Reserved cells

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

Operationg Systems

Examination task

12 February 2020

ID number _____ Surname _____ Name _____

Professor: ◯ Scanzio

It is not possible to consult texts, notes or to use calculators. The only material allowed consists in the forms distributed by the professor. Solve the exercises in the reserved spaces. Additional sheets are permitted only when strictly necessary. Report the main steps for solving exercises.
Duration: 100 **minutes.**

1. Suppose that the hard disk of a small embedded system is composed if 32 blocks of 1 MByte each, which are numbered from 0 to 31. Suppose that the operating system keeps track of the free (occupied) blocks indicating them in a vector with the value 0 (1), and that the current situation of the disk is represented by the following vector:

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

With reference to the file allocation methodologies *contiguous*, *linked*, *FAT* and *indexed*, indicate how the files `File1`, `File2` and `File3` (with dimension 4.4, 3.6 and 5.9 Mbyte, respectively) can be allocated.

Schematically report the information stored in the directory entry, and where that information is stored.

Ex 1

File 1 4.4 Mbyte → 5 used blocks
File 2 3.6 Mbyte → 4 used blocks
File 3 5.9 Mbyte → 6 used blocks

Contiguous

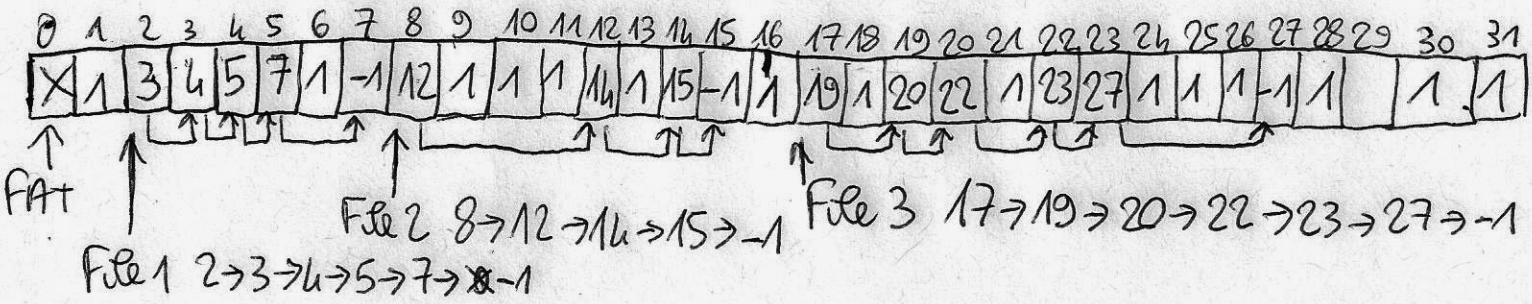
	Blocks	Dir. Start	Dir. Entry lenght
File 1	NOT ALLOCABLE	/	/
File 2	2,3,4,5	2	4
File 3	NOT ALLOCABLE	/	/

Linked

	Blocks	Start	End
File 1	0→2→3→4→5→-1	0	5
File 2	7→8→12→14→-1	7	14
File 3	15→17→19→20→22→23→-1	15	23

FAT

FAT in block 0



Indexed

	Dir entry	
File 1	block 0	2 3 4 5 7 -1
File 2	block 8	12 14 15 17 -1
File 3	block 9	~~20~~ ~~22~~ 23 27 29 NOT ENOUGH SPACE

2. Two processes (P_1 and P_2) must access in mutual exclusion to a critical section called *CS*. Solve this problem with the `testAndSet` procedure, and then with the use of semaphores and the `init`, `wait` and `signal` primitives. In addition, describe the main differences (advantages and disadvantages) of the two previous solutions.

   Finally, illustrate the implementations (in pseudo-code) of the three previous functions (`testAndSet`, `wait` and `signal`).

Ex 2

CS with testAndSet

```
char lock = FALSE;
P1 while (TRUE){
     while (testAndSet(&lock));
     //CS;
     lock = FALSE;
     //no CS;
   }
```

```
P2 while (TRUE){
     while (testAndSet(&lock));
     //CS;
     lock = FALSE;
     //no CS
   }
```

CS with init, wait and signal

```
init(m,1);
P1 while (TRUE){
     wait(&m);
     //CS;
     signal(&m);
     //no CS
   }
```

```
P2 while (TRUE){
     wait(&m);
     //CS;
     signal(&m);
     //no CS
   }
```

Advantages testAndSet

- Usable in multiprocessor environments
- extensible to N processes
- easy to use from the point of view of the software
- Symmetric

Disadvantages testAndSet

- Busy wait on spin-lock
- Must be implemented in hardware

Advantages semaphores

- Passive wait (i.e. no busy wait)
- Portability on different systems

Disadvantages semaphores

- Errors on their use can lead even real processes to deadlock

```
char testAndSet(char *lock){
   char val;
   val = *lock;
   *lock = TRUE;
   return val;
}
```

Possibility 1

```
signal(S){
   S++;
}
```

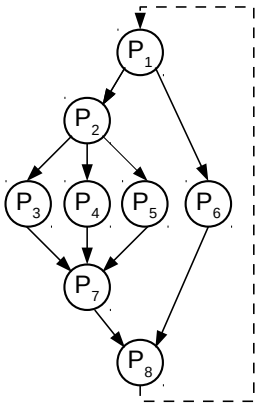
```
wait(S){
   while(S<=0);
   S--;
}
```

Possibility 2

```
signal(S){
   if(blocked())
      wakeup();
   else S++;
}
```

```
wait(S){
   if (S==0) block();
   else S--;
}
```

3. A concurrent program is composed of 8 processes (P_1, P_2,...,P_8), whose temporal relationship is illustrated by the following figure:



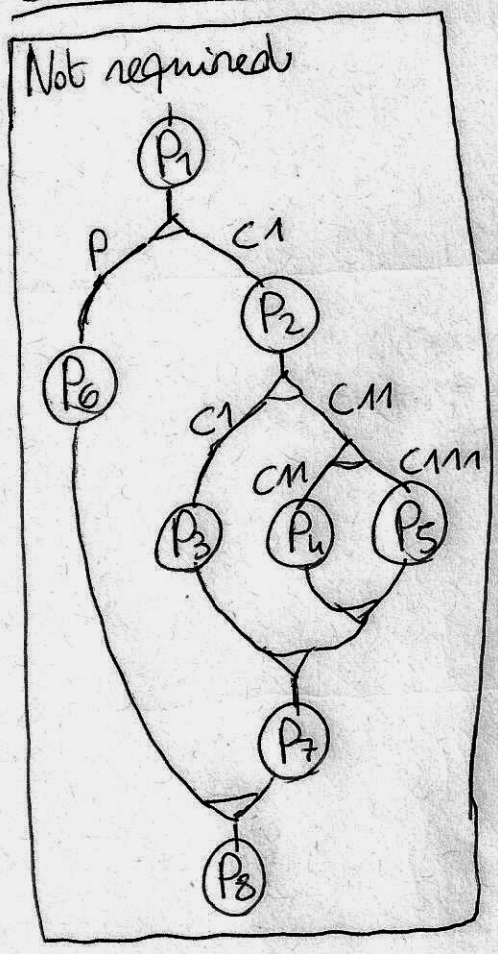
In the case of **not cyclic** processes, report the program that implements the previous precedence graph by using only the `fork`, `wait` and `exit` system calls. In this case, do not consider the dotted arc.

In the case of **cyclic** processes (i.e., function with body `while(1)`), report the body of processes P_1, P_2,...,P_8 using the primitives `init`, `signal`, `wait` and `destroy`. Report the initialization of the semaphores, and use the minimum number of semaphores. In this case, consider the dashed arch.

Ex 3

Not cyclic, fork, wait and exit

Solution 1

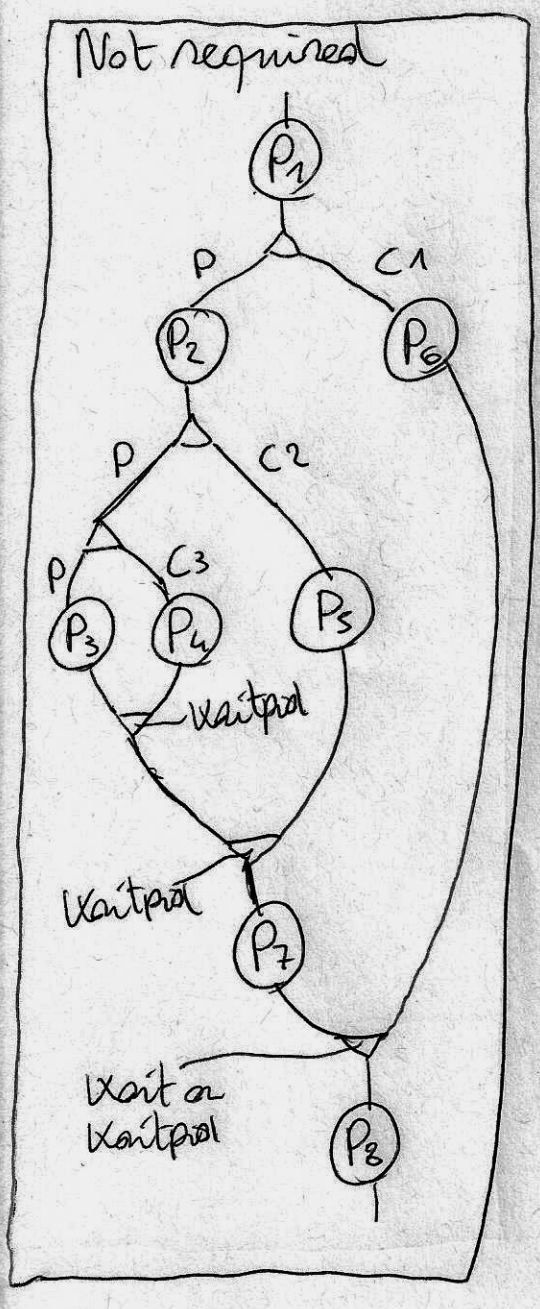



```
P1();
if (fork()){
   P6();
   wait((int *)0);
   P8();
}else{
   P2();
   if (fork()){
      P3();
      wait((int*)0);
      P7();
      exit(0);
   }else{
      if (fork()){
         P4();
         wait((int *)0);
      }else{
         P5();
         exit(0);
      }
      exit(0);
   }
}
```

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. Implement a BASH script capable to realize the "*trash*" functionality using the command line. The script manages the directory `TRASH` (which is assumed to exist and it is located in the home of the user). In this directory, the script stores all the files deleted by the user, and a hidden file `.TRASH_INDEX`, containing, for each deleted file, an entry with the following information:

   `filename absdir`

   The script must be able to perform three types of operations:

   - `--delete abspath`: deletion of the file with absolute path `abspath`. The script must check that the file to be deleted exists and is not duplicated in the `TRASH` directory, then it moves that file in `TRASH`, and it stores its name (`filename`) and the absolute path of the source directory (`absdir`) in the `.TRASH_INDEX` file.
   - `--restore filename`: restores the file with name `filename`. Use the content of the file `.TRASH_INDEX` to check for the presence of `filename`, and obtain the path of the original directory. Check the existence of this directory and move the file into it, then delete the entry related to the restored file from `.TRASH_INDEX`.
   - `--restore-all`: restores all the deleted files. This option performs the previous operation for all the files registered in `.TRASH_INDEX`, finally delete the contents of `.TRASH_INDEX`.

   Note that each invocation of the script must allow the execution of a single operation among the three described. In the case of problems during the deletion or restoration of a single file, the user must be warned with an error message, and the operation is not performed. Check for the correct number of parameters.

   Example (assuming that `trash` is the name of the script):

```
> ./trash --delete /dir1/pippo -> ("pippo" moved in TRASH, row "pippo /dir1/"
                                  added in .TRASH_INDEX)
> ./trash --delete /dir3/pippo -> ERROR  (filename "pippo" contained in .TRASH_INDEX)
> ./trash --restore pippo      -> ("pippo" moved in "/dir1/", row "pippo /dir1/"
                                  removed from .TRASH_INDEX)
```

   Suggestion: remember that the `grep` command with the `-v` option deletes all the lines that do not match the search criterion.

```
#!/bin/bash

################################################################################
# Exercise 4 - Exam 12/02/2020                                                 #
#   Launch as: ./ex4.sh --delete <abspath>                                     #
#              ./ex4.sh --restore <filename>                                   #
#              ./ex4.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 not found"
        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 "A file with name $FILE_NAME is already in the trash"
        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 "File with name $FILE_NAME not in the trash"
        exit 1
    fi

    # Check source folder existence
    if [ ! -d "$FILE_PATH/" ]; then
        echo "Source folder $FILE_PATH not found"
        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 "Source folder $FILE_PATH not found"
            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 "Usage: es4.sh < --delete | --restore | --restore-all > [arg]"
    exit 1
fi


exit 0
```

5. In digital image processing, the *smoothing* of an image consists in the application of a filter function whose purpose is to highlight significant patterns. Write the function:

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

which, after receiving as parameters the matrix of integers `mat` with `r` rows and `c` columns, performs the smoothing on all values, according to the following simplified algorithm.

Each value of the matrix must be replaced with the arithmetic mean of the adjacent elements (regardless of the number of adjacent elements). The function must perform the following steps:

- Define a temporary support matrix of the same size of the original source matrix (`mat`).
- Execute a number of threads equal to $(R \cdot C + 1)$.
  - The first set of $(R \cdot C)$ threads is executed immediately. Each of these threads manages a specific element of the matrix, and it is responsible for calculating the average value of the elements adjacent to it, and to store this average value in the support matrix in the corresponding position.
  - The last thread is executed only when all the previous $(R \cdot C)$ threads have terminated. It takes care of copying the temporary matrix in the original one. When this operation is finished, the `smoothing` function ends.

```c
/*
  Exam 2020/02/12  Ex. 5
 */

#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);
}
```

```c
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;
}
```

6. Consider the following set of processes:

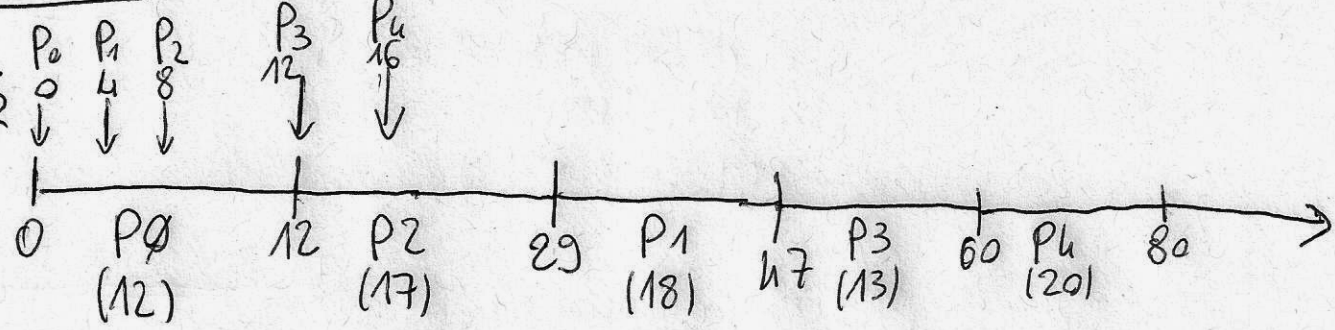
Process	Arrival Time	Burst Time	Priority
P_0	0	12	2
P_1	4	18	3
P_2	8	17	1
P_3	12	13	4
P_4	16	20	5

Represent using a Gantt diagram the execution of these processes using the scheduling algorithms PS (Priority Scheduling), RR (Round Robin), and SRTF (Shortest Remaining Time First). Compute the average *waiting time* for each process and for the global set of processes. Consider a temporal *quantum* of 15 units.

Illustrate which other evaluation metrics can be used in order to compare the previously indicated scheduling algorithms.

EX 6

PS

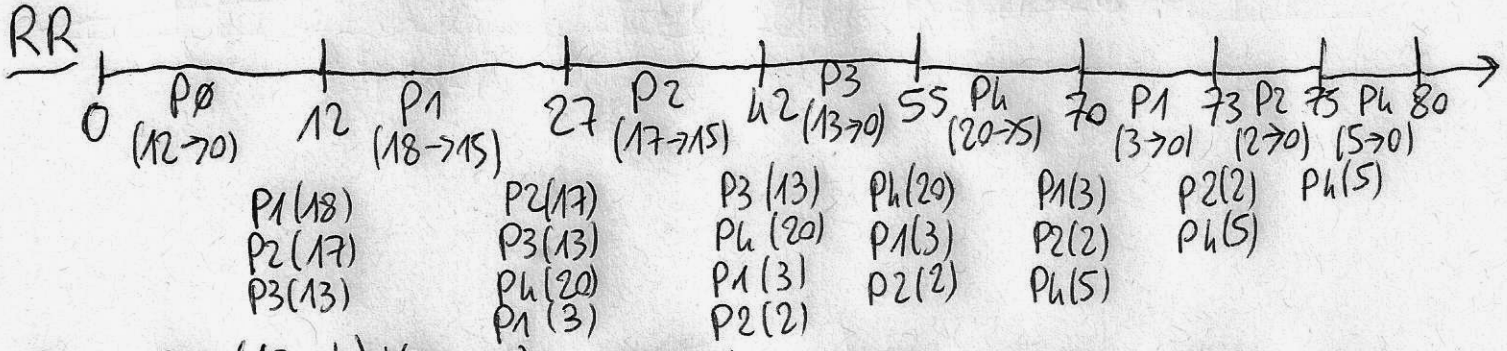


$T_w = 0 + (12-8) + (29-4) + (47-12) = 0+4+25+35+44 = 108$

(P_0, P_2, P_1, P_4)

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

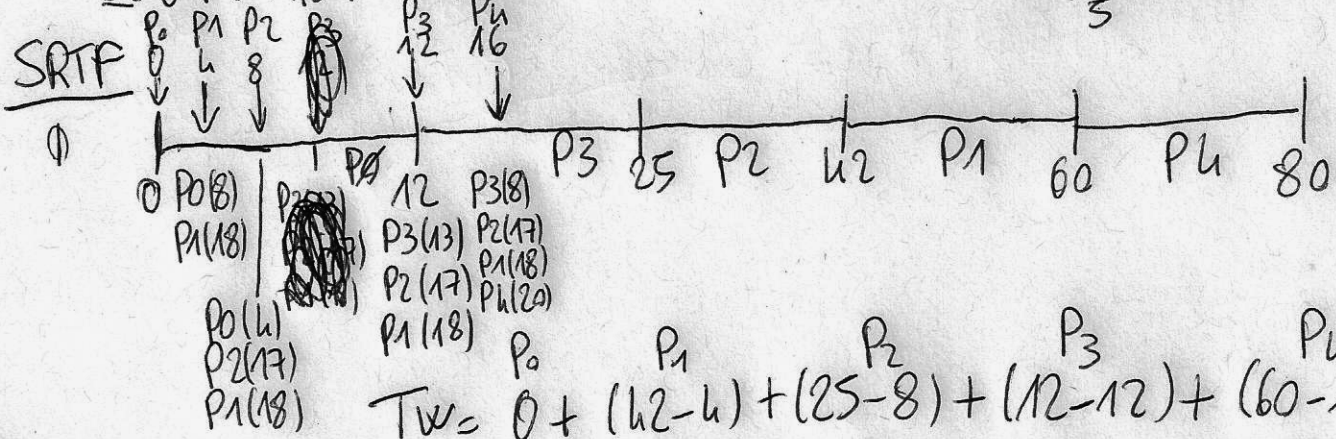
RR



$T_w = 0 + \underbrace{(12-4)+(27-27)}_{P_1} + \underbrace{(27-8)+(73-42)}_{P_2} + \underbrace{(42-12)}_{P_3} + \underbrace{(55-16)+(75-70)}_{P_4} =$

$= 0+8+43+19+31+30+39+5 = 175$ $\quad \overline{T_w} = \frac{175}{5} = 35$

SRTF



$T_w = \underbrace{0}_{P_0} + \underbrace{(42-4)}_{P_1} + \underbrace{(25-8)}_{P_2} + \underbrace{(12-12)}_{P_3} + \underbrace{(60-16)}_{P_4} =$

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

$= 99 \quad \overline{T_w} = \frac{99}{5} = 19.8$

Other metrics:
- CPU utilization
- Throughput
- Turnaround time
- Response Time