

Appunti di Sistemi Operativi

Jasin Atipi

September 2017

1 Marzo 2017

Processi e Thread (slides)

Il processo è l'istanza di un programma in esecuzione. Il programma è un concetto statico, il processo è dinamico. Il processo è eseguito in modo sequenziale, un'istruzione alla volta. In un sistema multiprogrammato i processi evolvono in modo concorrente. Le risorse fisiche e logiche sono limitate. Il S.O. stesso è un insieme di processi.

Il processo consiste di:

- Istruzioni (sezione Codice o Testo). Parte statica del codice.
- Dati (sezione Dati). Variabili globali.
- Stack. Chiamate a procedura e parametri, variabili locali.
- Heap. Memoria allocata dinamica.
- Attributi (id, stato, controllo).

Attributi (Process Control Block)

All'interno del Sistema Operativo ogni processo è rappresentato dal PCB, che contiene informazioni importanti:

stato del processo, program counter, valori dei registri, informazioni sulla memoria (registri limite, tabella pagine), informazioni sullo stato dell'I/O (richieste pendenti, file), informazioni sull'utilizzo del sistema (CPU), informazioni di scheduling.

Stato di un processo

Durante la sua exec., un processo evolve attraverso diversi stati. Diagrammi di stato diverso dipendentemente dal S.O.

Il processo può essere in exec o non in exec.

Scheduling

Selezione del processo da eseguire nella CPU al fine di garantire: multiprogrammazione e time-sharing.

Long-term scheduler: seleziona quali processi devono essere trasferiti nella coda dei processi pronti. Ordine dei secondi

Short-term scheduler: seleziona quali sono i prossimi processi ad essere eseguiti ed alloca la CPU. Ordine dei millisecondi

Ogni processo è inserito in una serie di code: Coda di processi pronti o Coda di un dispositivo.

Dispatcher

Cambio di contesto: salvataggio PCB del processo che esce e caricamento del PCB del processo che entra.

Passaggio alla modalità utente: all'inizio della fase di dispatch il sistema si trova in modalità kernel

Salto dell'istruzione da eseguire del processo appena arrivato nella CPU.

Il cambio di contesto (context switching) effettua il passaggio della CPU ad un nuovo processo.

Operazioni sui processi

Un processo può creare un figlio. Il figlio ottiene le risorse dal S.O. o dal padre. I tipi di esecuzione sono di tipo sincrona (il padre attende la terminazione dei figli) o asincrona (evoluzione parallela o concorrente di padre e figli).

La creazione di un processo in UNIX avviene in 3 modi:

System call fork: crea un figlio che è un duplicato del padre.

System call exec: carica sul figlio un programma diverso da quello del padre.

System call wait: per esecuzione sincrona tra padre e figlio.

8 Marzo 2018

Modelli di comunicazione

I thread fanno tutti parte di un solo processo (dal punto di vista del kernel), mentre dal punto di vista dell'utente i thread sono cose separate.

Solitamente i thread sono molti di più dei processi.

I processi sono due immagini separate nel sistema operativo, ma devono avere un modo di poter comunicare. Ci sono due modelli principali di comunicazione fra processi: scambio di messaggi (usando il kernel) o memoria condivisa.

La memoria condivisa è più efficiente però bisogna stare attenti a non far accedere processi non autorizzati a tale memoria.

Il message passing è un insieme di meccanismi utilizzati dai processi per comunicare e sincronizzare le loro azioni.

Lo scambio di messaggi consiste nella comunicazione di processi senza condividere variabili.

Le IPC forniscono due operazioni: send - receive.

Se P e Q vogliono comunicare, devono: aprire un canale di comunicazione; mandarsi messaggi send/receive.

Il canale di comunicazione è logico o fisico.

La nominazione (sorgente e destinazione) dipendono dai metodi di comunicazione che possono essere: diretto - indiretto.

Nella comunicazione diretta, send - receive devono nominarsi esplicitamente.

A sua volta esistono due metodi di comunicazione diretta: simmetrica - asimmetrica. Nella simmetrica: send (P1, message) - receive (P2, message). Nella asimmetrica: send(P1, message) - receive (id, message). Nella simmetrica i due processi devono conoscersi a vicenda. Nella asimmetrica quando spedisce so a chi spedire, quando ricevo invece non so chi mi sta spedendo il messaggio. Esiste uno svantaggio: se un processo cambia nome ci si confonde.

Le comunicazioni indirette si basano sul principio:

"All problems in computer science can be solved by another level of indirection".

I messaggi (nella comunicazione indiretta) sono spediti e ricevuti da mailboxes (anche chiamati porte). Ogni mailbox ha un unico id, i processi possono comunicare solo se condividono una mailbox.

Nelle comunicazioni indirette bisogna avere le seguenti operazioni:

creare una mailbox, send -receive tramite mailbox, eliminare una mailbox.

Condivisione di mailbox:

P_1, P_2, P_3 condividono la mailbox A.

P_1 , *spedisce*; P_2, P_3 ricevono.

Chi ottiene il messaggio?

Soluzioni:

Permettere ad un canale che sia associato con al più due processi

Sincronizzazione tra send e receive:

Lo scambio di messaggi può essere bloccante o non bloccante:

Bloccante (sincrono): send bloccante: il mittente si blocca finché il messaggio è stato ricevuto. Receive bloccante: il ricevente è bloccato finché il messaggio è disponibile.

Non-bloccante (asincrono): send non bloccante: il mittente spedisce il messaggio e continua, receive non bloccante: il ricevente riceve un messaggio valido o nulla

Memoria condivisa.

Esempio posix: processo prima crea il segmento di memoria condivisa:

```
segment id = shmget(IPC_PRIVATE, size, S_IRUSR | S_IWUSR);
```

Il processo che vuole accederci si attacca a questa:

```
shared memory = (char *) shmat(id, NULL, 0)
```

Ora il processo può scrivere nel segmento condiviso:

```
sprintf(shared memory, "Writing to shared memory");
```

Quando finito il processo stacca il segmento di memoria:

```
shmdt(shared memory);
```

Oer rimuovere il segmento di memoria condivisa:

```
shmctl(shm_id, IPC_RMID, NULL);
```

Scheduling dei processi

Lo scheduling è l'assegnazione di attività nel tempo.

L'utilizzo nella multiprogrammazione impone l'esistenza di una strategia per regolamentare:

1. ammissione dei processi nel sistema (memoria)
2. ammissione dei processi all'esecuzione (CPU)

Si divide in tre fasi: pronto (tanti processi) - in esecuzione (un processo solo) - in attesa (tanti processi)

Esistono molte code in cui i processi possono essere nella fase "pronto".

Scheduler a lungo e breve termine

Lo scheduler a lungo termine (job scheduler) seleziona quali processi devono essere portati dalla memoria alla ready queue.

Lo scheduler a breve termine (CPU scheduler) seleziona quale processo deve essere eseguito dalla CPU.

Lo scheduler a breve termine è invocato spesso molto veloce ($O(\mu s)$). Es: 100 *ms* per processo, 10 *ms* per scheduling. $10/110 = 9\%$ del tempo di CPU sprecato per scheduling. Lo scheduler a lungo termine è invocato più raramente. $O(ms)$. Controlla il grado di multiprogrammazione e il mix dei processi: I/O-bound: molti I/O, molti brevi burst di CPU. CPU-Bound: molti calcoli, pochi lunghi burst di CPU.

Esiste anche lo scheduler a medio termine. S.O. con memoria virtuale prevedono un livello intermedio di scheduling (a medio termine).

Per la momentanea rimozione forzata di un processo dalla CPU (serve per ridurre grado) multiprogrammazione.