **pass-up** dell'eccezione:
  - Se il processo che ha causato l'eccezione non ha definito un gestore di PgmTrap tramite la **SYS9** → Il processo viene terminato.
  - Se il processo ha definito un gestore di Pgm Trap tramite la **SYS9** → la gestione dell'eccezione viene delegata all'handler il cui indirizzo e' stato specificato nella TLB Exc NewArea.

Gestione delle Eccezioni

- Un'**eccezione** di tipo **TLB** viene sollevata quando uMPS2 non riesce a tradurre un indirizzo virtuale in un indirizzo fisico di memoria..

Routine del nucleo di gestione delle TLB Exception

(l'indirizzo della NewArea deve essere settato opportunamente in fase di system setup)

Gestione delle Eccezioni

- Il nucleo gestisce le eccezioni di tipo TLB effettuando un **pass-up** dell'eccezione:
  - Se il processo che ha causato l'eccezione non ha definito un gestore di PgmTrap tramite la **SYS10** → Il processo viene terminato
  - Se il processo ha definito un gestore di PgmTrap tramite la **SYS10** → la gestione dell'eccezione viene delegata all'handler il cui indirizzo e' stato specificato nella PgmTrap NewArea.

Gestione delle Eccezioni

- Il nucleo gestisce direttamente le SYS1-11 eseguite in modalita' *kernel*.
- Per tutte le altre eccezioni SYS/BP, o per SYS eseguite in modalita' *utente*:
  - Se il processo che ha causato l'eccezione non ha definito un gestore di SYS/BP Exc tramite la **SYS11** → Il processo viene terminato
  - Se il processo ha definito un gestore di SYS/BP Exc tramite la **SYS11** → la gestione dell'eccezione viene delegata all'handler il cui indirizzo e' stato specificato nella SYS/BP Exc NewArea.

PKaya Operating System

Organizzazione del Progetto --
Consegna

FASE 2

Anno Accademico 2011-2012

Gestione del progetto

- Lavoro di gruppo
- Strutturazione modulare del progetto fortemente consigliata …

ESEMPIO di strutturazione:

scheduler.c

handler.c

interrupts.c

main.c

utils.c → (funzioni ausiliarie)

Gestione del progetto

- Molte scelte sono LIBERE e DELEGATE al progettista (es. come gestire il supporto per MP)
- … Non esiste un'unica implementazione corretta!

CRITERI di VALUTAZIONE:

- *Correttezza*

  (non connessa solo al superamento del test …)

- *Prestazioni*

  (analisi di scalabilita' in ambiente MP)

- *Stile e leggibilita'*

  (presenza di commenti e documentazione di supporto)

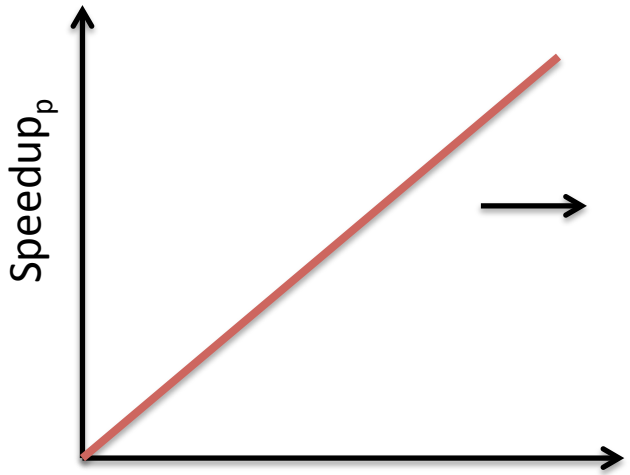
Gestione del progetto

- **Speedup** come metrica di scalabilita'

$$Speedup_p = \frac{T_{sequenziale}}{T_{parallelo}}$$

→ Tempo di esecuzione del test su architettura single-processor

→ Tempo di esecuzione del test su architettura con p-processori



→ Speedup Lineare (**caso ideale!!!!**)

→ Variare nel range [1:16]

Gestione del progetto

- Cosa consegnare:
  - Sorgenti (al completo)
  - Makefile
  - Documentazione (pdf)
  - file AUTHORS.txt, README.txt, etc

- Nella documentazione:
  - Scelte progettuali
  - Eventuali modifiche alle specifiche
  - Analisi di scalabilita'

Gestione del progetto

- **DATE** di consegna (indicative)
  - **I Consegna**: **10 Giugno 2012**
  - **II Consegna**: **8 Luglio 2012**
  - **III Consegna**: **9 Settembre 2012**

- La consegna deve essere effettuata come per Fase1 spostando l'archivio contenente il progetto nella directory di consegna di Fase2 (submit_phase2) associata al gruppo …