b)

```
package body Semaphore_Package_2 is
   task type body Semaphore(Initial: Natural := 1) is
      Value : Natural;
   begin
      Value := Initial;
      loop
         select
            when Value > 0 =>
            accept Wait do
            begin
               Value := Value - 1;
            end Wait;
         or
            accept Signal is do
            begin
               Value := Value + 1;
            end Signal;
         or
            terminate;
         end select;
      end loop;
   end Semaphore;
end Semaphore_Package_2;
```

a)
```
package body Semaphore_Package_1 is
   protected type Semaphore(Initial : Natural := 1) is
      entry Wait when Value > 0 is
      begin
         Value := Value - 1;
      end Wait;
      procedure Signal is
      begin
         Value := Value + 1;
      end Signal;
   begin
      Value := Initial;
   end Semaphore;
end Semaphore_Package_1;
```

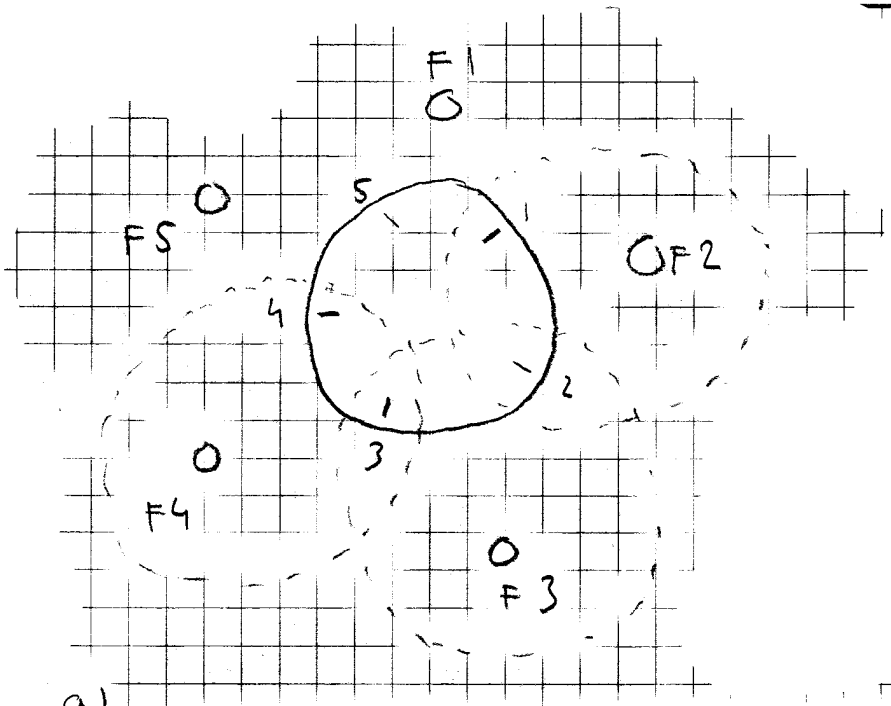
8

c) Första typen är bättre, eftersom den inte är en task (behöver inte schemaläggas, ingen context-switch för att anropa entries o.s.v.)

Rent funktionellt är de lika, men eftersom den skyddade typen tar mindre resurser i anspråk...

4

Poäng på uppgiften (ifylles av lärare)	Löpande sidnr 6
12	Uppgift nr 5



a) F3 kan inte äta då F2 eller F4 äter. Om antingen F2 eller F4 hela tiden äter så kan F3 aldrig äta. 4

b) Lägg till en privat variabel: Chopstick_Control
Num_Eating : Natural := 0;
I entry Take:
Lägg till vakten "when Num_Eating < 2"
Lägg till raden "Num_Eating := Num_Eating + 1;"
efter "begin".
I procedure Drop;
Lägg till "Num_Eating := Num_Eating - 1;"
före "end Drop;"
Då blir resultatet att den som börjat försöka äta alltid får göra det, och FIFO-kön på Take ser till att ingen blir tillbakaskuffad i kön. 8

a)

```
sem_t consumer_lock, producer_lock;

void produce ( int i)
{
    sem_wait ( &producer_lock);
    value = i;
    sem_post ( &consumer_lock);
}

int consume ()
{
    int v;
    sem_wait (&consumer_lock);
    v = value;
    sem_post ( &producer_lock);
    return v;
}

int main (void)
{
    // consume låser från början
    sem_init ( &consumer_lock, 0, 0);
    // produce är upplåst
    sem_init(&producer_lock, 0, 1);
    ... // resten av programmet
}
```

8

b)

```
pthread_cond_t  consumer_lock, producer_lock;

void produce( int v)
{
    pthread_cond_wait (&producer_lock, NULL);
    value = v;
    pthread_cond_signal (&consumer_lock);
}

int consume()
{
    int v;
    pthread_cond_wait (&consumer_lock, NULL);
    v = value;
    pthread_cond_signal (&producer_lock);
    return v;
}

int main(void)
{
    pthread_cond_init( &consumer_lock, NULL);
    pthread_cond_init (&producer_lock, NULL);
    // lås upp producer
    pthread_cond_signal (&producer_lock);
    ... // resten av programmet
}
```

mulex

3