9 - Input - Output

Principi dell'Hardware di I/O

Dispositivi I/O

I dispositivi di I/O si suddividono principalmente in due categorie:

1. **Dispositivi a blocchi:** Memorizzano informazioni in blocchi di dimensioni fisse, ciascuno con il proprio indirizzo. Tutti i trasferimenti di dati avvengono in unità di uno o più blocchi interi. Caratteristica principale: ogni blocco è indipendente dagli altri. Esempi includono dischi fissi magnetici, SSD, ecc.
2. **Dispositivi a caratteri:** Trasferiscono o accettano un flusso di caratteri senza una struttura a blocchi. Questi dispositivi non sono indirizzabili e non supportano operazioni di ricerca. Esempi comuni sono stampanti, mouse, tastiere, ecc.
   È importante notare che questa classificazione non include tutti i tipi di dispositivi di I/O, come ad esempio orologi, schermi o touchscreen.

Controller dei Dispositivi

I dispositivi di I/O sono composti da una parte elettronica chiamata controller del dispositivo, che è integrata nei computer sotto forma di chip o scheda a circuiti stampati. Questi controller possono gestire diversi dispositivi simili. Le interfacce tra il controller e il dispositivo possono essere standardizzate, come ANSI, IEEE, ISO, o de facto, come SATA, SCSI, USB o Thunderbolt. L'interfaccia tra il controller e il dispositivo è di livello molto basso; il compito del controller è convertire il flusso seriale di bit in blocchi di byte, correggere errori e trasferire dati in memoria centrale. Senza il controller, i programmatori dovrebbero gestire dettagli complessi come la modulazione di ciascun pixel; il sistema operativo inizializza il controller con parametri essenziali e delega la gestione dei dettagli complessi.

Da porta parallela a USB

- **Porta Parallela:** Tipo di interfaccia di comunicazione tra computer e dispositivi. Trasmette dati multi-bit simultaneamente su piú canali. Comunemente dotata di 25 o 36 pin.

- **Porta USB:** Universal Serial Bus (USB) è un'interfaccia standardizzata per collegare dispositivi al computer e facilitare la comunicazione e lo scambio di dati. Inoltre, USB può fornire alimentazione elettrica ai dispositivi collegati, rendendola versatile e facile da utilizzare grazie alla connessione plug-and-play.

I/O Mappato in Memoria

Ogni controller ha registri utilizzati per la comunicazione con la CPU. Il sistema operativo scrive in questi registri per dare comandi al dispositivo e legge da essi per conoscere lo stato del dispositivo. Alcuni dispositivi hanno anche un buffer di dati che il sistema operativo può utilizzare per scrivere e leggere dati, ad esempio la RAM video per visualizzare punti sullo schermo.

Port-Mapped I/O

Questo metodo utilizza numeri di porta di I/O per associare ogni registro di controllo. Ad esempio, per leggere da una porta si usa "IN REG, PORT", mentre per scrivere si usa "OUT PORT, REG". È cruciale mantenere separati gli spazi di indirizzi della memoria e dell'I/O. Pertanto, "IN R0, 4" legge dalla porta 4 e salva il valore in R0, mentre "MOV R0, 4" legge dall'indirizzo di memoria 4 e salva il valore in R0. Le due istruzioni con "4" si riferiscono a indirizzi differenti.

Memory-Mapped I/O

Questo metodo mappa ogni registro di controllo direttamente nello spazio di memoria, eliminando la necessità di istruzioni speciali come IN e OUT per l'I/O. I registri di controllo possono essere trattati come variabili in linguaggi come C, semplificando lo sviluppo dei driver direttamente in C anziché in assembly. Il sistema operativo deve solo evitare di mappare questi indirizzi nello spazio degli indirizzi virtuali dei processi utente per garantire la sicurezza. Utilizzando la gestione delle pagine di memoria, è possibile concedere controllo selettivo sui dispositivi specifici, eseguendo i driver dei dispositivi in spazi separati per migliorare la sicurezza e ridurre le dimensioni del kernel. Tuttavia, questo metodo può comportare una gestione meno efficiente della cache e dei registri mappati in memoria. Un approccio ibrido potenziale consiste nel filtrare gli indirizzi per distinguere tra memoria e dispositivi di I/O, inoltrando gli indirizzi di I/O direttamente ai dispositivi anziché alla memoria.

DMA

DMA, acronimo di Direct Memory Access, è una tecnologia che consente a dispositivi hardware di accedere direttamente alla memoria del computer senza passare attraverso la CPU. Questo miglioramento nell'efficienza consente trasferimenti più rapidi di dati tra dispositivi e la memoria del sistema, riducendo il carico sulla CPU e migliorando le prestazioni complessive del sistema.

Modalitá DMA e interazioni con il BUS

- **Cycle Stealing:** DMA trasferisce una parola per volta "rubando" cicli di CPU
- **Burst:** Controllo completo del BUS, esegue trasferimenti multipli
- **Fly-By Mode:** Trasferisce direttamente alla memoria principale

Interrupt

Si manifestano per determinate cause:

- **Trap:** Azione da parte del programma, trap nel kernel per una sysCall
- **Fault o Eccezione:** Errori di segmentazione o divisioni per 0
- **Interrupt Hardware:** Segnali da dispositivi Hardware esterni

Gestione degli Interrupt

- **Segnalazione dell'Interrupt alla CPU:** Il controller invia un segnale di interruzione alla CPU specificando il dispositivo richiedente.
- **Interruzione e Gestione da Parte della CPU:** La CPU interrompe il compito corrente e utilizza il numero fornito dal controller per accedere alla tabella del vettore degli interrupt.
- **Il Vettore degli Interrupt:** Indica l'inizio della procedura di servizio per l'interrupt corrispondente.
- **Conferma e Gestione degli Interrupt:** La procedura di servizio conferma l'interrupt scrivendo su una porta del controller degli interrupt.
- **Salvataggio dello Stato:** Il contatore di programma viene salvato per riprendere i processi interrotti. La CPU decide se utilizzare lo stack corrente del processo o lo stack del kernel, con relativi vantaggi e svantaggi.

Tipologie di Interrupt

Tipologie di Interrupt

1. **Interrupt Precisi:** Il sistema può determinare esattamente quali istruzioni sono state completate al momento dell'interrupt. Il PC (Program Counter) è salvato in un luogo noto. Tutte le istruzioni eseguite prima del PC sono completate, nessuna istruzione dopo il PC è stata eseguita e lo stato dell'istruzione puntata dal PC è noto. La CPU cancella gli effetti di eventuali istruzioni transitorie eseguite dopo il PC per garantire compatibilità e prevedibilità.
2. **Interrupt Imprecisi:** Diverse istruzioni vicino al contatore di programma si trovano in vari stati di completamento al momento dell'interrupt, rendendo incerto lo stato esatto del programma. La CPU deve salvare una grande quantità di stato interno sullo stack, rendendo il processo di gestione molto lento.

Principi del Software di I/O

Obiettivi

- **Indipendenza dal Dispositivo:** Il software di I/O dovrebbe permettere l'accesso a vari dispositivi senza specificarne il tipo.
- **Denominazione Uniforme:** I nomi di file o dispositivi dovrebbero essere stringhe o numeri indipendenti dal dispositivo.
- **Gestione degli Errori:** Gli errori vanno gestiti il più vicino possibile all'hardware, preferibilmente dal controller o dal driver del dispositivo, con ripetizioni per errori transitori.
- **Trasferimenti Sincroni vs Asincroni:** Sebbene l'I/O fisico sia spesso asincrono, i programmi utente lo trattano come sincrono per semplicità. Il sistema operativo fa sembrare le operazioni asincrone bloccanti, ma offre anche accesso asincrono per applicazioni ad alte prestazioni.
- **Buffering:** I dati spesso necessitano di un buffer temporaneo prima di raggiungere la destinazione finale, influenzando le prestazioni, specialmente per dispositivi con vincoli real-time.
- **Dispositivi Condivisibili vs Dedicati:** Dispositivi come dischi e SSD possono essere condivisi tra più utenti, mentre altri come stampanti e scanner sono solitamente dedicati.

Tipologie di Software per I/O

I/O Programmato

La CPU gestisce direttamente il trasferimento dei dati. Un esempio pratico:

1. Un processo utente prepara una stringa in un buffer dello spazio utente e richiede la stampa.
2. Il sistema operativo copia il buffer nello spazio kernel.
3. Invio dei caratteri alla stampante uno alla volta, aspettando che questa sia pronta per ogni carattere.
4. Il sistema operativo entra in un ciclo di polling, controllando il registro di stato della stampante e inviando un carattere alla volta.
   Questo processo occupa la CPU a tempo pieno durante l'I/O, risultando efficace solo quando il tempo di elaborazione di un carattere è breve.

```
copy_from_user(buffer, p, count);
for(int i = 0; i < count; i++){
while (*printer_status_reg != READY);
*printer_data_register = p[i];
}
return_to_user();
```

I/O Guidato da Interrupt

1. **Scenario:** Una stampante senza buffer stampa un carattere ogni 10 ms, consentendo alla CPU di svolgere altre attività in questo intervallo.
2. **Procedura:**
   - Dopo la chiamata di sistema per stampare, il buffer viene copiato nello spazio kernel.
   - Il primo carattere viene inviato alla stampante appena è pronta.
3. **Gestione CPU:** La CPU chiama lo scheduler ed esegue un altro processo mentre il processo di stampa è bloccato.
4. **Interrupt:** Quando la stampante è pronta per il carattere successivo, genera un interrupt che ferma il processo attuale e salva lo stato.
5. **Servizio di Interrupt:** La procedura di servizio dell'interrupt della stampante viene eseguita per continuare il processo di stampa.

I/O con DMA

1. **Limite dell'I/O Guidato dagli Interrupt:** Genera un interrupt per ogni carattere, sprecando tempo della CPU.
2. **Soluzione - DMA:** Utilizza un controller DMA per inviare i caratteri alla stampante senza coinvolgere la CPU per ogni carattere.
3. **Vantaggio:** Riduce il numero di interrupt da uno per carattere a uno per buffer, permettendo alla CPU di svolgere altre attività durante l'I/O.
4. **Requisito:** Richiede hardware speciale (controller DMA).

Struttura del Software di I/O

Il software di I/O é generalmente organizzato in quattro livelli, come mostrato nella tabella, ciascuno dei quali ha una funzione ben definita da eseguire e un'interfaccia ben definita verso i livelli adiacenti.

Software I/O a livello utente
Software del SO indipendente dal dispositivo
Driver Dispositivo
Gestore degli interrupt

Gestore degli Interrupt

1. **Salvataggio Registri:** Salva tutti i registri (incluso il PSW) non ancora salvati.
2. **Impostazione Contesto:** Configura il contesto per la procedura di servizio, includendo TLB, MMU e tabella delle pagine.
3. **Impostazione Stack:** Configura uno stack per la procedura di servizio dell'interrupt.
4. **Conferma al Controller:** Conferma al controller degli interrupt o riabilita gli interrupt se non c'è un controller centralizzato.
5. **Copia Registri:** Copia i registri salvati nella tabella dei processi.
6. **Esecuzione Procedura:** Esegue la procedura di servizio, estraendo informazioni dai registri del controller del dispositivo.
7. **Scelta Processo:** Decide quale processo eseguire successivamente, considerando le priorità dei processi bloccati ora pronti.
8. **Impostazione Contesto MMU:** Configura il contesto della MMU per il processo successivo e, se necessario, imposta il TLB.
9. **Caricamento Nuovi Registri:** Carica i registri del nuovo processo, incluso il PSW.
10. **Avvio Nuovo Processo:** Avvia l'esecuzione del nuovo processo.

Driver dei Dispositivi

I driver gestiscono i dispositivi di I/O attraverso registri specifici e possono essere specifici per un dispositivo o per una classe di dispositivi. Ogni dispositivo richiede un driver fornito dal produttore. Tecnologie come USB usano una pila di driver per supportare vari dispositivi.
I driver sono parte del kernel per accedere ai registri del controller, ma se usati nello spazio utente sono più facili da installare e meno rischiosi per il sistema operativo, sebbene più lenti. I driver sono installati come codice di terze parti e si trovano sotto il resto del sistema operativo. Nei sistemi moderni, i driver sono caricati dinamicamente.
I driver si classificano in due categorie: a blocchi (dischi) e a caratteri (stampanti, tastiere). Gestiscono letture, scritture, inizializzazioni del dispositivo, verificano la validità dei parametri di input e li traducono in comandi specifici per il dispositivo.

Software del SO indipendente dal Dispositivo

Il software di I/O indipendente dal dispositivo funge da intermediario tra i driver specifici dei dispositivi e le applicazioni utente. L'obiettivo è semplificare l'interazione con l'hardware fornendo un'interfaccia uniforme e gestendo operazioni comuni.

- **Interfaccia Uniforme dei Driver dei Dispositivi:** Standardizza un modello dove tutti i driver condividono la stessa interfaccia verso il SO, evitando modifiche al sistema operativo per l'introduzione di nuovi dispositivi. Ogni classe di dispositivi ha un set definito di funzioni che i driver devono implementare.
- **Buffering:** Gestisce i buffer per ottimizzare il trasferimento dati tra dispositivi e sistema.
- **Gestione degli Errori:** Identifica e comunica gli errori provenienti dai dispositivi agli utenti o ad altri sistemi.
- **Gestione dei Dispositivi Dedicati:** Assegna e rilascia dispositivi dedicati a specifici compiti o utenti.
- **Uniformitá della Dimensione dei Blocchi:** Assicura che la dimensione dei blocchi dati sia gestita in modo uniforme, indipendentemente dalle caratteristiche specifiche del dispositivo.

Software per I/O a livello utente

Le librerie di I/O semplificano le chiamate di sistema per l'input e l'output, ad esempio con funzioni come `write()` e `read()`. Funzioni come `printf()` e `scanf()` gestiscono dati e trasformazioni prima di richiamare le funzioni di sistema, migliorando la facilità di programmazione. Lo spooling è utilizzato nei sistemi multiprogrammati per gestire dispositivi dedicati attraverso un processo daemon e una directory di spooling, migliorando l'efficienza nell'uso dei dispositivi e la gestione delle risorse.