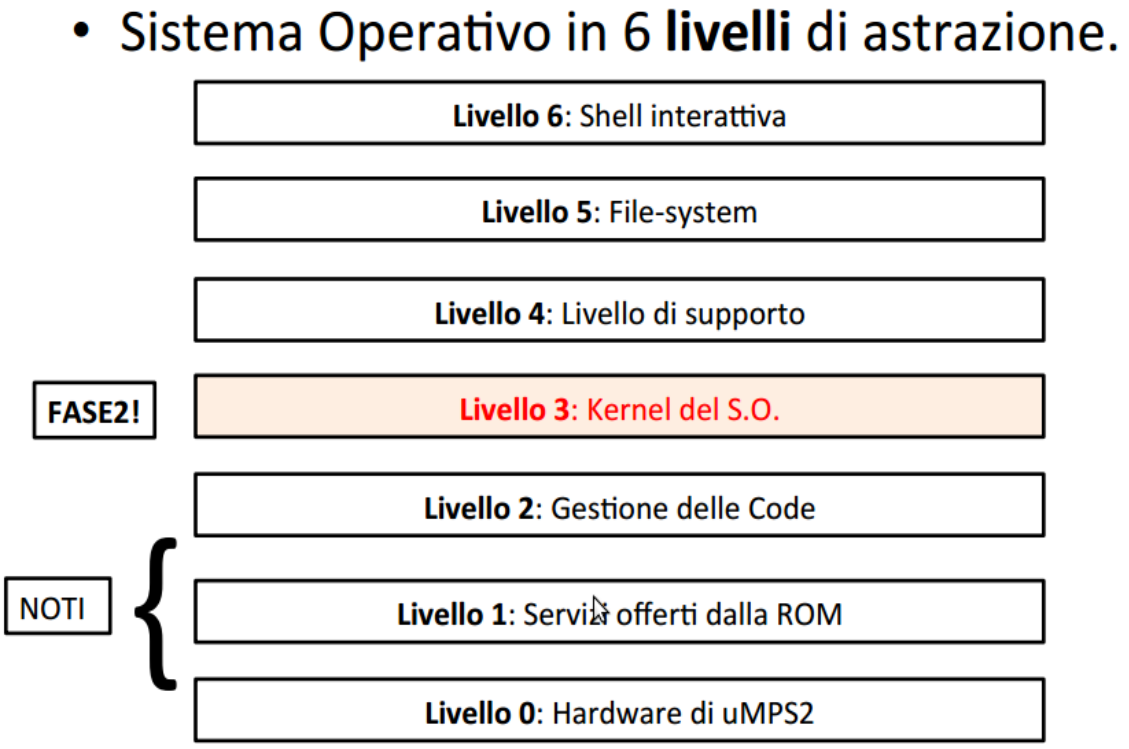
Pkaya Operating System

2011-2012

INDICE

Introduzione

Lo scopo di questa seconda fase consiste nello sviluppo del livello 3 di Amikaya, sfruttando le strutture dati create nel livello 2.
All'interno di questo livello si trova il nucleo del sistema operativo stesso; il suo obiettivo è quello di fornire un ambiente in cui è possibile avere la presenza di diversi processi, asincroni e sequenziali, che vengono eseguiti condividendo tra loro l'utilizzo della CPU.
Il kernel deve occuparsi della gestione di eccezioni, systemcall e interrupt, nonché dello scheduling dei processi per l'alternanza dell'uso della CPU. Tra le novità di quest'anno troviamo il supporto di uMPS2 alla programmazione concorrente, ovvero simulando la presenza di più processori. Per la gestione delle sezioni critiche del kernel è disponibile una funzione hardware di test&set CAS().



INIZIALIZZAZIONE DEL SISTEMA:

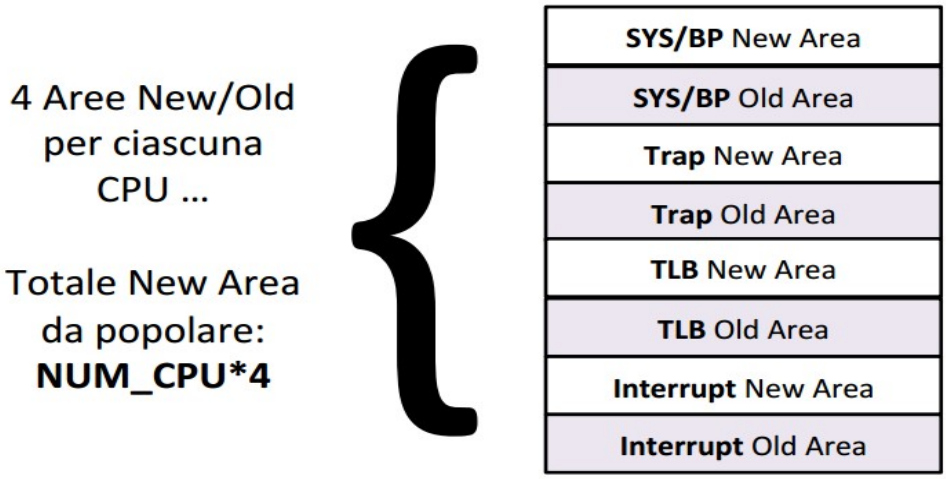
In questa fase il kernel si incarica di inizializzare tutte le strutture dati:

- **PCB: le strutture dati associate ai processi.**
- **ASL: le strutture dati associate ai semafori.**
- **Mutex: le strutture dati per la test&set.**
- **Pids' array: il vettore per la gestione dei pid.**
- **Queues: le code per la gestione dello scheduling.**
- **CurrentProcs: il vettore per tenere traccia dei processi correnti.**
- **Devstatus: la matrice per la gestione degli interrupts.**

Le new/old area:

- **INTNEW: la new area che verrà caricata al verificarsi di un interrupt.**
- **INTOLD: l´area dove verrà salvato lo stato precedente del processore.**
- **SYSNEW: la new area che verrà caricata al verificarsi di un una syscall.**
- **SYSOLD: l´area dove verrà salvato lo stato precedente del processore.**
- **TLBNEW: la new area che verrà caricata al verificarsi di una TLB.**
- **TLBOLD: l´area dove verrà salvato lo stato precedente del processore.**
- **PGMNEW: la new area che verrà caricata al verificarsi di un PGM TRAP.**
- **PGMOLD: l´area dove verrà salvato lo stato precedente del processore.**

Quando le strutture dati sono pronte, il kernel istanzia il processo test e lo inserisce nella ready queue.



SCHEDULER.C:

La politica di scheduling implementata è di tipo TIME SLICE, con un timer di 5ms senza preemption.
Quando lo scheduler viene invocato, si controlla che la ready queue non sia vuota, in questo caso si controlla l'expired queue e se sono presenti dei processi che avevano scaduto il time slice, si cambiano le due liste per evitare starvation. Se entrambe le code dovessero essere vuote, si possono verificare 3 differenti situazioni:

• HALT(): non ci sono più processi da eseguire.
• PANIC(): il sistema è in una situazione di deadlock
• WAIT(): se non esistono processi pronti nella readyQueue si abilitano gli Interrupt affinché un processo precedentemente in attesa di un IO venga sbloccato e spostato nella readyQueue.

Exception.c

Int_handler:

Quando si verifica un interrupt di tipo PLT, lo stato attuale del processo viene salvato in modo che la prossima esecuzione sia sincronizzata.
Quando si verifica un interrupt di tipo IT, si eseguono una verhogen sul semaforo dello pseudo-clock per ogni processo bloccato e si reimposta il timer.
Quando si verifica un altro tipo di interrupt, lo stato del device che ha causato l'interrupt viene salvato in una apposita tabella e il controllo viene ceduto alla funzione device_handler() in interrupt.c.

Tlb_handler:

Un’eccezione di tipo TLB viene sollevata quando uMPS2 non riesce a tradurre un indirizzo virtuale in un indirizzo fisico di memoria. Il processo può definire dei propri gestori per le eccezioni TLB.

Pgmtrap_handler:

Un’eccezione di tipo pgmtrap viene sollevata quando un programma utente prova ad eseguire un’azione illegale o non definita.
Se il processo ha definito un suo handler allora si copia la old area nella zona definita dal processo e si delega la gestione dell'eccezione, altrimenti il processo viene killato. Il processo può definire un proprio gestore per le eccezioni di tipo Program Trap.

Sysbk_handler:

Quando viene invocata una syscall, l'handler si incarica di salvare lo stato del processo in caso di syscall bloccante e incrementa il program counter del processo per evitare un loop sulla chiamata quando riprenderà l'esecuzione. È possibile, da parte del processo, specificare un proprio gestore delle syscall.

Interrupt.c

Quando un device causa un interrupt, lo stato del device che ha causato l'interrupt viene salvato in una apposita tabella e si controlla se qualche processo era in attesa di questo interrupt oppure l'interrupt è precedente alla WAIT_IO(). Di conseguenza il comportamento di device_handler() e WAIT_IO() sono strettamente collegati.
Dopo aver gestito correttamente l'interrupt, si ricarica il processo che stava eseguendo sulla CPU0 se presente oppure viene richiamato lo scheduler.

- 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

Syscall.c

Questo modulo viene invocato nel momento in cui viene eseguita un'istruzione SYSCALL.
Nel caso di una istruzione SYS eseguita in kernel-mode, si va ad eseguire la syscall, nel caso sia stata invocata in user mode si controlla che il processo abbia definito un suo handler, altrimenti viene killato.
Sono state implementate le seguenti syscall:

create_process:
Quando viene invocata la sys1 determina la creazione di un processo figlio del processo chiamante, ha come valori di ritorno 0 nel caso di creazione corretta oppure 1 nel caso contrario.

create_brother:
Quando viene invocata la sys2 determina la creazione di un processo fratello del processo chiamante,ha come valori di ritorno 0 nel caso di corretta oppure 1 nel caso contrario.

terminate_process:
Quando viene invocata la sys3 termina il processo corrente e tutta la sua progenie.

verhogen:
Quando viene invocata la sys4 viene eseguita una V sul semaforo con chiave semKey.

passeren:
Quando viene invocata la sys5 esegue una P sul semaforo con chiave semKey.

get_cpu_time:
Quando invocata, la sys6 restituisce il tempo di CPU (in microsecondi) usato dal processo corrente.

wait_for_clock:
Quando viene 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).

wait_for_io_Device:
Quando viene invocata la sys8 esegue una P sul semaforo associato al device identificato da intNo, dnume e waitForTermRead

specify_prg_state_vector:
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 processo (da utilizzare nel caso si verifichi un PgmTrap)

specifiy_tlb_state_vector:
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)

specify_sys_state_vector:
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).

GESTIONE DEI PROCESSORI

Il nostro kernel supporta fino a 16 CPU concorrenti: perché ciò sia possibile bisogna gestire la mutua esclusione sugli accessi a variabili globali tramite un meccanismo hardware di Test&Set, fornito nativamente da umps2. Sono presenti:

-una mutex per accedere al process counter, alla ready queue ed expired queue.
-una mutex per accedere ai semafori e alle liste dei semafori.
-una mutex per accedere alla variabile softBlockCounter.

ISTRUZIONI PER LA COMPILAZIONE:

per compilare il sorgente, spostarsi in ./src ed eseguire:

-make clean && make kernelphase2lab per compilare il test in laboratorio
-make clean && make per compilare il test a casa
-make clean && make scalabilitylab per compilare il test di scalabilità in laboratorio
-make clean && make scalability per compilare il test di scalabilità a casa

per visualizzare la documentazione generata da doxygen eseguire:
-make docs

CONTATTI:
hassna.elfilali@studio.unibo.it
francesco.soncina@studio.unibo.it
katia.fraulini@studio.unibo.it