LEZIONE 4

**Concetto di processo Un processo è un programma in esecuzione;** l’esecuzione dei processi deve avvenire in modo sequenziale**.** Per l’elaborazione di un insieme di processi si possono considerare due tipi di elaborazioni:

* **Un sistema a lotti (batch system)** esegue lavori (jobs): i processi che si hanno a disposizione vengono mandati tutti insieme in elaborazione, ma non c’è possibilità d’interazione con l’utente, in quanto una volta in elaborazione il comando è totalmente del sistema e quindi si perde ogni contatto con essi.

**Questo tipo di sistema è più performante in situazioni di alto traffico**

* **Un sistema a partizione del tempo (time-shared system)** esegue programmi utenti o task. Questo comporta la partizione del tempo in slices (fette) le quali vengono concesse ai vari processi che ne fanno richiesta.

**Lo svantaggio del time-shared è** che una percentuale del tempo va utilizzata per la gestione di questo meccanismo di scambio. **Questo tipo di sistema è più performante in situazioni di basso traffico**

* **Un processo comprende:**

1. **Contatore di programma (program counter)**: è una variabile che contiene il punto dell’indirizzo nel quale è stata eseguita l’ultima istruzione. Questo è utile perché se per qualche motivo il processo viene congelato, con il program counter si può conoscere il punto di ripresa.
2. **Pila (stack):** particolare struttura dati nel quale l’accesso è solo da un verso ( si inseriscono i dati in cima per poi riprenderli sempre dalla cima)
3. **Sezione di dati (data section)**

Stato di un processo Mentre un processo è in esecuzione, è soggetto a cambiamenti di **STATO:**

1. **NUOVO**: si crea il processo
2. **ESECUZIONE**: le istruzioni vengono eseguite
3. **ATTESA**: il processo attende che si verifichi qualche evento
4. **PRONTO**: il processo attende di essere assegnato a un’unità d’elaborazione
5. **TERMINATO**: il processo ha terminato

Diagramma di transizione di stati di un processo

**Spiegazione** Per poter convertire un pezzo di programma in processo, bisogna allocare una serie d’informazioni; di conseguenza non può essere ancora elaborato, dato che sta in fase di vestizione. Quando il processo è in questa fase si start up, si dice che si trova in uno **stato di** **NUOVO.** Una volta finita tale fase il processo passa nello **stato PRONTO** (ciò non comporta di essere subito eseguito dato che ci sono anche altri processi). A seconda di alcuni criteri poi il processo scelto passa allo **stato di esecuzione.** Dallo stato di esecuzione si può uscire in 3 modi: 1. Si va **nello stato di ATTESA,** ovvero il processo per qualche ragione non può continuare l’esecuzione; quando poi tale evento (es. dati recepiti) è terminato ì, questo ripassa allo stato di pronto rimettendosi in discussione con gli altri processi. 2. Si ha un’interruzione momentanea non per volontà del processo che ha bisogno di qualcosa, ma è il SO che lo interrompe momentaneamente in quanto arriva un processo con priorità maggiore (di cui il primo non sa nulla), questo però senza mandarlo in stato di attesa. 3. Si va allo **stato terminato,** ovvero il processo ha terminato l’esecuzione. **La differenza tra il punto 1 e 2 è che il processo va nello stato di attesa volutamente in quanto ha bisogno di qualche dato, per poi ripassare (una volta ricevuti i dati) di nuovo allo stato pronto; mentre nel secondo caso il processo viene interrotto non per sua scelta ecco perché passa subito allo stato di pronto.**

Blocco di controllo di un processo (PCB) Quando un processo nasce, bisogna attivare una serie di informazioni che vanno contenute in u blocco dati detto PCB. Le informazioni connesse ad un processo specifico sono:

* **Stato del processo**
* **Contatore di programmi**
* **Registri di CPU**
* **Informazioni sullo scheduling di CPU**
* **Informazioni sulla gestione della memoria**
* **Informazioni di contabilizzazione delle risorse**
* **Informazioni sullo stato dell’I/O**

**PCB**

Immagine che contiene screenshot

Descrizione generata automaticamenteIl puntatore punta al prossimo stato del PCB

Il PCB è una sorta di fotografia del processo per consentire la ripresa di questo in un seguito a un eventuale congelamento.

**4**

Immagine che contiene orologio

Descrizione generata automaticamenteLa CPU può essere commutata tra processi

**Spiegazione** Attraverso una system call un processo va in esecuzione, dopo di che c’è un’interruzione (freccia dopo system call) a causa di un secondo processo (seconda linea verticale). Durante l’elaborazione si verifica una seconda interruzione (nested interrupt); quindi ci sta una interruzione nidificata, ovvero in un processo ci sta un secondo processo anch’esso interrotto a causa di un terzo processo che però conclude la sua elaborazione. Una volta finita si ritorna indietro, quindi al punto A e cosi via fino al processo iniziale. Tutte queste riprese sono state possibili grazie al PCB.

**Immagine che contiene screenshot

Descrizione generata automaticamenteFunzionamento nello specifico**

**Spiegazione** P0 è in esecuzione, ad un certo punto ha un’interruzione e il SO prende il controllo, salva lo stato di p0 in PCB0. Dopo di che il SO prende P1 e lo ripristina (lo manda in esecuzione). Quando P1 ha terminato o è stato sospeso, il SO prende di nuovo il controllo, salva lo stato si P1 in PCB1 e ripristina P0 da dove lo aveva congelato

Code di scheduling dei processi

* **Coda di processi (job queue):** insieme di tutti i processi del sistema.
* **Coda dei processi pronti (ready queue):** processi presenti nella memoria centrale, pronti e in attesa di essere eseguiti.
* **Coda del dispositivo (device queue):** elenchi dei processi che attendono la disponibilità di un particolare dispositivo di I/O.

L’appartenenza ad una coda non comporta staticità, ovvero se un processo nasce in una coda non è detto che muoia in quella stessa cosa. **Coda dei processi pronti e diverse code di dispositivi I/O:**

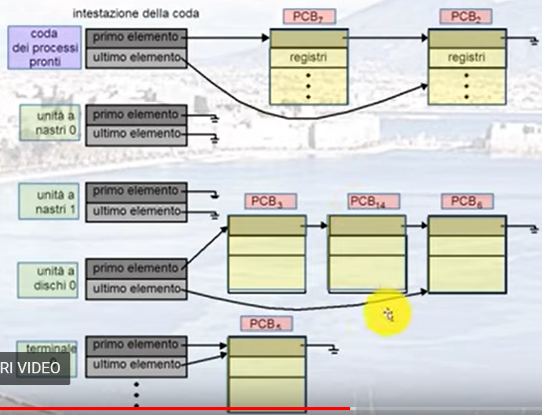


Diagramma di accodamento per lo scheduling dei processi

Immagine che contiene screenshot, orologio

Descrizione generata automaticamente**Spiegazione** Dall’esterno arriva un processo che va nella coda dei processi pronti (alla coda dei processi pronti si arriva da diversi punti come mostra il disegno); da qui viene scelto quale dare alla CPU. La CPU può terminare il processo. Oppure si verificano delle interruzioni temporanee (es. 4 voci scritte sotto CPU) e alla fine dell’interruzione il processo torna alla coda dei processi pronti.

Scheduler

* **Scheduler a lungo termine (job scheduler)**: seleziona **i processi che devono essere caricati nella coda dei processi pronti.**

**Nota:** poco critico in quanto ha molto tempo per effettuare una scelta oculata dei processi.

* **Scheduler di CPU (short-term scheduler o CPU scheduler):** fa la selezione tra **i lavori pronti per essere eseguiti e assegna la CPU a uno di essi.**

**Nota:** molto critico perché ha poco tempo a disposizione e non può permettersi di sbagliare altrimenti ci sono molti problemi.

**Dispatch:** è un modulo e serve per caricare il processo pronto e farlo andare in esecuzione.

**Latenza di dispatch:** il tempo impiegato dal dispatch per sospendere un processo o avviare l’esecuzione di un nuovo processo.

**Nota:** più basso è questo tempo, meglio è l’efficienza.

* **Scheduling a medio termine e diagramma di accodamento** Ha il compito di rimuovere processi dalla memoria (e dalla contesa per la CPU)

Immagine che contiene screenshot

Descrizione generata automaticamente

Scheduler (cont)

* **Lo scheduler di CPU** viene invocato frequentemente (millisecondi) => (deve essere veloce).
* **Lo scheduler a lungo termine** viene invocato meno frequentemente (secondi, minuti) => (può essere lento).
* Lo scheduler a lungo termine controlla il **grado di multiprogrammazione.**
* I processi possono essere caratterizzati come:
* **Processo con prevalenza di I/O (I/O-bound process):** impiega la maggior parte del proprio tempo nell’esecuzione di operazioni di I/O.
* **Processo con prevalenza d’elaborazione (CPU-bound process):** richiede poche operazioni di I/O e impiega la maggior parte del proprio tempo nelle elaborazioni.

**Nota:** i processi che si scelgono devono essere un mix di processi con prevalenza di I/O e processi con prevalenza d’elaborazione.

Cambio di contesto

* Passaggio della CPU a un nuovo processo, con conseguente registrazione dello stato del processo vecchio e il caricamento dello stato precedentemente registrato del nuovo processo.
* **Il tempo necessario al cambio di contesto è puro sovraccarico:** il sistema non compie alcun lavoro utile durante la commutazione. Di conseguenza la durata del tempo di cambio deve essere il più breve possibile.
* La durata del cambio di contesto dipende molto dall’architettura.

Creazione di un processo

* Un processo genitore può creare numerosi processi figli che, a loro volta, possono creare altri processi creando un **ALBERO DI PROCESSI.**
* **Condivisione delle risorse**
* Genitore e figlio condividono tutte le risorse.
* Il processo figlio condivide un sottoinsieme delle risorse del genitore.
* Genitore e figlio non condividono alcuna risorsa.
* **Esecuzione**
* Genitore e figlio possono essere eseguiti in modo concorrente.
* Il processo genitore attende che alcuni o tutti i suoi processi figli terminino. **(Questi due punti significano che è inimmaginabile che un figlio non abbia un padre; di conseguenza se il padre termina l’esecuzione, il sistema fa crollare a cascata i figli)**

**Spazio d’indirizzi**

* Il processo figlio è un duplicato del processo padre.
* Nel processo figlio si carica un programma.

**Esempi UNIX**

* la system call **fork** crea un nuovo processo.
* La system call **exec** dopo una fork sostituisce lo spazio di memoria del processo con un nuovo programma.

**Immagine che contiene orologio

Descrizione generata automaticamenteEsempio albero dei processi**

Terminazione di un processo

* **Un processo termina quando finisce l’esecuzione della sua ultima istruzione e chiede al SO di essere cancellato usando la system call EXIT.**
* il processo figlio può riportare alcuni dati al processo genitore attraverso la system call **wait.**
* Tutte le risorse del processo sono liberate dal SO.
* **Un processo genitore può porre termine all’esecuzione di uno dei suoi processi figli (system call ABORT) perché:**
* Il processo figlio ha ecceduto nell’uso di alcune risorse.
* Il compito assegnato al processo figlio non è più richiesto.
* Il processo genitore termina.

**Il SO non consente ad un processo figlio di continuare l’esecuzione in tale circostanza.**

* **TERMINAZIONE A CASCATA (CASCATING TERMINATION)**

Processi cooperanti

* Un processo è **indipendente** se non può influire su altri processi nel sistema o subirne l’influsso.
* Un processo è **cooperante** se influenza o può essere influenzato da altri processi in esecuzione nel sistema.
* **Vantaggi dei processi cooperanti**
* Condivisione d’informazioni
* Accelerazione del calcolo
* Modularità
* Convenienza

Problema produttore-consumatore

* Usuale paradigma per i processi cooperanti: un processo **produttore** produce informazioni che sono consumate da un processo **consumatore.**
* **Memoria illimitata** (unbounded-buffer): non pone limiti alla dimensione del vettore.
* **Memoria limitata** (bounded-buffer): presuppone l’esistenza di una dimensione fissata di memoria.

Ovviamente non esiste uno schema a memoria illimitata; quindi sia chi produce che chi consuma devono far riferimento ad un’area sulla quale devono depositare gli oggetti da consumare o da produrre. La domanda che si deve porre il produttore è se la memoria è libera (essendo questa limitata).

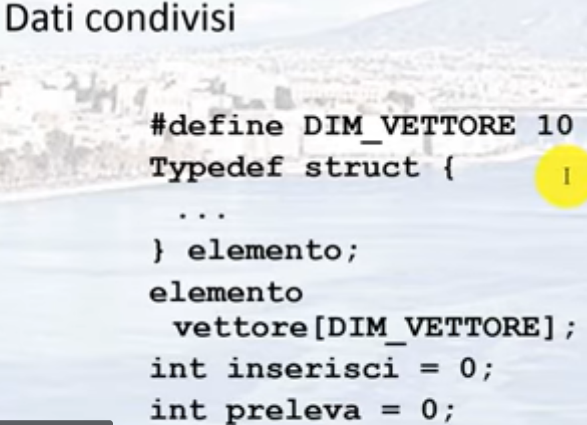
Memoria limitata-soluzione con memoria condivisa Con memoria limitata si suppone di trovare una soluzione con memoria condivisa.

Si suppone di usare una zona di **memoria condivisa** e di usare un vettore circolare di grandezza **DIM\_VETTORE** e che fa uso di 2 indici per accedere al vettore. Con l’indice **inserisci** si punta alla prima casella disponibile del vettore. Con l’indice **preleva** si punta alla prima posizione occupata del vettore. Al momento iniziale si pongono a 0 sia **inserisci** che **preleva.** Da tala definizioni si ricava che:

* **Se inserisci == preleva ALLORA il vettore è vuoto**
* **Se ((inserisci+1) % DIM\_VETTORE) == preleva ALLORA il vettore è pieno**
* **Max elementi nel vettore = DIM\_VETTORE -1**

**Immagine che contiene disegnando

Descrizione generata automaticamente**

**Spiegazione**  si definisce l’area dei dati condivisi facendo una #**define DIM\_VETTORE** di ad esempio 10 locazioni; ci sta una struct elemento con vettore DIM\_VETTORE, e ci sono due interi uguali a 0 (inserisci e preleva).

**Immagine che contiene uccello

Descrizione generata automaticamente**Processo produttore

**Spiegazione [**WHILE (1) è un modo compatto di descrivere un loop forever]. Si definisce un elemento appena\_Prodotto; dopo di che si fa il while(1), quindi si entra nel loop infinto al cui interno ci sta un altro while che dice fino a quando il vettore è pieno non fare niente ; appena questa condizione è falsa si esce dal while e nella posizione inserisci ci si mette l’elemento appena\_Prodotto e bispogna poi aggiornare il vettore inserisci. **NOTA: il while interno si deve sbloccare vuol dire che inserisci e preleva sono due variabili condivise, quindi se qualcuno da un’altra parte le modifica, queste si modificano pure qui.**

Processo consumatore

**Immagine che contiene uccello

Descrizione generata automaticamenteSpiegazione** Si definisce l’elemento da\_Consumare; dopo di che si fa il while (1), quindi si entra nel loop infinto al cui interno c’è un altro while che dice fin quando il vettore è vuoto, non fare nulla; appena questa condizione è falsa, si esce dal while e prendo l’elemento, quindi da\_Consumare = vettore[preleva] e preleva = (preleva + 1) % DIM\_VETTORE

Comunicazione tra processi (IPC)

* Attraverso le funzioni di IPC i processi possono comunicare tra loro e sincronizzare le proprie azioni.
* Un sistema di scambio di messaggi permette ai processi di comunicare tra loro senza ricorrere a dati condivisi.
* Un sistema IPC fornisce le due opzioni:
* **Send (messaggio):** dimensione fissa o variabile
* **Receive (messaggio)**
* Se i processi P e Q vogliono comunicare, devono:
* Stabilire un **canale di comunicazione** tra loro
* Scambiare messaggi mediante **send/receive**
* Realizzazione di un canale di comunicazione
* Fisica (es. memoria condivisa, bus, reti)
* Logica (es. proprietà logiche)

**Questioni implementative** Sono le domande da porsi e le problematiche che possono sorgere:

* Come sono stati creati i canali di comunicazione?
* Può un canale essere associato a più di due processi?
* Quanti canali possono esserci tra ogni coppia di processi comunicanti?
* Qual è la capacità di un canale di comunicazione?
* La dimensione di un messaggio deve essere fissa o variabile?
* Il canale di comunicazione è unidirezionale o bidirezionale?

Comunicazione diretta

* I processi devono nominarsi reciprocamente in modo esplicito:
* **Send (P, messaggio):** invia messaggio al processo P
* **Receive (Q, messaggio):** riceve, in messaggio, un messaggio dal processo Q P e Q ovviamente si devono conoscere.
* Caratteristiche di un canale di comunicazione:
* I canali vengono stabiliti automaticamente
* Un canale è associato esattamente a una coppia di processi
* Esiste esattamente un canale tra ciascuna coppia di processi
* Il canale può essere unidirezionale, ma è in genere bidirezionale

Comunicazione indiretta

* I messaggi vengono inviati a delle **porte** (dette anche mailbox) che li ricevono
* Ciascuna porta è identificata in modo univoco
* Due processi possono comunicare solo se condividono una porta
* Caratteristiche di un canale di comunicazione:
* Tra una coppia di processi si stabilisce un canale solo se entrambi i processi condividono una stessa porta
* Un canale può essere associato a più di due processi
* Ciascuna coppia di processi può condividere più canali di comunicazione
* Il canale può essere unidirezionale o bidirezionale

Il vantaggio è che, l’implementazione è più facile in quanto i processi non si devono conoscere per forza tra loro.

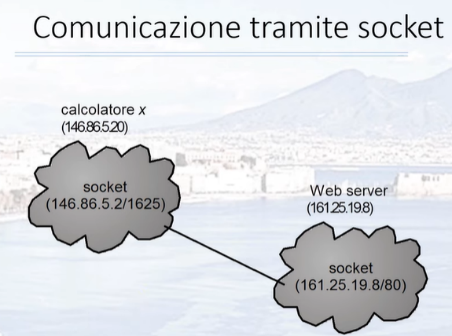
* Operazioni:
* Creare una nuova porta
* Inviare e ricevere messaggi tramite la porta
* Rimuovere una porta
* Primitive:

**send (A, messaggio):** invia messaggio alla porta A **receive (a, messaggio):** riceve, in messaggio, un messaggio dalla porta A

* Condivisione della porta:
* P1, P2, P3 condividono la porta A
* P1 invia un messaggio ad A; P2 e P3 eseguono una **receive** da A
* Quale processo riceverà il messaggio?
* Soluzioni:
* Consentire che un canale sia associato al massimo a due processi
* Consentire a un solo processo alla volta di eseguire un’operazione **receive**
* Consentire al sistema di decidere arbitrariamente quale processo riceverà il messaggio. Il sistema può comunicare l’identità del ricevente al trasmittente

Sincronizzazione Con i messaggi sincronizzati si sta a ipotizzare uno scambio d’informazioni tra due processi che comporta la copresenza dei due processi. Lo scambio può essere **sincrono (bloccante), oppure asincrono (non bloccante).** Le primitive di riferimento sono **send and receive** le quali possono essere anch’esse sincrone o asincrone. Code di messaggi (buffering) Se la comunicazione è diretta o indiretta, i messaggi scambiati tra i processi comunicanti risiedono in code temporanee. Fondamentalmente esistono 3 modi per realizzare queste code:

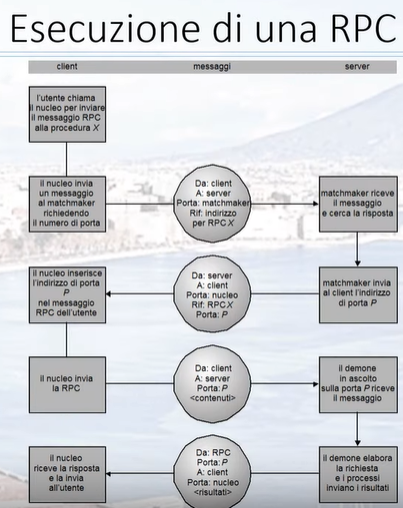
1. **Capacità zero:** 0 messaggi. Il trasmittente deve fermarsi finchè il ricevente prende in consegna il messaggio
2. **Capacità limitata:** la coda ha lunghezza finita n. Il trasmittente deve attendere se il canale è pieno
3. **Capacità illimitata:** la coda ha lunghezza potenzialmente infinita. Il trasmittente non si ferma mai

Comunicazione client-server

* Socket

Immagine che contiene bottiglia, sedendo, carta, tavolo

Descrizione generata automaticamente

* Immagine che contiene screenshot

  Descrizione generata automaticamenteChiamate di procedure remote, RPC
* Invocazione di metodi, RMI (java)

Immagine che contiene screenshot

Descrizione generata automaticamenteImmagine che contiene testo

Descrizione generata automaticamente