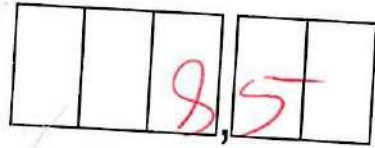




BELOEIL
Sébastien



M1 RS
2013

Exercice 1.

2/9

a) Services : 2 bus de communication unidirectionnel, $E \rightarrow R$ et $R \rightarrow E$
 $E \rightarrow R$ n'est pas fiable.

Hypothèses :

- ~~pas de perte de message~~ $E \rightarrow R$: ~~pas plus de deux~~ ~~consécutives~~ (sans accuser de réception du 2^e message précédent)
- canal de retour fiable (pas de perte / ajout / ...)

pas de duplication, intercalation ; ni en insertion ni en suppression

vocabulaire : ack positif : envoi positif

nack négatif : envoi en erreur

err : erreur.

format :

ack avec bit de priorité pour le message et l'accuse de réception

nack message de retour négatif

err

message d'erreur : valeur propre

regles procedurales

E envoie $\rightarrow R$ ack ou nack

E renvoie si pas de ack ou nack

si R err (dans le cas de deux demandes d'ack avec bit de priorité pour le premier envoi : retour à état stable).

12 fetch.

recepteur

env

ack

nach.

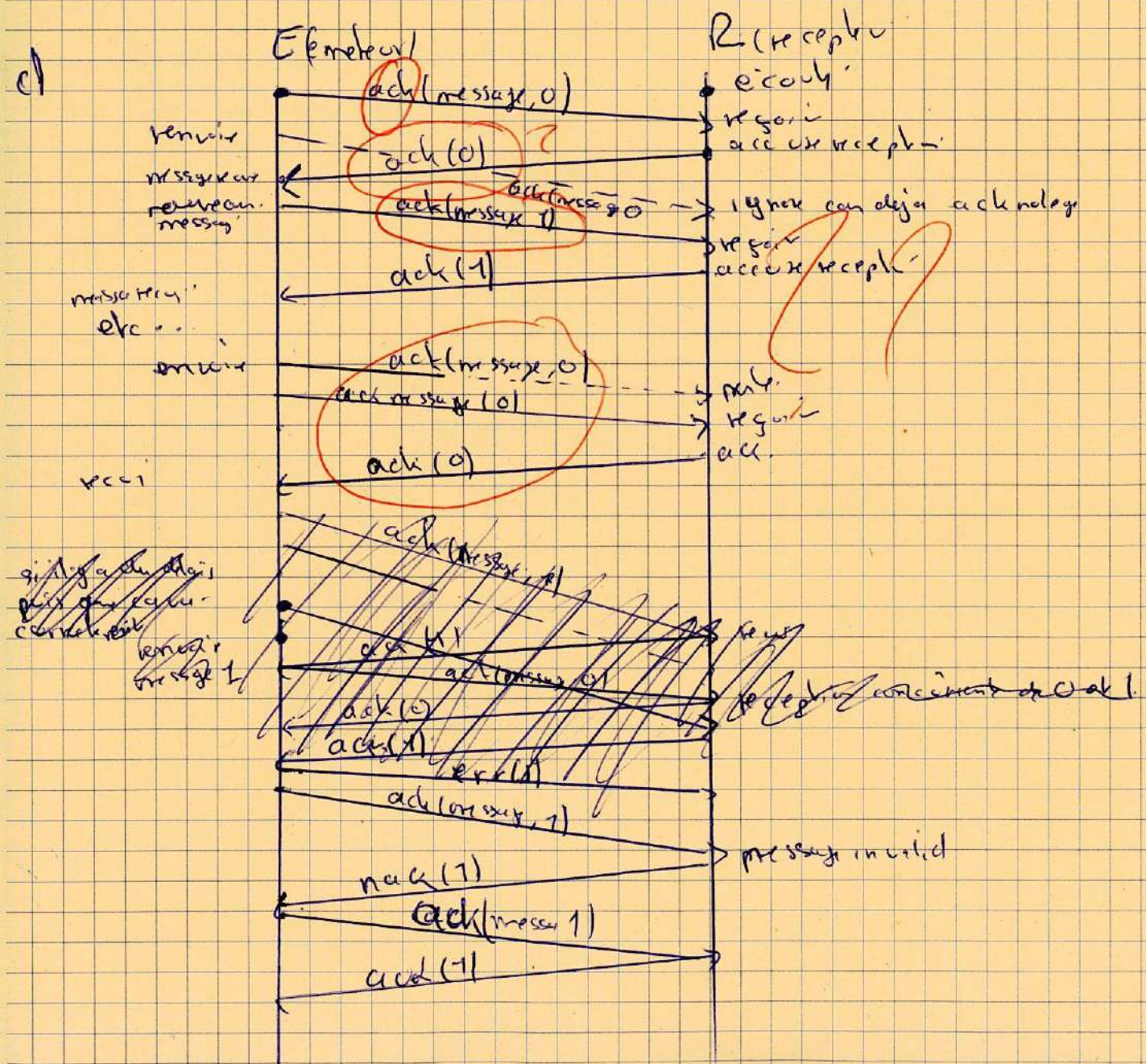
~~reprise~~ transmission

~~ajoutabil~~ de point

renvoi le message

envoi

Incomplete



Exercice 2. 10

```

procedure Enl {
  chan B = message [16];
  chan R = message [8];
}

```

```

procedure Recv {
  chan R = message [16];
  chan B = message [8];
}

```

```


procedure Env {
  message [16] = message;
  chan message [16] = message;
  E! mess;
  return true;
}


```

```


procedure Env {
  Init;
  Env;
  Recv;
  if Env;
}


```

```


procedure Env {
  message [16] = "something";
  ack = true;
  chan message = mess + ack;
  if R? then != ack;
  E! message;
  R? return = ack;
  E! err;
  O! message;
  E! Simon B! message;
  Fi;
  S;
  Init;
  Env;
  while (1) { // ceci n'est pas la bonne façon de faire
    Env; // il aurait fallu gérer les erreurs au client.
  }
}


```

```

procedure Recv {
  local acc;
  chan message [16];
  R? message;
  R! ack message [16]; // dénie le bon message
  // do something with mess;
  if
  :: message correct;
  E! message [16]; // dénie le bon message
  ack = message [16];
  :: message incorrect;
  R! ack;
  ack = message [16];
  ack = !ack;
  E! ack;
  :: message error; return false;
  Fi;
  return true;
}
Init;
Recv;
while (R! = err R);
while (R! = false);
// leg message recv;
}
}

```

whilst (1) { // ceci n'est pas la bonne façon de faire
Env; // il aurait fallu gérer les erreurs au client.
}

3)

Deadlock

20
: lorsque l'on attend par accede a des ressource,
mais que plusieurs entité attende par des ressource on peut
se retrouver dans un etat d'attente entité-attente
indefinite.

exemple

A attend la ressource de B
B de C et C de A. personne n'a rien,

Live lock

Quand plusieurs entité veulent la même ressource
avec des mutex. par exemple A veut M et B et veut M
si A a réservé la ressource B doit attendre que M soit libéré.

Starvation

Lorsque qu'un système arrive à épuiser de ressource
il ne peut plus traiter normalement des messages
c'est un état de starvation

Ingénierie des Protocoles M1

Devoir Écrit

Aucun document ou appareil électronique n'est autorisé.

Renseignez vos nom, prénom.

NOM (en majuscules)	Prénom
BELCIEL	Sébastien

Exercice 1

On considère un système qui consiste en un émetteur et un récepteur de messages qui communiquent via deux canaux, l'un avec pertes de messages entre émetteur et récepteur, l'autre sûr entre récepteur et émetteur. Afin de garantir la bonne réception des messages, l'émetteur retransmet régulièrement le même message jusqu'à réception d'un accusé de réception de la part du récepteur. L'émetteur envoie alors le message suivant.

Cependant, l'émetteur doit pouvoir différencier les accusés de réception pour ce nouveau message de ceux qui pourraient dater du message précédent. L'émetteur fait donc alterner un bit attaché à chaque nouveau message, et le récepteur renvoie ce bit en guise d'accusé de réception.

- Proposer une modélisation en utilisant les cinq éléments d'un protocole vu en cours (le service, les hypothèses, le vocabulaire, le format des messages et les règles procédurales).
- Utiliser le diagramme de transmission d'état («State Transition Diagram») pour illustrer les règles procédurales de la modélisation de votre protocole.
- Utiliser le diagramme de séquence-temps (Time Sequence Diagram) pour illustrer le fonctionnement de votre protocole.

Exercice 2 Proposer une modélisation PROMELA du protocole de la question précédente.

Exercice 3 Donner la définition des termes suivants

- (a) Deadlock
- (b) Livelock
- (c) Starvation

Exercice 4 Cocher les réponses correctes aux questions ci-dessous. Les questions peuvent admettre de 0 à plusieurs bonnes réponses: pour valider une question, toutes les bonnes réponses attendues (0, 1, 2, 3 ou 4) doivent être cochées, et celles-là seulement.

1) Concernant les sockets en général :

- ☒ La création d'un socket nécessite trois paramètres, le dernier généralement optionnel.
- ☒ Un socket ne peut être utilisé que pour envoyer ou que pour recevoir des données.
- ☒ Un socket doit obligatoirement être associé à une adresse en appelant bind().
- ☒ Un socket est identifié par une adresse IP et un numéro de port.