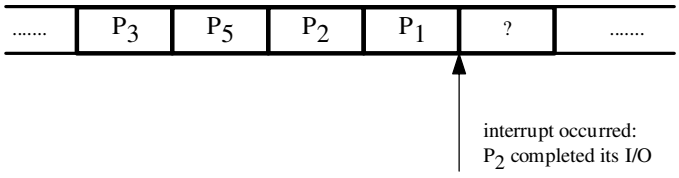
Problem

Gantt chart of the execution of several processes is shown in the figure below. Execution of process P_1 is interrupted due to the completion of P_2's I/O. There are 1.2 time units remaining in the current P_1's CPU-burst interval. Lengths of next CPU-burst intervals of processes currently in the ready queue are: P_2 (1), P_3(0.5), P_4(1), P_5(1.5). Priorities of processes are: P_1(2), P_2(1), P_3(2), P_4(4), P_5(1). Lower number represents higher priority.



Which process will be selected to run next if the following scheduling algorithms are applied:

(a) round-robin (P_1's time slice expired, too)?
(b) non-preemptive SJF scheduling (without priority considered)?
(c) preemptive SJF scheduling (without priority considered)?
(d) non-preemptive priority scheduling (in the case of tie SJF is applied)?
(e) preemptive priority scheduling (in the case of tie SJF is applied)?

Problem

What are the functions of the following modules and which transitions in a state diagram do they cause?

(a) long-term scheduler
(b) medium-term scheduler
(c) short-term scheduler

Problem

Explain the function of dispatcher.

Problem

Explain what may cause the problem in the use of the following monitor code.

```
monitor EXAMPLE is
    var P: array[0..2] of boolean;
        X: condition;

    procedure do1(id1: integer, id2: integer);
    begin
        if P[0] and P[1] and P[2] then X.wait(id1);
        if not P[0] then
            id2 := 0
        else
        if not P[1] then
            id2 := 1
        else
            id2 := 2
        endif;
        P[id2] := true;
    end;
```

```
    procedure do2(id: integer);
    begin
        P[id] := false;
        X.wait(id)
    end;

begin
    P[0] := false;
    P[1] := false;
    P[2] := false
end;
```

Problem
Explain semantics of the signal operation on semaphore and signal operation on monitor's condition variable. What is the difference?

Problem
Describe semantics of conditional critical region.

Problem
What can be the output of the following program?

```
program EXAMPLE;
        var x: integer;

        process one;
                begin
                        x := x + 1;
                end;
        process two;
                begin
                        x := 4;
                end;
begin
        x := 1;
        cobegin
                one;
                two
        coend;
        write(x);
end.
```

Problem
What is an interrupt vector?

Problem
Describe the following transitions in the process state model:

a) hold → ready
b) running → ready
c) running → waiting
d) running → complete

Problem

Describe the concept of process image.

Problem
Which transitions trigger short-term scheduling?

Problem
Let us observe two concurrent processes P_1 and P_2 that use shared real varible *x* initialized to 5.

```
P1::    .....                    P2::    .....
        .....                            if x > 4 then
        x := x - 1;                              x := 15 / x;
        .....                            end if;
        print(x);                        .....
```

a) What may be the output of process P_1?
b) Ensure mutually exclusive access to variable *x* using semaphores without changing semantics of processes P_1 and P_2.
c) Ensure mutually exclusive access to variable *x* using CCR without changing semantics of processes P_1 and P_2.

Problem
Consider simultaneous execution of the following two processes

```
P1::    .....                                P2::    .....
        with A when x > 0 do                         with A when x ≤ 0 do
                x := x - 5;                                  x := x + 3;
                if x = 1 then                                when B do y := 5 * x; end;
                   with A do x := x + 4; end;        end;
                end if;                              .....
        end;                                         .....
        .....                                        .....
```

What may be the problem in this code? How it can be conceptually resolved?

Problem
Define sleeping barber problem and relate it to operating system issues.

Problem
Monitor is implemented using semaphores in the following manner:

i. Each entry procedure F is replaced by

```
wait(mutex);
-- body of F
if next_count > 0 then
        signal(next);
else
        signal(mutex);
end if;
```

ii. For each condition *x*, *x.wait* operation is replaced by

```
x_count := x_count + 1;
if next_count > 0 then
        signal(next);
else
        signal(mutex);
end if;
```

```
                wait(x_sem);
                x_count := x_count - 1;
```

iii. For each condition *x*, *x.signal* operation is replaced by

```
                if x_count > 0 then
                        next_count := next_count + 1;
                        signal(x_sem);
                        wait(next);
                        next_count := next_count - 1;
                end if;
```

Is the implementation correct? Explain.

Problem

Protecting the operating system is crucial to ensuring that the computer system operates correctly. Provision of this protection is the reason behind dual-mode operation, memory protection, and the timer. To allow maximum flexibility, however, we would also like to place minimal constraints on the user. The following is a list of operations that are normally protected. What is the minimal set of instructions that must be protected? Identify instructions that must be protected.

(a) Change to user mode.
(b) Change to monitor mode.
(c) Read from monitor memory.
(d) Write into monitor memory.
(e) Fetch an instruction from monitor monitor memory.
(f) Turn on timer interrupt.
(g) Turn off timer interrupt.

Problem

Two cooperating processes P_1 and P_2 use shared varibles *x* and *y*.

```
P1::    x := 1;                          P2::    y := 1;
        while y = 1 loop                         while x = 1 loop
                null;                                    null;
        end loop;                                end loop;
        -- compute z                             -- compute z
        x := 0;                                  y := 0;
        print(z);
```

Do these processes always terminate? If not, explain why and correct the code to guarantee termination.

Problem

Distinguish between scheduling and dispatching.

Problem

What may be the problem in the following semaphore solution to the cigarette smokers problem and how would you remedy it.

```
procedure CIGARETTE_SMOKERS is
        INGREDIENT: array(0..2) of  SEMAPHORE := (0..2 => 0);
        SYNC: SEMAPHORE := 1;

        task type SMOKER;
        task body SMOKER is
        begin
                loop
```

```
                        WAIT(INGREDIENT(I));
                        WAIT(INGREDIENT((I + 1) mod 2));
                        -- smoke
                        SIGNAL(SYNC);
                end loop;
        end SMOKER;

        task AGENT;
        task body AGENT is
                I, J: INTEGER range 0..2;
        begin
                loop
                        I := RANDOM(0..2);
                        J := RANDOM(0..2);
                        while I = J loop
                                J := RANDOM(0..2);
                        end loop;
                        WAIT(SYNC);
                        SIGNAL(INGREDIENT(I));
                        SIGNAL(INGREDIENT(J));
                end loop;
        end AGENT;

        A_SMOKER: array (0..2) of SMOKER;
begin
        null;
end CIGARETTE_SMOKERS;
```

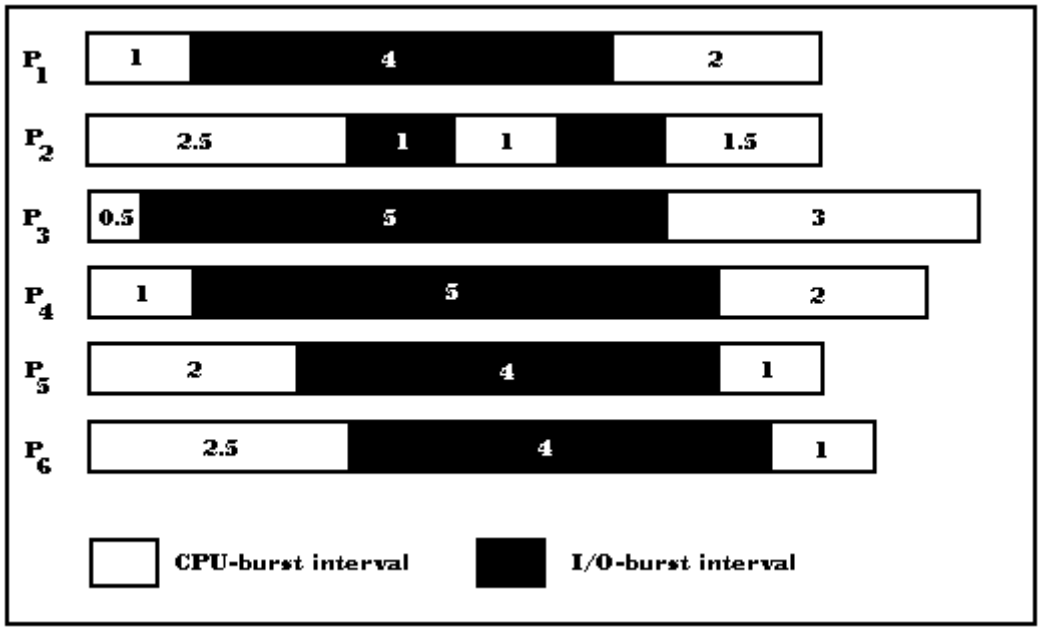
Problem
What is a resident monitor?

Problem
What is the purpose of system calls?

Problem
Processes in the uniprocessor system are assigned permanently to two ready queues. Scheduling algorithm for the foreground ready queue is round-robin with the time slice of 1 unit. Processes in the background ready queue are scheduled for execution according to the SJF algorithm only if the foreground ready queue is empty. Scheduling is preemptive. Processes P_1, P_3 and P_4 are assigned to the foreground ready queue. Processes P_2, P_5 and P_6 are assigned to the background ready queue. Arrival times are given in the following table.

Process	Arrival Time
P_1	0
P_2	0.5
P_3	2
P_4	2
P_5	2.5
P_6	2.5

Distribution of the CPU and I/O burst times for each process is given in the following figure.



Compute CPU utilization, throughput and average turnaround time.

Problem
Implement statement "**region** v **do** S ;" using semaphore.

Problem
What is buffering?

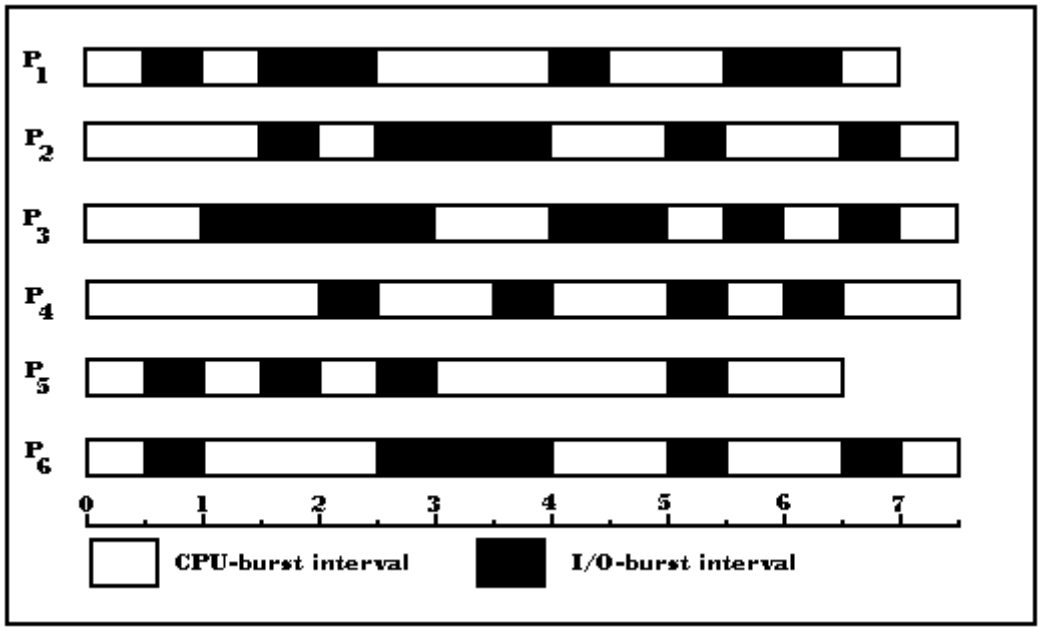
Problem
What is an interrupt vector?

Problem
Define the difference between preemptive and nonpreemptive scheduling.

Problem
Processes in the uniprocessor system are scheduled for the execution according to the SJF algorithm. Scheduling is preemptive. Arrival times are given in the following table.

Process	Arrival Time
P_1	0
P_2	0.5
P_3	1
P_4	2
P_5	1.5
P_6	0.5

Distribution of the CPU and I/O burst times for each process is given in the following figure.



Compute CPU utilization and throughput. Use

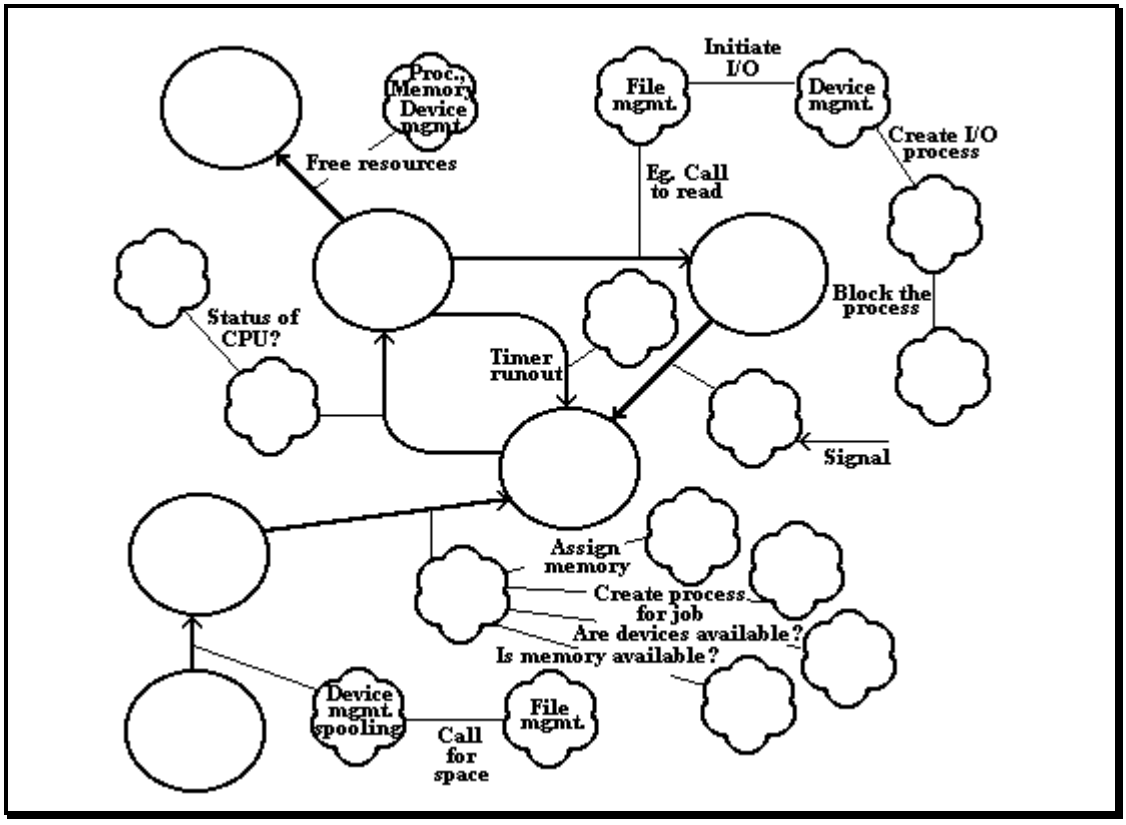
(a) Future knowledge of the length of CPU-burst intervals (given in the figure).
(b) $\tau_{n+1} = 0.4t_n + 0.6\tau_n$ to predict value of the following CPU-burst interval (t_n contains the most recent information, τ_n stores the past history, $\tau_0 = 1$).

Compute α in the following formula $\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$ which will predict most accurately values of the CPU-burst intervals in this particular case.

Problem
A binary semaphore is a semaphore with an integer value that can range only between 0 and 1. Show how a general semaphore can be implemented using binary semaphores.

Problem
What is a spooling?

Problem
Fill in the empty space in the state model diagram.



Problem
Five jobs A through E, arrive at the same time. They have estimated running times of 10, 6, 2, 4 and 8. Their externally determined priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the average process turnaround time. Ignore process switching overhead.

(a) Priority scheduling.
(b) Shortest job first.
(c) Round robin with time slice 2.
(d) FCFS (run in order 10, 6, 2, 4, 8).

All jobs are completely CPU bound.

Problem]
Some systems allow an interrupt of interrupt service routine (ISR). In that case, interrupt is enabled only after ISR reaches certain safe point in its execution. In your opinion, what is a job ISR should complete without being interrupted?

Problem
Table below shows the status of ready queue and waiting queue at the instants when scheduling decisions are made. For the execution of all five processes CPU has used 5.5 time units.

Time	Ready Queue	Waiting Queue	Process Scheduled for execution
0	P_1-P_2-P_4	P_3	P_1
1	P_2-P_4	P_3-P_1	P_2
1.5	P_2-P_4-P_3	P_1	P_2

2	P_4-P_3	P_1-P_2	P_4
3	P_3-P_5	P_1-P_2-P_4	P_3
3.5	P_1-P_2-P_3-P_4-P_5	-	P_1
4	P_2-P_3-P_4-P_5	-	P_2
4.25	P_3-P_4-P_5	-	P_3
4.5	P_4-P_5	-	P_4
5	P_5	-	P_5
5.5	-	-	-

(a) Define waiting time.
(b) Compute average waiting time of five processes in the table.
(c) Compute throughput.

Problem
List several events that might cause a dispatcher to replace one program with a new program.

Problem
In the solution to the producer-consumer problem with one producer and one consumer we were able to use just two semaphores for synchronization purposes. There is no need for mutually exclusive access to the buffer, because both producer and consumer never attempt to use the same element of the buffer at the same time. This is not true any more if we introduce multiple producers and consumers. In the solution below, third semaphore is introduced to protect critical section.

```
program PROD_CON;
    const   MAX = 1000;
            N = 20;
    var     empty, full, mutex: semaphore;
            I, O, K: integer;
            buffer: array [1..MAX] of integer;

    process type PRODUCER;
        var item: integer;
    begin
        repeat
            item := random(100);
            wait(mutex);
            wait(empty);
            buffer(I) := item;
            I := I mod MAX + 1;
            signal(full);
            signal(mutex)
        forever
    end;

    process type CONSUMER;
        var item: integer;
    begin
        repeat
            wait(mutex);
            wait(full);
            item := buffer(O);
            O := O mod MAX + 1;
            signal(empty);
```

```
                signal(mutex);
                writeln(item)
            forever
        end;
        var PROD: array[1..N] of PRODUCER;
            CONS: array [1..N] of CONSUMER;
begin
    I := 1;
    O := 1;
    initial(empty, MAX);
    initial(full, 0);
    initial(mutex, 1);
    cobegin
        for K :=1 to N do
            begin
                PROD[K];
                CONS[K]
            end;
    coend
end.
```

(a) Proposed solution is incorrect. Explain why.
(b) What changes would you make to correct the program?

Problem
Consider the following proposal for implementing statement "**with** region_name **when** B **do** S;" using semaphores:

```
var e: semaphore := 1;
    d: semaphore := 0;
    nd: integer := 0;

wait(e);
while not B do
    begin
        nd := nd + 1;
        signal(e);
        wait(d);
        wait(e)
    end;
S;
while nd > 0 do
    begin
        nd := nd - 1;
        signal(d)
    end;
signal(e);
```

Does this solution work correctly? If so, give a convincing argument that it correctly simulates a region statement. If not, clearly explain what the problem is, and give an example of an execution sequence that causes problems.

Problem
State the dining philosophers problem. Describe possible deadlock and starvation situation.

Problem
Following code represents a monitor solution to the dining philosophers problem.

```
monitor DINING_PHILOSOPHERS is
    var     STATE: array [0..4] of (THINKING, HUNGRY, EATING);
            SELF:  array [0..4] of condition;

    procedure entry PICKUP(i: integer 0..4);
            begin
                STATE[i] := HUNGRY;
                TEST(i);
                if STATE[i] <> EATING then SELF[i].wait endif;
            end;

    procedure entry PUTDOWN(i: integer 0..4);
            begin
                STATE[i] := THINKING;
                TEST(i + 4 mod 5);
                TEST(i + 1 mod 5);
            end;

    procedure TEST(k: integer 0..4);
            begin
                if  STATE[k + 4 mod 5] <> EATING
                    and STATE[k] = HUNGRY
                    and STATE[k + 1 mod 5] <> EATING
                then
                     STATE[k] := EATING;
                     SELF[k].signal;
                end if;
            end;

    begin
            for i :=0 to 4 do
                STATE[i] := THINKING;
    end;
```

(a) How should processes for philosophers look like?
(b) Explain how this solution works.

Problem
Consider the following proposal for implementing statement "**with** region_name **when** B **do** S;" using semaphores:

```
var e: semaphore := 1;
    d: semaphore := 0;
    g: semaphore := 0;
    nd: integer := 0;
    ng: integer := 0;
```

```
wait(e);
while not B do
    begin
        nd := nd + 1;
        if ng > 0 then signal(g); else signal(e); endif;
        wait(d);
        nd := nd - 1;
        ng := ng + 1;
        if nd > 0 then signal(d); else signal(g); endif;
        wait(g)
        ng := ng - 1;
    end;
S;
if nd > 0 then signal(d);
        else if ng > 0 then signal(g);
                       else signal(e);
             endif;
endif;
```

Does this solution work correctly? If so, give a convincing argument that it correctly simulates a region statement. If not, clearly explain what the problem is, and give an example of an execution sequence that causes problems.