Answer to exercises K.1

- *refer to the slides 12 for the instructions*

- a. *F&A (s,x): <rtn := s; s := s+x; return (rtn)>*

  $s_0 = 0;$

  P(s): rtn := F&A (s, 1);

  while *rtn ≠ 0* do

  rtn := F&A(s, -1);

  od

  V(s): rtn := F&A(s, -1);

- b. *C&S (b,o,n,s): <b := (s=o); if b then s := n fi>*

  $s_0 = 0;$

  P(s): o := 0; n := 1; do *C&S(b,o,n,s)* while *(!b)*

  V(s): s := 0;

Answer to exercises K.3

- The interrupt handler cannot do this in general. Only if the buffer would be in globally shared kernel space.

Answer to exercises chap 4.1

A. Let *k* be the break-even point, i.e., the number of steps at which two implementations take the same time to complete. Then the following holds

- The linked list implementation needs $k * tl$ time units.
- In the array implementation, half of the operations will be inserts and half will be removes, on average. An insert may cause an overrun and hence the time for one insert is $ta + oh * p$. The time to one remove is unchanged, ta. Thus for k steps the array implementation needs:

$$k/2 * ta + k/2 * (ta + oh * p) = k * (ta + oh * p/2) \quad \text{time units.}$$

Equating the time for two implementations allows us to derive p:

$$k * tl = k * (ta + oh * p/2)$$

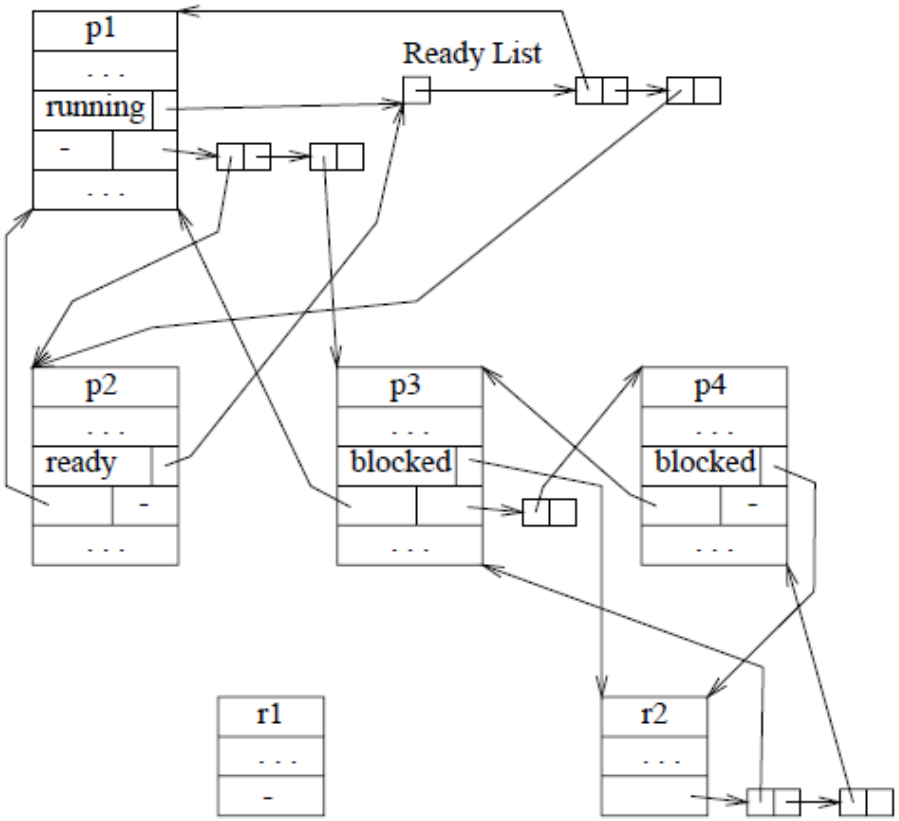
$$\Rightarrow \quad p = 2(tl - ta) / oh$$

B. Substitute the given values into the formula for p:

$$p = 2\,(10ta - ta)/\,100ta = 0.18$$

Answer to exercises chap 4.2

Refer to slide 31 or book page 115 for the structure of a process descriptor.



Answer to exercises chap 4.5

- Refer to the book page 122 for the Destroy() function.
- If a process executed *Destroy (self),* the *if* statement in the *Kill_Tree* procedure would have to be bypassed. Otherwise, the process would find its own status type equals '*running*', and would interrupt and free its own CPU, before completing the operation and invoking the scheduler.

Answer to exercises chap 4.6

- There is no change in *Create* and *Activate.*

- In *Suspend* and *Destroy,* remove the entire *if* statement that tests whether the current status type of the process is *running.* This will always be false, since only one process can be running in a single-processor machine-----the process executing the *Suspend or Destroy* statement.

Answer to exercises Chap 4.8

- SWAP(R, M): <temp := R; R : = M; M := temp}

- Pb(sb): *R :=0;* do *SWAP(R, sb)* while (*!R*)
- Vb(sb): *sb := 1;*

Answer to the given exercises on the course page

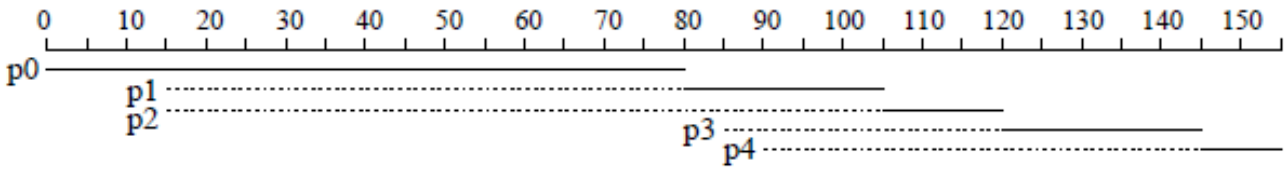
- Response time:
  - our book: time elapsing from arrival to beginning of service
- Turnaround time:
  - our book: time elapsing from arrival to completion

- Solution can be found at

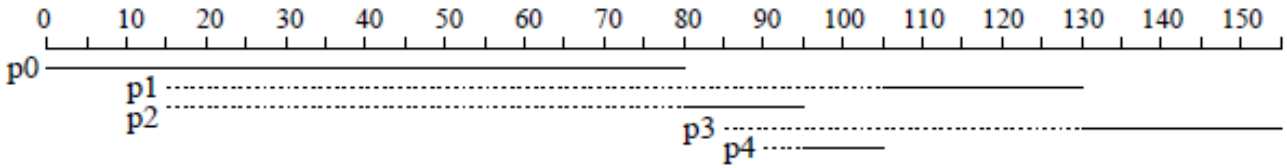
http://www.win.tue.nl/~johanl/educ/2IN05/Schedules.xls

Answer to exercises chap 5.3 (a)- (d)

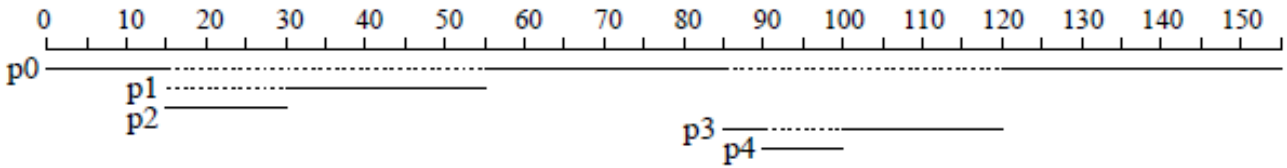
(a) FIFO



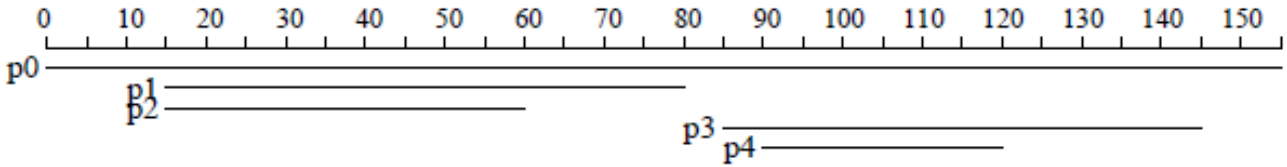
(b) SJF



(c) SRT



(d) RR (When two or three processes are shown as running concurrently, they run at 1/2 or 1/3 of their normal CPU speed.)



Answer to exercises chap 5.4 (a) – (d)

- Average turnaround time: $\left(\sum_{i=1}^{n} r_i\right) \Big/ n$

  where r_i is the real time each process *i* spends in the systems during its lifetime and *n* is the total number of active processes.

- (a) FIFO: (80 + 90 + 105 + 60 + 65) / 5 = 80
- (b) SJF: (80 + 115 + 80 +70 + 15) / 5 = 72
- (c) SRT: (155 + 40 + 15 + 35 + 10) / 5 = 51
- (d) RR: (155 + 65 + 45 + 60 + 30) / 5 = 71
  - Note that this number is only approximate; for example, the 5 time quanta during 85 and 90 cannot be divided evenly between p0 and p3; instead, one process would get 3 quanta and the other would get 2.

Answer to exercises chap 5.6

- (a) each quantum q is followed by a context switch s. Thus to cycle through n processes requires $n(s + q)$ time units. This time must not exceed the limit t, i.e.,

  $n(s + q) \leq t$. The largest q that satisfies this condition is:

  $$q = t/n - s$$

- (b) Substitute the given values into the formula for q

  $q = 1/100 - 0.001 = 0.009$

  $q = 1/100 - 0.01 = 0$

  - Note that the second case is infeasible since 100% of the CPU is devoted to the context-switch overhead; there is no time left to run the actual computations.

Answer to exercises chap 5.7 (a)

- i. When $t < q$, the quantum never expires. The execution repeats the timing sequence: $t, s, t, s, \ldots$ Thus for every t seconds of execution, $t+s$ CPU seconds are needed. The fraction of overhead is

  $$s / (t + s)$$

- ii. When $t >> q$, each execution of t needs many quanta to complete. Each quantum is followed by a context switch. The fraction of wasted CPU time is

  $$s / (s + q)$$

  - This value is approximate since t is, in general, not a multiple of q and hence the last quantum will not be used completely.

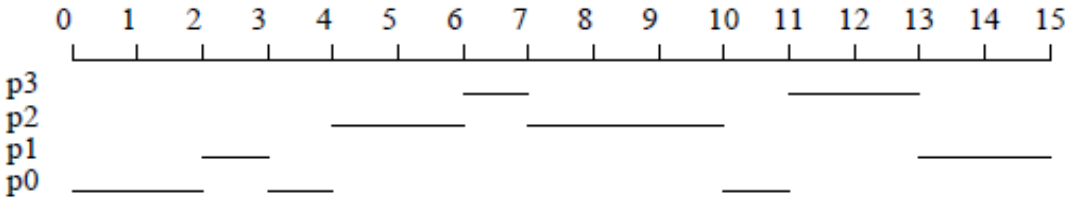
- iii. When q approaches 0, the value of $s/(s+q)$ approach 1. (100% overhead)

Answer to exercises chap 5.7 (b)

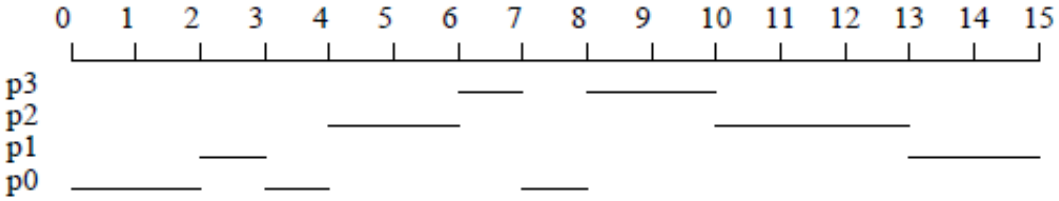
- (b) 50% overhead means $s / (t + q) = 0.5$, which is true when $q = s$.

Answer to exercises chap 5.12

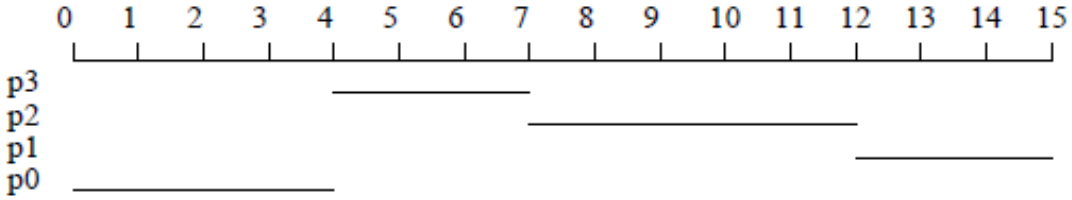
- (a)



- (b)



- (c)



Answer to exercises chap 5.13

- With priority scheduling, the following scenario will cause a deadlock: a low-priority process enters the critical section. While inside the critical section, it is preempted by a higher-priority process (that woke up from a wait or was newly created). If the process tries to enter the critical region, it will be stuck forever in the loop

  do TS(R, sb) while (!R), because *sb* will never be set to *1*.
- Under RR, this problem cannot occur, since all processes get a turn periodically, and hence the first process will eventually leave the critical section and set *sb* to *1*.