Caselle riservate

Ex. 1	
Ex. 2	
Ex. 3	
Ex. 4	
Ex. 5	
Ex. 6	
Tot.	

SOLUZIONE

Sistemi Operativi

Compito d'esame

12 Febbraio 2020

Matricola _____ Cognome _____ Nome _____

Docente: ◯ Quer ◯ Sterpone

Non si possono consultare testi, appunti o calcolatrici a parte i formulari distribuiti dal docente. Risolvere gli esercizi negli spazi riservati. Fogli aggiuntivi sono permessi solo quando strettamente necessari. Riportare i passaggi principali.
Durata della prova: 100 **minuti.**

1. Si supponga che il disco rigido di un piccolo sistema embedded sia costituito da 32 blocchi di 1 MByte, che tali blocchi siano numerati da 0 a 31, che il sistema operativo mantenga traccia dei blocchi liberi (occupati) indicandoli in un vettore con il valore 0 (1), e che la situazione attuale del disco sia rappresentata dal seguente vettore:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	1	0	0	0	0	1	0	0	1	1	1	0	1	0	0	1	0	1	0	0	1	0	0	1	1	1	0	1	0	1	1

Con riferimento alle metodologie di allocazione di file contigua, concatenata, FAT e indicizzata, indicare come possono essere allocati in sequenza i file `File1`, `File2` e `File3` di dimensione uguale a 4.4, 3.6 e 5.9 Mbyte, rispettivamente. Si ipotizzi che la FAT possa essere memorizzata all'interno di un blocco.

Per ogni strategia, indicare le informazioni memorizzate nella directory e dove tali informazioni vengono memorizzate.

Contigua

	Blocchi	Inizio	lunghezza
F1	non ci sta	/	/
F2	2, 3, 4, 5	2	4
F3	non ci sta	/	/

5 4 6 Blocchi occupati

Concatenata

	Blocchi	Inizio	Fine
F1	0 → 2 → 3 → 4 → 5 → -1	0	5
F2	7 → 8 → 12 → 14 → -1	7	14
F3	15 → 17 → 19 → 20 → 22 → 23 → -1	15	23

FAT

FAT nel blocco 0

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
X	2	3	4	5	7	2	-1	12	2	2	2	14	2	15	-1	2	19	2	20	22	2	23	27	2	2	2	-1	2	2	2	

FAT

F1 2 → 3 → 4 → 5 → 7 → X

F2 8 → 12 → 14 → 15 → X

F3 17 → 19 → 20 → 22 → 23 → 27 → X

Indicizzata

INDICI

F1 → blocco 0 | 2 3 4 5 7 -1 |

F2 → blocco 8 | 12 14 15 17 -1 |

F3 ✗ blocco 9 | 20 22 23 27 29 | non ci sta !

le directory entry di F1 e F2, puntano ai blocchi indice 0 e 8.

2. Due processi (P_1 e P_2) devono accedere in mutua esclusione a una sezione critica di nome SC. Si risolva tale problema mediante la procedura `testAndSet` e quindi mediante l'utilizzo dei semafori e le primitive `init`, `wait` e `signal`. Si descrivano infine le principali differenze (vantaggi e svantaggi) delle due soluzioni precedenti.

   Si illustrino inoltre le implementazioni (senza busy-waiting e in pseudo-codice) delle tre funzioni precedenti (`testAndSet`, `wait` e `signal`).

ES 2

SC con testAndSet

```
char lock=FALSE;
P1 while(TRUE){
     while(testAndSet(&lock));
     //SC;
     lock=FALSE;
     //noSC;
   }
```

```
P2 while(TRUE){
     while(testAndSet(&lock));
     //SC;
     lock=FALSE;
     //noSC;
   }
```

SC con init, wait e signal

```
init(m,1);
P1 while(TRUE){
     wait(&m);
     //SC;
     signal(&m);
     //noSC;
   }
```

```
P2 while(TRUE){
     wait(&m);
     //SC;
     signal(&m);
     //noSC;
   }
```

Vantaggi testAndSet:

- utilizzabile in ambiente multiprocessore
- estendibile a N processi
- facile da usare dal punto di vista software
- Simmetrica

Svantaggi testAndSet

- busy wait su spin-lock
- deve essere implementata in hardware

Vantaggi semafori

- Attesa passiva (i.e. no busy wait)
- Portabilità su sistemi diversi

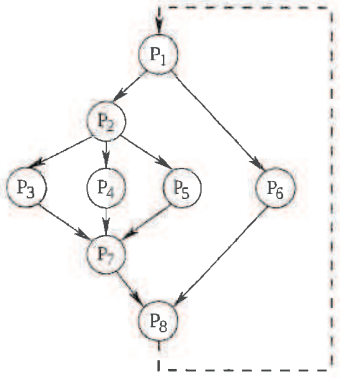
Svantaggi semafori

- Errori nel loro utilizzo possono portare processi coinvolti in deadlock

```
char testAndSet(char *lock){
   char val;
   val = *lock;
   *lock = TRUE;
   return val;
}
wait(sem_t *s){
   s->cnt--;
   if(s->cnt<0){
      push P to s->head;
      block P;
   }
}
```

```
signal(sem_t *s){
   s->cnt++;
   if(s->cnt<=0){
      pop P from s->head;
      wakeup P;
   }
}
```

3. Un programma concorrente è costituito da 8 processi $(P_1, P_2, \ldots, P_8)$ la cui relazione temporale è illustrata dalla figura successiva:



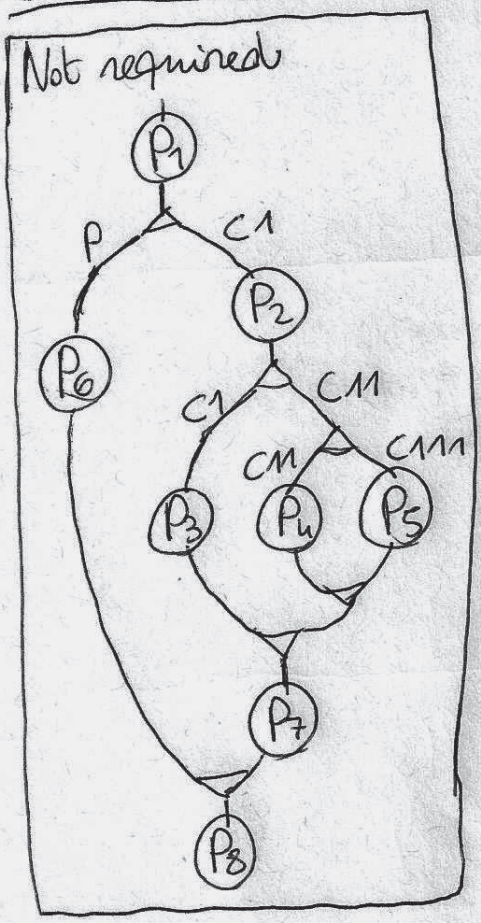
Nel caso i processi **non siano ciclici**, si riporti il programma in grado di realizzare il precedente grafo di precedenza utilizzando solo le system call `fork` e `wait`. In questo caso si trascuri l'arco tratteggiato.

Nel caso i processi **siano ciclici** (ovvero con corpo del tipo `while(1)`), si riporti il corpo processi $(P_1, P_2, \ldots, P_8)$, utilizzando le primitive `init`, `signal`, `wait` e `destroy`. Riportare inoltre l'inizializzazione dei semafori. In questo caso si consideri l'arco tratteggiato.

Ex 3

Non cyclic, fork, wait and exit

Solution 1

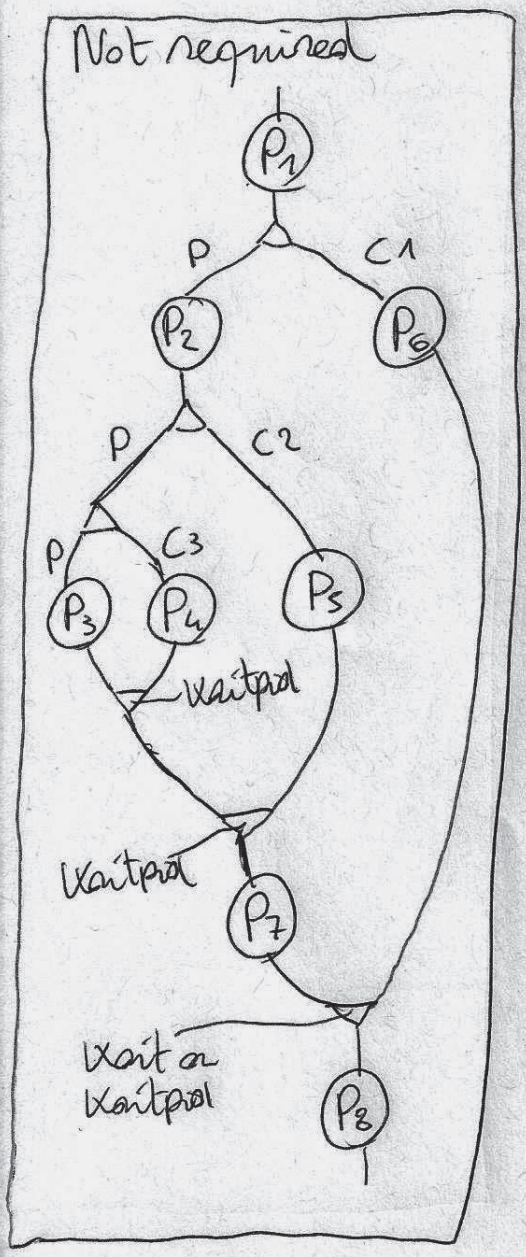



```
P1();
if (fork()){
   P6();
   wait((int *)0);
   P8();
}else{
   P2();
   if (fork()){
      P3();
      wait((int*)0);
      P7();
      exit(0);
   }else{
      if (fork()){
         P4();
         wait((int *)0);
      }else{
         P5();
         exit(0);
      }
      exit(0);
   }
}
```

EX 3

no cyclic, fork, wait and exit

Solution 2




```
P1();
pidC1 = fork();
if (pidC1){
    P2();
    pidC2 = fork();
    if (pidC2){
        pidC3 = fork();
        if (pidC3){
            P3();
            waitpid(pidC3, (int*)0, 0);
        }else{
            P4();
            exit(0);
        }
        waitpid(pidC2, (int*)0, 0);
    }else{
        P5();
        exit(0);
    }
    P7();
    wait((int*)0);
}else{
    P6();
    exit(0);
}
P8();
```

EX 3

cyclic, init, signal, wait and destroy

```
init(S1,1);   init(S2,..S8,0);

P1 while(1){
     wait(S1);
     P1();
     signal(S2);
     signal(S6);
   }

P2 while(1){
     wait(S2);
     P2();
     signal(S3);
     signal(S4);
   } signal(S5);

P3 while(1){
     wait(S3);
     P3();
     signal(S7);
   }

P4 while(1){
     wait(S4);
     P4();
     signal(S7);
   }

P5 while(1){
     wait(S5);
     P5();
     signal(S7);
   }

P6 while(1){
     wait(S6);
     P6();
     signal(S8);
   }

P7 while(1){
     wait(S7);
     wait(S7);
     wait(S7);
     P7();
     signal(S8);
   }

P8 while(1){
     wait(S8);
     wait(S8);
     P8();
     signal(S1);
   }

destroy(S1...S8);
```

4. Scrivere uno script BASH in grado di realizzare la funzionalità di "*cestino*" operando su linea di comando. Lo script gestisce una cartella `TRASH` (che si ipotizzi esistente), situata nella home dell'utente, in cui memorizza tutti i file cancellati dall'utente. In tale cartella, lo script memorizza anche il file nascosto `.TRASH_INDEX`, contenente, per ciascun file cancellato, una entry con le seguenti informazioni:

   `filename absdir`

   Lo script deve essere in grado di eseguire operazioni di tre tipi:

   - `--delete abspath`: cancellazione del file di path assoluto `abspath`. Si verifichi che il file da cancellare esista e non sia duplicato nel direttorio `TRASH`, quindi si sposti tale file in `TRASH` e si memorizzino il nome (`filename`) e il percorso assoluto della cartella d'origine (`absdir`) nel file `.TRASH_INDEX`.
   - `--restore filename`: ripristino del file di nome `filename`. Si consulti `.TRASH_INDEX` per verificare la presenza di `filename` ed ottenerne il percorso della cartella d'origine. Si verifichi l'esistenza di tale cartella e si sposti in essa il file, infine si cancelli da `.TRASH_INDEX` la entry relativa al file ripristinato.
   - `--restore-all`: ripristino di tutti i file cancellati. Si effettui l'operazione precedente per tutti i file registrati in `.TRASH_INDEX`, infine si cancelli il contenuto di `.TRASH_INDEX`.

   Si noti che ogni invocazione dello script deve permettere l'esecuzione di una singola operazione tra le tre descritte. In caso di problemi durante la cancellazione o il ripristino di un singolo file, si avverta l'utente con un messaggio d'errore e non si esegua l'operazione. Si verifichi il passaggio del numero corretto di parametri.

   Esempio (supponendo che `trash` sia il nome dello script):

```
> ./trash --delete /dir1/pippo -> ("pippo" spostato in TRASH, riga "pippo /dir1/"
                                  aggiunta in .TRASH_INDEX)
> ./trash --delete /dir3/pippo -> ERRORE  (filename "pippo" presente in .TRASH_INDEX)
> ./trash --restore pippo      -> ("pippo" spostato in "/dir1/", riga "pippo /dir1/"
                                  rimossa da .TRASH_INDEX)
```

   Suggerimento: si ricorda che il comando `grep` con l'opzione `-v` elimina dell'input tutte le righe che non soddisfano il criterio di ricerca.

```
#!/bin/bash

#
#   Launch as: ./es4.sh --delete <abspath>
#              ./es4.sh --restore <filename>
#              ./es4.sh --restore-all
#

TRASH_DIR="~"
TRASH_INDEX=".TRASH_INDEX"

# Manage delete operation
if [ $1 == "--delete" ] && [ $# -gt 1 ]; then

    # Split file name and path
    FILE_NAME=$(basename "$2")
    FILE_PATH=$(dirname "$2")

    # Check file existence
    if [ ! -f "$FILE_PATH/$FILE_NAME" ]; then
        echo "File non trovato"
        exit 1
    fi

    # Check if a file with the same name is already in the trash folder
    if [ -f "$TRASH_DIR/$FILE_NAME" ]; then
        echo "Esiste già un file di nome $FILE_NAME nel cestino"
        exit 1
    fi

    # Move the file into the trash folder
    mv "$FILE_PATH/$FILE_NAME" "$TRASH_DIR/"

    # Log new delete file entry into the trash index
    echo "$FILE_NAME $FILE_PATH" >> "$TRASH_DIR/$TRASH_INDEX"

# Manage restore operation
elif [ $1 == "--restore" ] && [ $# -gt 1 ]; then

    # Set file name to restore
    FILE_NAME=$2

    # Retrieve file path
    FILE_PATH=$(grep -e "^$FILE_NAME" "$TRASH_DIR/$TRASH_INDEX" | cut -d " " -f 2)

    # Check if file name was found in trash index
    if [ "$FILE_PATH" == "" ]; then
        echo "Nessun file di nome $FILE_NAME nel cestino"
        exit 1
    fi

    # Check source folder existence
    if [ ! -d "$FILE_PATH/" ]; then
        echo "Cartella di origine $FILE_PATH non trovata"
        exit 1
    fi

    # Restore file
    mv "$TRASH_DIR/$FILE_NAME" "$FILE_PATH/"

    # Delete index entry
    grep -v -e "^$FILE_NAME" "$TRASH_DIR/$TRASH_INDEX" >>
"$TRASH_DIR/~$TRASH_INDEX"
    mv "$TRASH_DIR/~$TRASH_INDEX" "$TRASH_DIR/$TRASH_INDEX"
```

```
# Manage restore all operation
elif [ $1 == "--restore-all" ]; then

    # Scan all lines in the trash index
    while read FILE_NAME FILE_PATH; do

        # Check source folder existence
        if [ ! -d "$FILE_PATH/" ]; then
            echo "Cartella di origine $FILE_PATH non trovata"
            continue
        fi

        # Restore file
        mv "$TRASH_DIR/$FILE_NAME" "$FILE_PATH/"

    done < "$TRASH_DIR/$TRASH_INDEX"

    # Clear trash index content
    echo -n "" > "$TRASH_DIR/$TRASH_INDEX"

# Manage wrong parameters
else
    echo "Utilizzo: es4.sh < --delete | --restore | --restore-all > [arg]"
    exit 1
fi
```

5. In elaborazione digitale delle immagini, lo *smoothing* di una immagine consiste nell'applicazione di una funzione di filtro il cui scopo è quello di evidenziare i pattern significativi. Si scriva la funzione

```
void smoothing (int **mat, int r, int c);
```

che, una volta ricevuti quali parametri la matrice di interi `mat` contenente `r` righe e `c` colonne effettua lo smoothing di tutti i suoi valori, secondo il seguente algoritmo semplificato.

Ciascun valore della matrice deve essere sostituito dalla media aritmetica degli elementi adiacenti (qualunque sia il loro numero). La funzione deve procedere come segue:

- Definire una matrice temporanea di supporto della stessa dimensione di `mat`.
- Eseguire un numero di thread uguale a `(R · C + 1)`.
  - I primi `(R · C)` thread vengono eseguiti immediatamente. Ognuno di questi thread si concentra su un elemento specifico della matrice e si occupa di calcolare il valore medio degli elementi ad esso adiacenti e quindi di memorizzare tale valore nella matrice di supporto nella posizione corrispondente.
  - L'ultimo thread, viene eseguito solo quando tutti i precedenti `(R · C)` thread hanno terminato. Esso si occupa di ricopiare la matrice temporanea su quella originale. Quando questa operazione è terminata la funzione `smoothing` termina anch'essa.

LA SOLUZIONE RIPORTA IL PROGRAMMA COMPLETO.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

#define R 4
#define C 5

void smoothing (int **mat, int r, int c);

// For average
typedef struct {
  int **source_mat;
  int r, c; // Dimension of the source matrix
  int i, j; // Position of the element to analyze
  int *dest; // Destination element
  sem_t *sync_sem; // To synchronize the end of average computation with the start
of the copy
} args_t;

// For copy
typedef struct {
  int **source_mat;
  int r, c; // Dimension of the source matrix
  int **dest_mat; // Destination element
  sem_t *sync_sem; // To synchronize the end of average computation with the start
of the copy
} args_copy_t;

void *average(void *arg) {

    // Get arguments
    int **mat = ((args_t *) arg)->source_mat;
    int r = ((args_t *) arg)->r;
    int c = ((args_t *) arg)->c;
    int i = ((args_t *) arg)->i;
    int j = ((args_t *) arg)->j;
    int *dest = ((args_t *) arg)->dest;
    sem_t *sem = ((args_t *) arg)->sync_sem;
    int n_elem = 0;

    // Compute average of adjacent elements
    *dest = 0;
    for (int x=-1; x<=1; x++)
      for (int y=-1; y<=1; y++)
        if ( (x!=0 || y!=0) && i+x>=0 && i+x<r && j+y>=0 && j+y<c) {
          *dest += mat[i+x][j+y];
          n_elem++;
        }
    *dest /=n_elem;

    /* IMP NOTE: instead of using this semaphore, solutions based on pthread_join
are also acceptable */
    sem_post(sem);

    pthread_exit(0);
}


void *copy(void *arg) {
  // Get arguments
  int **source_mat = ((args_copy_t *) arg)->source_mat;
  int r = ((args_copy_t *) arg)->r;
```

```
  int c = ((args_copy_t *) arg)->c;
  int **dest_mat = ((args_copy_t *) arg)->dest_mat;
  sem_t *sem = ((args_copy_t *) arg)->sync_sem;

  // Wait the finish of all the average threads
  for (int i=0; i<r*c; i++)
    sem_wait(sem);

  // Copy into the destination matrix
  for (int i=0; i<r; i++)
    for (int j=0; j<c; j++)
      dest_mat[i][j] = source_mat[i][j];

  pthread_exit(0);
}


void smoothing (int **mat, int r, int c) {
  pthread_t *tids;
  args_t *args;
  args_copy_t args_copy;
  sem_t sem;

  // Allocate temporary matrix
  int **tmp_mat;
  tmp_mat = (int**)malloc(r*sizeof(int*));
  for (int i=0; i<r; i++)
    tmp_mat[i] = (int*)malloc(c*sizeof(int));

  // Allocate thread id array
  tids = (pthread_t *) malloc((r*c+1)*sizeof(pthread_t));

  // Synchronization semaphore initialization
  sem_init(&sem, 0, 0);

  // Allocate array of args
  args = (args_t *) malloc(r*c*sizeof(args_t));

  // Args for average
  for(int i=0; i<r; i++) {
    for(int j=0; j<c; j++) {
      args[i*c+j].source_mat=mat;
      args[i*c+j].r = r;
      args[i*c+j].c = c;
      args[i*c+j].i = i;
      args[i*c+j].j = j;
      args[i*c+j].dest = &tmp_mat[i][j];
      args[i*c+j].sync_sem = &sem;
    }
  }

  // Args for copy
  args_copy.source_mat = tmp_mat;
  args_copy.r = r;
  args_copy.c = c;
  args_copy.dest_mat = mat;
  args_copy.sync_sem = &sem;

  // Start r*c threads that computes the average of adjacent element
  for(int i=0; i<r*c; i++) {
    pthread_create(&tids[i], NULL, average, &args[i]);
  }
  // Start copy thread
  pthread_create(&tids[r*c], NULL, copy, &args_copy);
```

```
  // Wait that the copy is completed
  pthread_join(tids[r*c], NULL);

  // Free memory
  free(tids);
  for (int i=0; i<r; i++)
    free(tmp_mat[i]);
  free(tmp_mat);
}


int main(int argc, char **argv) {
  int **mat;
  int n=0;

  mat = (int**)malloc(R*sizeof(int*));
  for (int i=0; i<R; i++)
    mat[i] = (int*)malloc(C*sizeof(int));

  for(int i=0; i<R; i++)
    for(int j=0; j<C; j++)
      mat[i][j] = n++;

  printf("BEFORE smoothing\n");
  for(int i=0; i<R; i++) {
    for(int j=0; j<C; j++)
      printf("%d ", mat[i][j]);
    printf("\n");
  }

  smoothing (mat, R, C);

  printf("AFTER smoothing\n");
  for(int i=0; i<R; i++) {
    for(int j=0; j<C; j++)
      printf("%d ", mat[i][j]);
    printf("\n");
  }


  for (int i=0; i<R; i++)
    free(mat[i]);
  free(mat);

  return 0;
}
```

- Utilizzo CPU
- Produttività (Throughput)
- Tempo completamento
- Tempo risposta

(+ Tempo attesa)

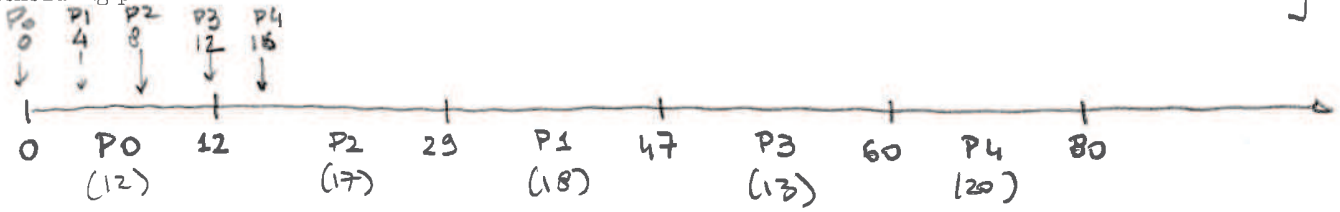
6. Si consideri il seguente insieme di processi:

Processo	Tempo arrivo	Burst Time	Priorità
P_0	0	12	2
P_1	4	18	3
P_2	8	17	1
P_3	12	13	4
P_4	16	20	5

Rappresentare mediante diagramma di Gantt l'esecuzione di tali processi utilizzando gli algoritmi di scheduling PS (Priority Scheduling), RR (Round Robin) e SRTF (Shortest Remaining Time First). Calcolare il tempo di attesa medio per ciascun processo e quello globale. Si consideri un quantum temporale di 15 unità di tempo.

Si illustrino quali altre metriche di valutazioni sarebbe possibile utilizzare al fine di confrontare gli algoritmi di scheduling precedentemente indicati.

PS

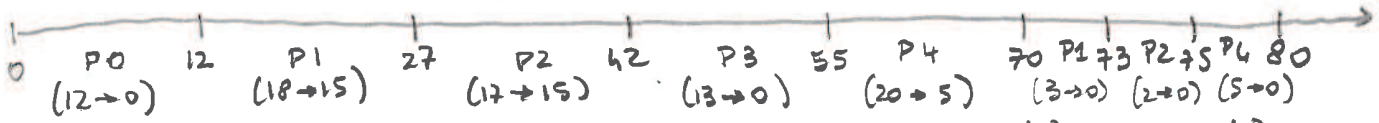


$T_a = 0 + (12-8) + (29-4) + (47-12) + (60-16) = 0 + 4 + 25 + 35 + 44 = 108$

$P_0 \quad P_2 \quad P_1 \quad P_3 \quad P_4$

$\overline{T_a} = \frac{108}{5} = 21.6$

RR



Q: $P_0(12)$ | $P_1(18)\ P_2(17)\ P_3(13)$ | $P_2(17)\ P_3(13)\ P_4(20)\ P_1(3)$ | $P_3(13)\ P_4(20)\ P_1(3)\ P_2(2)$ | $P_4(20)\ P_1(3)\ P_2(2)$ | $P_1(3)\ P_2(2)\ P_4(5)$ | $P_4(5)$

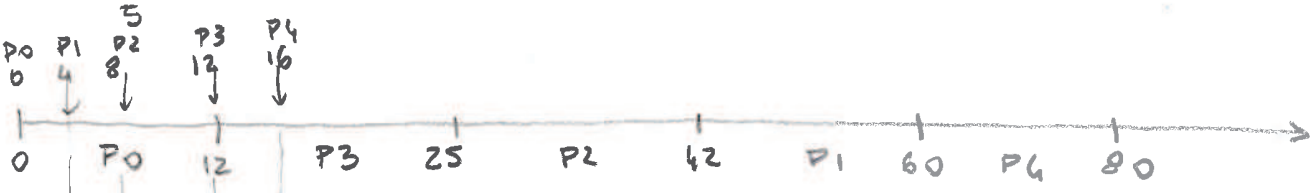
$T_a = 0 + (12-4) + (27-8) + (42-12) + (55-16) = 0 + 8 + 19 + 30 + 39 =$
$(70-27) \quad (73-42) \quad (75-70) \qquad 43 \quad 31 \quad 5$

$= 0 + 51 + 50 + 30 + 44 = 175$

$P_0 \quad P_1 \quad P_2 \quad P_3 \quad P_4$

$\overline{T_a} = \frac{175}{5} = 35.0$

SRTF



P0(8) P4(18)
P0(4) P2(17) P1(18)
P3(13) P2(17) P1(18)
P3(8) P2(17) P1(18) P4(20)

$P_0 \quad P_1 \quad P_2 \quad P_3 \quad P_4$

$T_a = 0 + (42-4) + (25-8) + (12-12) + (60-16)$

$= 0 + 38 + 17 + 0 + 44$

$= 99$

$\overline{T_a} = \frac{99}{5} = 19.8$