

Pandosplus Fase 2

Stefano Staffolani, Filippo Speziali, Gabriele Buttinoni, Denis Gjonaj

19 aprile 2022

1 Introduzione

Questa è la documentazione riguardante la fase 2 del progetto PandOsPlus. Per la documentazione completa e ufficiale: https://www.cs.unibo.it/~renzo/so/pandosplus/docs/pandosplus_phase2.v1.pdf In questa fase viene implementato il kernel, in particolare la gestione e sincronizzazione dei processi in modalità kernel, uno scheduler dei processi, i device interrupt handlers, la rilevazione di deadlock e multiprogrammazione. L'obiettivo di questa fase è quindi quello di arrivare a comprendere la logica interna del nucleo e la creazione dello stesso in ogni sua parte.

2 Implementazione

2.1 Inizializzazione

All'interno del file di inizializzazione è contenuta la funzione main, ovvero la prima funzione a venire chiamata e serve a inizializzare il BIOS. Viene inizializzato il Pass-up Vector con gli indirizzi dell'handler per gli eventi di tipo TLB-refill, il valore dello stack pointer e l'exception handler; viene poi creato e impostato a dovere il processo init e la lista di dSemaphores, viene infine chiamato lo scheduler.

2.2 Scheduler

Lo scheduler si occupa di decidere quale processo entra in esecuzione ed è costituito da due livelli di priorità uno alto e uno basso. Una volta lanciato lo scheduler vengono eseguiti i processi presenti nelle code ad alta e bassa priorità fino a che entrambe non sono vuote. Durante l'esecuzione abbiamo tre possibili scenari: il primo avviene quando non ci sono più programmi da eseguire e quindi viene invocata la funzione HALT(); il secondo avviene quando non vengono eseguite nessuna istruzioni, cioè quando prCount e sbCount sono entrambi maggiori di 0. Viene quindi chiamata la funzione WAIT(); il terzo avviene quando si entra in deadlock, cioè quando il prCount è uguale a zero mentre il sbCount è maggiore di zero. In questo caso viene chiamata la funzione PANIC() indicando il fatto che qualcosa è andato storto durante l'esecuzione.

2.3 ExceptionHandler

L'exception handler prende l'exception state situato all'inizio della BIOS Data Page così da estrarre l'exception code ed eseguire l'azione relativa al suo valore. Per il valore 0 (interrupt) bisogna passare il controllo al device interrupt handler; per i valori da 1 a 3 (TLB exceptions) bisogna passare il controllo al TLB exception handler, per i valori da 4 a 7 e da 9 a 12 (program traps) bisogna passare il controllo al Program Trap exception handler, per il valore 8 (SYSCALL) bisogna passare il controllo al SYSCALL exception handler. Spetta all'Exception handler il compito di determinare se il processo corrente era in esecuzione in kernel-mode o in user-mode.

2.4 SYSCALLExceptionHandler

Quando viene eseguita un'istruzione assembly di tipo SYSCALL viene sollevata una System Call exception. Il processo in esecuzione passa nei registri da a0 a a3 valori specifici che poi verranno utilizzati in seguito all'istruzione SYSCALL. Il kernel eseguirà uno dei servizi per il processo in base

al valore inserito nel registro a0 e il processo passerà eventuali parametri usando i registri da a1 a a3. Se il processo era in kernel-mode e a0 conteneva un valore negativo allora il kernel dovrà eseguire uno dei servizi descritti (NSYS)

2.5 SYSCALL

- NSYS1 Create_Process quando chiamata questa System Call crea un nuovo processo che diventa un “progeny” del processo chiamante assegnandoli un pid, uno stato iniziale e una priorità oltre ad una struttura di supporto.
- NSYS2 Terminate_Process: quando chiamata questa System Call termina il processo chiamante se il parametro passato è 0, altrimenti termina il processo il cui pid corrisponde all’argomento passato, in ogni caso termina anche tutta la progenie del processo da terminare.
- NSYS3 Passeren: quando chiamata questa System Call esegue una P operation sul semaforo binario il cui indirizzo viene passato come parametro tramite il registro a1, in base al valore del semaforo il controllo o viene passato al processo corrente o questo processo passa da “in corso” a “bloccato” per poi chiamare lo scheduler.
- NSYS4 Verhogen: quando chiamata questa System Call esegue una V operation sul semaforo binario il cui indirizzo viene passato come parametro tramite il registro a1.
- NSYS5 Do_IO_Device: quando chiamata questa System Call utilizza i valori passati tramite i registri a1 e a2 (cmdAddr e cmdValue) per eseguire una P operation sul semaforo che il nucleo mantiene per i dispositivi di input/output, dato che l’operazione di P verrà eseguita su un synchronization semaphore questa chiamata bloccherà sempre il processo corrente per poi chiamare lo scheduler.
- NSYS6 Get_CPU_Time: quando chiamata questa System Call inserisce nel registro v0 il tempo impiegato dal processo richiedente. Il valore viene aggiornato nel campo p_time del processo corrente.
- NSYS7 Wait_For_Clock: quando chiamata questa System Call esegue una P operation sullo Pseudo-clock semaphore. Essendo quest’ultimo un synchronization semaphore il processo attualmente in esecuzione viene bloccato chiamando la system call 3. Viene quindi spostato lo stato del processo da “in corso” a “bloccato”.
- NSYS8 Get_SUPPORT_Data: quando chiamata questa System Call salva nel registro v0 il valore contenuto nel campo p_supportStruct del processo corrente. Se non è presente nessun valore viene restituito NULL
- NSYS9 Get_Process_ID: quando chiamata questa System Call controlla se il campo p_parent è uguale a 0, se ciò avviene viene salvato nel registro v0 il valore presente nel campo p_pid del processo corrente, altrimenti viene salvato il valore p_pid del padre.
- NSYS10 Yield: quando chiamata questa System Call il processo corrente viene sospeso e spostato in fondo alla coda della sua priorità. Dopo essersi sospeso il processo corrente deve attendere prima di poter ripartire.

2.6 InterruptExceptionHandler

...

3 Difficoltà e scelte implementative

- Per quanto riguarda lo scheduler le difficoltà maggiori sono state trovate nell’impostare STATUS in modo da abilitare gli interrupt e disabilitare invece il PLT utilizzando apposite funzioni.
- Nelle System Call le difficoltà maggiori sono state riscontrate nelle NSYS2, NSYSs ...

- Nella NSYS2 (Terminate_Process) non ci era subito chiaro come capire se il processo che si stava cercando di terminare fosse bloccato su un device semaphore o su un non-device semaphore
- Nella SYS5 (Do_IO_Device) la difficoltà principale è stata quella di riuscire a utilizzare cmdAddr e cmdValue in modo da poter ottenere l'indirizzo del semaphore che il nucleo mantiene per il dispositivo di input/output