BIKAYA Operating System

Specifiche di Progetto

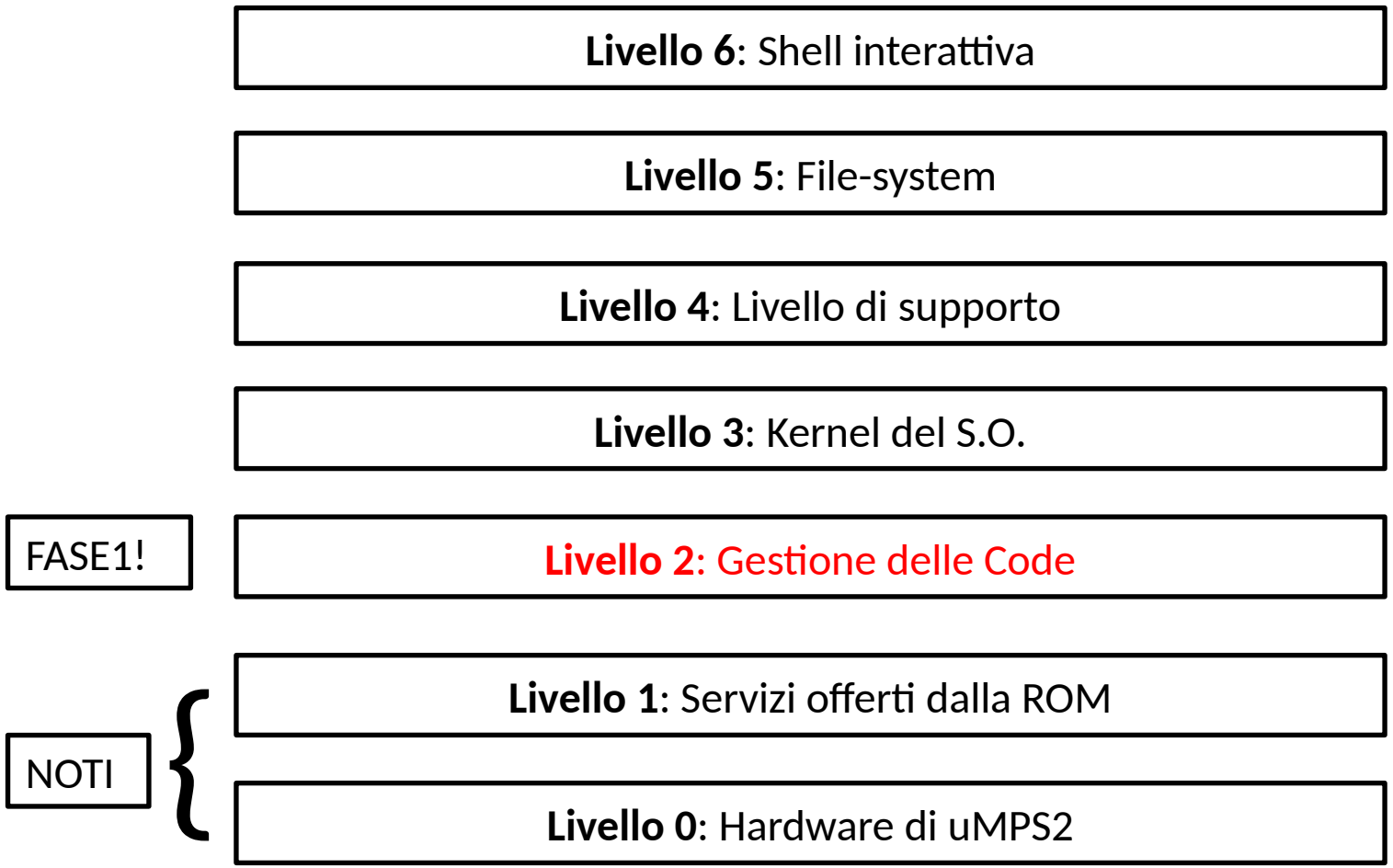
FASE 2

v.0.1

Anno Accademico 2019-2020
(da un documento di Marco di Felice)

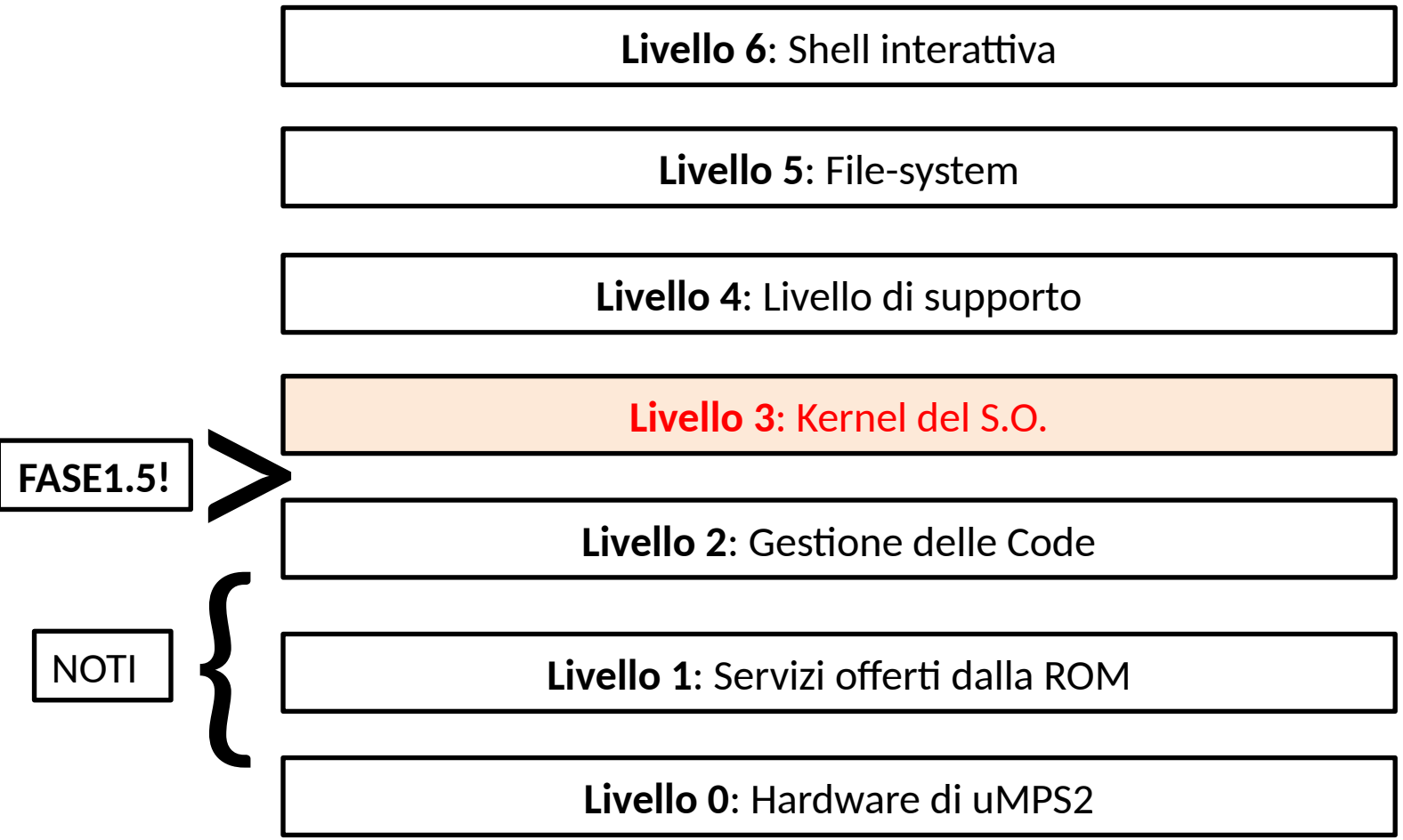
BIKAYA OS

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



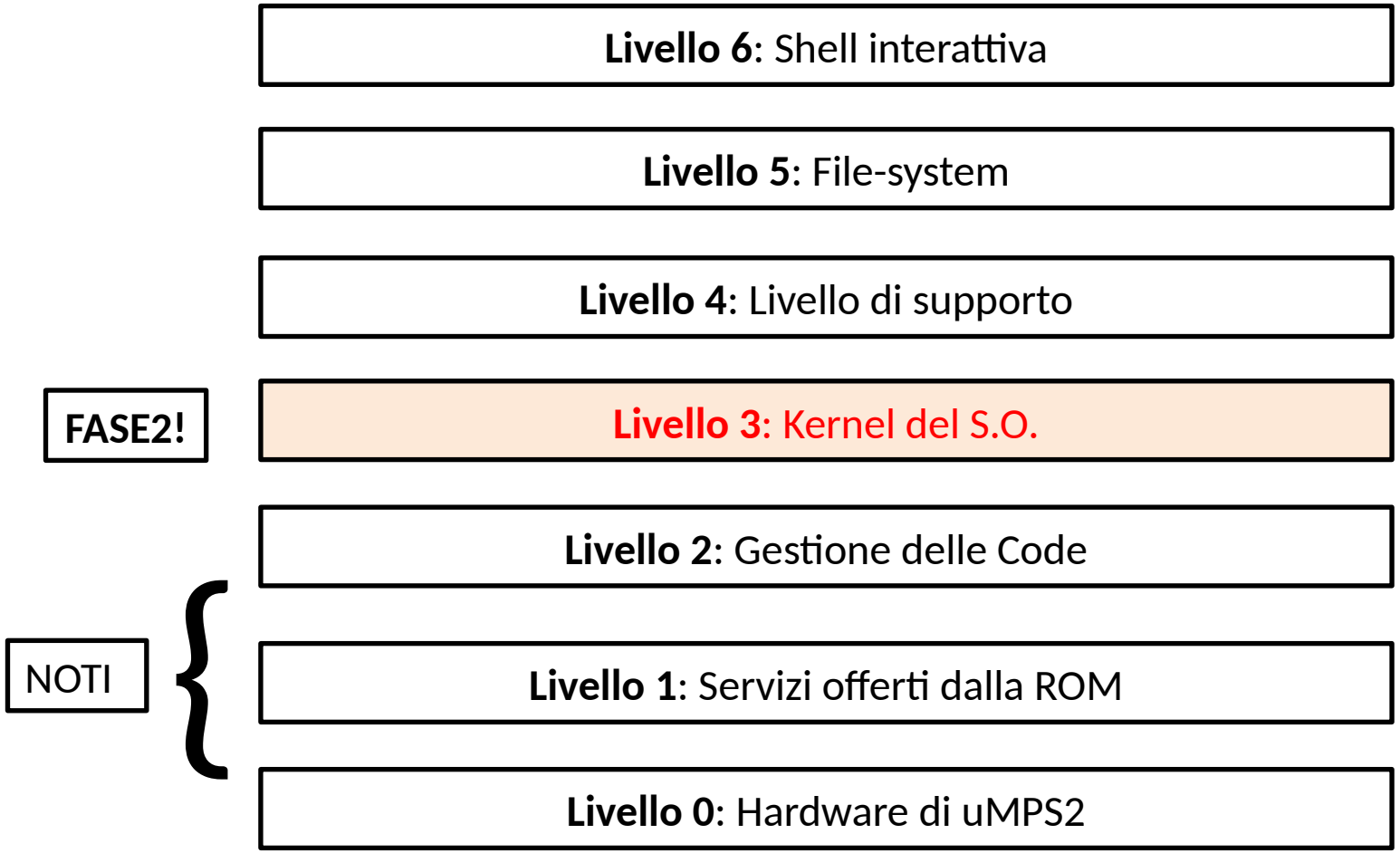
BIKAYA OS

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



BIKAYA 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**

Nella fase 2 dovrete completare le funzionalità già abbozzate durante la fase 1.5: gestione di interrupt e system call.

Livello 3 del S.O.

Vengono utilizzate le strutture dati relative ai **pcb** per gestire le code dei processi.

I processi sono organizzati in alberi per tenere traccia della loro genealogia e nelle liste dei semafori (**asl**) sui quali si bloccano per l'accesso alle risorse del sistema.

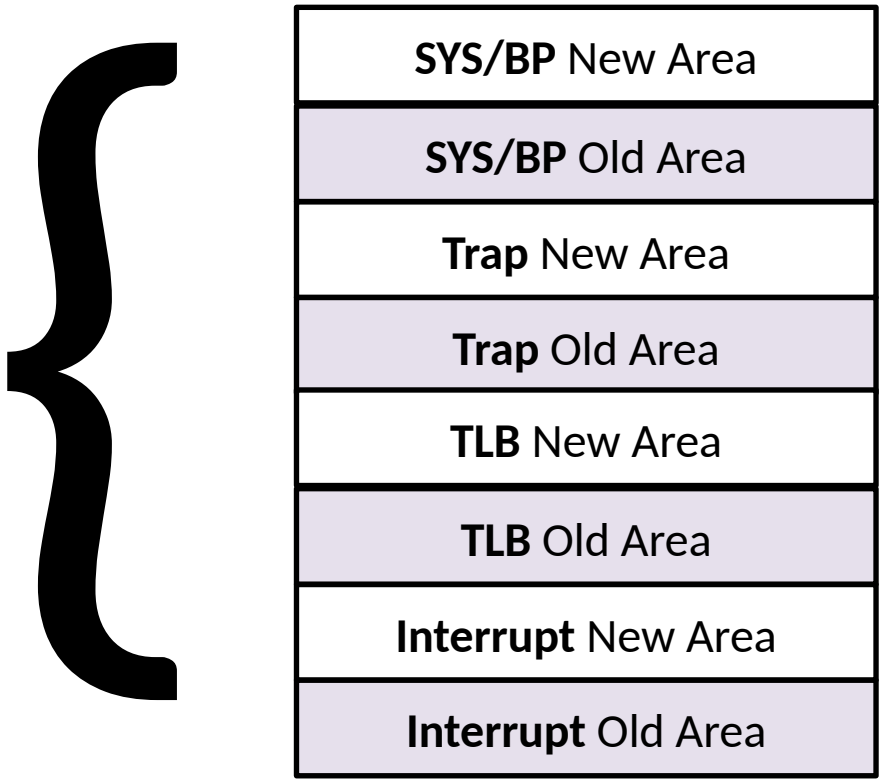
Livello 3 del S.O.

- **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)

Inizializzazione del sistema

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

4 Aree New/Old presenti in locazioni di memoria predefinite



Inizializzazione del sistema

Non cambia molto rispetto alla fase 1.5.

- Inizializzare **strutture dati** di Phase1, questa volta anche con i semafori.

```
initPcbs();
initAsl();
```

Inizializzazione del sistema

- **Instanziare** il PCB e lo stato del singolo processo di **test**
  - Interrupt abilitati
  - Virtual Memory OFF
  - $SP=RAMTOP-FRAMESIZE
  - priorita’ = *1*
  - Settare PC all’entry-point dei test

- **Inserire** il processo nella Ready Queue

Il processo di test

Il processo che si occupa di verificare le funzionalità di test va lanciato alla fine dell'inizializzazione e lasciato operare senza interferenze fino alla fine.

Sarà sua responsabilità creare nuovi processi usando la system call preposta.

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)

Scheduler di Sistema

- Context switch e scheduling rimangono gli stessi di fase 1.5; idealmente i meccanismi implementati dovrebbero scalare senza problemi nella fase successiva (salvo errori)

- Si aggiunge un tracciamento del tempo di esecuzione di ogni processo, che lo scheduler deve accumulare in nuovi campo della struttura pcb_t

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)

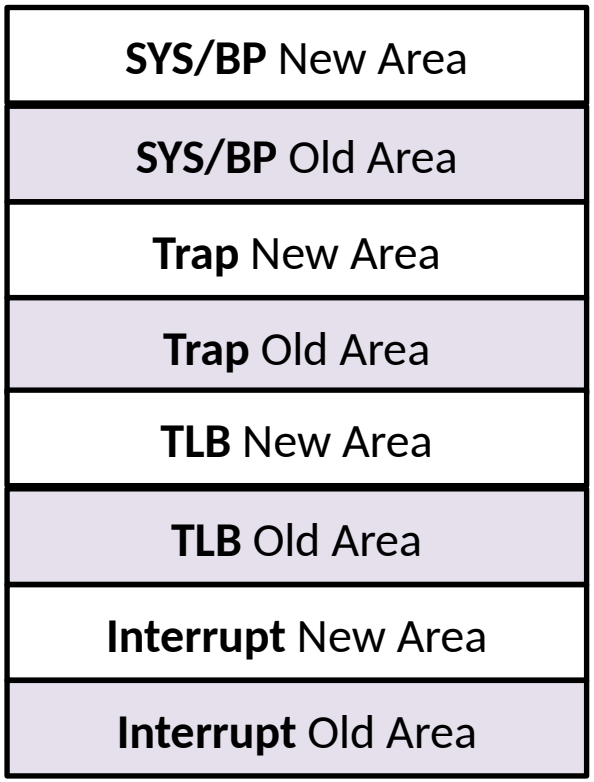
Gestione delle SYSCALL

- Gestione delle SYSCALL e BREAKpoint
  - Una SYSCALL si distingue da un BREAKpoint attraverso il contenuto del registro Cause. I parametri della SYSCALL/BP si trovano nei registri a0-a3 o a1-a4 (su uMPS2 o uARM, rispettivamente)
  - Nel caso delle SYSCALL, il registro a0/a1 identifica la SYSCALL specifica richiesta
  - Al termine della SYSCALL, il valore di ritorno puo' essere passato al processo tramite il registro v0 o a1 (su uMPS2 o uARM, rispettivamente)
  - 8 possibili SYSCALL, con codici [1…8]

Gestione delle SYSCALL

- Numero della SYS specificata nel registro **a0/a1**

…



Routine del nucleo di gestione delle SYS/BP

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

Gestione delle SYSCALL

Nota: su uMPS2 perche' un processo continui l'esecuzione dopo aver invocato una system call e' necessario **incrementare** il suo program counter di una word (i.e. 4 byte).

Su uARM non sono necessarie operazioni aggiuntive.

Gestione delle SYSCALL

- SYSCALL 1 (**SYS1**) **Get_CPU_Time**

```
void SYSCALL(GETCPUTIME, unsigned int *user, unsigned int *kernel, unsigned int *wallclock)
```

  - Quando invocata, la **SYS1** restituisce il tempo di esecuzione del processo che l'ha chiamata fino a quel momento, separato in tre variabili:
  - Il tempo usato dal processo come utente (user)
  - Il tempo usato dal processo come kernel (tempi di system call e interrupt relativi al processo)
  - Tempo totale trascorso dalla prima attivazione del processo.

Gestione delle SYSCALL

- SYSCALL 2 (**SYS2**) **Create_Process**

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

  - Questa system call crea un nuovo processo come figlio del chiamante. Il program counter, lo stack pointer, e lo stato sono indicati nello stato iniziale. Se la system call ha successo il valore di ritorno è zero altrimenti è -1. Se cpid != NULL e la chiamata ha successo *cpid contiene l'identificatore del processo figlio, rappresentato dall'indirizzo del suo pcb_t.

Gestione delle SYSCALL

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

```
Int SYSCALL(TERMINATEPROCESS, void * pid, 0, 0)
```

  - Quando invocata, la **SYS3** termina il processo identificato da pid (il proc. corrente se pid == NULL) insieme alla sua progenie. pid e' un puntatore a pcb_t cosi' come viene inizializzato dalla system call Create_Process
  - Restituisce 0 se ha successo, -1 per errore (e.g. il pid non corrisponde a un processo esistente).

Gestione delle SYSCALL

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

```
void SYSCALL(VERHOGEN, int *semaddr, 0, 0)
```

  - Operazione di rilascio su un semaforo. Il valore del semaforo è memorizzato nella variabile di tipo intero passata per indirizzo. L'indirizzo della variabile agisce da identificatore per il semaforo.

Gestione delle SYSCALL

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

```
void SYSCALL(PASSEREN, int *semaddr, 0, 0)
```

  - Operazione di richiesta di un semaforo. Il valore del semaforo è memorizzato nella variabile di tipo intero passata per indirizzo. L'indirizzo della variabile agisce da identificatore per il semaforo.

Gestione delle SYSCALL

- SYSCALL 6 (**SYS6**) **Do_IO**

int SYSCALL(IOCOMMAND, unsigned int command, unsigned int *register, int subdevice)

  - Questa system call attiva una operazione di I/O copiando parametro command nel campo comando del registro del dispositivo indicato come puntatore nel secondo argomento.
  - L'operazione è bloccante, quindi il chiamante viene sospeso sino alla conclusione del comando. Il valore ritornato è il contenuto del registro di status del dispositivo.

Gestione delle SYSCALL

- SYSCALL 6 (**SYS6**) **Do_IO**

  - Il quarto parametro indica a quale sottodevice si sta facendo riferimento nel caso in cui si voglia portare avanti un'operazione su un terminale. 0 corrisponde alla trasmissione, 1 alla ricezione.

Gestione delle SYSCALL

- SYSCALL 7 (**SYS7**) **Spec_Passup**

```
int SYSCALL(SPECPASSUP, int type, state_t *old, state_t *new)
```

  - Questa chiamata registra quale handler di livello superiore debba essere attivato in caso di trap di Syscall/breakpoint (type=0), TLB (type=1) o Program trap (type=2). Il significato dei parametri old e new è lo stesso delle aree old e new gestite dal codice della ROM: quando avviene una trap da passare al gestore lo stato del processo che ha causato la trap viene posto nell'area old e viene caricato o stato presente nell'area new. La system call deve essere richiamata una sola volta per tipo (pena la terminazione). Se la system call ha successo restituisce 0, altrimenti -1.

Gestione delle SYSCALL

- SYSCALL 8 (**SYS8**) **Get_pid_ppid**

```
Void SYSCALL(GETPID, void ** pid, void ** ppid, 0)
```

  - Questa system call assegna il l'identificativo del processo corrente a *pid (se pid != NULL) e l'identificativo del processo genitore a *ppid (se ppid != NULL)

Gestione delle SYSCALL

- SYSCALL > 8

Devono essere inoltrati al gestore di livello superiore se presente (i.e. se è stato specificato da una Spec_Passup), altrimenti causano la terminazione del processo.

Stesso dicasi per le eccezioni di tipo TLB e TRAP.

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)

-

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

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

Nota: su uARM al momento di un interrupt e' necessario **decrementare** di una word (i.e. 4 byte) il program counter del processo che e' stato interrotto perche' questo continui correttamente l'esecuzione (ripetendo l'istruzione che non e' stata portata a termine a causa dell'eccezione).

Su uMPS2 non e' necessaria alcuna operazione aggiuntiva.

Gestione degli Interrupt

- Il nucleo deve gestire le linee di interrupt da 1 a 7.

- **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.

- Interrupt con numero di linea più bassa hanno priorità più alta, e dovrebbero essere gestiti per primi.

Gestione degli Interrupt

Utilizzate un semaforo per ogni device per “risvegliare” il processo che ha richiesto l’operazione di I/O con la SYS6 (due semafori per i terminali che sono device “doppi”).

Notate che le linee di interrupt per i dispositivi di I/O (dalla linea 3 in poi) possono essere relative a istanze multiple, per cui bisogna distinguere quale di esse abbia effettivamente lanciato l’eccezione.

Gestione di BP/TLB/Trap

Le rimanenti eccezioni vengono gestite da un processo soltanto tramite la syscall Spec_Passup. Se ne viene sollevata una e il processo corrente non risulta aver registrato degli handler alternativi deve essere terminato.

A quel punto il controllo passa a un altro processo.

Riassumendo

Nel file p2test_bikaya.c viene fornita la funzione di test, che si occupa di verificare le funzionalità richieste.

L’esecuzione del test e’ corretta se questo arriva al termine senza andare in PANIC.

BIKAYA Operating System

Organizzazione del Progetto --
Consegna

FASE 2

Anno Accademico 2019-2020

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 difficolta'/errori presenti.

Gestione del progetto

- **DATE** di consegna

  24 maggio 2020 ore 23:59

  2 Luglio 2020, ore 23:59

  6 Settembre 2020, ore 23:59

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