

# Pandos+ Operating System

Specifiche di Progetto

**FASE 2**

**v.0.1**

Anno Accademico 2021-2022  
(da un documento di Marco di Felice)

# Pandost+

- 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

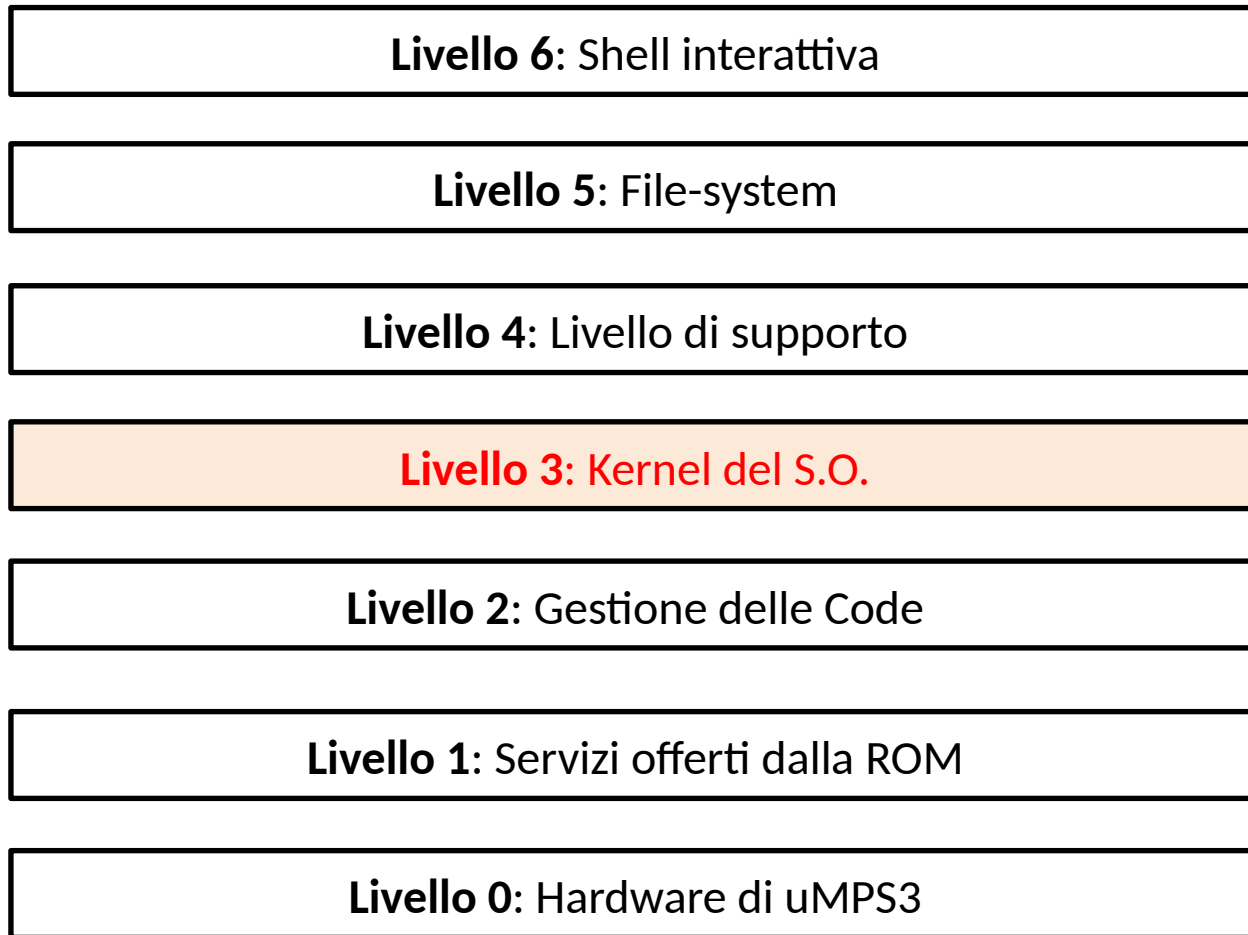
NOTI

**Livello 1:** Servizi offerti dalla ROM

**Livello 0:** Hardware di uMPS3

# Pandost+

- 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 **eccezioni**

**Quella che segue e' una visione panoramica delle specifiche; per le informazioni dettagliate e' indispensabile fare riferimento ai manuali di PandOS/PandOS+ (capitolo 3 rivisto) e uMPS3.**

# Livello 3 del S.O.

- Funzionalità che il nucleo deve gestire:
  - **Inizializzazione del sistema**
  - **Scheduling** dei processi
  - Gestione delle **eccezioni**

# Strutture dati necessarie

Variabili globali per:

- Conteggio dei processi vivi
- Conteggio dei processi bloccati
- Coda dei processi “ready”
- Puntatore al processo correntemente attivo
- Un semaforo (e.g. una variabile int) per ogni (sub) dispositivo. Non necessariamente tutti questi semafori sono sempre attivi.
- Strutture dati già gestite in fase 1

# Inizializzazione: strutture dati

- Al contrario di fase 1 il vostro codice ha controllo a partire dal `main()`
- Inizializzare i moduli di fase 1 (`initPcbs()` e `initSemd()`)
- Inizializzare le variabili elencate nella slide precedente
- Popolare il pass up vector con gestore e stack pointer per eccezioni e TLB-Refill

# Inizializzazione: dispositivi

- E' sufficiente caricare l'Interval Timer con in valore corrispondente a 100 millisecondi, scrivendolo nel registro corrispondente
- Questo valore dipende dalla frequenza di esecuzione del processore, non puo' essere una semplice costante.



# Inizializzazione: scheduler

- Allocare un processo (pcb\_t) in kernel mode, con interrupt abilitati, stack pointer a RAMTOP e program counter che punta alla funzione test() (fornita da noi).
- Inserire questo processo nella Ready Queue a bassa priorita'.
- invocare lo scheduler.

# Pass Up Vector

Nell'evento di un'eccezione uMPS3 salva lo stato del processore in una specifica locazione di memoria (0x0FFFF000) e carica PC e SP che trova nel Pass Up Vector, una struttura che dovete popolare all'indirizzo 0x0FFFF900.

Il Pass Up Vector distingue tra TLB-Refill e tutte le altre eccezioni (per distinguere ulteriormente si veda il registro Cause).

Le eccezioni non possono essere annidate.

# Livello 3 del S.O.

- Funzionalità che il nucleo deve gestire:
  - Inizializzazione del sistema
  - **Scheduling dei processi**
  - Gestione delle eccezioni

# Scheduler

Una volta che lo scheduler viene lanciato la prima volta il controllo non dovrebbe piu' tornare al main().

Da qui in avanti l'esecuzione e' in mano al processo di test; l'unico momento in cui si torna al vostro kernel e' nell'eventualita' di un'eccezione.

# Scheduler

Lo scheduler di PandOS+ ha due livelli di priorit : bassa e alta.

Questi due livelli si implementano con due code di processi gestite separatamente e con il campo `p_prio` di `pcb_t`.

Il ruolo dello scheduler  , fondamentalmente, decidere quale processo deve entrare in esecuzione.

# Scheduler: alta priorit 

Se la coda ad alta priorit  non   vuota si estrae il primo processo disponibile da li  e lo si carica in esecuzione.

I processi ad alta priorit  non sono soggetti a preemption (i.e. non vengono mai interrotti), e vengono estratti dalla coda in ordine FIFO.

# Scheduler: bassa priorit 

Se la coda dei processi ad alta priorit  e' vuota lo scheduler ne sceglie uno dalla coda dei processi a bassa priorit .

Questa e' soggetta a un algoritmo round robin preemptive con time slice da 5 millisecondi per processo. = I processi vengono estratti dalla coda dei processi ready a bassa priorit 

Quando un processo viene estratto si carica il PLT con l'opportuno time slice.

# Stato del processore

```
typedef struct state {  
    unsigned int entry_hi;  
    unsigned int cause;  
    unsigned int status;  
    unsigned int pc_epc;  
    unsigned int gpr[STATE_GPR_LEN];  
    unsigned int hi;  
    unsigned int lo;  
} state_t;
```

**Nota:** Per assegnare il registro pc e' necessario scrivere lo stesso valore anche nel registro t9

t9 viene fornito  
come costante  
simbolica di un  
registro



# Stato del processore

Caricare un processo in esecuzione significa copiare il suo stato salvato (il campo `p_s` del relativo `pcb_t`) nel processore.

A questo scopo esiste la funzione LDST. LOAD STATE

# Scheduler

Un processo puo' essere:

- ready
- running
- blocked/waiting

Il contatore di “soft blocked” indica il numero di processi bloccati su un'operazione di IO (che quindi finira') e non un semaforo.

Man mano che i processi passano da uno stato all'altro e' vostra responsabilita' aggiornare questo contatore.

# Scheduler

Se ad un certo momento entrambe le code sono vuote, lo scheduler deve verificare alcune condizioni:

- Se non ci sono piu' processi (Process Count == 0), spegnere la macchina con HALT().
- Se Process Count > 0 e Soft Blocked Count > 0 il processore deve essere messo in stato di attesa (funzione WAIT()). wait creata per ridurre il consumo di energia e di evitare di fare aspettare tempo per il prossimo
- Se Process Count > 0 e Soft Blocked Count == 0 lo scheduler e' in deadlock: invocare PANIC().

# Il processo di test

funzione Test da invocare nel  
main

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.

- Funzionalità che il nucleo deve gestire:
  - ☑ Inizializzazione del sistema
  - ☑ Scheduling dei processi
  - Gestione delle eccezioni

# TLB-Refill

Eccezione che viene sollevata quando nel TLB non viene trovata una entry valida.

Non e' da gestire in questa fase; potete popolare la relativa parte del Pass Up Vector con la funzione `uTLB_RefillHandler()` che troverete nel file `p2test.c`

# Tutte le altre eccezioni

Nel caso di un'eccezione che non sia TLB-Refill, l'altro handler del Pass Up Vector viene invocato dal BIOS.

Per distinguere effettivamente di che eccezione si tratta bisogna leggere il registro Cause.ExcCode:

- 0 = Interrupt
- 1-3 = TLB Trap
- 4-7,9-12 = Program Trap
- 8 = Syscall

# Tutte le altre eccezioni

Per quanto riguarda TLB Trap e Program Trap, tutto quello che si deve fare e' passare il controllo a un gestore indicato dal processo corrente (se presente) oppure ucciderlo.

I processi possono specificare questo gestore al momento della creazione tramite System Call; e' salvato nel campo p\_supportStruct della struttura pcb\_t.



# Gestione delle SYSCALL

Le eccezioni di classe System Call vengono sollevate con l'istruzione SYSCALL.

Per passare dei parametri all'eccezione vengono usati i registri a0, a1, a2 e a3.

In base al codice in a0 si esegue una delle seguenti operazioni, usando a1 - a3 come argomenti.

# Gestione delle SYSCALL

codici negativi: da eseguire a lvl del kernel  
codici positivi: da eseguire

- SYSCALL -1: Create\_Process

```
int SYSCALL(CREATEPROCESS, state_t *statep, int prio, support_t *supportp)
```

- Questa system call crea un nuovo processo come figlio del chiamante. Il primo parametro contiene lo stato che deve avere il processo. Se la system call ha successo il valore di ritorno è zero altrimenti è -1.  
se ha successo ritorna l'id del nuovo processo creato, altrimenti -1 se non si è riusciti a crearlo
- prio indica se si tratta di un processo ad alta priorita'
- supportp e' un puntatore alla struttura di supporto del processo  
la struttura di supporto contiene il pc delle trap e va inserito nella struttura pcbt
- restituisce il pid del processo

# Gestione delle SYSCALL

- SYSCALL -1: **Create\_Process**

Al momento della creazione di un processo e' necessario creare per questo un **id univoco** che lo identifichi.

L'id puo' essere (per esempio) un numero progressivo oppure l'indirizzo della struttura `pcb_t` corrispondente.

# Gestione delle SYSCALL

- SYSCALL -2: **Terminate\_Process**

```
void SYSCALL(TERMPROCESS, int pid, 0, 0)
```

- Quando invocata, la **SYS2** termina il processo indicato dal secondo parametro insieme a tutta la sua progenie.
- Se il secondo parametro e' 0 il bersaglio e' il processo invocante. => 0 non può essere un id di un processo. Servirà un metodo dato il pid cercare il descrittore del processo

N.B/. quando un processo viene ucciso, dobbiamo stare attenti che magari ha dei processi che un semaforo deve essere liberato se il processo non esiste più. Anche se viene ucciso il processo chiamante, ognuno può essere in queste istruzioni.

# Gestione delle SYSCALL

- SYSCALL -3: **Passeren**

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

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

Quindi abbiamo processi bloccati sia con val 0 sia con val 1

# Gestione delle SYSCALL

- SYSCALL -4: **Verhogen**

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

- Operazione di rilascio su un semaforo binario. 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

metodo per lanciare operazioni di tipo I/O

- SYSCALL -5: **DO\_IO**

```
int SYSCALL(DOIO, int *cmdAddr, int cmdValue, 0)
```

- Effettua un'operazione di I/O scrivendo il comando cmdValue nel registro cmdAddr, e mette in pausa il processo chiamante fino a quando non si è conclusa.
- L'operazione è bloccante, quindi il chiamante viene sospeso sino alla conclusione del comando. Il valore ritornato deve essere il contenuto del registro di status del dispositivo.

cmdAddr = indirizzo di dove inserire il comando

cmdValue = indica il valore del comando

Le operazioni di I/O non sono istantanee, c'è un lasso di tempo di ricezione, nel frattempo non fa nulla, vogliamo che sia sincrona, dunque il processo viene messo in sospensione in attesa della ricezione.

Ogni device ha un registro di comando che va settato per ultimo, e sarà quello che effettivamente farà partire il processo I/O: invece di farlo a mano si chiama la Syscall

# Gestione delle SYSCALL

- SYSCALL -6: **Get\_CPU\_Time**

```
int SYSCALL(GETTIME, 0, 0, 0)
```

- Quando invocata, la **NSYS6** restituisce il tempo di esecuzione (in microsecondi) del processo che l'ha chiamata fino a quel momento.
- Questa System call implica la registrazione del tempo passato durante l'esecuzione di un processo.

return del tempo di esecuzione fino a quel momento, ritorno un campo della pcb

p\_time = campo di esecuzione del processo fino a quel momento, può essere arbitrariamente precisa.



# Gestione delle SYSCALL

- SYSCALL -7: **Wait\_For\_Clock**

```
int SYSCALL(CLOCKWAIT, 0, 0, 0)
```

- Equivalente a una Passeren sul semaforo dell'Interval Timer.
- Blocca il processo invocante fino al prossimo tick del dispositivo.

i processi chiedono all'Interval Timer di sollevare i loro interrupt e liberare il processo (credo u.u)

# Gestione delle SYSCALL

- SYSCALL -8: **Get\_Support\_Data**

```
support_t* SYSCALL(GETSUPPORTPTR, 0, 0, 0)
```

- Restituisce un puntatore alla struttura di supporto del processo corrente, ovvero il campo p\_supportStruct del pcb\_t.

tenere traccia del puntatore

# Gestione delle SYSCALL

- SYSCALL -9: **Get\_Process\_Id**

```
int SYSCALL(GETPROCESSID, int parent, 0, 0)
```

- Restituisce l'identificatore del processo invocante se parent == 0, quello del genitore del processo invocante altrimenti.

# Gestione delle SYSCALL

- SYSCALL -10: Yield

```
int SYSCALL(YIELD, 0, 0, 0)
```

- Un processo che invoca questa system call viene sospeso e messo in fondo alla coda corrispondente dei processi ready.
- Il processo che si e' autosospeso, anche se rimane "ready", non puo' ripartire immediatamente

il processo decide di lasciare il controllo ad un altro processo, una volta invocato da un processo esso viene messo in fondo.

Nota che se il processo è di alta priorità ed è l'unico, mettendolo in fondo verrebbe cmq richiamato lui, ma deve essere lasciato "in pausa"

# Gestione delle SYSCALL

Le System Call elencate identificate da un codice negativo sono invocabili solo da processi in kernel mode; se si riconosce che il processo chiamante e' in user mode quest'ultimo deve essere terminato.

Il processo in kernel mode deve essere terminato anche se chiama una System Call con codice ma inesistente.

! codice negativo ma inesistente !

# Gestione delle SYSCALL

SYSCALL non e' una convenzionale chiamata di funzione. Lo stato del processo viene salvato e si passa al contesto del kernel: i parametri vengono letti nei registri a0-a3, e il valore di ritorno viene passato al processo chiamante nel registro v0.

Il processo viene interrotto nell'istruzione SYSCALL; per procedere il program counter deve essere incrementato di una word (4 byte).

# 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

interrupt del processore, noi ne abbiamo solo 1

→ Un solo dispositivo

⌈ Otto dispositivi per  
Ciascuna linea



Distinguere tra sub-device in ricezione o trasmissione



# 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 SYSCALL 5 (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.

# Syscall > 0, Trap

Se si verifica una System Call con codice > 0 o una eccezione Trap, il controllo dovrebbe passare per la struttura di supporto indicata al momento della creazione del processo. Se non e' stata indicata (NULL), il processo deve terminare.

# Struttura di supporto

```
typedef struct context t {  
    unsigned int stackPtr, status, pc;  
} context t;
```

```
typedef struct support t {  
    int sup_asid;  
    state_t sup_exceptState[2];  
    context_t sup_exceptContext[2];  
} support t;
```

# Riassumendo

Nel file `p2test.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.

**NB alcuni errori non interrompono il test, dobbiamo controllare bene!!**

# Riassumendo

Tutti i dettagli sono spiegati in profondita' nel capitolo 3 del libro (rivisto dal professore per PandOS+):

[https://www.cs.unibo.it/~renzo/so/pandosplus/docs/pandosplus\\_phase2.pdf](https://www.cs.unibo.it/~renzo/so/pandosplus/docs/pandosplus_phase2.pdf)

E nel manuale di uMPS3:

<https://www.cs.unibo.it/~renzo/doc/umps3/uMPS3principOfOperations.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.

inserire qualche commento

# Gestione del progetto

- **DATA** di consegna

**19 Aprile 2022, ore 23:59**

- 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.

Buone Regole da seguire:

1. evitare di mandare screen del codice, mandarlo in forma testuale.
2. collaborazione di metodi risolutivi è ben accetta, ma ognuno deve portare il proprio codice alla valutazione. Va bene tutto ma deve essere nostro. Gli aiuti da parte loro (prof e tutor) ci saranno sempre se siamo onesti con la provenienza del codice.

**su [wiki.virtualsquare.org](https://wiki.virtualsquare.org) ci sono delle indicazioni per il debugging di uMPS3**