

PKaya Operating System

Specifiche di Progetto

FASE 1.5

v.0.1

Anno Accademico 2018-2019
(da un documento di Marco di Felice)

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.

FASE1!

Livello 2: Gestione delle Code

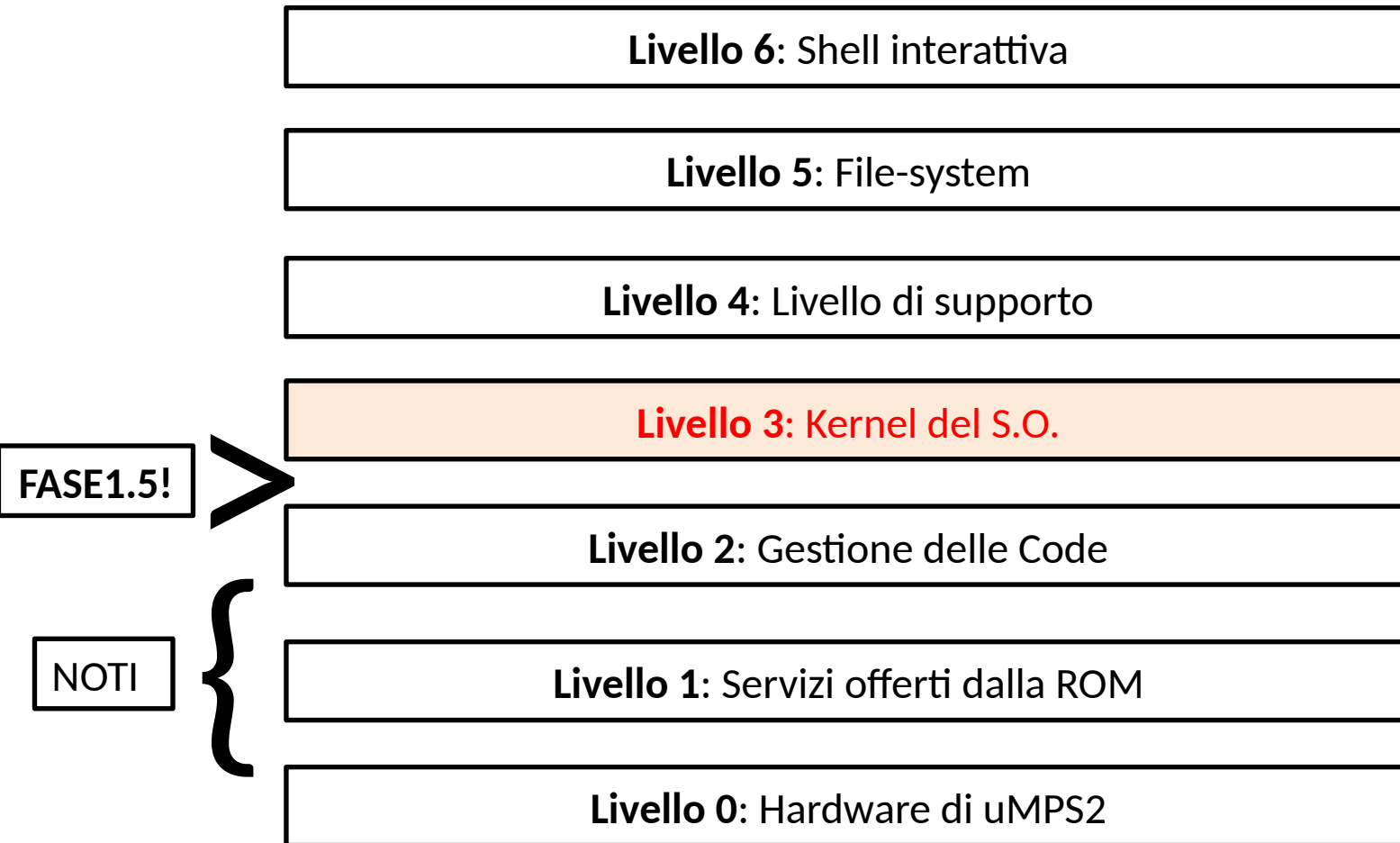
Livello 1: Servizi offerti dalla ROM

NOTI

Livello 0: Hardware di uMPS2

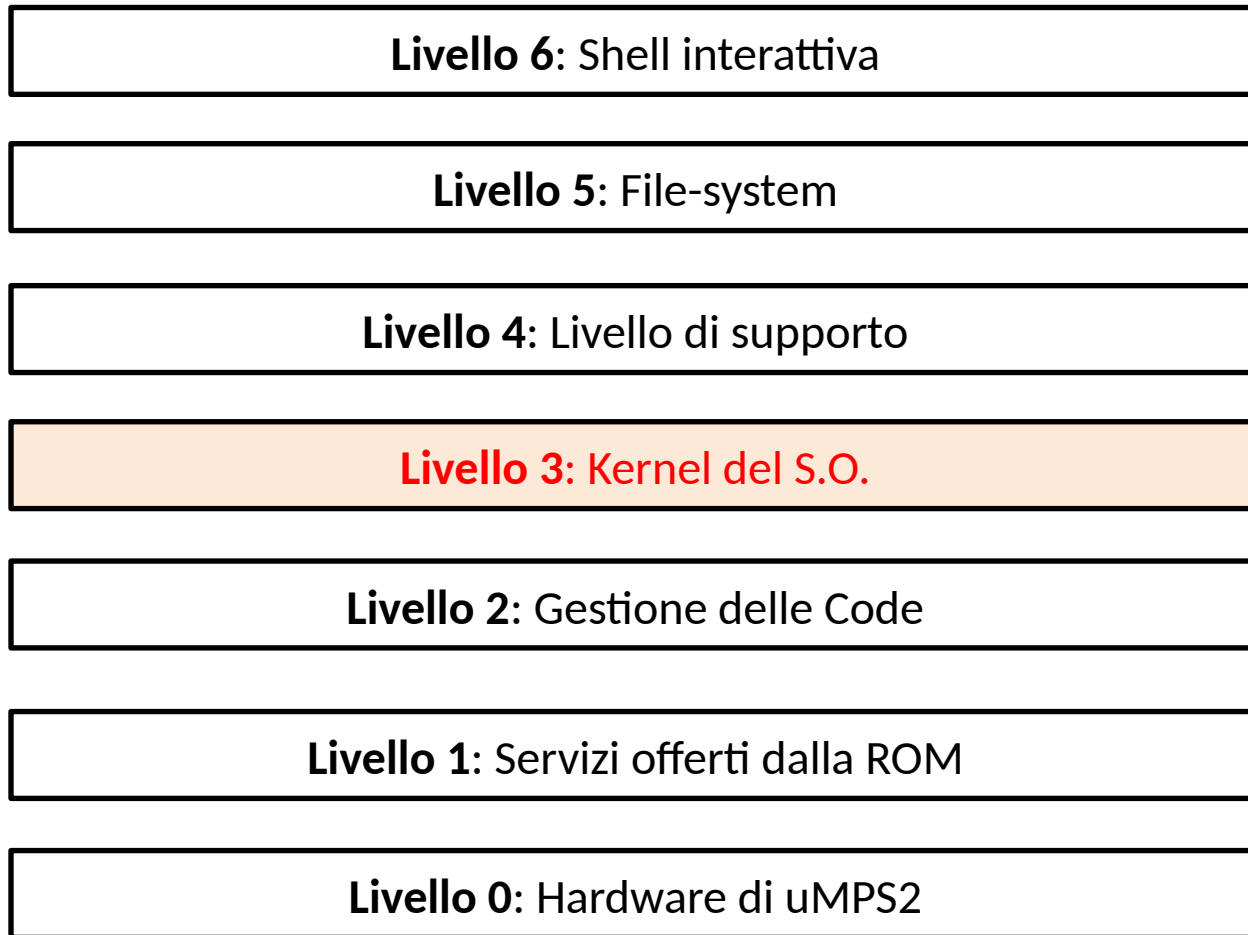
pKaya OS

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



pKaya OS

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



FASE2!

NOTI



Livello 3 del S.O.

- **Funzionalità'** che il nucleo deve gestire:
 - **Inizializzazione** del sistema
 - **Scheduling** dei processi
 - Gestione delle **syscall**
 - Gestione degli **interrupt**

Nella fase 1.5 dovrete implementarle in maniera soltanto parziale, per poi completare il tutto nella fase 2.

Livello 3 del S.O.

Delle strutture dati e funzioni sviluppate nella fase 1 dovete utilizzare quelle relative ai **pcb**.

Dovrete mantenere e gestire una (o piu') liste di processi pronti all'esecuzione.

Non e' (ancora) richiesta la gestione dei semafori (anche se avete gia' sviluppato le funzioni relative).

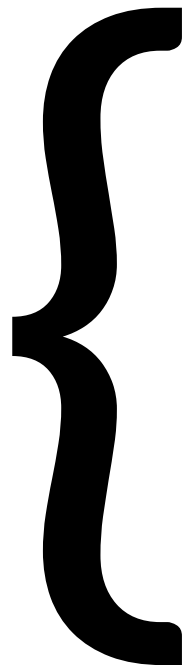
Livello 3 del S.O.

- 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)~~

Inizializzazione del sistema


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

4 Aree New/Old
presenti in
locazioni di
memoria
predefinite



SYS/BP New Area
SYS/BP Old Area
Trap New Area
Trap Old Area
TLB New Area
TLB Old Area
Interrupt New Area
Interrupt Old Area

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 un timer

Inizializzazione del sistema

- Inizializzare **strutture dati** di Phase1 (solo i pcb):
`initPcb()`

- Inizializzare **variabili** del kernel:
Per ora, solo la lista dei processi

```
LIST_HEAD(ready_queue);
```

Inizializzazione del sistema



- **Instanziare** il PCB e lo stato dei 3 processi di **test**
 - Interrupt abilitati
 - Virtual Memory OFF
 - Processor Local Timer abilitato
 - Kernel-Mode ON
 - $\$SP = \text{RAMTOP} - \text{FRAME SIZE} * n$
 - $\text{priorita}' = n$
 - Settare PC all'entry-point dei test
 $\text{pstate.pc_epc} = (\text{memaddr}) \text{ test}n$
- **Inserire** i processi nella Ready Queue

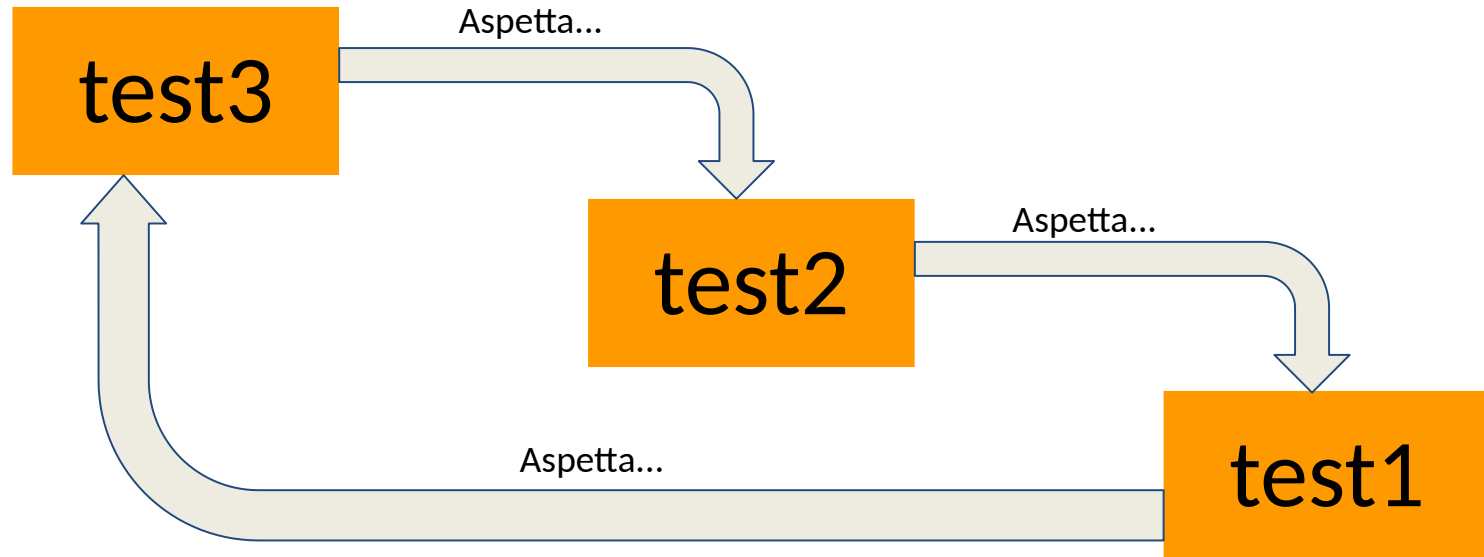
I 3 processi di test

test1, test2 e test3 sono funzioni che si alternano nello stampare un'immagine sul terminale.

Usano un semplice sistema di semafori “usa e getta” che gli permette di sincronizzarsi.

Oltre a stampare aspettano per dei tempi randomizzati, emulando un lavoro piu' complesso.

I 3 processi di test




Comincia test1 e il ciclo si ripete per 6 volte (18 stampe).

Livello 3 del S.O.

- Funzionalità che il nucleo deve gestire:
 - Inizializzazione del sistema
 - **Scheduling dei processi**
 - Gestione delle **syscall**
 - Gestione degli **interrupt**
 - ~~Gestione delle **eccezioni** (BreakPoints, PgmTrap, TLB Exceptions)~~

Scheduler di Sistema

- Funzionalità dello scheduler:
 -  **Context-switch** tra processi. Ad ogni processo deve essere assegnato un time-slice di 3 millisecondi (**TIME_SLICE**).
 - **Meccanismo di aging**: per evitare starvation delle priorità più basse queste devono essere incrementate man mano che il processo aspetta nella Ready Queue. Per farlo è necessario aggiungere un campo nella struttura `pcb_t` (**original_priority**) che salvi la priorità originale, da ripristinare quando il processo viene schedulato.

Scheduler di Sistema

- Funzionalità dello scheduler:
 - **Log dei context switch:** per verificare che i processi vengano alternati correttamente, ogni volta che si verifica un context switch dovete chiamare la funzione `log_process_order` passando come parametro la priorità originale del processo.

`log_process_order` è fornita da
`p1.5test_rikaya.c`

Livello 3 del S.O.

- **Funzionalità'** che il nucleo deve gestire:
 - **Inizializzazione** del sistema
 - **Scheduling** dei processi
 - **Gestione delle syscall**
 - Gestione degli **interrupt**
 - ~~Gestione delle eccezioni (BreakPoints, PgmTrap, TLB Exceptions)~~

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]

Per ora dovete gestire solo la numero 3.

Gestione delle SYSCALL

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

SYS/BP New Area
SYS/BP Old Area
Trap New Area
Trap Old Area
TLB New Area
TLB Old Area
Interrupt New Area
Interrupt Old Area



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 3 (**SYS3**) Terminate_Process

```
void SYSCALL(TERMINATEPROCESS, 0, 0, 0)
```

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

Livello 3 del S.O.

- **Funzionalità'** che il nucleo deve gestire:
 - **Inizializzazione** del sistema
 - **Scheduling** dei processi
 - Gestione delle **syscall**
 - **Gestione degli interrupt**
 - ~~Gestione delle **eccezioni** (BreakPoints, PgmTrap, TLB Exceptions)~~

Gestione degli interrupt

- **Interrupt**=eventi asincroni legati ad IO/Timers

SYS/BP New Area
SYS/BP Old Area
Trap New Area
Trap Old Area
TLB New Area
TLB Old Area
Interrupt New Area
Interrupt Old Area




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 per la fase 1.5 (uno dei due, a scelta).

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.

Per lo scopo della fase 1.5 non e' strettamente necessario identificare e distinguere la sorgente dell'interrupt in quanto ne serve (per ora) uno solo.

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
- Siccome non dovete gestire processori multipli, potete scegliere liberamente quale timer usare per lo scheduler.

Riassumendo

Nel file `p1.5test_rikaya.c` sono forniti:

- 3 funzioni di test (`test1`, `test2` e `test3`) da inserire in altrettanti processi
- la funzione `log_process_order`, da chiamare ogni volta che un processo viene scambiato dallo scheduler

L'esecuzione del test e' corretta se vengono stampate 18 righe in ordine crescente e il diagramma di Gantt e' sensato.

Riassumendo

Dovete implementare:

- L'inizializzazione del sistema
- Un interrupt timer con time slice di 3 ms
- Una system call (terminate process)
- Uno scheduler con priorit  ed aging che intervalli correttamente tre processi, chiamando la funzione `log_process_order` a ogni context switch

PKaya Operating System

Organizzazione del Progetto --
Consegna
FASE 1.5

Anno Accademico 2018-2019

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
- ... Non esiste un'unica implementazione corretta!

CRITERI di VALUTAZIONE:

- *Correttezza*
(non connessa solo al superamento del test ...)
- *Prestazioni*
(eventuali accortezze che migliorano il sistema)
- *Stile e leggibilita'*
(presenza di commenti e documentazione di supporto)

Gestione del progetto

- Cosa consegnare:
 - Sorgenti (al completo)
 - Makefile o build tool ananlogo
 - 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

- **DATA** di consegna

23 Aprile 2019, ore 23:59

- La consegna deve essere effettuata come per Fase1 spostando l'archivio contenente il progetto nella directory di consegna di Fase1.5 (submit_phase1.5) associata al gruppo ...