Documentazione

Nikolas Acquaviva,
Stefano Carminati,
Matteo Manuelli,
Michele Monteferrante

Aprile 2022

1 Introduzione

Questa documentazione serve a spiegare come abbiamo deciso di affrontare la seconda fase del progetto di Sistemi Operativi.

In questa fase abbiamo dovuto costurire lo scheduler, l'exception handler, l'interrupt handler e inizializzare le strutture dati utili per implementare il kernel. Vediamo ora nel dettaglio le nostre implementazioni:

2 Init.c

Init.c inizializza il Bios: ovvero vengono inizializzate le strutture dati quali: passup Vector, processi e semafori. Dopo questa chiamata Init.c esaurisce la sua funzione e passa totalmente il controllo.

3 Scheduler.c

Lo scheduler si occupa di gestire i processi: più precisamente di decidere qual è il prossimo processo da eseguire, fa in modo che il processo su cui ha eseguito yield() non venga mandato subito dopo in esecuzione a meno che non sia l'unico processo inviabile, poiché bisogna evitare che tali processi riprendano immediatamente dopo l'operazione yield(). Inoltre gestisce le situazioni di HALT, WAIT e, in caso di deadlock, PANIC.

4 Exception Handler.c

4.1 General Exception Handler

Ottiene il contenuto del registro cause, per poi avere il codice eccezione, in base al suo valore scopriamo che tipo di eccezione dobbiamo gestire.

4.2 Pass Up Or Die

Viene realizzata nei casi in cui si abbia program trap o tlb exeption oppure nei casi di syscall con codice insesistente, positivo oppure negativo ma in user mode.

Tramite index scopriamo se siamo nel caso di un'eccezione generale o in una page fault, dopodiché se il processo corrente non ha struttura di supporto, lo terminiamo, altrimenti settiamo i giusti parametri della stessa.

4.3 Syscall Exception Handler

Gestisce le syscall, controllando prima il codice della syscall se il codice è negativo e nel range (-1, -10) facciamo il branching con lo switch sul codice syscall per capire quale chiamare ed infine caricare lo stato corretto dopo aver incrementato il program counter di una word per non avere il loop infinito di syscalls.

Analizziamo ora le Syscall più rilevanti:

4.3.1 TerminateProcess

Abbiamo, per pulizia, ma anche per necessità nelle Syscall successive, deciso di creare quattro funzioni ausiliarie che in questa Syscall fanno la maggior parte del lavoro; Terminate controlla il primo parametro, ovvero il pid passato in input, se è 0 termina il processo chiamante, altrimenti tramite FindProcess trova il processo attraverso il pid e lo termina.

Analizziamo meglio come avviene la terminazione del processo:

se il processo da terminare non ha figli, viene chiamata la funzione Die:

che prende in input il processo e un intero: se l'intero passato è 1 e il processo non ha padre, si esegue un semplice outchild, così da togliere il processo dalla lista dei figli del padre, si controlla se il campo semAdd è NULL, se non lo è si guardano i deviceSemaphore e, se si trova il processo, si decrementa il numero di processi bloccati da essi. Se non viene trovato si controllano i semafori che non gestiscono i device e, se il processo è bloccato, semAdd ha valore 0 e la coda dei processi bloccati da quel semaforo è vuota, si incrementa il valore del campo semAdd, altrimenti si usa removeBlocked e si rimuove dalla coda dei processi bloccati, infine si esegue una verhogen sul semaforo tenedno conto del fatto che sia un deviceSemaphore o meno. Se invece il campo semAdd risulta NULL allora si controlla se il processo corrente è quello da terminare: se non è così si rimuove dalla coda dei processi pronti. Infine si libera il processo tramite FreePcb e si decrementa il numero di processi attivi.

Se ha figli invece si chiama la recursiveDie, che tramite visita in profondità per ogni pcb appartenente al sottoalbero richiama la Die citata precedentemente.

4.3.2 Do - Io

Controlla se il device è il terminale, che ha registri di trasmissione e ricezione, o se invece è di tipo generale, ovvero con solo command register. Se è il terminale setta la line a 4, poiché è costante. Se è di altro tipo incrementa la line con un ciclo for annidato per ogni device, dopodiché si setta a cmdValue il comando

del device. Tramite la variabile isRecvTerm controlla se cmdAddr è un registro di ricezione e in caso setta l'indice a line*8 + numDevice + 8, altrimenti a line*8 + numDevice. Dopodiché inserisce il processo nella coda dei processi bloccati associata al semaforo ed incrementa il softBlockCount, decrementando il valore del deviceSemaphore, richiama lo scheduler e, dopo la chiamata, si controlla quale device è e, se è un non-terminale, si ritorna lo stato del device, se isRecvTerm è 1, si ritorna lo stato di ricezione del terminale, altrimenti si ritorna lo stato di trasmissione.

4.3.3 Yield

Modifica lo stato del processo corrente e lo inserisce nella coda di priorità corretta, dopodiché memorizza in una variabile il puntatore al processo su cui ha eseguito yield() in modo che lo scheduler realizzi la politica di dispatching del prossimo processo correttamente.

5 InterruptHandler.c

Entriamo in InterruptExceptionHandler che tramite getInterruptInt ottiene la linea sulla quale l'interrupt si è acceso e ne controlla il valore:

se è 0 va in PANIC;

se è 1 fa un update dello stato del current process, per poi inserirlo nella coda di priorità corretta, richiamando poi lo scheduler;

se è 2 sblocca tutti i processi bloccati dall'interval-timer semaphore, decrementa il softBlockCount adeguatamente e inserisce il processo nella coda di priorità corretta.

se invece è maggiore di 2 passiamo il controllo alla NonTimerHandler, un'altra funzione:

questa esegue un ulteriore controllo iniziale:

se la linea è compresa fra 2 e 7 si invia un ACK e salva lo stato da ritornare;

se la linea è 7 si casta il device register al terminal register, se il terminale non è pronto a ricevere salva lo stato da ritornare, invia un ACK e setta come pronta la ricezione di un interrupt, altrimenti salva solo lo stato da ritornare e invio un ACK.

Se si è entrati nella NonTimerHandler, dopo i controlli sulle linee si eseguono le seguenti operazioni:

troviamo l'indirizzo del deviceSemaphore, lo incrementiamo, e lo sblocchiamo;

se c'era almeno un processo bloccato inserisce lo stato da ritornare nel registro v0, sblocca il processo, ricalcola il tempo del processo, diminuisce il softBlockCount e, se il processo è diverso da quello corrente, lo inserisce nella coda di priorità corretta poi, se il processo corrente è uguale a NULL, chiama lo scheduler, altrimenti, se il processo sbloccato ha una priorità maggiore rispetto al processo corrente, copia lo stato del processore nel pcb del processo corrente

e lo inserisce nella coda di priorità adeguata, dopodiché chiama comunque lo scheduler. In tutti gli altri casi carica il vecchio stato;

infine se il processo corrente è uguale a NULL richiama lo scheduler, altrimenti carica lo stato del BIOS.