REAL-TIME SYSTEMS AND PROGRAMMING LANGUAGES: 4th Edition
Solutions to Selected Exercises

A. Burns and A.J. Wellings
Department of Computer Science, University of York, UK

email burns,andy@cs.york.ac.uk

May 2012

Abstract

This note gives examples of solutions to the exercises involving problems in our book "Real-Time Systems and Programming Languages: 4th Edition". Answers to questions involving more general discussion topics, or understanding of material presented in the chapters, are not given. *No guarantee is given that the questions or answers are correct or consistent.*

Question 2.3

```
ensure Array_Is_Sorted
by
  Bubble sort
else by
  Exchange sort
else
  error
```

Question 2.5

If an error is detected at P1 at time t, P1 is rolled back to recovery point R13. No other processes are affected.

If an error is detected at P2 at time t, P2 is rolled back to R23, unfortunately, it has communicated with P1, so P1 is rolled back to R12.

If P3 detected error, it is rolled back to R32, this requires communication with P2 and P4 to be undone; therefore P2 must be rolled back to R22 and P4 to R42. Rolling back P2 requires P1 to be rolled back to R12.

If P4 detected error, it is rolled back to R43, this requires communication with P3 to be undone; P3 is rolled back to R32. This requires communication with P2 and P4 to be undone; therefore P2 must be rolled back to R22. Rolling back P2 requires P1 to be rolled back to R12. Finally, P4 must be rolled back again to R42.

Question 3.2

1. any assertion check
2. a watch dog timer detecting a missed deadline in another processes
3. array bounds violation
4. failure of a health monitoring check

Question 3.3

```
with character_io;
package body look is

 function test_char(c : character)
              return boolean is
 begin
    -- returns true is valid alpha
    -- numeric character else false
 end;

 function read return punctuation is
    c :character;
 begin
    loop
      c := character_io.get;
      if test_char(c) /= true then
        if c = ‘.’ then
             character_io.flush;
             return period;
        end if;
        if c = ‘,’ then
             character_io.flush;
             return comma;
        end if;
        if c = ‘;’ then
             character_io.flush;
             return semicolon;
        end if;
        raise ILLEGAL_PUNCTUATION;
      end if;
    end loop;
 exception
    when ILLEGAL_PUNCTUATION =>
       character_io.flush;
       raise;
       -- or raise ILLEGAL_PUNCTUATION
    when IO_ERROR =>
```

```
       character_io.flush;
       raise;
 end read;

begin
    null;
end;
```

```
with look;
function get_punctuation return punctuation is
   p : punctuation;
begin
   loop
     begin
       p := look.read;
       exit;
     exception
       when others =>
           null;
     end;
   end loop;
   return p;
end get_punctuation;
```

Question 3.4

```
procedure reliable_heater_off is
  type stage is (first, second, third, fourth);
begin

  for i in stage loop
    begin
      case i is
        when first =>
          heater_off;
          exit;
        when second =>
          increase_coolant;
          exit;
        when third =>
          open_valve;
          exit;
        when fourth =>
           panic;
           exit;
      end case;
    exception
      when heater_stuck_on |
           temperature_still_rising |
           valve_stuck =>
        null;
    end;
  end loop;
end reliable_heater_off;
```

Question 3.8

In the first code fragment, the exception cannot be handled by the `Do_Something` procedure. The domain is the calling code.

In the second code fragment, the exception can be handled by the `Do_Something` procedure. The domain is the procedure.

In the third code fragment, the exception can be handled within the inner block declared within the procedure. This block is the domain.

Question 3.10

All the alternatives will fail in the recovery block environment because the state is restored have each has failed. Hence, the else clause is executed.

In the exception handling environment, no state restoration occurs so the secondary sets I to 20.

Question 3.13

The equivalent expression of the temperature control class would be:

```
public class ValveStuck extends Exception;
public class HeaterStuckOn extends Exception;
public class TemperatureStillRising extends Exception;


public class TemperatureControl
{
  public void heaterOn();

  public void heaterOff() throws HeaterStuckOn;

  public void increaseCoolant()
        throws TemperatureStillRising;

  public void openValve() throws ValveStuck;

  public void panic();
}
```

And the turning the heater off reliably:

```
  public static void reliableHeaterOff()
  {
    TemperatureControl TC = new TemperatureControl();
    boolean done = false;
    int I = 1;

    while(!done) {
      try {
        if(I == 1) {
          TC.heaterOff(); done = true;
        } else if(I == 2) {
          TC.increaseCoolant(); done = true;
        } else if(I == 3) {
```

```
          TC.openValve(); done = true;
        } else {
          TC.panic(); done = true;
          System.out.println("panic␣called");
        }
      }
      catch (Exception E) { I++; }
    }
  }
```

Question 3.14

If an object is thrown which is not a subclass of `Exception`. It will not be caught by the catch statement.

Question 4.1

```
with Ada.Text_IO; use Ada.Text_IO;
procedure Main is

    procedure One is
    begin
      delay 1.0;
      put_line("one finished");
    end One;


    procedure Two is
    begin
      delay 2.0;
      put_line("two finished");
    end Two;

    procedure Three is
    begin
      delay 3.0;
      put_line("three finished");
    end Three;

    type Pointer is access procedure;

    type Parameters is array(Positive range <>)
         of Pointer;


    procedure Run_Concurrently(These : Parameters) is

      task type Worker(This: Pointer);
      task body Worker is
      begin
        This;
      end Worker;
      type Worker_Pointer is access Worker;
      Starter : Worker_Pointer;
```

```
    begin
      Put_Line("Run started");
      for i in These'Range loop
        Starter := new Worker(These(i));
      end loop;
      Put_Line("Run Finishing");
    end Run_Concurrently;

begin
  Put_Line("Main started");
  Run_Concurrently((One'access, Two'access,
                    Three'access));
  Put_Line("Main Finished");
end Main;
```

Question 4.2

Although arrays of tasks can be created easily, assigning values to their discriminants is awkward. An early version of Ada 95 incorporated new syntax to allow this; however, during the language's scope reduction, this was removed. The same effect can be achieved by calling a function with a side effect.

```
package Count is
  function Assign_Number return Index;
end Count;

task type X(I : Index :=
                Count.Assign_Number);

type XA is array (Index) of X;
```

Where the body of `Count` is

```
package body Count is
  Number : Index := Index'First;
  function Assign_Number return Index is
  begin
    return Number;
    Number := Number + 1;
  end Assign_Number;
end Count;

Xtasks : XA;
```

Question 4.3

For two tasks:

```
procedure Cobegin is

  task A;
  task B;

  task body A is separate;
  task body B is separate;
```

```
begin
  null;
end Cobegin;
```

Question 4.4

A fork is simply creation of a process dynamically. This can be done easily in Ada. The join is more problematic. Notionally, given the identity of the task created, the parent can busy-wait using the 'terminated flag. However, if busy-waiting is to be avoided then some form of IPC mechanism must be used.

Question 4.5

2^{11} *we think*!

Question 4.9

The following points need to be made:

- an exception will not propagate beyond a task, even if it caused the task to terminate
- a scope cannot exit (even to propagate an exception) if they are none terminated tasks dependent on that scope.
- an exception raised in the elaboration of the declarative part of a task will cause that task to fail (without beginning its execution); moreover the parent of that task will have "tasking_error" raised before it can state executing.

These points are illustrated by the behaviour (output) of the program. If C = 2 then no exceptions are raised and the following output is generated:

```
A Started
Main Procedure Started
P Started
P Finished
(* DELAY 10 second *)
T Finished
A Finished
```

Note the output "Main Procedure Started" could occur first second or third.

When C = 1 then the procedure P will fail. But the exception (constraint_error or numeric_error) cannot propagate for 10 seconds until task T has completed. Its propagation will cause A to fail (hence no A finished message) but the main program will be unaffected:

```
A Started
Main Procedure Started
P Started
(* DELAY 10 second *)
T Finished
```

Finally when C=0 then A will fail during elaboration of its declarative part and hence it will never start to execute; moreover the main program will get tasking_error raised at its starts and hence it will not begin. The output is simply:

```
Main Procedure Failed
```

Question 4.10

For task `A`, the parent is the main task, its children are `C` and `D`, its master is `Hierarchy` and its dependents are `C` and `D`.

For task `Pointerb.all`, the parent is `C`, it has no children, its master is `Hierarchy` and it has no dependents.

For task `Another_Pointerb.all`, the parent is `D`, it has no children, its master is `Hierarchy` and it has no dependents.

For task `C`, the parent is `A`, its child is `Pointerb.all`, its master is `A` and it has no dependents.

For task `D`, the parent is `A`, its child is `Another_Pointerb.all`, its master is `A` and it has no dependents.

Procedure `Main` has no direct parent, children, master or dependents

Procedure `Hierarchy` has no direct parent, children, or master. Its dependents are `A`, `Pointerb.all`, and `Another_Pointerb.all`.

Question 4.12

`MyCalculation.run();` runs the run procedure sequentially.

whereas the

```
new Thread(MyCalculation).start();
```

will create a new thread.

Question 5.2

The algorithm works by keeping four slots for the data: two banks of two slots. The reader and the writer never access the same bank of slots at the same time. The atomic variable `Latest` contains the index of the bank to which the last data item was written and the `Next_Slot` array indexed by this value indicates which slot in that bank contains the data.

1. a `Write` is followed by a `Read` – the reader gets the last value written
2. a `Read` is preempted by a `Write` – the writer will write to a different bank of slots than being accessed by the current reader.
3. a `Write` is preempted by a `Read`– the reader will read from a different bank of slots than being accessed by the current writer.
4. a `Read` is preempted by more than on `Write` – the first writer will write to one of the slots in the bank which is not being currently read, the other will write into the other slot.

5. a `Write` is preempted by more than one `Read` – all the readers will read from the same slot in the bank other than the one being written to.

Consider some arbitrary time when the latest value of the data item is in `Four_Slot(Second, First)`. In this case `Latest` equals `Second` and `Next(Second) = First`. Assume also that this is the last value read. If another read request comes in and is interleaved with a write request, the write request will chose the first bank of slots and the first slot, and so on. Thus it is possible that the reader will obtain an old value but never an inconsistent one. If the write comes in again before the read has finished, it will write to the first bank and second slot, and then the first bank and first slot. When the reader next comes in, it will obtain the last value that was completely written (that is, the value written by the last full invocation of `Write`).

Question 5.3

```
var  mutex( initial  1);
            wrt( initial  1);
            readcount := 0;

(* reader processes *)

P(mutex)
  readcount := readcount +1;
   if  readcount = 1  then  P(wrt);
V(mutex)

  (* read data structure *)

P(mutex)
  readcount := readcount -1;
   if  readcount = 0  then  V(wrt);
V(mutex)

(* writer processes *)

P(wrt)
  (* write data structure *)
V(wrt)
```

Question 5.4

Hoare's conditional critical region is of the form:

region x **when** B **do** S1;

In the following solution: x_wait is a semaphore used for processes waiting for their guard to be TRUE; and x_count is the number waiting. When the region is left we scan down the queue to see if any process can continue. We use x_temp to keep track of the number of processes we have rescheduled (otherwise we would just loop).

```
var  x_mutex  semaphore  ( initial  1);
        x_wait   semaphore  ( initial  0);

        x_count : integer :=0;
        x_temp   : integer :=0;

   P(x_mutex)
    if not  B  then
       x_count := x_count +1;
       V(x_mutex);
       P(x_wait);
        while not  B  do
           x_temp := x_temp +1;
            if  x_temp =< x_count  then
               V(x_wait);
            else
               V(x_mutex);
            end if ;
           P(x_wait);
        end loop ;
       x_count := x_count -1;
    end if ;

   S1;

    if  x_count > 0  then
       x_temp := 1;
       V(x_wait);
    else
       V(x_mutex);
    end ;
```

Question 5.5

```
For mutual exclusion:
    mutex : semaphore(initial 1);
For signalling processes suspended:
    next  : semaphore(initial 0);
For each condition:
    x_sem : semaphore(0);
Counts:
    next_count : integer :=0;
    x_count    : integer :=0;

For each procedure:
    P(mutex);
        body;
     if  next_count > 0  then
       V(next);
     else
        V(mutex);
     end if ;

For each wait on condition x

    x_count := x_count +1;
     if  next_count > 0  then
```

```
        V(next);
     else
        V(mutex);
    P(x_sem);
    x_count := x_count -1;

For each signal on condition x

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

Question 5.6

```
interface module  semaphores;
     define  semaphore, P, V, init;
     type  semaphore =  record  taken : boolean;
                                 free : signal;
                       end ;

     procedure  P(var s:semaphore);
     begin
         if  s.taken  then  wait(s.free)  end;
        s.taken := true;
     end  P;


     procedure  V( var  s:semaphore);
     begin
        s.taken := false;
        send(s.free);
     end ;

     procedure  init( var  s:semaphore)
     begin
      s.taken := false;
     end ;

end ;
```

To extend to the general semaphore the boolean becomes a count. Init takes an initial value of type integer. P only blocks when the count is 0.

Question 5.7

The criticism of condition synchronisation is that wait and signal are like sempahore operations and therefore unstructured. A signal on the wrong condition may be difficult to detect. They are therefore too low level and unstructured.

The WaitUntil primitive removes the need for condition variables and therefor wait and signal. the problem how to implement it. If a process is blocked because its boolean expression is false then it must give up the monitor lock to allow other processes to enter and change the variables in the expression. So the issue is when to re-evaluate the guards. As the monitor does not know if the variables in a boolean expression have been altered, all boolean expressions must be revaluated EVERY TIME a process releases the monitor lock. This is inefficient and therefore not used.

The object to this approach may be invalid if we have a multi processor system with shared memory where each processor only executes a single process. If this is the case then the process can continually check its guards as the processor is unable to execute another process.

```
buffer_controller :  monitor

   type  buffer_t  is record
     slots      :  array (1..N)  of  character;
     size       : integer  range  0..N;
     head, tail : integer  range  1..N;
   end record ;

  buffer : buffer_t;


   procedure  produce(char : character);
     ...
      WaitUntil  buffer.size < N

     -- place char in buffer
   end ;

   function  consume  return  character;
    ...
     WaitUntil  buffer.size > 0

    -- take char out of buffer;
     return  char;
   end;
 end;
```

Question 5.8

Here we show a solution in the Modula language.

```
MODULE smokers ;
 TYPE
    ingredients = (TOBandMAT,TOBandPAP,PAPandMAT);

 INTERFACE MODULE controller;
    DEFINE  get, put, release;
    USE ingredients;
    VAR
      TP, MP, PP : signal;
      TAM, TAP, PAM : signal;

    PROCEDURE get ( ingred : ingredients);
    BEGIN
```

```
   (* called by smokers *)
   CASE ingred OF
     TOBandMAT: BEGIN
                  (* Paper Process *)
                  IF NOT awaited(PP) THEN
                    wait(TAM)
                  END;
                  (* ingredients available *)
                END;

      TOBandPAP: BEGIN
                  (* Matches Process *)
                  IF NOT awaited(MP) THEN
                    wait(TAP)
                  END;
                  (* ingredients available *)
                END;

      PAPandMAT: BEGIN
                  (* Tobacco Process *)
                  IF NOT awaited(TP) THEN
                    wait(PAM)
                  END;
                  (* ingredients available *)
                END
     END
 END get;

 PROCEDURE release ( ingred : ingredients);
 BEGIN
   (* called by smokers *)
   (* releases agent *)
      CASE ingred OF
         TOBandMAT: BEGIN
              (* Paper Process *)
              send (PP)
         END;

         TOBandPAP: BEGIN
              (* Matches Process *)
              send (MP)
         END;

         PAPandMAT: BEGIN
              (* Tobacco Process *)
              send (TP)
         END
       END
END release;

PROCEDURE put ( ingred : ingredients);
BEGIN
  (* called by agent *)
  (* place ingredients on the table *)
  CASE ingred OF
     TOBandMAT: BEGIN
           (* looking for Paper Process *)
           IF awaited(TAM) THEN
```

```
                send(TAM)
              END;
              wait(PP)
        END;

        TOBandPAP: BEGIN
              (* Matches Process *)
              IF awaited(TAP) THEN
                 send(TAP)
              END;
              wait(MP)
         END;

         PAPandMAT: BEGIN
               (* looking for Tobacco Process *)
               IF awaited(PAM) THEN
                  send(PAM)
               END;
               wait(TP)
         END
      END
    END put;
 END controller;

 PROCESS smoker (requires : ingredients);
 BEGIN
   LOOP
     controller.get(requires);
     (* roll and smoke cigarette *)
     controller.release(requires)
   END
 END smoker;

BEGIN
    smoker(TOBandMAT);
    smoker(TOBandPAP);
    smoker(PAPandMAT)
END smokers.
```

Question 5.10

```
#include "mutex.h"

typedef struct
  pthread_mutex_t mutex;
  pthread_cond_t ok_to_read;
  pthread_cond_t ok_to_write;
  int readers;
  int writing;
  int waiting_writers;
  /* data */
   shared_data;

int start_read(shared_data *D)
  PTHREAD_MUTEX_LOCK(&D->mutex);
  while(D->writing > 0 || D->waiting_writers > 0)
```

```
    PTHREAD_COND_WAIT(&(D->ok_to_read), &(D->mutex));

  D->readers++;
  PTHREAD_MUTEX_UNLOCK(&D->mutex);
  return 0;
;

int end_read(shared_data *D)
  PTHREAD_MUTEX_LOCK(&D->mutex);
  if(--D->readers == 0 )
    PTHREAD_COND_SIGNAL(&D-> ok_to_write);
  ;
  PTHREAD_MUTEX_UNLOCK(&D->mutex);
  return 0;


int start_write(shared_data *D)
  PTHREAD_MUTEX_LOCK(&D->mutex);
  D->waiting_writers++;
  while(D->writing > 0 || D->readers > 0)
    PTHREAD_COND_WAIT(&(D->ok_to_write), &(D->mutex));

  D->writing++;
  D->waiting_writers--;
  PTHREAD_MUTEX_UNLOCK(&D->mutex);
  return 0;
;


int end_write(shared_data *D)
  PTHREAD_MUTEX_LOCK(&D->mutex);
  D->writing = 0;
  if(D->waiting_writers != 0 )
    PTHREAD_COND_SIGNAL(&D-> ok_to_write);

  else
    PTHREAD_COND_BROADCAST(&D-> ok_to_read);

  PTHREAD_MUTEX_UNLOCK(&D->mutex);
  return 0;


int main()
  /* ensure all mutexes and condition
     variables are initialised */
```

Question 5.11

```
#include "mutex.h"

const int N = 32;
typedef struct
  pthread_mutex_t mutex;
  pthread_cond_t free;
  int free_resources;
```

```
    resource;

void allocate(int size, resource *R)
  pthread_mutex_lock(&R->mutex);
    while(size > (R->free_resources))
       pthread_cond_wait(&(R->free), &(R->mutex));

    R->free_resources = R->free_resources - size;
  pthread_mutex_unlock(&R->mutex);


void deallocate(int size, resource *R)
  pthread_mutex_lock(&R->mutex);
    R->free_resources = R->free_resources + size;
    pthread_cond_broadcast(&R-> free);
  pthread_mutex_unlock(&R->mutex);


void initialize(resource *R)
  R->free_resources = N;
  /* initialise mutex and condition variables */
```

Question 5.14

```
with ada.text_io; use ada.text_io;
with ada.exceptions; use ada.exceptions;
with ada.numerics.discrete_random;
procedure smokers2 is

  type need is (t_p, t_m, m_p);
  package smoking_io is new enumeration_io(need);
  package random_ingredients is new
          ada.numerics.discrete_random(need);

  task type smoker(my_needs : need);

  task active_agent;

  protected agent is
    procedure give(ingredients : need);
    entry give_matches(n : need; ok : out boolean);
    entry give_paper(n : need; ok : out boolean);
    entry give_tobacco(n : need; ok : out boolean);
    procedure cigarette_finished;
    entry wait_smokers;
  private
    t_available, m_available,
      p_available : boolean := false;
    allocated_t, allocated_p,
      allocated_m : boolean;
    all_done : boolean := false;
    ingredients : need;
  end agent;

  task body smoker is
```

```
    got : boolean;
  begin
    smoking_io.put(my_needs);
    loop

      got := false;

      case my_needs is
         when t_p =>
           while not got loop
             agent.give_tobacco(my_needs, got);
             delay 0.1;
           end loop;
           agent.give_paper(my_needs, got);
         when t_m =>
           while not got loop
             agent.give_tobacco(my_needs, got);
             delay 0.1;
           end loop;
           agent.give_matches(my_needs, got);
         when m_p =>
           while not got loop
             agent.give_matches(my_needs, got);
             delay 0.1;
           end loop;
           agent.give_paper(my_needs, got);
       end case;
      if not got then raise program_error; end if;
      -- make and smoke cigarette
      smoking_io.put(my_needs); put_line(" smoking!");
      delay 1.0;
      agent.cigarette_finished;
    end loop;
  exception
    when e: others =>
       put(exception_name(e));
       put(" exception caught in ");
      smoking_io.put(my_needs);put_line("smoker");
  end smoker;

  task body active_agent is
    gen : random_ingredients.generator;
    ingredients : need;
  begin
    random_ingredients.reset(gen);
    loop
      -- chose two items randomly and set
      -- t_available, m_available, p_available to
      -- true or false correspondingly
      ingredients := random_ingredients.random(gen);
      agent.give(ingredients);
      agent.wait_smokers;
    end loop;
  end active_agent;

  protected body agent is
     procedure give(ingredients : need) is
```

```
      begin
        case ingredients is
          when t_p =>
             t_available := true;
             p_available := true;
             m_available := false;
             put("agent has ");
          when t_m =>
             t_available := true;
             m_available := true;
             p_available := false;
          when m_p =>
             m_available := true;
             p_available := true;
             t_available := false;
        end case;
        put("agent has "); smoking_io.put(ingredients);
        put_line(" for smokers");
        allocated_t := false;
        allocated_p := false;
        allocated_m := false;
      end give;

      entry give_tobacco(n : need; ok : out boolean)
            when t_available and not allocated_t is
      begin

        if (allocated_m and n = t_m) or
           (allocated_p and n = t_p ) or
           (m_available and n = t_m) or
           (p_available and n = t_p) then
               ok := true;
               allocated_t := true;
        else
               ok := false;
        end if;
             if ok then
                  put("agent: given out tobacco to ");
                  smoking_io.put(n); put_line(" smoker");
             else
                  put("agent: refusing tobacco to  ");
                  smoking_io.put(n); put_line(" smoker");
             end if;
       end give_tobacco;

       entry give_matches(n : need;  ok : out boolean)
             when m_available and not allocated_m is
       begin
          if (allocated_t and n = t_m) or
             (allocated_p and n = m_p) or
             (t_available and n = t_m) or
             (p_available and n = m_p) then
               ok := true;
               allocated_m := true;
             else
               ok := false;
             end if;
```

```
            if ok then
                 put("agent: given out matches to ");
                 smoking_io.put(n); put_line(" smoker");
            else
                 put("agent: refusing matches to  ");
                 smoking_io.put(n); put_line(" smoker");
            end if;
       end give_matches;

       entry give_paper(n : need;  ok : out boolean)
         when p_available  and not allocated_p is
       begin
         if (allocated_m and n = m_p) or
            (allocated_t and n = t_p) or
            (m_available and n = m_p) or
            (t_available and n = t_p )  then
              ok := true;
              allocated_p := true;
            else
              ok := false;
            end if;
            if ok then
                 put("agent: given out paper to ");
                 smoking_io.put(n); put_line(" smoker");
            else
                 put("agent: refusing paper to  ");
                 smoking_io.put(n); put_line(" smoker");
            end if;
        end give_paper;

        procedure cigarette_finished is
        begin
          all_done := true;
        end;

        entry wait_smokers when all_done is
        begin
          all_done := false;
        end wait_smokers;

  end agent;

  tp : smoker(t_p);
  tm : smoker(t_m);
  mp: smoker(m_p);

begin
   null;
end smokers2;
```

Question 5.17

Client tasks call the Wait entry and are queued until there are "Needed" number of tasks. At which point they are all released. The value of "Needed" is determined when an instance of the protected type is created.

```
with Pthreads; use Pthreads;
```

```
package Barriers is

  type Barrier(Needed : Positive) is limited private;

  procedure Wait(B : in out Barrier);
  procedure Initialise(B : in out Barrier);

private

  type Barrier(Needed : Positive) is
    record
      M : Mutex_T;
      C : Cond_T;
      Arrived : Natural;
    end record;

end Barriers;

package body Barriers is


  procedure Wait(B : in out Barrier) is
  begin
    mutex_lock(B.M);
    B.Arrived := B.Arrived + 1;
    if Arrived = B.Needed then
      B.Arrived := 0;
      Cond_Broadcast(B.C);
    else
      Cond_Wait(B.C, B.M);
    end if;
    mutex_unlock(B.M);
  end Wait;


  procedure Initialise(B : in out Barrier) is
  begin
    Mutex_Initialise(B.M);
    Cond_Initialise(B.C);
  end Initialise;

end Barriers;
```

Question 5.18

```
package body MULTICAST is

  package COND_SEM is new SEMAPHORE_PACKAGE(0);
  package BINARY_SEM is new SEMAPHORE_PACKAGE(1);

  use BINARY_SEM; use COND_SEM;

  DATA_AVAILABLE : COND_SEM.SEMAPHORE;
  MUTEX : BINARY_SEM.SEMAPHORE;
  WAITING : INTEGER := 0;
```

```

  THE_DATA : INTEGER;
  procedure SEND(I : INTEGER) is
  begin
    WAIT(MUTEX);
      THE_DATA := I;
      if WAITING /= 0 then
        SIGNAL(DATA_AVAILABLE);
      else
        SIGNAL(MUTEX);
      end if;
  end;

  procedure RECEIVE(I: out INTEGER) is
  begin
    WAIT(MUTEX)
      WAITING := WAITING + 1;
    SIGNAL(MUTEX);
    WAIT(DATA_AVAILABLE);
    WAITING := WAITING -1;
    I := THE_DATA;
    if WAITING /= 0 then
      SIGNAL(DATA_AVAILABLE);
    else
       SIGNAL(MUTEX)
    end if;
  end RECEIVE;

end MULTICAST;
```

Question 5.19

```
package body BROADCAST is

  package COND_SEM is new SEMAPHORE_PACKAGE(0);
  package BINARY_SEM is new SEMAPHORE_PACKAGE(1);

  use BINARY_SEM; use COND_SEM;

  DATA_AVAILABLE : COND_SEM.SEMAPHORE;
  RECEIVERS_READY : COND_SEM.SEMAPHORE;
  MUTEX : BINARY_SEM.SEMAPHORE;
  NEW_SEND : BINARY_SEM.SEMAPHORE;
  WAITING : INTEGER := 0;

  THE_DATA : INTEGER;

  procedure SEND(I : INTEGER) is
  begin
    WAIT(NEW_SEND);
    WAIT(MUTEX);
      THE_DATA := I;
      if WAITING = 10 then
        SIGNAL(DATA_AVAILABLE);
      else
        SIGNAL(MUTEX);
        WAIT(RECEIVERS_READY);
```

```
      end if;
    SEND(NEW_SEND);
  end;

  procedure RECEIVE(I: out INTEGER) is
  begin
    WAIT(MUTEX)
      WAITING := WAITING + 1;
      if WAITING = 10 then SIGNAL(RECEIVERS_READY);
      else
        SIGNAL(MUTEX);
        WAIT(DATA_AVAILABLE);
      end if;
      I := THE_DATA;
      WAITING := WAITING -1;
      if WAITING /= 0 then
        SIGNAL(DATA_AVAILABLE);
      else
       SIGNAL(MUTEX)
      end if;
  end RECEIVE;

end BROADCAST;
```

Question 5.20

```
protected Total_Count is
  procedure Car_In(Upper_Just_Passed : out Boolean);
  procedure Car_Out(Lower_Just_Passed : out Boolean);
private
  Total_Cars : Natural := 0;
  Upper_Threshold_Passed : Boolean := False;
  Lower_Threshold_Passed : Boolean := False;
end Total_Count;

protected body Total_Count is
  procedure Car_In(Upper_Just_Passed : out Boolean) is
  begin
    Total_Cars := Total_Cars + 1;
    if Total_Cars >= Maximum_Cars_In_City_For_Red_Light
       and then not Upper_Threshold_Passed then
      Upper_Threshold_Passed := True;
      Upper_Just_Passed := True;
      Lower_Threshold_Passed := False;
    else
      Upper_Just_Passed := False;
    end if;
  end Car_In;

  procedure Car_Out(Lower_Just_Passed : out Boolean) is
  begin
    Total_Cars := Total_Cars - 1;
    if Total_Cars <= Minimum_Cars_In_City_For_Green_Light
       and then not Lower_Threshold_Passed then
      Lower_Just_Passed := True;
      Lower_Threshold_Passed := True;
    UpperThreshold_Passed := False;
```

```
    else
      Lower_Threshold_Passed := False;
    end if;
  end Car_out;
end Total_Count;

task body Bar_Controller is
  City_Just_Full : Boolean;
  City_Just_Space : Boolean;
begin
  loop
    City_Just_Full := False;
    City_Just_Space := False;
    select
      accept Car_Entered do
        Total_Count.Car_In(City_Just_Full);
      end Car_Entered;
    or
      accept Car_Exited do
        Total_Count.Car_Out(City_Just_Space);
      end Car_Exited;
    end select;
    if City_Just_Full then
      City_Traffic_Lights_Controller.City_Is_Full;
    elsif City_Just_Space then
      City_Traffic_Lights_Controller.City_Has_Space;
    end if;
  end loop;
end Bar_Controller;
```

Question 5.22

```
public class ReadersWriters2
{

  private int readers = 0;
  private int waitingReaders = 0;
  private int waitingWriters = 0;
  private boolean writing = false;

  ConditionVariable OkToRead = new ConditionVariable();
  ConditionVariable OkToWrite = new ConditionVariable();

  public void startWrite() throws InterruptedException
  {
    synchronized(OkToWrite) // get lock on condition variable
    {
      synchronized(this) // get monitor lock
      {
        if(writing | readers > 0 | waitingReaders > 0) {
          waitingWriters++;
          OkToWrite.wantToSleep = true;
        } else  {
          writing = true;
          OkToWrite.wantToSleep = false;
        }
      } //give up monitor lock
```

```
    if(OkToWrite.wantToSleep) {
      OkToWrite.wait();
    }
  }
}

public void stopWrite()
{
System.out.println("StopWrite␣Called␣");

  synchronized(OkToRead)

  {
    synchronized(OkToWrite)
    {
      synchronized(this)
      {
        if(waitingReaders > 0) {
          writing = false;
          readers = waitingReaders;
          waitingReaders = 0;
          OkToRead.notifyAll();
        } else if(waitingWriters > 0) {
          waitingWriters--;
          OkToWrite.notify();
        } else writing = false;
      }
    }
  }
}

public synchronized void startRead()
       throws InterruptedException
{
  synchronized(OkToRead) {
    synchronized(this)
    {
      if(writing) {
        waitingReaders++;
        OkToRead.wantToSleep = true;
      } else {
        readers++;
        OkToRead.wantToSleep = false;
      }
    }
    if(OkToRead.wantToSleep) {
      OkToRead.wait();
    }
  }
}

public synchronized void stopRead()
{
  synchronized(OkToWrite)
  {
    synchronized(this)
    {
      readers--;
```

```
        if(readers == 0 & waitingWriters > 0) {
          waitingWriters--;
          writing = true;
          OkToWrite.notify();
        }
      }
    }
  }
}
```

Question 5.23

```
public class ResourceManager
{

  private final int maxResources = 15;
  protected int resourcesFree;

  public ResourceManager()
  {
    resourcesFree = maxResources;
  }

  public synchronized void allocate(int size) throws
    TooManyResourcesRequested, InterruptedException

  {
    if(size > maxResources) throw new
                      TooManyResourcesRequested();
    while(size > resourcesFree) {
       wait();
    }
    resourcesFree = resourcesFree - size;
  }

  public synchronized void deallocate(int size)
  {
    resourcesFree = resourcesFree + size;
    System.out.println("resources␣left␣" + resourcesFree);
    notifyAll();
  }
}
```

Question 5.24

```
public class QuantitySemaphore
{

  int value;

  public QuantitySemaphore(int I)
  {
    value = I;
  }

  public synchronized void wait(int I)
```

```
  {
    try {
      while(I > value) wait();
      value = value - I;
    }
    catch (Exception E) { };
  }

  public synchronized void signal(int I)
  {
    value = value + I;
    notifyAll();
  }
}
```

Question 5.25

Given the class specification, it would be very messy to implement the algorithm. It would be necessary to keep a count of high priority waiters. The signaller would notifyAll. All low priority waiters would wait again if there were high priority waiters. High priority waiters would need to decide amongst themselves wich one should continue, the others would wait again.

If we change the specification so that the methods are not synchronized, then the following solution is possible.

```
public class Event
{
  private int highPriorityWaiting;
  private int lowPriorityWaiting;
  private ConditionVariable highWaiter;
  private ConditionVariable lowWaiter;

  public Event()
  {
    highPriorityWaiting = 0;
    lowPriorityWaiting = 0;
    highWaiter = new ConditionVariable();
    lowWaiter = new ConditionVariable();
  }

  public  void highPriorityWait()
  {
    synchronized(highWaiter) {
      synchronized(this) {
        highPriorityWaiting++;
      }
      try {
        highWaiter.wait();
      } catch(Exception E) {};
    }
  };

  public synchronized void lowPriorityWait()
  {
    synchronized(lowWaiter) {
```

```
      synchronized(this) {
        lowPriorityWaiting++;
      }
      try {
        lowWaiter.wait();
      } catch(Exception E) {};
    }
  };
  public synchronized void signalEvent()
  {
    synchronized(highWaiter) {
      synchronized(lowWaiter) {
        synchronized(this) {
            if(highPriorityWaiting > 0) {
              highWaiter.notify();
              highPriorityWaiting--;
            } else if (lowPriorityWaiting > 0) {
              lowWaiter.notify();
              lowPriorityWaiting--;
            }
        }
      }
    }

  };
}
```

To modify this, so that an individual thread is woken, requires a notifyAll, and each thread to test an indended ID with their own Id. Messy.

Question 6.1

Yes: protected objects are flexible, they can implement semaphores, and therefore can implement the same expressive power as the rendezvous.

No: Although it may have the same expressive power, it is not easy to program a selective waiting construct. It is not possible for example to have a multi-way select and therefore the "ease of use" is sacrificed.

Question 6.2

NO, each queue is in FIFO order, there is no time information available.

Question 6.3

```
subtype Binary_Value is Integer range 0 ..1 ;
task type semaphore(Value :
          Binary_Value := 0)  is
  entry P;
  entry V;
end semaphore;
```

```
task body semaphore is
begin
  loop
    select
      when value = one =>
        accept P;
        value := 0;
      or
        accept V;
        value := 1;
      or
         terminate;
    end select;
  end loop;
end semaphore;
```

If task is aborted, the semaphore is deadlocked.

Question 6.4

A tasking implementation might be more expensive but it allows the semaphore to timeout on the signal operation and release it for other tasks.

Question 6.5

Assuming a task which implements a sempahore.

```
For mutual exclusion:
    mutex : semaphore;
For signalling processes suspended:
    next  : semaphore;
For each condition:
    x_cond : semaphore;

next and x_cond must be initialised so
    next.P;
    x_cond.P;

For each procedure:
    mutex.P;
        body;
     if  next.P'count > 0  then
        next.V;
     else
        mutex.V
     end if

For each wait on condition x
     if  next.wait'count > 0  then
        next.V;
     else
        mutex.V;
    x_cond.P;

For each signal on condition x
     if  x_cond.P > 0  then
```

```
    x_cond.V;
    next.P;
  end if ;
```

This will not work because of 'count; we must keep explicit count to avoid the race condition.

Question 6.7

```
task ENTRY_DETECTOR;
task EXIT_DETECTOR;
task LIGHTS_CONTROLLER is
   entry CARS_LEFT(X : NATURAL);
   entry CARS_ARRIVED(X : NATURAL);
end LIGHTS_CONTROLLER;

task body ENTRY_DETECTOR is
   TMP : NATURAL;
begin
  loop
    TMP := CARS_ENTERED;
    if TMP > 0 then
        LIGHTS_CONTROLLER.CARS_ARRIVED(TMP);
    end if;
    delay 10.0;
  end loop;
end ENTRY_DETECTOR;
```

```
task body EXIT_DETECTOR is
   TMP : NATURAL;
begin
  loop
    TMP := CARS_EXITED;
    if TMP > 0 then
        LIGHTS_CONTROLLER.CARS_LEFT(TMP);
    end if;
    delay 10.0;
  end loop;
end EXIT_DETECTOR;
```

```
task body LIGHTS_CONTROLLER is
    N : constant NATURAL := ??;
       -- varies according to tunnel
    CURRENT : NATURAL := 0;
begin
  loop
    select
      accept CARS_ARRIVED(X : NATURAL) do
        CURRENT := CURRENT + X;
      end CARS_ARRIVED;
      if CURRENT > N then
        SET_LIGHTS(RED);
      end if;
    or
     accept CARS_LEFT(X : NATURAL) do
       CURRENT := CURRENT - X;
```

```
      end CARS_LEFT;
      if CURRENT < N then
        SET_LIGHTS(GREEN);
      end if;
    or terminate;
    end select;
  end loop;
end LIGHT_CONTROLLER;
```

Question 6.8

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Exceptions; use Ada.exceptions;
with Ada.Numerics.Discrete_Random;
procedure smokers is


  type NEED is (T_P, T_M, M_P);
  package Smoking_io is new Enumeration_IO(NEED);
  package random_ingredients is
          new Ada.Numerics.Discrete_random(Need);

  task type Smoker(MY_NEEDS : NEED);

  task AGENT is
    entry GIVE_MATCHES(N : NEED; OK : out BOOLEAN);
    entry GIVE_PAPER(N : NEED; OK : out BOOLEAN);
    entry GIVE_TOBACCO(N : NEED; OK : out BOOLEAN);
    entry CIGARETTE_FINISHED;
  end AGENT;

  task body Smoker is
    GOT : BOOLEAN;
  begin
    Smoking_io.put(My_Needs);
    loop

      GOT := FALSE;

      case My_Needs is
         when T_P =>
           while not GOT loop
             AGENT.GIVE_TOBACCO(MY_NEEDS, GOT);
           end loop;
           AGENT.GIVE_PAPER(MY_NEEDS, GOT);
         when T_M =>
           while not GOT loop
             AGENT.GIVE_TOBACCO(MY_NEEDS, GOT);
           end loop;
           AGENT.GIVE_Matches(MY_NEEDS, GOT);
         when M_P =>
           while not GOT loop
             AGENT.GIVE_Matches(MY_NEEDS, GOT);
           end loop;
           AGENT.GIVE_PAPER(MY_NEEDS, GOT);
       end case;
      if not GOT then raise PROGRAM_ERROR; end if;
```

```
        -- make and smoke cigarette
        Smoking_io.put(My_Needs); put_line(" Smoking!");
        delay 1.0;
        AGENT.CIGARETTE_FINISHED;
      end loop;
    exception
      when e: others =>
         put(Exception_Name(e));
         put(" exception caught in ");
        Smoking_io.put(My_Needs);put_line("smoker");
    end Smoker;

    TP : Smoker(T_P);
    TM : Smoker(T_M);
    MP: Smoker(M_P);

    task body AGENT is
      T_AVAILABLE, M_AVAILABLE,
        P_AVAILABLE : Boolean;
      ALLOCATED_T, ALLOCATED_P,
        ALLOCATED_M : BOOLEAN;
      Gen : random_ingredients.generator;
      Ingredients : Need;
    begin
      random_ingredients.reset(gen);
      loop
        -- chose two items randomly and set
        -- T_AVAILABLE, M_AVAILABLE, P_AVAILABLE to
        -- TRUE or FALSE correspondingly
        Ingredients := random_ingredients.random(Gen);
        case Ingredients is
           when T_P =>
              T_AVAILABLE := True;
              P_AVAILABLE := True;
              M_AVAILABLE := False;
              put("Agent has ");
           when T_M =>
              T_AVAILABLE := True;
              M_AVAILABLE := True;
              P_AVAILABLE := False;
           when M_P =>
              M_AVAILABLE := True;
              P_AVAILABLE := True;
              T_AVAILABLE := False;
         end case;
         put("Agent has "); Smoking_io.put(Ingredients);
         put_line(" for smokers");
        ALLOCATED_T := FALSE;
        ALLOCATED_P := FALSE;
        ALLOCATED_M := FALSE;
        loop
          select
           when T_AVAILABLE and not ALLOCATED_T =>
            accept GIVE_TOBACCO(N : NEED;
                   OK : out BOOLEAN) do
              if (Allocated_M and N = T_M) or
                 (Allocated_P and N = T_P ) or
```

```
              (M_AVAILABLE and N = T_M) or
              (P_AVAILABLE and N = T_P) then
             OK := TRUE;
             ALLOCATED_T := TRUE;
           else
             OK := FALSE;
           end if;
           if OK then
                put("Agent: given out Tobacco to ");
                Smoking_io.put(N); put_line(" Smoker");
           else
                put("Agent: refusing Tobacco to  ");
                Smoking_io.put(N); put_line(" Smoker");
           end if;
         end GIVE_TOBACCO;
       or
        when M_AVAILABLE and not ALLOCATED_M =>
         accept GIVE_MATCHES(N : NEED;
                OK : out BOOLEAN) do
           if (Allocated_T and N = T_M) or
              (Allocated_P and N = M_P) or
              (T_AVAILABLE and N = T_M) or
              (P_AVAILABLE and N = M_P) then
             OK := TRUE;
             ALLOCATED_M := TRUE;
           else
             OK := FALSE;
           end if;
           if OK then
                put("Agent: given out Matches to ");
                Smoking_io.put(N); put_line(" Smoker");
           else
                put("Agent: refusing Matches to  ");
                Smoking_io.put(N); put_line(" Smoker");
           end if;
         end GIVE_MATCHES;
       or
        when P_AVAILABLE  and not ALLOCATED_P =>
         accept GIVE_PAPER(N : NEED;
                OK : out BOOLEAN) do
           if (Allocated_M and N = M_P) or
              (Allocated_T and N = T_P) or
              (M_AVAILABLE and N = M_P) or
              (T_AVAILABLE and N = T_P )  then
             OK := TRUE;
             ALLOCATED_P := TRUE;
           else
             OK := FALSE;
           end if;
           if OK then
                put("Agent: given out Paper to ");
                Smoking_io.put(N); put_line(" Smoker");
           else
                put("Agent: refusing Paper to  ");
                Smoking_io.put(N); put_line(" Smoker");
           end if;
         end GIVE_PAPER;
```

```
        end select;
        if (ALLOCATED_P and ALLOCATED_T) or
           (ALLOCATED_M and ALLOCATED_T) or
           (ALLOCATED_P and ALLOCATED_M) then
          accept Cigarette_Finished;
          exit;
        end if;
      end loop;
    end loop;
  exception
    when e: others =>
       put(Exception_Name(e));
       put_line(" exception caught in Agent");
  end AGENT;
begin
  null;
end smokers;
```

Question 6.9

```
task body SERVER is
  type SERVICE is (A, B, C);
  next : SERVICE := A;
begin
  loop
    select
      when NEXT = A or
         (next = B and SERVICE_B'count = 0
          and SERVICE_C'count = 0) or
         (next = C and SERVICE_C'count = 0) =>
            accept SERVICE_A do next := B; end;
    or
      when NEXT = B or
         (next = C and SERVICE_A'count = 0
          and SERVICE_C'count = 0) or
         (next = A and SERVICE_A'count = 0) =>
            accept SERVICE_B do next := C; end;

    or
      when NEXT = C or
         (next = A and SERVICE_A'count = 0
         and SERVICE_B'count = 0) or
         (next = B and SERVICE_B'count = 0) =>
           accept SERVICE_C do next := A; end;
        or
      terminate;
    end select;
  end loop;
end SERVER;
```

Question 6.10

(1) If exception A is raised it is trapped inside the rendezvous and therefore the only message to appear is “A trapped in sync”.

(2) If exception B is raised it is trapped inside the rendezvous but then re-raised. This will propagate the exception to task ONE where it

will be trapped and C raised but unhandled (note the exception doesn't propagate to main). The exception will also propagate to block Z is task TWO where it will be handled, the handler however raises exception C which is handled by block Y. The following will therefore be printed: "B trapped in sync", "B trapped in block Z", "C trapped in Y" and "B trapped in one"

(3) If exception C is raised it is trapped inside the rendezvous. The handler then raises D which is propagated the task ONE and block Z. Block Z catches D with when others and then raises C which is trapped by block Y's when others. The following will therefore be printed: "C trapped in sync", "others trapped in Z", "C trapped in Y" and "C trapped in one".

(4) If exception D is raised it is not trapped by sync and therefore propagates to ONE. Block Z catches D with when others and then raises C which is trapped by block Y's when others. The following will therefore be printed: "others trapped in Z", "C trapped in Y" and "D trapped in one".

Question 7.8

Fragment 1
Case (1): Flag = A
Case (2): Flag = B
Case (3): Flag = A
Case (4): Flag = A
Fragment 2
Case (1): Flag = A
Case (2): Flag = B
Case (3): Flag = B
Case (4): Flag = B
Fragment 3
Case (1): Flag = A
Case (2): Flag = B
Case (3): Flag = A
Case (4): Flag = A
Fragment 4
Case (1): Flag = B
Case (2): Flag = A
Case (3): Flag = B
Case (4): Flag = A

1 Question 7.9

```
protected Controller is
  entry Stop(At_Location : out Array_Bounds);
  procedure Found(At_Location : in Array_Bounds);
private
  Found_At : Array_Bounds;
  Found_String : Boolean := False;
```

```
end Controller;

protected body Controller is
  entry Stop(At_Location : out Array_Bounds)
        when Found_String is
  begin
    At_Location := Found_At;
  end Stop;

  procedure Found(At_Location : in Array_Bounds) is
  begin
    Found_At := At_Location;
    Found_String := True;
  end Found;
end Controller;

task body Searcher is
  Found : Boolean := false;
  At_Loc : Array_Bounds;
  Str : Search_String;
begin
  accept Find (Looking_For : Search_String) do
    Str := Looking_For;
  end Find;

  loop
    select
      Controller.Stop(At_Location);
    then abort
      Search_Support.Search(Search_Array, Lower, Upper,
                            Str, Found, At_Loc);
      if Found then
        Controller.Found(At_Loc);
      end if;
    end select;
    -- At_Loc is location of string

    select
      accept Get_Result (At_Location : out Array_Bounds) do
        At_Location = At_Loc;
      end select;
    or
      accept Find (Looking_For : Search_String) do
        Str := Looking_For;
      end Find;
    end select;
  end loop;
end Searcher;
```

Question 7.10

There are three possible interleavings of interest.

1. `Error_1` is raised before `Watch` executes its raise statement. In this case, the then abort clause is abandoned and the message "Error_1 Caught" printed.

2. `Error_2` is raised before `Signaller` calls `Go` and causes `Error_1` to be raised. Furthermore, the exception propagates outside the select statement before this happens. In this case, entry is cancelled and the message "Error_2 Caught" printed.
3. `Error_2` is raised before `Signaller` calls `Go` and causes `Error_1` to be raised. However, before the exception propagates outside the select statement, `Error_1` is raised. In this case, the then abort clause is abandoned, the `Error_2` exception lost and the message "Error_1 Caught" printed.

Question 7.11

```
#include "sig.h"


#define MODE_A 1
#define MODE_B 2
#define MODE_CHANGE SIGRTMIN +1

int mode = MODE_A;

void change_mode(int signum, siginfo_t *data, void *extra)
  mode = data -> si_value.sival_int;


int main2()

  sigset_t mask, omask, allmask;
  struct sigaction s, os;
  int local_mode;

  s.sa_flags = 0;
  s.sa_mask = mask;
  s.sa_sigaction = & change_mode;


  /* mask used to mask out mode changes whilst accessing */
  /* current mode */
  sigemptyset(&mask);
  sigaddset(&mask, MODE_CHANGE);

  sigaction(MODE_CHANGE, &s, &os);

  /* allmask used to mask all signals except mode change, */
  /* whilst waiting for the new mode */
  sigfillset(&allmask);
  sigdelset(&mask, MODE_CHANGE);

  while(1)

    sigprocmask(SIG_BLOCK, &mask, &omask);
      local_mode = mode;
    sigprocmask(SIG_UNBLOCK, &mask, &omask);
```

```
    /* periodic operation using mode*/

    if(local_mode != MODE_A)
      /* wait for mode change */
      sigsuspend(&allmask);
     else
      /* code for mode A */
      WAIT_NEXT_PERIOD;


  return 0;
```

Question 8.1

It will fail because of the execution of the select statement is not an atomic operation. It is possible for a task calling the urgent entry to timeout or abort after the evaluation of the guards to the medium or low priority entries but before the rendezvous is accepted. If this is the only task on the queue then both the other entries are closed even though there is no available entry on the urgent entry. An entries on the medium and low priority queues are blocked.

The reason you cannot extend the solution to a numeric solution with a range of 0 to 1000 is that it is not practical to enumerate all the members of the family.

```
subtype level is integer range 0 .. 1000;
task controller is
   entry sign_in(l : level)
   entry request(level)(d:data);
end controller;
```

```
task body controller is
   total :integer;
   pending : array (level) of integer;
begin
  -- init pending to 0
  loop
    if total = 0 then
      accept sign_in(l:level) do
         total := total + 1;
         pending(l) := pending + 1;
      end;
    else
      loop
        select
          accept sign_in(l:level) do
            total := total + 1;
            pending(l) := pending + 1;
          end;
        else
          exit;
        end select;
     end loop;

```

```
    for i in level loop
       if pending(i) > 0 then
          accept(i)(d:data) do null end;
          pending(i) := pending -1;
          total := total - 1;
          exit; -- look for new calls
       end if;
    end loop;
  end loop;
end controller;
```

Question 8.2

```
public class ResourceManager
{

  private final int maxResources = 100;
  private int resourcesFree;

  public ResourceManager()
  {
    resourcesFree = maxResources;
  }

  public synchronized void allocate(int size)
         throws IntegerConstraintError
    // see ** for definition of IntegerConstraintError
  {
    if(size > maxResources) throw
      new IntegerConstraintError(1,maxResources, size);
    while(size< resourcesFree) wait();
    resourcesFree = resourcesFree - size;
  }

  public synchronized void free(int size)
  {
    resourcesFree = resourcesFree + size;
    notifyAll();
  }
}
```

Question 8.4

Assuming that tasks are queued on entries in priority order:

```
type request_range is range 1..MAX;

protected resource_controller is
  entry request(R : out resource;
                amount : request_range);
  procedure free(R : resource;
                 amount : request_range);
private
  freed : request_range := request_range'last;
  Queued : Natural := 0;
```

```
  ...
end resource_controller;
```

```
protected body resource_controller is
  entry request(R : out resource;
               amount : request_range)
        when freed > 0 and
             Queued < request'Count is
  begin
    if amount <= freed then
      freed := freed - amount;
      -- allocate
     Queued := 0;
    else
      Queued := request'Count + 1;
      requeue request;
    end if;
  end request;
```

```
  procedure free(R : resource;
                 amount : request_range) is
  begin
    freed := freed + amount;
    -- free resources
    Queued := 0;
  end free;
end resource_controller;
```

Question 8.5

```
#include "mutex.h"

const int N = 32;
typedef struct
  pthread_mutex_t mutex;
  pthread_cond_t free;
  int free_resources;
   resource;

void allocate(int size, resource *R)
  pthread_mutex_lock(&R->mutex);
    while(size > (R->free_resources))
       pthread_cond_wait(&(R->free), &(R->mutex));

    R->free_resources = R->free_resources - size;
  pthread_mutex_unlock(&R->mutex);


void deallocate(int size, resource *R)
  pthread_mutex_lock(&R->mutex);
    R->free_resources = R->free_resources + size;
    pthread_cond_broadcast(&R-> free);
  pthread_mutex_unlock(&R->mutex);
```

```
void initialize(resource *R)
  R->free_resources = N;
  /* initialise mutex and condition variables */
```

Question 9.1

A timing failure is defined to be the delivery of a service outside its defined delivery interval - typically beyond some defined deadline. Often the service is delivered late because of the time needed to construct the correct value for the service. If the system was designed to always deliver a value in the correct interval then the "lack of time" would be manifest as an incorrect value (delivered on time). Hence timing and value failures cannot be considered orthogonal.

The converse to the above can also be true. A service that has failed because the value it delivers is incorrect may be able to deliver a correct value if it is given more (CPU) time; hence a correct value may be delivered but too late. However this is not universally true; a value failure (or error) can be due to many reasons (e.g. software error, hardware error) other than insufficient allocation of processor cycles.

Question 11.1

With the rate monotic scheduling approach, all processes are allocated a priority according to their periods. The shorter the period the higher the priority. Process P would there have a higher priority than Q which would have a higher priority than S. A preemptive scheduler is used and therefore the processes would be scheduled in the following order.

time	process	total time for current period
1	P	1
2	Q	1
3	Q	2
4	P	1
5	S	1
6	S	2
7	P	1
8	Q	1
9	Q	2
10	P	1
11	S	3
12	S	4
13	P	1
14	Q	1
15	Q	2
16	P	1
17	S	5
18	idle	

The three processes may be scheduled using the cyclic executive approach by splitting up process S into 5 equal parts S1, S2, S3, S4, and S5. (from the above rate monotonic solution). The loop is given by:

```
loop
  P; Q; P; S1; S2;
  P; Q; P; S3; S4;
  P; Q; P; S5
end loop;
```

Question 11.2

There is actually a typo in this question! Q should be the second most important (after P) and with requirement of 6,1.

a As P has the highest priority it will run first for 30 ms. Then Q will run for 1 ms; unfortunately it has missed its first five deadline at 6ms, 12ms, 18, 24ms and 30ms. S will run last (after 31ms) but have missed its deadline at 25ms.

b Utility of P is 30%. Utility of Q is 16.67%. Utility of S is 20%. Total utility is 66.67%.

c Two approaches could be used. If scheduling is based on earliest deadline then the test is that total utilisation is less than 100priority model is used then the rate monotonic test could be applied. *It will not be assumed that the general test can be remembered by the student, although the lower bound value of 69% should be.* As total untilisation is less than 69% the process set is scheduable. The rate monotonic scheme assigns priorities in an inverse relation to period length.

d For rate monotonic Q will have highest static priority, then S and then P. The execution sequence will be:

Process	Execution Time	Total Time
Q	1	1
S	5	6
Q	1	7
P	5	12
Q	1	13
P	5	18
Q	1	19
P	5	24
Q	1	25
S	5	30
Q	1	31
P	5	36
Q	1	37
P	5	42
Q	1	43
P	5	48
Q	1	49
idle	1	50

For earliest deadline the execution sequence will be the same up to the first idle time.

Question 11.3

At the minimum execution of R its utility is 10% (Total now 76.67%). At the maximum execution R utility is 50% (Total now 116.67%). As R is not safety critical then it must miss its deadline (if any process must). The earliest deadline scheme will not ensure this. The approach that should be taken is to use rate monotonic scheme and to transform P so that its period is less than R; ie P becomes a process with a period of 10 and a requirement of 3ms (per period). With the new scheme Q will still have the highest static priority, then P, then S and lowest priority will go to R. The execution sequence will be:

Process	Execution Time	Total Time
Q	1	1
P	3	4
S	2	6
Q	1	7
S	3	10
P	2	12
Q	1	13
P	1	14
R	4	18
Q	1	19
R	1	20
P	3	23
R	1	24
Q	1	25
S	5	30
Q	1	31
P	3	34
R	2	36
Q	1	37
R	3	40
P	2	42
Q	1	43
P	1	44
R	4	48
Q	1	49
R	1	50

With this scheme S gets 16ms in its first period.

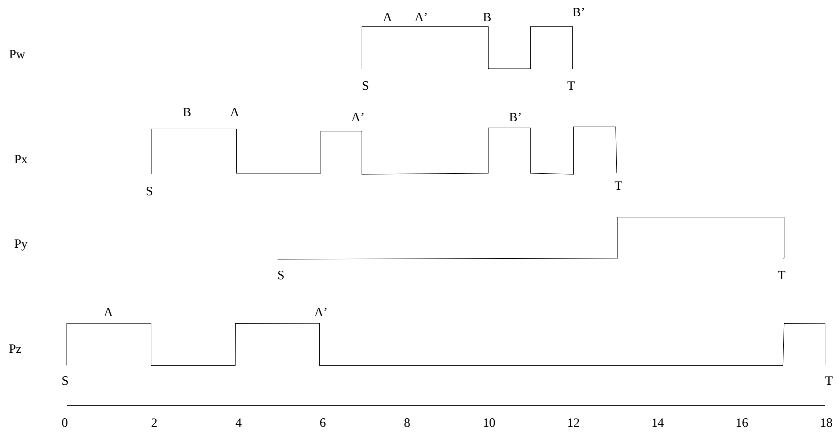
Question 11.4

The new diagram will look as follows:

The result of inheritance is that the highest priority process now finishes at time 12 (rather than 16) and P1 (the next highest) now completes at time 13 (rather than 17). There are also less context switches.

Question 11.6

The rule for calculating blocking is: the maximum critical section with a ceiling equal or higher than the task but used by a task with a priority



lower.

The first part is to calculate the ceilings of each resource. We shall let the letters A,B,C,D,E stand for the tasks and their priorities.

Resource	Used By	Ceiling
R1	B,D	B
R2	B,E	B
R3	A,C	A
R4	C	C
R5	C,D	C
R6	D,E	D

Note R4 is only used by one task and hence can be ignored.

Applying the rule to Task A: R3 has a ceiling equal and is used by C (lower) hence blocking is 75ms.

Task B: R1, R2 and R3 all have higher or equal ceilings and are used by lower; the maximum value is thus 150ms.

Task C: Resources to consider, R1, R2, R5 (not R3 as it is not used by lower); the maximum value is thus 250ms.

Task D: All ceilings are higher (or equal) but only R2 and R6 are used by E; the maximum value is thus 175ms.

The lowest priority task cannot experience a block; hence maximum is 0.

Question 11.7

The task set is schedulable as

U(P1) = 0.2

U(P2) = 0.25

U(P3) = 0.3

Hence U = 0.75 which is below the threshold of 0.780 for thress processes.

Question 11.8

The task set is unschedulable because priorities have been assigned that are not optimal. It must have period transformation applied to it. P1 is transformed to a task that has period 6 and computation time 1. RMS now gives P1 the highest priority (ie P1 is highest and the allocation is optimal).

The utilisation of the task set is .1666 + .333 + .25 which is below the bound for schedulability. Hence system is OK.

Question 11.9

The key here is to note that the formulae is necessary but not sufficient. This means that if a task set passes the test it is schedulable but if it failes the test it may or may not be scheduable. The worst phases for periods is when they are relative primes. The given task set has one task as half the frequency of the other and hence the utilisation bound on this set is above that predicted by the formulae.

Question 11.13

The key here is to recognise that the period of the process has to be at least half the deadline of the event.

Question 11.14

The processes must be given priorities in rate order so C_Event, E_Event, B_Event, A_Event and D_Event is the correct order.

Utilisation for each process must be worked out:

A_Event 11.11%

B_Event 8.33%

C_Event 20%

D_Event 16.67%

E_Event 16.67%

Giving a total of 72.78%

The lower bound test of 69% would imply not schedulable. But for 5 tasks the bound is 74the process set is schedulable

Question 11.15

To schedule optimally the priorities must be assigned by the deadline monotonic rule. This gives process `b` the highest priority, then process `a`, then process `c`. Premptive scheduling must be used. Applying equation the response time equation gives R_b = 4, R_a = 8 and R_c = 29 Hence all tasks are schedulable.

Question 11.16

In any interval being considered, it is possible to calculate the number of time, K, the clock handler could have executed:

$$K = \left\lceil \frac{R_i}{T_{clk}} \right\rceil$$

It is also possible to calculate the number of movements, V, there has been from the delay queue to the dispatch queue:

$$V = \sum_{g \in \Gamma p} \left\lceil \frac{R_i}{T_g} \right\rceil$$

where Γp is the set of periodic tasks.

If $K \geq V$, we must assume, for the worst case, that each movement occurs on a different clock tick (and hence must be costed at CT^c). If this is not the case, a reduuced cost can be used. Hence equation 16.4 becomes:

$$\begin{aligned} R_i &= CS^1 + C_i + B_i \\ &+ \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (CS^1 + CS^2 + C_j) \\ &+ \sum_{k \in \Gamma s} \left\lceil \frac{R_i}{T_k} \right\rceil IH + \left\lceil \frac{R_i}{T_{clk}} \right\rceil CT_c \\ &+ I^{\Gamma p} \end{aligned}$$

$$\begin{aligned} I^{\Gamma p} &= if\ K \geq V : V * CT^s \\ &\quad else\ K * CT^s + (V - K) * CT^m \end{aligned}$$

Question 11.18

This periodic task suffers release jitter. It the worst case this is 20 milliseconds. Use equations 13.11 and 13.12 to calculate response time with this jitter.

Question 12.1

Basically, WCET and blocking time can be wrong, sporadics can be invoked more often that anticipated, and the application periodic processes could compute the wrong delay value. The tools may be wrong.

RTS could handle the timing events for periodics, and check that sporadics don't go off more often than anticipate. Watdog timers?

Still can't really guarantee without memory firewalls.

Question 12.2

With the mode change example, a number of tasks need to have their priorities changed. Typically, these changes should take place atomically (that is, all changed together). To achieve this, a protected object with a high ceiling priority could be used. For example, in the following, a group of `N` tasks can exist in one of four modes. A call of `Set_Mode` will change the priorities of the tasks. Each task must, however, first call `Register`, so that its identity can be held:

```
with Ada.Task_Identification;
with Ada.Dynamic_Priorities;
use Ada.Dynamic_Priorities;
with System;
package Flight_Management is
  N : constant Positive := ...;
  type Task_Range is range 1..N;
  type Mode is (Taxiing, Take_Off, Cruising, Landing);

  Mode_Priorities : array(Task_Range, Mode) of System.Priority;
  -- priorities are set during an
  -- initialisation phase of the program

  type Labels is array(Task_Range) of
       Ada.Task_Identification.Task_Id;

  protected Mode_Changer is
    pragma Priority(System.Priority'Last);
    procedure Register(Name : Task_Range);
    procedure Set_Mode(M : Mode);
  private
    Current_Mode : Mode := Taxiing;
    Task_Labels : Labels;
  end Mode_Changer;
end Flight_Management;
```

The body of `Mode_Changer` will be

```
protected body Mode_Changer is
  procedure Register(Name : Task_Range) is
  begin
    Task_Labels(Name) :=
               Ada.Task_Identification.Current_Task;
  end Register;

  procedure Set_Mode(M : Mode) is
  begin
    if M /= Current_Mode then
      Current_Mode := M;
      for T in Task_Range loop
        Set_Priority(Mode_Priorities(T,M),Task_Labels(T));
      end loop;
    end if;
  end Set_Mode;
end Mode_Changer;
```

Note that this will only work if all the tasks first register.

Question 14.1

```
-- assuming System is already withed and used

Heart_Monitor : constant Interrrupt_Id := ....;

word : constant := 2; -- number of storage units in a word
Bits_In_Word : constant := 16; -- bits in work

type csr is new integer;
for Csr'Size use Bits_In_Word;
for Csr'Alignment use Word;
for Csr'Bit_Order use Low_Order_First;

protected Interrupt_Handler is
  entry wait_heart_beat;
private
  procedure Handler;
  pragma Attach_Handler(Handler, heart_monitor);
  pragma Interrupt_Priority(Interrupt_Priority'Last);
  Interrupt_Occured : Boolean := False;
end Interrupt_Handler;

task patient_monitor;

protected body Interrupt_Handler is
  procedure Handler is
  begin
    Interrupt_Occured := True;
  end handler;

  entry wait_heart_beat when Interrupt_Occured is
  begin
    Interrupt_Occured := False;
  end wait_heart_beat;
end Interrupt_Handle;

task body patient_monitor is
  Control_Reg_Addr : constant Address := 8##177760#;
  ecsr : csr;
  for ecsr'Address use Control_Reg_Addr;
  volts : csr := 5;
begin
  loop
    select
      Interrupt.wait_heart_beat;
      volts := 5;
    or
      delay 5.0;
      select
        supervisor.sound_alarm;
      else
        null;
      end select;
      ecsr := volts;
      volts := volts +1;
    end select;
```

```
  end loop;
end patient_monitor;
```

Question 14.2

This is an Ada 83 solution. The Ada 2005 is TBD.

```
package Motorway_Charges is
end Motorway_Charges;

with Journey_Details; use Journey_Details;
with DISPLAY_INTERFACE; use DISPLAY_INTERFACE;
with system; use system;
package body Motorway_Charges is

  type Transmit_T is (No, Yes);
  for Transmit_T use (No => 0, Yes => 1);

  type CR_T is record
     -- have not used string because of the length tag
    C1 : Character;
    C2 : Character;
    C3 : Character;
    C4 : Character;
    C5 : Character;
    C6 : Character;
    C7 : Character;
    C8 : Character;

    Go : Transmit_T;

    Details : Travel_Details;
    Code : Security_Code;
  end record;

  for CR_T use record at mod 2;
    C1 at 0 range 0 .. 7;
    C2 at 0 range 8 .. 15;
    C3 at 1 range 0 .. 7;
    C4 at 1 range 8 .. 15;
    C5 at 2 range 0 .. 7;
    C6 at 2 range 8 .. 15;
    C7 at 3 range 0 .. 7;
    C8 at 3 range 8 .. 15;

    Go at 4 range  0 .. 0;

    Details at 4 range  1 .. 4;
    Code at 4 range  5 .. 15;
  end record;

  CR : CR_T;
  for CR use at 8#177762#;

  Shadow : CR_T;

  DR : integer;
  for CR use at 8#177760#;
```

```

  Current_Cost : Integer := 0;

  task Handler is
    entry Interrupt;
    for Interrupt use at 8#60#;
    pragma Hardware_Priority(6);
    -- other solutions acceptable
  end Handler;

  task body Handler is
  begin
    Shadow.C1 := Registration_Number(1);
    Shadow.C2 := Registration_Number(2);
    Shadow.C3 := Registration_Number(3);
    Shadow.C4 := Registration_Number(4);
    Shadow.C5 := Registration_Number(5);
    Shadow.C6 := Registration_Number(6);
    Shadow.C7 := Registration_Number(7);
    Shadow.C8 := Registration_Number(8);
    Shadow.Details := Current_Journey;
    Shadow.Code := Code;
    Shadow.Go := Yes;
    loop
      accept Interrupt do
         CR:= Shadow;
         Current_Cost := Current_Cost + DR;
         select
             Display_Driver.Put_Cost(Current_Cost);
         else
             null;
         end select;
      end Interrupt;
    end loop;
  end Handler;

end Motorway_Charges;
```