Sistemi operativi 2/ed
Paolo Ancilotti, Maurelio Boari, Anna Ciampolini, Giuseppe Lipari

SOLUZIONI DEI PROBLEMI DEL CAPITOLO 2

1. Soluzione:

Trasferimento tra l'ambiente dei processi e l'ambiente del nucleo:

Poiché le funzioni del nucleo girano in stato privilegiato mentre il codice dei processi gira in stato non privilegiato, come è stato messo in evidenza nel testo, il trasferimento del controllo tra l'ambiente dei processi e l'ambiente del nucleo può avvenire soltanto in seguito ad un'interruzione, sia essa un'interruzione asincrona (o esterna) o un'interruzione sincrona, corrispondente all'esecuzione di un'istruzione tipo "*chiamata a supervisore*" (ad esempio INT o SVC a seconda delle macchine).

In seguito all'accettazione dell'interruzione da parte della CPU, vengono inseriti, via hardware, i valori dei registri PS (Program Status Word) e PC (Program Counter) in cima allo stack del processo che era in esecuzione all'arrivo dell'interruzione stessa. Al posto di tali valori, nei registri di macchina PC e PS vengono caricati, sempre via hardware, i valori corrispondenti rispettivamente all'indirizzo della prima istruzione della procedura di risposta alle interruzioni e alla parola di stato relativa all'ambiente di nucleo (stato privilegiato). Come conseguenza di tali operazioni eseguite via hardware, inizia l'esecuzione della procedura di risposta alle interruzioni. Avendo disponibile un solo insieme di registri generali, al fine di evitare di cancellare i valori presenti nei registri al momento in cui l'interruzione è stata accettata, è necessario salvarli per poterli poi ripristinare. Per prima cosa viene quindi eseguita la funzione di *salvataggio dello stato* del processo interrotto che ha il compito di trasferire i valori di tutti i registri generali dai registri di macchina al campo *contesto* del descrittore del processo stesso. Successivamente, anche i valori di PC e PS, precedentemente inseriti nello stack, vengono salvati nel descrittore del processo. A questo punto l'intero stato del processore virtuale corrispondente al processo interrotto è presente nel campo *contesto* del suo descrittore e tutti i registri di macchina possono essere utilizzati dalla procedura di risposta alle interruzioni. Tale procedura, una volta terminata la gestione dell'interruzione, deve restituire il controllo all'ambiente processi. E ciò può avvenire o riprendendo l'esecuzione del processo interrotto a partire dall'istruzione successiva all'ultima eseguita, se la gestione dell'interruzione non prevede nessuna commutazione di contesto, oppure mandando in esecuzione un diverso processo se, a causa della gestione dell'interruzione, si e reso necessario commutare il contesto dal processo interrotto a un diverso processo da mandare in esecuzione.

Trasferimento tra l'ambiente dei processi e l'ambiente del nucleo:

In entrambi i due casi precedenti, il trasferimento del controllo dall'ambiente di nucleo a quello dei processi avviene con lo stesso meccanismo che, praticamente, corrisponde al duale del precedente. Infatti, noto il processo a cui deve essere ceduto il controllo (o quello che ha subito interruzione o quello che viene prescelto per essere eseguito in caso di cambio di contesto) vengono prelevati dal campo *contesto* del suo descrittore tutti i valori da ricaricare nei registri generali di macchina. Sempre dal campo contesto vengono prelevati i valori dei registri PS e PC del processore virtuale del processo e tali valori vengono inseriti in cima allo stack. La procedura termina quindi eseguendo l'istruzione IRET (ritorno da interruzione) che, via hardware carica nei registri di machina PS e PC i valori presenti in cima allo stack. In questo modo riparte l'esecuzione del processo schedulato a partire dall'istruzione successiva all'ultima eseguita, con la parola di stato corrispondente allo stato del processore non privilegiato e trovando nei registri generali di macchina gli stessi valori che erano presenti nel momento in cui il processo ha precedentemente perso il controllo.

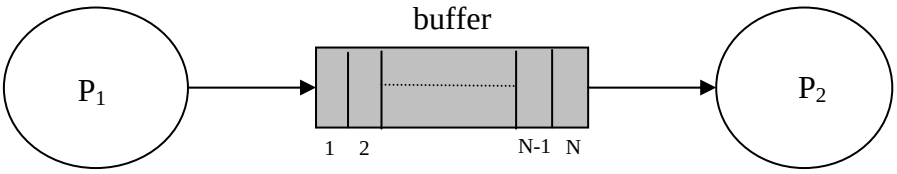
2. Soluzione:

Essendo in esecuzione il processo P2, la cui priorità è 2, cioè la più bassa fra le tre priorità previste, ciò significa che tutti i processi con priorità superiore (cioè quelli con priorità 0 oppure 1) sono bloccati. Inoltre essendo P2, fra quelli con priorità 2, il primo processo in coda, ciò significa che il successivo processo nella coda di priorità 2, e cioè P4, è pronto in attesa di ricevere il suo quanto di tempo. Per cui gli stati dei processi sono i seguenti:

P1 sbloccato
P2 in esecuzione
P3 bloccato
P4 pronto
P5 bloccato
P6 bloccato

3. Soluzione:

Nel paragrafo 2.7, con riferimento alla figura 2.8, dove il buffer ha la capacità di un solo messaggio, e nell'ipotesi che il buffer sia inizialmente vuoto, come è stato mostrato, la sincronizzazione fra le operazione di *inserimento* del messaggio nel buffer da parte del processo produttore e l'operazione di *prelievo* da parte del processo consumatore, deve necessariamente garantire che tali operazioni siano eseguite nell'unica sequenza: *inserimento-prelievo-inserimento-prelievo.....* Ogni altra sequenza porterebbe a delle interferenze tra i due processi. Nel caso in cui fossero eseguiti due inserimenti consecutivi (*inserimento-inserimento-prelievo..)* il secondo inserimento porterebbe alla cancellazione del messaggio già depositato ma non ancora prelevato (condizione di *overflow*). Analogamente, nell'ipotesi di due prelievi consecutivi (*inserimento-prelievo-prelievo...*) verrebbe prelevato consecutivamente lo stesso messaggio (condizione di *underflow*).
Riportiamo adesso la figura 2.8 citata nel testo, ma con la differenza, rispetto a quanto indicato nel paragrafo 2.7, che la capacità del buffer sia di N messaggi.



Adesso il produttore può inviare fino a N messaggi consecutivi prima che si verifichi una condizione di *overflow*. Analogamente, se ad un certo istante le N posizioni del buffer contengono tutte dei messaggi, il consumatore può prelevare fino a N messaggi consecutivamente prima che si verifichi una condizione di *underflow*. Per questi motivi, i vincoli di sincronizzazione tra le due operazioni di *inserimento* e di *prelievo* devono adesso consentire molte possibili sequenze in più rispetto al caso riportato nel testo. Per garantire ancora che la cooperazione tra i due processi sia corretta, e cioè che non produca condizioni né di *overflow* nè di *underflow*, possiamo definire nel seguente modo in nuovi vincoli di sincronizzazione tra le due operazioni:

- l'i_{esimo} *prelievo* deve necessariamente essere preceduto dall' i_{esimo} inserimento. Infatti, per evitare di prelevare più volte lo stesso messaggio (condizione di *underflow*) il numero di messaggi prelevati non deve mai superare il numero dei messaggi inviati;

- l' i_{esimo} prelievo deve necessariamente precedere l'$(i+N)_{esimo}$ inserimento. Infatti, per evitare di sovrascrivere messaggi già inviati ma non ancora ricevuti (condizione di *underflow*) non deve mai avvenire che più di N messaggi siano già stati inviati ma non ancora ricevuti.

4. Soluzione:

L'ipotesi prevista dal problema è quella in cui il processo P1 sia in esecuzione, che il processo P2 sia in stato pronto e, in particolare, fra tutti i processi pronti sia quello che verrà scelto dall'algoritmo di *short term scheduling* per andare in esecuzione quando P1 perderà il controllo della CPU. Inoltre, è previsto dal testo del problema che il processo P1, in esecuzione, venga interrotto dal timer in seguito allo scadere del suo quanto di tempo. Ciò implica che l'algoritmo di *short term scheduling* sia di tipo preemptive come, ad esempio nel caso dell'algoritmo *Round-Robin*. Quindi, con riferimento alla figura 2.3 (o alla figura 2.4) relativa agli stati di un processo e alle relative transizioni di stato, all'arrivo dell'interruzione di timer deve essere eseguita la transizione di stato relativa alla revoca della CPU al processo P1 e al suo inserimento nella coda dei processi pronti. Possiamo quindi descrivere nel seguente modo le operazioni relative al cambio di contesto fra i due processi P1 e P2:

- il contesto del processo P1 viene salvato nel proprio descrittore. In particolare, ciò significa che tutte le informazioni contenute nei registri di macchina vengono trasferite nell'area *contesto* del descrittore del processo P1;
- il descrittore del processo P1 viene inserito nella coda dei processi pronti. In particolare, con l'algoritmo di *short term scheduling* di tipo *Roun-Robin*, tale inserimento viene effettuato in modo tale che P2 sia l'ultimo della coda;
- l'algoritmo di scheduling seleziona il primo processo presente nella coda dei processi pronti: in base all'ipotesi previste dal problema, tale processo corrisponde a P2;
- il contesto di P2 viene caricato nei registri di macchina. In particolare tutti i valori presenti nel campo *contesto* del descrittore del processo P2 vengono trasferiti nei registri di macchina. In seguito a questa operazione il processo P2 riprende la sua esecuzione a partire dall'istruzione successiva a quella in cui aveva precedentemente perso il controllo o, nell'ipotesi di prima attivazione, a partire dalla sua prima istruzione.

5. Soluzione:

Il problema prevede che un processo (indichiamolo genericamente con Q) sia in attesa dell'arrivo del segnale d'interruzione. Inoltre prevede che, a causa dell'arrivo del segnale d'interruzione non avvenga un cambio di contesto, cioè che il processo in esecuzione (indichiamolo genericamente con P) non perda il controllo a favore del processo Q.

Nel descrivere la soluzione del problema faremo la stessa ipotesi prevista nel testo che l'architettura di macchina preveda due set di registri generali (R_1, R_2,… R_n e R_1', R_2',…R_n') e due registri *stack pointer* (SP ed SP'), associati rispettivamente all'ambiente dei processi e all'ambiente di nucleo (nel caso di un solo set dei registri si veda la risposa al problema N.1):

a) in seguito all'accettazione da parte della CPU del segnale d'interruzione, il processore passa in stato *supervisore*; i valori dei registri PC (*Program Counter*) e PS (*Program Status Word*) contenenti i valori del contatore delle istruzioni e della parola di stato, relativi al processo P che ha subito interruzione, vengono salvati in cima allo stack del nucleo. Al loro posto, nei registri di macchina PC e PS vengono caricati, via hardware, i valori corrispondenti, rispettivamente, all'indirizzo della prima istruzione della procedura di risposta alle interruzioni e alla parola di stato relativa all'ambiente di nucleo. Passa quindi in esecuzione

la procedura di risposta all'interruzione che opera utilizzando l'insieme dei registri generali dell'ambiente di nucleo (R_1', R_2',…R_n'). In base alle ipotesi previste dal problema, tale procedura, verificata l'esistenza del processo Q bloccato in attesa del segnale d'interruzione, deve eseguire l'operazione relativa alla transizione di stato di Q da bloccato a pronto;

b) tale operazione consiste in due azioni consecutive: la prima relativa all'estrazione del descrittore del processo Q dalla coda dei processi bloccati nella quale si trova; la seconda nell'inserimento del descrittore di Q nella coda dei processi pronti. A questo punto viene verificato, in base ai criteri previsti dall'algoritmo di scheduling, se deve riprendere l'esecuzione del processo P interrotto oppure se, a causa del fatto che adesso anche il processo Q è pronto per l'esecuzione, non sia necessario mandare Q in esecuzione revocando la CPU al processo P. L'ipotesi prevista dal problema corrisponde alla prima di queste due eventualità;

c) con queste ipotesi, non è quindi necessario effettuare un cambio di contesto e quindi la proceduta di risposta all'interruzione deve terminare restituendo il controllo al processo interrotto P. Per tornare all'ambiente utente è sufficiente che la procedura di risposta alle interruzioni termini eseguendo l'istruzione IRET che, trasferendo via hardware i valori presenti in cima allo stack del nucleo nei registri di macchina PC e PS, rimette in esecuzione il processo P il quale può riprendere la sua esecuzione a partire dall'istruzione successiva all'ultima eseguita prima dell'interruzione. Non essendoci stata commutazione di contesto, il processo P ritrova nei registri generali R_1, R_2,… R_n i valori in essi presenti al momento dell'interruzione. Infatti, la funzione di risposta alle interruzioni non li ha modificati avendo usato durante la sua esecuzione i registri di nucleo R_1', R_2',…R_n'.

Per generalità, riportiamo, di seguito, anche la soluzione allo stesso problema, ma nell'ipotesi che al punto c) del testo fosse prevista la commutazione di contesto fra il processo P e il processo Q, e cioè che il precedente punto c) del problema fosse formulato nel seguente modo

c) al ritorno all'ambiente utente nell'ipotesi che sia previsto, come conseguenza dell'interruzione, cambio di contesto tra i processi.

Soluzione:

La soluzione del problema in questo secondo caso non è molto diversa dalla precedente. In particolare resta invariata la parte a) della soluzione. Anche la parte b) non cambia se non nel fatto che, una volta terminata l'operazione di attivazione di Q mediante l'inserimento del suo descrittore nella coda dei processi pronti, l'algoritmo di scheduling prevede che la procedura di risposta alle interruzioni prosegua con un cambio di contesto revocando la CPU a P e mandando Q in esecuzione.

c) Ciò che cambia rispetto alla soluzione precedente è proprio questo ultimo punto c). Nella nuova ipotesi, è quindi necessario effettuare un cambio di contesto. Tutti i valori dei registri generali (R_1, R_2,… R_n) vengono salvati nel campo contesto del descrittore di P. In tale campo vengono anche trasferiti i valori relativi al contatore delle istruzioni e alla parola di stato propri del processo P, prelevandoli dalla cima allo stack del nucleo dove sono stati precedentemente salvati. Il descrittore di P viene quindi inserito nella coda dei processi pronti in quanto, pur perdendo il controllo della CPU a causa del cambio di contesto, non viene bloccato ma resta pronto (revoca della CPU). Nei registri generali (R_1, R_2,… R_n) vengono caricati i corrispondenti valori prelevati dal campo contesto del descrittore di Q; da tale campo vengono prelevati anche i valori relativi al contatore delle istruzioni e alla parola di stato propri del processo Q, valori che vengono inseriti in cima allo stack del nucleo. A questo punto la proceduta di risposta all'interruzione termina mediante l'istruzione IRET

che, trasferendo via hardware i valori presenti in cima allo stack del nucleo nei registri di macchina PC e PS, rimette in esecuzione il processo Q terminando quindi il cambio di contesto e riportando la CPU in ambiente utente.

6. <u>Soluzione</u>:

a)

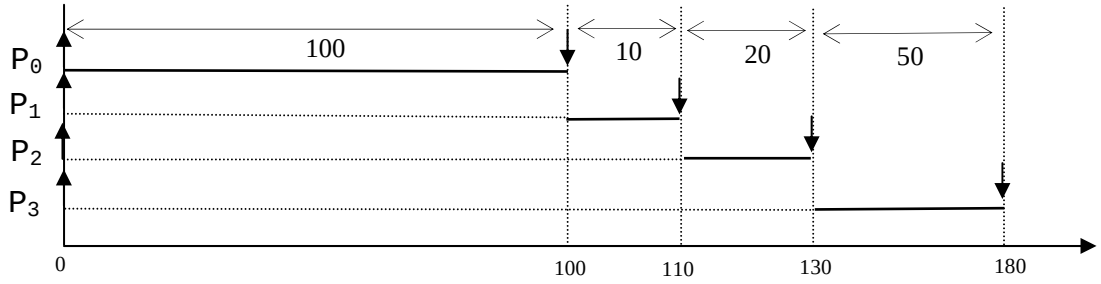


Diagramma temporale relativo all'algoritmo FCFS

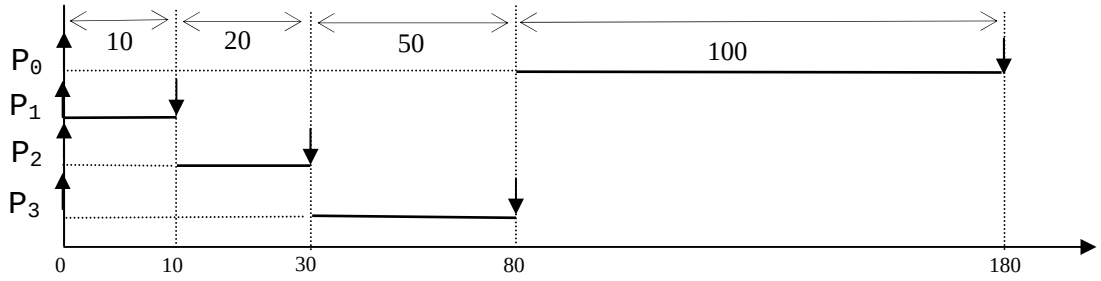


Diagramma temporale relativo all'algoritmo SJF

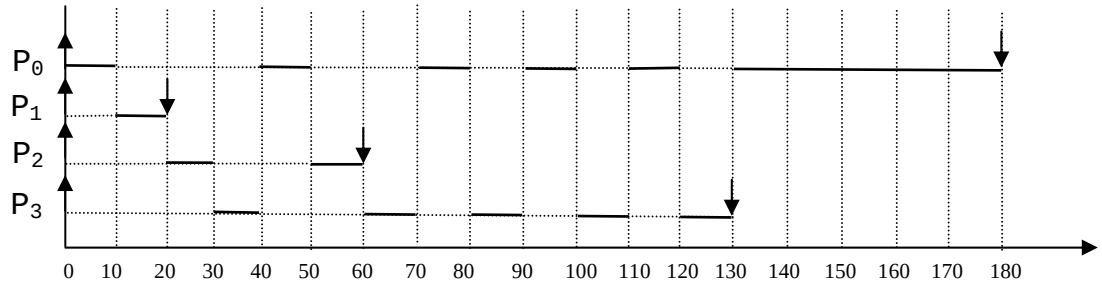


Diagramma temporale relativo all'algoritmo RR con quanto=10

b)

- Tempi di completamento dei singoli processi e tempo medio di completamento nel caso di algoritmo FCFS: T_0=100; T_1=110; T_2=130; T_3= 180

  da cui il turnaround medio: T =(100+110+130+180)/4=130

- Tempi di completamento dei singoli processi e tempo medio di completamento nel caso di algoritmo SJF: T_0=180; T_1=10; T_2=30; T_3= 80

  da cui il turnaround medio: T =(180+10+30+80)/4=75

- Tempi di completamento dei singoli processi e tempo medio di completamento nel caso di algoritmo RR: T_0=180; T_1=20; T_2=60; T_3= 130

  da cui il turnaround medio: T =(180+20+60+130)/4=97.5

c)

- Tempi di attesa dei singoli processi e tempo di attesa medio nel caso di algoritmo FCFS: A_0=0; A_1=100; A_2=110; A_3= 130

  da cui il tempo di attesa medio: A =(100+110+130)/4=85

- Tempi di attesa dei singoli processi e tempo di attesa medio nel caso di algoritmo SJF: A_0=80; A_1=0; A_2=10; A_3= 30

  da cui il tempo di attesa medio: A =(80+10+30)/4=30

- Tempi di attesa dei singoli processi e tempo di attesa medio nel caso di algoritmo RR: A_0=80; A_1=10; A_2=40; A_3= 80

  da cui il tempo di attesa medio: A =(80+10+40+80)/4=52.5

7. <u>Soluzione</u>:

a)

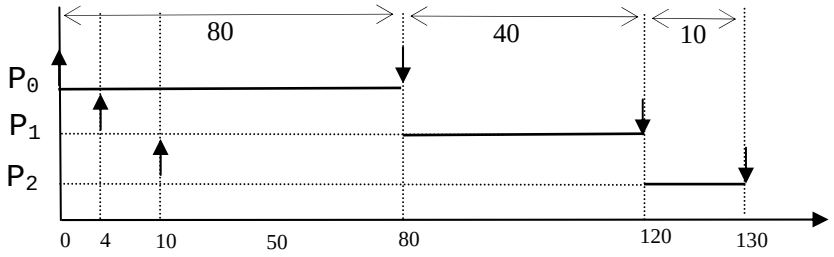


Diagramma temporale relativo all'algoritmo SJF

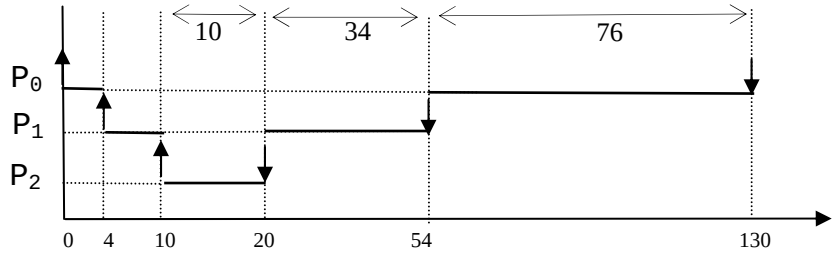


Diagramma temporale relativo all'algoritmo SRTF

b)

- Tempi di completamento dei singoli processi e tempo medio di completamento nel caso di algoritmo SJF: T_0=80; T_1=116; T_2=120;

  da cui il turnaround medio: T =(80+116+120)/3=105.3

- Tempi di completamento dei singoli processi e tempo medio di completamento nel caso di algoritmo SRTF: T_0=130; T_1=50; T_2=10;

  da cui il turnaround medio: T =(130+50+10)/3=63.3

c)

- Tempi di attesa dei singoli processi e tempo di attesa medio nel caso di algoritmo SJF: A_0=0; A_1=76; A_2=110;

  da cui il tempo di attesa medio: A =(76+110)/3=62

- Tempi di attesa dei singoli processi e tempo di attesa medio nel caso di algoritmo SRTF: A_0=50; A_1=10; A_2=0;

  da cui il tempo di attesa medio: A =(50+10)/3=20

8. <u>Soluzione</u>:

a)

$$U = C_0/T_0 + C_1/T_1 = 3/6 + 4/10 = \text{ percentuale } 0.5 + 0.4 = 0.9$$

La percentuale di idle time è pari al 10%. Il minimo comune multiplo dei due periodi è 30. In tale periodo P_0 utilizza per 5 volte 3 unità di tempo pari ad un totale di 15 unità di tempo, mentre P_1 utilizza per 3 volte 4 unità di tempo per un totale di 12 unità. La somma delle unità di tempo utilizzate complessivamente è quindi 27 su 30 e cioè pari al 90%, come indicato dal valore del fattore U di utilizzazione. Quindi la percentuale di idle time è del 10%.

b)

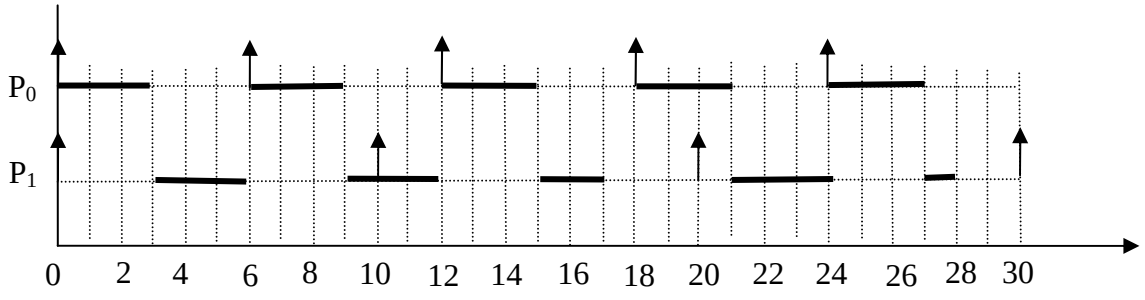


Diagramma temporale relativo all'algoritmo RM

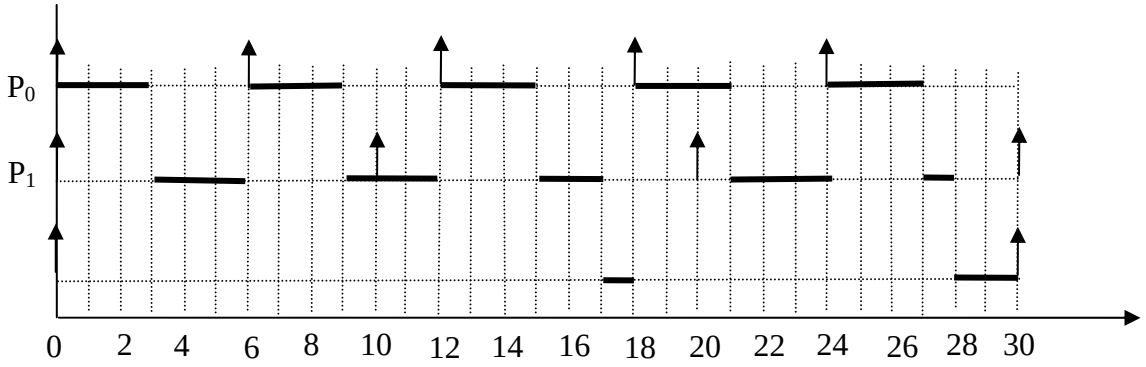
Pur essendo il fattore di utilizzazione $U > 2(2^{1/2} - 1)$ (infatti $2(2^{1/2} - 1)=0.82$) i processi risultano schedulabili cion RM.

c)

Per ottenere un fattore di utilizzazione pari a 1 il terzo processo dovrebbe utilizzare gli intervalli durante i quali la CPU è idle. Questi, come indicato dal digramma temporale, sono costituiti dagli intervalli 17-18 e 28-30. Affinché, con l'introdzione del terzo processo P_2 i processi siano ancora schedulabili, i parametri temporali di P_2 dovrebbero quindi essere C_2=3 e T_2=30: In questo caso infatti:

$$U = C_0/T_0 + C_1/T_1 + C_2/T_2 = 3/6 + 4/10 + 3/30 = 1$$

e, come evidenziato dal nuovo diagramma temporale riportato di seguito, tutti i processi rispettano i propri vincoli temporali.



d)

La risposta è no. Come si può facilmente verificare dal diagramma temporale relativo all'algoritmo RM, anche il più piccolo incremento del tempo di esecuzione di P_1 non consentirebbe a quest'ultimo di completare la sua esecuzione prima della prima scadenza e provocherebbe quindi un overflow del processo al tempo 10.

e)

Per portare il fattore di utilizzazione U a 1 aumentando il tempo di esecuzione di P_1, dovremmo incrementare C_1 in modo tale che C_1/T_1 =1 e quindi aumentare C_1 da 4 a 5. Tale incremento, come visto al punto precedente non consentirebbe la schedulabilità dei due processi con RM. Ma, come indicato nel successivo diagramma temporale, utilizzando l'algoritmo EDF i due processi sono ovviamente schedulabili essendo U=1.

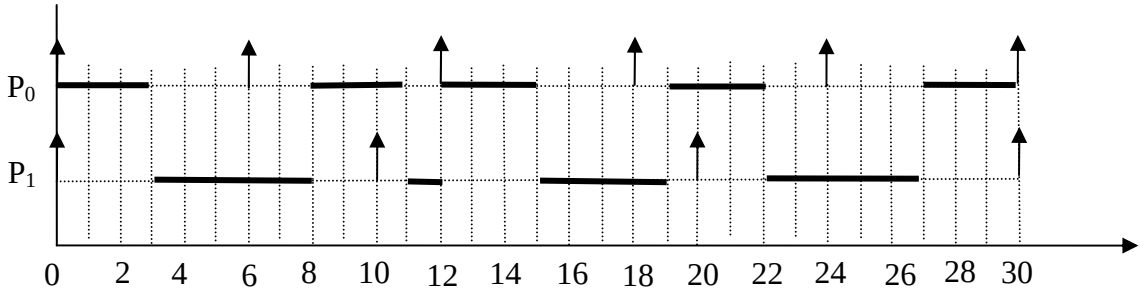


Diagramma temporale relativo all'algoritmo EDF