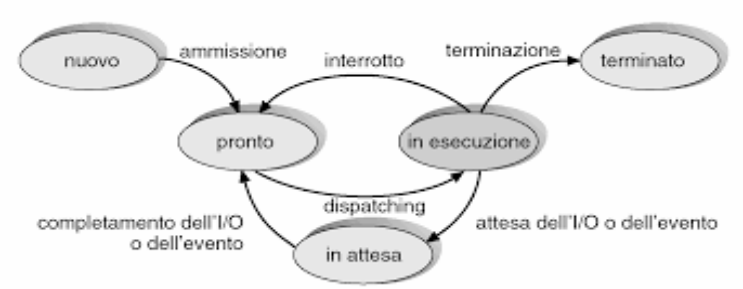
Autore : Vendrame Mario Tobia

DOMANDE PER LA TEORIA DI SISTEMI OPERATIVI(doc non completo) :

1) Descrivere l'evoluzione di un processo tramite diagramma a stati con esempio.



Un processo viene svolto in un determinato tempo (tempo finale – tempo iniziale), esso dipende da vari fattori (CPU, agenti interni/esterni ecc..).

Il processo è una funzione che prendi in input dei dati iniziali e da in output dei risultati, generalmente un processo ha anche bisogno di risorse che vengono ottenute tramite comunicazione con altre entità (Es SO).

**********AGGIUNGERE ESEMPIO **********

2a) Definire le condizioni necessarie per lo stallo

Il deadlock si raggiunge quando il sistema non può raggiungere il suo stato finale, le condizioni necessarie sono :

- Mutua esclusione : Le risorse coinvolte non sono condivisibili
- Allocazione parziale : Un processo non richiede tutte le risorse necessarie in un unico step, ma in diversi momenti della sua evoluzione.
- Non sottraibilità: solamente il processo che sta usando la risorsa è in grado di rilasciarla
- Attesa circolare: I processi sono in attesa di risorse occupate da altri processi e a loro volta in attesa di risorse occupate da altri.

2b) Risoluzione dello stallo grazia all'algoritmo del banchiere

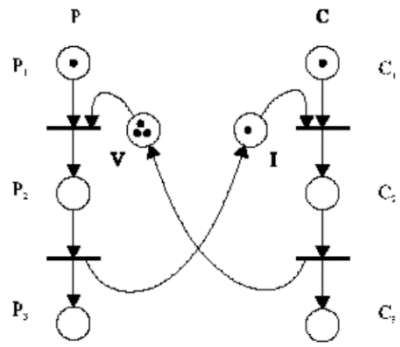
Con l'algo del banchiere le risorse vengono concesse ad un processo solo se le risorse residue sono sufficienti alla terminazione del processo → rimozione dell'attesa circolare

3)Allocazione globale definizione vantaggi e svantaggi

L'allocazione globale prevede di allocare le risorse a priori, in modo tale che I processi ne usufruiscano da subito. Questo genere di allocazione rimuove la condizione di allocazione parziale che porta I processi in stallo, ma è applicabile solo quando sono note le risorse utilizzate dai processi durante la loro evoluzione, altrimenti non è applicabile.

++++++++++++INSERIRE ESEMPIO DI USO CONTEMPORANEO DI DUE RISORSE NON CONDIVISIBILI

4)Rete di petri produttore consumatore



Esso non va mai in stallo perchè un consumatore può prendere la risorsa solo è disponibile (ossia il produttore la ha prodotta), altrimenti deve aspettare. La risorsa acquisita è solo del consumatore e di nessun altro.

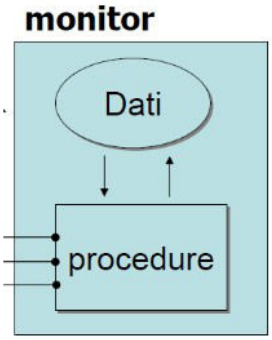
5) Descrizione semaforo

I semafori sono delle variabili booleane che permettono di mettere in esclusione mutua → risolvendo in questo modo l'utilizzo contemporaneo di una stessa risorsa.

Le sue primitive sono WAIT() o lock() e SIGNAL() o unlock(), wait fa attendere il processo, signal avvisa del libermento della risorsa.

Per risolvere il problema della richiesta di utilizzo simultaneo di una risorsa si mettono in mutua esclusione anche I semafori con semafori privati.

6) Descrizione monitor



Un monitor è un oggetto che permette di mettere in mutua esclusione sia I dati che le procedure, ogni processo una volta entrato nel monitor protrà agire solo su queste procedure. Una volta che il processo acquisisce il monitor detiene il lock, se esso non completa la procedura puo essere messo in coda all'interno del monitor(si sospende con una wait).

Ci sono più tipi di monitor :

-Hoare

-Brich Hansen

7)Costrutto di monitor e differenza tra monitor e regione critica

Monitor mio_monitor{

//dati locali

//costruttore

Entry metodo1()

Entry metodoN()

}

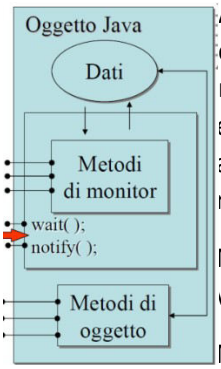
Monitor Hansen: Nel monitor di hansen la signal() è l'ultima istruzione in modo tale da liberare altri processi sospesi in coda, se la coda è vuota → viene abilitato l'accesso al monitor fuori dalla coda.

Monitor di hoare: La funzione signal() può essere richiamata più volte all'interno della stessa procedura, in questo monitor la priorità è data al processo risvegliato. Il processo risvegliante attende in una coda chiamata urgent che il processo risvegliato completi l'esecuzione all'interno del monitor.

Le regioni critiche sono regioni di codice con esclusione mutua, esse a differenza dei monitor non risolvono I problemi di sync dei processi.

8) Spiegare il ruolo del semaforo urgent in hoare

9)Monitor di java, notify() e notifyAll(), differenze tra monitor java ,hoare, regione critica



Ad ogni oggetto viene associato un monitor, ovvero una regione critica condizionale con 2 code, 2 per I thread che devono entrare,e una per I thread che non hanno trovato la condizione verificata. La JVM associa ad ogni oggetto un lock, e ogni processo per poter accedere al monitor deve possederlo, se lo possiede può accedere e utilizzarlo, all'interno il processo può eseguire la wait su se stesso → rilasciando il lock e inserendosi in coda wait set diventando not runnable.

Notify() libera un thread dalla cosa wait set, esso prima di diventare runnable deve valutare la condizione, se essa non è verificata torna in wait set

NotifyAll() estrae dal wait set e rende runnable tuttti I thread in attesa

Il motive per cui esiste la notifyAll() → perchè puo acccadere che un processo in testa alla wait list non soddisfi mai la condizione e per questo motivo mandi in starvation tutti, anche quelli che potrebbero soddisfarla.

Il monitor di java è un vero e proprio oggetto gestito dalla JVM che gestisce interamente lock – unlock , la regione critica invece è una zona di codice in mutua esclusione che non risolve il problema della sincro fra processi.

Monitor di java processo gestito dalla JVM , monitor di hoare ha una coda urgent nella quale ci sono I processi risveglianti.

10) Parametri di scheduling RR vs PS

Parametri : Utilizzo CPU(minimizzare periodo attività),Throughput(processi su unità di tempo), Turnaround time (minimizzare tempo di completamento), Reaction Time, Deadline(tempo massimo entro il quale il processo deve iniziare o terminare), Predicibilità, Equità, Bilanciamento delle risorse

RR (Round Robin)

Esso è preemptive, la sua ready list è FIFO, Utilizza Time slice → quanto temporale, allo scadere di esso il processo running viene messo in fondo alla ready list. Esso è un algo preemptive, con overhead minimo, buon throughput, trattamento equo a tutti I proessi

PS (Priority scheduling)

Ad ogni processo è associato un indicatore di priorità, ready list è ordinata per priorità decrescente(N liste, una per ogni livello di priorità). L'algoritrmo può essere preempritve, o non preemptive. L'overheaad può essere alto. Il rischio di starvation (I processi con priorità più bassa

possono rimanere in attesa infinita se ci sono sempre processi con priorità più alta) è eliminato adottando la tecnica di aging (aumenta la priorità col tempo di attesa)

11)Esempio priority inversion

- Un processo a bassa priorità esegue il lock() di una risorsa, poco dopo un processo a alta priorità cercherà di fare altrettanto , senza però riuscirci, esso si mette in coda per aspettare l'unlock, prima che il processo a bassa priorità sblocchi la risorda un processo intermendio, che vuole utilizzare una risorsa diversa da quella del processo in bassa priorità esegue un preemption sul processo a bassa priorità, mettendolo in coda e indirettamente sorpassando anche quello di alta priorità che stava attendendo.

-Di conseguenza processi con priorità inferiore bloccano quello con priorità più elevata

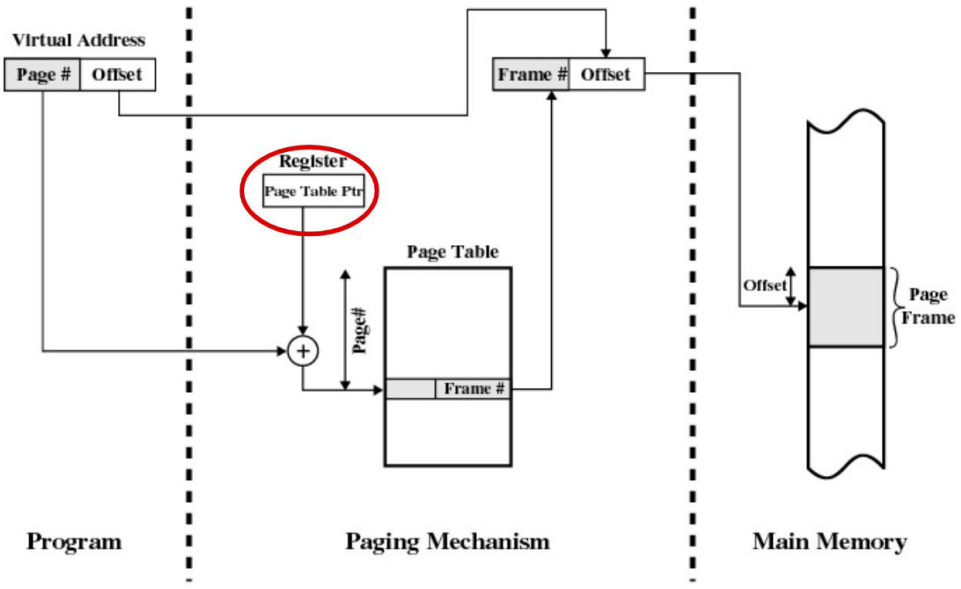
Per risolvere questo problema, il processo che impiega il semaforo eredita temporaneamente la priorità più alta tra quelle dei processi sullo stesso semaforo. (aumenta l'overhead)

12)Descrizione paginazione

La memoria fisica viene divisa in blocchi uguali, di dimensioni spesso uguali al settore fisico del disco.

Il SO mantiene una page table per ciascu processo, quando un processo è caricato in memoria essa indica in quale frame si trova ciascuna sua pagina (I frame possono non essere contigui).

Nella page table , l'elemento di indice I corrisponde alla i-esima pagina del processo e contiene : l'indice del frame nella mem fisica in cui è allocata l'iesima pagina del processo, alcuni bit di controllo P = pagina presente nella mem fisica, M = pagina modificata durante la permanenza in mem fisica, bit di protezione



Translation logical (virtual) address -> physical address

13)Segmentazione

Ogni indirizzo è costituito da un indice di segmento e un offset. Ciascun processo è costituito da un insieme di segmenti di dimensioni diverse e da una segment table contenente per ciascuno dei suoi segmenti :

- Indirizzo iniziale

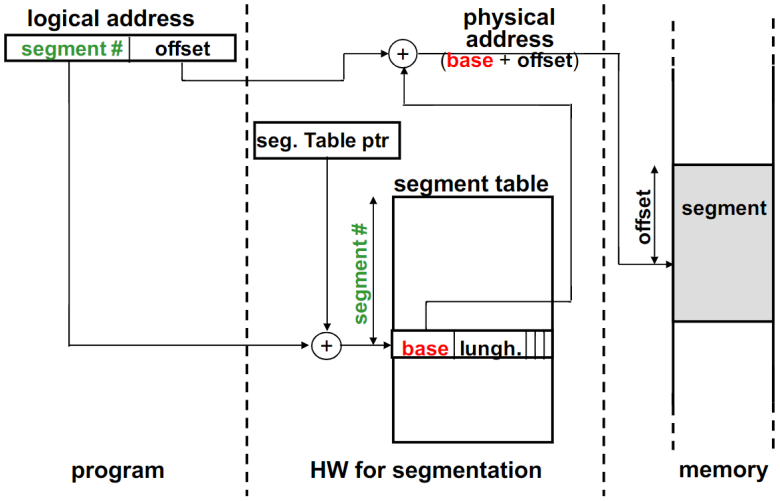
- Dimensione del segmento

- Bit controllo e bit di protezione

Indirizzo virtuale to indirizzo fisico

L'indirizzo virtuale definito dal codice è composto da un indice (#seg) e un offset (Off), l'indice indica un elemento della segment table da cui si estrae : indirizzo iniziale del segmento (base) e la dimensione del segmento (Dim). Se off>dim → indirizzo non valido altrimenti valido.

L'indirizzo fisico è dato da base+off, per realizzare un sistema di memoria virtuale(MV) è necessario che nell'HW sia presente un meccaniscmo di paginazione o segmentazione o entrambi



14) Strategia del working set

Esso è l'insieme di pagine usate effettivamente dal processo, è costituito da W pagine usate nelle ultime delta unità di tempo in cui un processo è stato in esecuzione.

Si tratta di un metodo per far in modo che il resident set di un processo coincida con il suo working set, infatti il SO rimuove periodicamente le pagine che non sono più nel working set, il processo in questione può diventare running solo se tutto il suo WS è in memoria.

Utilizzare questa strategia comporta overhead (valore ottimo di delta non è noto a priori), per questo ci sono alcune approssimazioni, quali : Page-fault frequency : al verificarsi di un page fault si valuta il tempo trascorso dal page fault precedente, se troppo piccolo si aggiunge un frame al WS altrimenti se troppo grande si rimuove con use bit=0 , oppure si utilizza WS a intervallo variabile , ad ogni intervallo di tempo gli use bit del resident set sono azzerati, ad ogni page fault il resident set cresce, al termine dell'intervallo ri rimuovono le pagine con use=0, la durata dell'intervallo dipende dal numero di page fault.

15) Ruolo device handler

Struttura device handler :

while(true){

wait(RA) // wait for signal from DOIO

wait(OC) // wait on OC semaphore

signal(RS) //suspended process becomes ready

}

Il device handler è un processo di sistema che gestice l'I/O del particolare dispositivo in concomitanza con la routine di servizio delle interruzioni ovvero l' ISR del dispositivo, per ottimizzare il tempo totale

Esso effettua le operazioni più complesse, quali transcodifica, controlli ecc.. specifici per il dispositivo, costituisce la parte più importante del driver

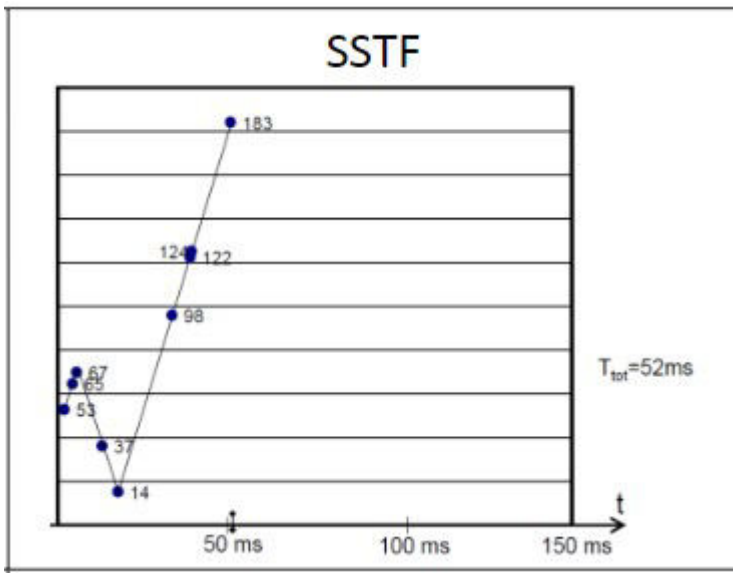
16) Raid 2

Usa un Error Correcting code di N bit (codice di hamming) per rilevare/correggere gli errori negli stripe registrati in posizioni corrispondenti, servono N dischi in più per contenere questo codice. Consente la correzzioe di un singolo bit errato e il rilevamento di errori su più bit. Non è più usato nei sistemi commerciali.

17) SSTF

In questo metodo di schedulazione degli accessi al disco viene scelta ogni volta come richiesta successiva da evadere quella situata sulla traccia più vicina alla posizione attuale della testina.

Tempo complessivo più breve del FIFO, però se continuano a pervenire richieste per tracce vicine alla attuale può provocare starvation delle richieste più lontane.



18) Descrivere il C-LOOK

L'ordine di evasione delle richieste è lo stesso dello scan(le richieste vengono evase secondo l'ordine con cui vengono incontrate dalla testina, che si sposta nella stessa direzione da una estremità all'altra del disco), ma la testina inverte la direzione quando ha raggiunto l'ultima richiesta in quella direzione. Stesse caratteristiche dello scan e dello cscan ma più efficienti, evita il rischio di starvation ed ha una buona efficienza, favorisce però le richieste che si trovano alle estremità e anche quelle arrivate per ultime.

19) Confronto SCAN e C-SCAN

SCAN :Le richieste vengono evase secondo l'ordine con cui vengono incontrate dalla testina che si sposta nella stessa direzione da una estremità all'altra del disco e inverta la direzione solo quando ha raggiunto l'estremità

C-SCAN : Le richieste vengono evase solo mentre la testina sta procedendo in una direzione (il ritorna all'altra estremità avviene senza accessi al disco), ma è anche meno efficiente.

Entrambi evitano il rischio di starvation e hanno una buona efficienza.

20) Ruolo i-node

L'i-node è l'index node una struttura dati sul file system che contiene le informazioni per la gestione del file (64 bytes)(ad ogni file è associato un i-node) esso gestisce diversi parametri, tra I quali : mode,link count,owner id, group id, file size,last accessed, last modified, i-node modified e 13 elementi dell'indice di allocazione.

21) Gestione blocchi liberi

Per gestire I blocchi liberi il SO deve conoscere quali sono liberi, grazie alla DAT, ci sono diverse tecniche : Bit table (Ogni blocco ha ha un bit 0 o 1 in base all'occupazione, questo genere di tecnica occupa uno spazio ridotto) Linked list : Gruppi contigui di blocchi liberi (#di blocchi liberi e link al prossimo gruppo, questa tecnica non occupa spazio su disco), index table, free block list.

Per la gestione dei blocchi liberi dunque servono due tabelle : DAT e FAT, esse possono essere inserite o in memoria o sul disco (letura lenta, molti accessi)

22)Tecnica dello spooling

Un processo che richieda l'uso di un dispositivo non condivisibile deve attendere che si liberi, se esso è in uso. Per evitare queste attese il SO utilizza una tecnica di spooling (simultaneous peripheral operation on line) che consiste nell'assegnare ad ogni processo che lo richieda un dispositivo virtuale realizzato da un file sul disco.

- Ciascun processo ottiene senza attesa l'accesso al file che rappresenta il dispositivo
- Le richieste dei processi di trasferire dati al dispositivo vengono trasformate in trasferimenti sul file assegnato
- I/O più rapido (dischi veloci)
- Quando il processo chiude il proprio dispositivo virtuale il suo file è inviato a una coda per il dispositivo fisico
- La coda dei file si mantiene tra attivazioni di sistema

23) Effetti della duplicazione durante la chiamata fork()

La primitiva fork() crea un nuovo processo e adotta un modello a clonazione, ossia alloca spazio per le regioni data e stack del processo figlio e vi copia l'attuale contenuto delle regioni del padre, opzionalmente lo può fare anche per lea regione text altrimenti essa è condivisa. Alla funzione invocante (padre) viene ritornato 0 se è andato tutto secondo I piani altrimenti -1 → fallimento.

24) Immagine e contesto di un processo UNIX

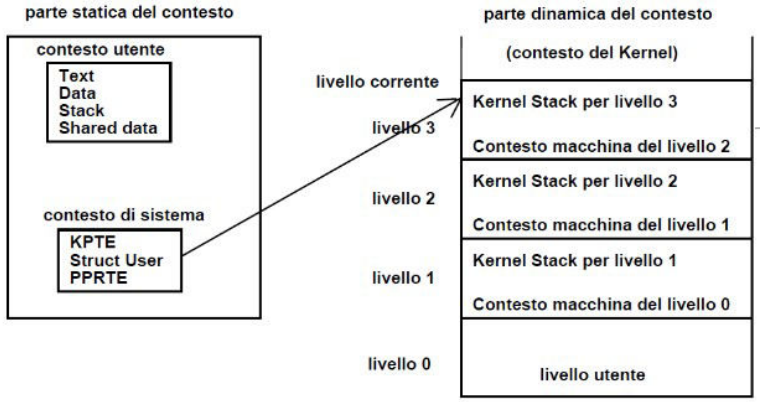
Ogni processo (figlio) è creato da un processo (padre), cio crea una gerarchia che collega padri e figli, a ogni processo sono associati 7 identificatori:

- PID (process ID) assegnato alla creazione che lo identifica univocamente nel sistema
- PID del padre
- User ID reale (utente che esegue il processo)
- User ID effettivo
- Group ID reale (gruppo utente che esegue il processo)
- Group ID effettivo
- Process group ID (gruppo di processi che condividono lo stesso contesto di standard I/O)

Il descrittore di processo è diviso in 2 parti : La prima risiede sempre in memoria centrale e contiene stato,priorità,utilizzo tempo di CPU ecc.. e la seconda può stare anche in mem secondaria essa contiene altri ID puntatore alla prima struttura , inode directory corrente, directory root,UFDT ecc.

Contesto di processo : Il processo può operare in modo utente o in modo kernel. Gli ingressi al kernel sono 3 : chiamata di sistema, interrupt, eccezzione. In modo utente utilizza lo user stack allocato nella regione di stack, in modo kernel utilizza lo stack kernel. Quando si trova in modalità kernel e il SO stabilisce un cambio di processo, viene salvato il contesto del processo corrente e

ripreso quello nuovo da eseguire. Il contesto è dato da: Contesto utente → regioni del processo, contesto macchina → registri macchina, contesto di sistema → strutture user e proc, kernel stack



25)Rate monotonic scheduling

Alcune applicazioni Real-time sono costituite da processi periodici dei quali si conoscono le caratteristiche : R (Ready Time ossia istante in cui è pronto per iniziare) S (Starting deadline) C (completition deadline) E (tempo di esecuzione) T (periodo di attivazione)

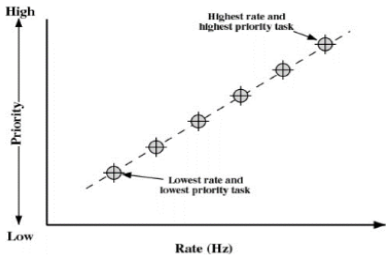
Per task periodici, assegna la priorità sulla base del valore del periodo, le priorità più elevate sono assegante ai task di periodo inferiore (priorità cresce con il crescere della frequenza). Bisogna verificare se e con quale margine tutti I deadline siano soddisfatti. In generale per qualsiasi algo di schedulazipne di n task periodici, la condizione necessaria perchè I deadline siano soddisfatti è la seguente

Σi Ci / Ti ≤ 1

Ci = tempo di esecuzione in ciascun periodo del task i

Ti = periodo del task i

Ci / Ti = percentuale di utilizzazione della CPU



Il valore 1 corrisponde alla piena utilizzazione della CPU nel caso di un algoritmo di schedulazione idealmente perfetto (si trascurano gli overhead di sistema); la relazione limita il numero di task schedulabili

Nel caso **RMS** la **condizione** da soddisfare è più stringente all'aumentare di n: **Σi Ci / Ti ≤ n ($2^{1/n} - 1$)**

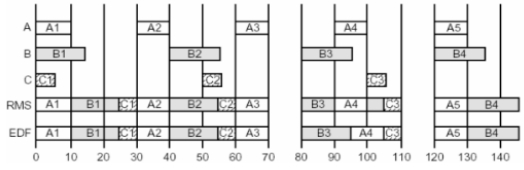
Il termine a destra tende a ln 2=0.693 per n→∞

Confronto tra RMS e EDS

TA=30ms TB=40ms TC=50ms → prioA=33 prioB=25 prioC=20

CA=10ms CB=15ms CC=5ms (execution time)

Σi Ci / Ti = 1/3 + 3/8 + 1/10 = 0.808



TA=30ms TB=40ms TC=50ms → prioA=33 prioB=25 prioC=20

CA=**15ms** CB=15ms CC=5ms (execution time)

Σi Ci / Ti = 1/2 + 3/8 + 1/10 = 0.975 3(21/3 – 1) = 0.779