Progetto μMPS3, fase 2

Ahmed Ayad
ahmed.ayad@studio.unibo.it

Andrea Zanella
andrea.zanella7@studio.unibo.it

Gabriele Bernardi
gabriele.bernardi5@studio.unibo.it

Lorenzo Giarrusso
lorenzo.giarrusso@studio.unibo.it

May 7, 2024

Contents

1 Modifiche alle strutture dati della fase 1 **3**

2 main.c **3**

3 scheduler.c **4**
3.1 Estrazione e esecuzione dei processi dalla ready queue 4
3.2 Gestione dell'SSI 4
3.3 Attesa di interrupt 4
3.4 Gestione dei deadlock 4

4 exceptions.c **5**

5 Modulo ssi.c: Gestione dei PCB e Implementazione dei servizi della SSI **7**

6 Gestione delle Interrupts (`interrupts.c`) **7**

1 Modifiche alle strutture dati della fase 1

Nella fase 2 del progetto, si é deciso di introdurre una modifica alle strutture dati esistenti per includere un campo aggiuntivo chiamato `int dev_n` nella struttura `pcb_t`. Questo campo é stato aggiunto per rappresentare il numero del dispositivo su cui il PCB si é bloccato in attesa di un'operazione di I/O. Questa modifica é stata necessaria per gestire in modo piú efficiente i processi bloccati sui dispositivi, consentendo un'identificazione piú rapida e precisa del dispositivo coinvolto.

2 main.c

Il main é strutturato seguendo fedelmente le specifiche richieste, suddividendosi in tre fasi principali:

1. **Inizializzazione di strutture dati e variabili:** Questa fase é dedicata alla preparazione dell'ambiente di esecuzione. Qui vengono create e inizializzate le strutture dati necessarie e le variabili che saranno utilizzate durante l'esecuzione del programma.

2. **Istanziazione dei PCB per l'SSI e per il processo di test:** In questa fase vengono istanziati i Process Control Block (PCB) per la System Service Interface (SSI) e per il processo di test. Ogni PCB rappresenta un programma in esecuzione e contiene le informazioni necessarie per il suo controllo e la sua gestione.

3. **Chiamata dello scheduler:** L'ultima fase comporta la chiamata dello scheduler, che é responsabile della gestione della schedulazione dei processi. Lo scheduler determina quale processo debba essere eseguito in base a politiche predefinite. Si usa uno scheduling di tipo round-robin.

Le decisioni implementative rilevanti e meritevoli di spiegazione sono le seguenti:

- I PCB bloccati sui dispositivi sono gestiti tramite code multiple, una per ogni tipo di dispositivo. In particolare, i terminali hanno una gestione separata per trasmissione e ricezione. Questa scelta é stata fatta dopo aver valutato un'alternativa che prevedeva l'utilizzo di una singola coda generale per tutti i processi bloccati. Tuttavia, l'implementazione di una coda separata per ogni tipo di dispositivo é stata preferita perché l'alternativa avrebbe richiesto controlli e operazioni piú onerose per lo sblocco dei PCB, compromettendo la fairness del sistema.

- I PCB bloccati in attesa di un messaggio non dispongono di una coda dedicata, ma sono identificati "per esclusione": se un PCB non é nella ready queue, allora é necessariamente bloccato e pertanto va sbloccato. Questo approccio semplifica la gestione dei PCB bloccati, garantendo che i processi vengano correttamente gestiti senza dover mantenere code separate per ciascun tipo di blocco.

- Dato che altri moduli possono necessitare di accedere a variabili definite in questo contesto, come il puntatore `current_process` e i contatori dei processi esistenti e di quelli bloccati (`process_count`, `softblock_count`),

queste variabili sono dichiarate come globali. Sebbene questa scelta possa apparire poco elegante, é stata preferita per evitare l'eccessiva complessitá delle chiamate a funzioni che sarebbe stata causata dal passaggio, ad ogni chiamata, dei parametri necessari.

3 scheduler.c

Nel file `scheduler.c`, l'implementazione dello scheduler segue rigorosamente le specifiche fornite. Esaminiamo dettagliatamente il funzionamento del codice, che si articola nei seguenti quattro punti:

3.1 Estrazione e esecuzione dei processi dalla ready queue

Il concetto di "ready queue" rappresenta una coda dei processi che sono pronti per essere eseguiti dalla CPU. Quando un processo é pronto, significa che ha completato le operazioni di inizializzazione e attesa di risorse esterne e ora é in attesa di tempo di CPU per essere eseguito. L'estrazione di un processo dalla ready queue implica che il sistema operativo selezioni uno dei processi pronti per l'esecuzione in base a un algoritmo di scheduling specifico. L'assegnazione del "Process Local Timer" (PLT) e della variabile "startTime" é cruciale per le informazioni di accounting temporale e dunque per far rispettare il time slice assegnato ad ogni processo.

3.2 Gestione dell'SSI

La "System Service Interface" (SSI) é un'entitá critica nel sistema operativo a microkernel che gestisce le interfacce tra il kernel stesso e i servizi di sistema. Quando l'unico processo attivo é l'SSI, il sistema entra in uno stato di HALT, indicando che non vi sono altre attivitá da eseguire al momento. Questo puó accadere, ad esempio, durante il boot del sistema o in situazioni in cui tutti i processi utente sono terminati.

3.3 Attesa di interrupt

L'attesa di interrupt é una fase in cui il sistema operativo si mette in attesa di segnali esterni o eventi hardware che richiedono una risposta immediata. Gli interrupt possono provenire da periferiche esterne, timer hardware o altri eventi che richiedono l'attenzione immediata del sistema operativo. Durante questa fase, il sistema rimane in uno stato di inattivitá relativa, consumando meno risorse finché non si verifica un interrupt.

3.4 Gestione dei deadlock

Il "deadlock" é una situazione critica in cui due o piú processi si bloccano reciprocamente, ognuno aspettando che l'altro rilasci una risorsa che ha in uso, impedendo cosí l'avanzamento. Nel contesto del sistema operativo, un deadlock puó verificarsi se tutti i processi sono in attesa di una risorsa che non puó essere rilasciata fino a quando un altro processo non rilascia una risorsa che detiene. La gestione del deadlock é fondamentale per garantire la stabilitá del sistema operativo e puó comportare l'identificazione dei processi coinvolti nel deadlock

e l'adozione di misure correttive, come la terminazione di alcuni processi o il rilascio forzato delle risorse. Nel nostro caso, si è scelto di utilizzare l'*ostrich algorithm*, semplicemente mandando il sistema in PANIC qualora si individui una situazione di deadlock.

4 exceptions.c

Oltre alle tre funzioni principali previste dalle specifiche (`passUpOrDie`, `syscallHandler` e `exceptionHandler`), sono qui definite alcune funzioni di servizio:

- `update_time`: Questa funzione aggiorna il campo `p_time` di un Process Control Block (PCB), che contiene il tempo di CPU occupato dal PCB dalla sua ultima esecuzione.
- `copyProcessorState`: Questa funzione copia un valore di tipo `state_t` da una locazione all'altra. Essendo un'operazione frequente, `copyProcessorState` aiuta a evitare la riscrittura del codice.
- `die`: Questa funzione si occupa del ramo "die" del `passUpOrDie`, inviando una richiesta all'SSI per terminare il processo attuale e chiamando lo scheduler.
- `send_msg`: Questa funzione crea un messaggio proveniente da un mittente specifico e lo inserisce nell'inbox del destinatario. Gestisce il caso in cui non sia possibile allocare un altro messaggio restituendo il valore `MSGNOGOOD`; altrimenti, restituisce 0 se l'invio ha successo.

Come specificato, la funzione `passUpOrDie` viene utilizzata dall'exception handler per gestire gli eventi non gestibili da interrupt handler e syscall handler. Se il campo `p_supportStruct` del processo attuale è NULL, il controllo passa al ramo "die"; altrimenti, le informazioni di supporto trovate in `p_supportStruct` vengono passate al livello di supporto.

La funzione `syscallHandler` viene chiamata dall'exception handler quando riceve una richiesta SYS1 (syscall per inviare un messaggio) o SYS2 (syscall per ricevere un messaggio). Queste chiamate di sistema possono essere soddisfatte solo se il processo che le richiede è in esecuzione in modalità kernel. Pertanto, viene controllato il bit corrispondente nel registro di controllo Status tramite `getSTATUS`, isolando il bit Kernel/User tramite un AND bitwise con la costante `USERPON`, e facendo lo shift di 3 posizioni. Se il bit Kernel/User vale 1, il processo stava eseguendo in user mode e viene generata un'eccezione con il codice `GENERALEXCEPT` specificando la causa tramite la costante `PRIVINSTR`. Se il bit Kernel/User è 0, il processo era in modalità kernel e la richiesta può essere elaborata.

Se il registro `a0` contiene `SENDMESSAGE`, viene tentato l'invio di un messaggio. Il destinatario è specificato in `a1`, il payload in `a2`, e il risultato dell'invio viene memorizzato in `reg_v0` (`DEST_NOT_EXIST` se il destinatario non esiste, 0 se l'invio è completato con successo, `MSGNOGOOD` altrimenti). La logica di invio comprende:

1. Verifica dell'esistenza del PCB destinatario ("è attualmente assegnato ad un processo?")

2. Se il destinatario non esiste, viene restituito `DEST_NOT_EXIST` in `reg_v0`

3. Se il destinatario è nella ready queue o il messaggio è diretto al mittente stesso, il messaggio viene inserito nella sua casella di posta tramite `send_msg` e il valore di ritorno viene assegnato a `reg_v0`

4. Se il destinatario è bloccato, il messaggio viene inserito nella sua casella di posta tramite `send_msg`, il valore di ritorno viene assegnato a `reg_v0`, e il destinatario viene sbloccato (reinserito nella ready queue).

Se il registro `a0` contiene `RECEIVEMESSAGE`, viene richiesta la ricezione di un messaggio. Il mittente desiderato è specificato in `a1` (se `ANYMESSAGE`, qualsiasi mittente va bene), il payload può essere memorizzato in `a2`, e l'identificatore del mittente deve essere restituito in `reg_v0`. La logica di ricezione comprende:

1. Ricerca di un messaggio nell'inbox del chiamante proveniente dal mittente desiderato.

2. Se non esiste un messaggio soddisfacente, il chiamante viene bloccato.

3. Se viene trovato un messaggio soddisfacente, viene ricevuto senza bloccare il chiamante. L'identificatore del mittente viene memorizzato in `reg_v0`, il payload può essere copiato in `a2`, il messaggio viene reinserito nella lista dei messaggi liberi, e l'esecuzione procede normalmente.

La funzione `exceptionHandler` gestisce tutte le eccezioni ricevute controllando il codice dell'eccezione per determinare quale funzione chiamare per gestirle. Prima di tutto, la funzione salva le informazioni sullo stato dell'eccezione nel puntatore `state_t* exc_state`, che viene poi utilizzato per la gestione. L'exception state è contenuto in `BIOSDATAPAGE`, mentre il codice dell'eccezione è codificato nel campo `exc_code` del registro Cause nell'exception state. Il codice dell'eccezione viene estratto e interpretato per determinare l'azione da intraprendere:

1. Se `exc_code == IOINTERRUPTS`, l'eccezione è un'interrupt, quindi viene chiamato l'`interruptHandler`.

2. Se `exc_code` è compreso tra 1 e 3, si tratta di una Page Fault Exception, quindi viene chiamato `passUpOrDie` specificando il tipo di eccezione come `PGFAULTEXCEPT`.

3. Se `exc_code` è compreso tra 4 e 7 oppure tra 9 e 12, si tratta di una Program Trap, quindi viene chiamato `passUpOrDie` specificando il tipo di eccezione come `GENERALEXCEPT`.

4. Se `exc_code == SYSEXCEPTION`, viene chiamato `syscallHandler`.

5. Se nessuna delle condizioni precedenti è soddisfatta, il codice dell'eccezione non può essere gestito e il sistema va in PANIC.

5 Modulo ssi.c: Gestione dei PCB e Implementazione dei servizi della SSI

Il modulo `ssi.c` gestisce i Process Control Blocks (PCB) e implementa i servizi della System Service Interface (SSI). Esso definisce due funzioni di servizio per il bloccaggio dei PCB e implementa i vari servizi richiesti dalle specifiche.

Le funzioni di servizio sono:

- `blockPCB`: Blocca il PCB sulla linea specificata e lo inserisce nella coda di attesa corrispondente, distinguendo fra i casi di trasmissione e ricezione per i terminali.
- `blockPCBfromAddr`: Identifica il dispositivo corrispondente a un indirizzo dato e blocca il PCB specificato tramite la funzione `blockPCB`.

La funzione principale `ssi` gestisce l'SSI secondo le specifiche. All'interno di un ciclo infinito, essa attende e processa le richieste, comunicando i risultati ai richiedenti. Le richieste sono processate dalla funzione `ssi_request`, che si basa sul campo `service_code` nel payload del messaggio di richiesta.

I servizi offerti includono:

1. `ssi_create_process`: Crea un processo come figlio del richiedente.
2. `ssi_terminate_process`: Termina un processo e il suo sotto-albero, rimuovendo il PCB dalla coda di attesa, se necessario.
3. `ssi_DoIO`: Blocca il richiedente sulla coda del dispositivo corrispondente, supportando solo I/O sincrono.
4. `GetCPUTime`: Restituisce il tempo di CPU del chiamante dal campo `p_time`.
5. `ssi_waitForIT`: Mette il chiamante in attesa del prossimo tick dell'Interval Timer.
6. `GetSupportData`: Restituisce i dati di supporto del chiamante dal campo `p_supportStruct`.
7. `ssi_getPID`: Ottiene il PID del chiamante o del suo PCB padre, come specificato nelle specifiche.

In conclusione, il modulo `ssi.c` fornisce una gestione efficiente dei PCB e implementa in modo accurato i servizi necessari per il funzionamento del sistema, garantendo un'interfaccia coerente e affidabile per la gestione dei processi e delle interrupt.

6 Gestione delle Interrupts (`interrupts.c`)

Il modulo `interrupts.c` contiene l'implementazione delle funzioni necessarie per gestire le interrupt nel sistema. Le funzioni principali definite in questo modulo sono:

- `unblockPcbByDevNum`: Questa funzione si occupa di sbloccare il Process Control Block (PCB) che è stato precedentemente bloccato su un dispositivo specifico, identificato tramite il numero di dispositivo fornito come parametro `devnum`.

- `getHighestPriorityDevNo`: Data una bitmap dei dispositivi su una certa linea, questa funzione restituisce il numero del dispositivo con la massima priorità fra quelli che attualmente hanno un'interrupt attiva.

La funzione principale, `interruptHandler`, è responsabile di processare gli interrupts ricevuti e smistarli alle funzioni appropriate in base alla loro provenienza. Per determinare quali linee hanno attualmente un'interrupt attiva, il sistema controlla il campo `IP` (Interrupt Pending) del registro `Cause` e ne effettua un'operazione di AND bitwise con le costanti specifiche per ogni linea di interrupt (`IL_CPUTIMER`, `IL_TIMER`, ecc.).

Si noti che il controllo delle linee di interrupt viene eseguito partendo dalla linea 0 fino alla 7, garantendo così la priorità definita nelle specifiche del sistema. Se più linee hanno un'interrupt attiva contemporaneamente, viene gestito prima l'interrupt della linea con il numero più basso.

In base al tipo di interrupt ricevuto, la funzione `interruptHandler` chiama una delle seguenti funzioni di gestione:

- `PLT_interruptHandler`: Chiamata per un interrupt PLT (Process Local Timer), quando il processo in esecuzione esaurisce il suo tempo di esecuzione allocato. Questa funzione reimposta il timer, aggiorna il campo `p_time` del processo, lo rimette in fondo alla coda dei processi pronti (*ready queue*) e chiama lo scheduler per selezionare il prossimo processo da eseguire.

- `IT_interruptHandler`: Chiamata per un interrupt generato dall'Interval Timer, ovvero per un tick dello *Pseudo-clock*. Questa funzione reinizializza il valore dell'Interval Timer, sblocca tutti i processi in attesa di un tick dello *Pseudo-clock*, e continua con l'esecuzione del processo attuale se presente, altrimenti chiama lo scheduler per selezionare un nuovo processo da eseguire.

- `deviceInterruptHandler`: Chiamata per un interrupt proveniente da un dispositivo. Questa funzione gestisce il dispositivo identificando prima quello con la massima priorità e un'interrupt attiva sulla linea più alta. Se il dispositivo è un terminale, gestisce separatamente i casi di trasmissione e ricezione. Dopo aver identificato e gestito il dispositivo, viene inviato un *acknowledgement* al dispositivo stesso scrivendo la costante `ACK` sul suo registro. Successivamente, viene sbloccato in modo FIFO un PCB fra quelli in attesa sulla coda di quel dispositivo. Se nessun PCB era in attesa, la funzione procede senza ulteriori azioni. Altrimenti, il PCB viene sbloccato, un messaggio viene inviato al processo e il PCB viene inserito nella coda dei processi pronti (*ready queue*). Infine, se c'era un processo in esecuzione prima dell'interrupt, questo viene ri-caricato e ripreso. In caso contrario, viene chiamato lo scheduler per selezionare un nuovo processo da eseguire.