Memoria Bloc 2 AST

Daniel Vilardell

Índex

Aquest document no sera molt detallat ja que el codi en si es prou entenedor i esta comentat lo suficient com per veure el que fa cada secció.

1 Practica 3

1.1 TSocketSend

- **sendData:** S'encarrega de enviar les dades amb la restricció de mida del canal, i per això es segmenten abans.
- **segmentize:** Segmenta els segments de dades.
- **sendSegment:** Envia un segment per el canal.

1.2 TSocketRecv

- **ReceiverTask:** Implementa la classe runnable i es l'encarregat de rebre paquets i enviarlos a processar.
- **receiveData:** Metode encarregat de al rebre un segment de dades enviarlo a consumir.
- **processReceivedSegment:** Mira si la cua de recepció esta plena i si no es el cas introdueix el segment.
- **consumeSegment:** Consumeix els segments. Si son massa grans ho fa a troços i no els elimina fins que no els ha consumit del tot.

1.3 MonitorChannel

- **send:** Simula el canal i les seves perdues, fins a la practica 6 les perdues de canal son nules.
- **getMMS:** Informa de la mida màxima del camp de dades dels paquets de nivell de xarxa, per a evitar la fragmentació.

1.4 Sender & Receiver

Finalment comentar que Sender i Receiver creen els seus sockets pertinents encarregats de enviar les dades i rebreles. Si incrementem la velositat de enviament i disminuim la de recepció el programa perd segments ja que no tenim cap metode de control de fluxe.

2 Practica 4

2.1 Main

- **Host1:** Implementa runnable i crea dos sockets de recepció, cada un que rep d'un port diferent.
- **Host2:** Implementa runnable i crea dos sockets de emisió, cada un que emet a un port diferent.

2.2 Sender & Receiver

Aquestes classes utilitzen els seus respectius sockets per a enviar i rebre dades. Es pot modificar la velisitat de emisio i recepció junt amb el buffer de emisió i de recepció.

2.3 ProtocolSend & ProtocolRecv

Les classes son les encarregades de gestionar els sockets de enviament i recepcio, i cada cop que es vol establir una conexió es crea un nou socket i s'afegeix a la llista de sockets oberts.

- **openForOutput:** Metode de ProtocolSend que gestiona els sockets per a poguer tractar amb multiples comunicacions alhora amb diferents sockets.
- **openForInput:** Fa el mateix que el metode anterior el que dins de la classe ProtocolRecv.
- **ipInput:** Al rebre un segment analitza el port d'origen i de desti i l'envia al socket a processar pertinent.
- **getMatchingTSocket:** Busca a la llista de sockets el socket amb port entrada i sortida pertinent.

Les classes TSocketRecv i TSocketSend son les mateixes que a la practica 3 i funcionen de la mateixa forma.

3 Practica 5

3.1 Main

Es el mateix que a la practica 4, obre dos threads, un d'ells obre un emisor i l'altre un receptor amb els seus ports origen i desti corresponents.

3.2 Protocol

En aquesta practica la classe protocol no es distingeix de la part emisor i receptor, sino que ho englova tot. Té també un arraylist de sockets on guarda els sockets oberts.

- **openWith:** Crea un nou socket i l'afegeix a la llista.
- **ipInput:** Del segment que rep mira els camps de port origen i port destí i busca el socket que compleix aquests requisits. Un cop trobat envia a processar el segment al socket.
- **getMatchingTSocket:** Busca el segment amb el port origen i desti iguals als passats com a parametre
- **ReceiverTask:** Es un thread que esta sempre executantse a la escolta de la rebuda de missatges pel canal.

3.3 Receiver & Sender

Aquestes classes fan el mateix, envien i reven varios segments de dades per el canal usant un TSocket.

3.4 TSocket

Aquesta classe es la mes complexa de la practica 5. Es la encarregada de fer el protocol Stop&Wait i en la seguent practica de gestionar el control de fluxe mitjançant ARQ. Per a tractar amb el cas de finestra zero tenim un boolea que ens marcara si tenim problemes de finestra zero al emisor o al receptor (per si en algun moment ho volguessim passar a full duplex). A mes hem afegit dos condicions, una que tractarà amb el cas de cua buida i l'altre que tractarà el cas de espera de ACK.

- **sendData:** Es fragmenten els segments i es van enviant. Mentres no hagi rebut el segment de reconeixement o estigui en el cas de finestra zero s'esperarà. En els dos casos el receptor enviarà el ACK per informar que pot seguir enviant dades.
- **segmentize:** Segmenta el fragment de dades.
- **sendSegment:** Introdueix el segment al canal.
- **receiveData:** Rep les dades i crida el metode que les consumeix.
- **consumeSegment:** En el cas que la cua estigui buida el thread s'adorm. Despres es va recuperant poc a poc en funció de la mida maxima de recepció de segments. En el cas que estiguem en finestra zero s'envia un missatge al emisor per a avisar que ja s'ha consumit un segment i per tant la cua no està plena.
- **sendAck:** Omple varios parametres del ACK i l'envia.
- **processReceivedSegment:** Si es ACK mira el parametre window i actualitza si el receptor es troba en estat zeroWindow. Si es missatge de dades el guarda a la cua i envia ACK.

4 Practica 6

4.1 Main, Protocol, Receiver & Sender

Totes aquestes classes son iguals a les de la practica 5. S'ha afegit un ratio de perdua de fitxers però.

4.2 TSocket

Ara s'ha hagut d'implementar el sistema per evitar perdua de fitxers. Hem afegit els atributs nextAcknowledged, nextSend i nextReceive per a tractar amb aquest tema.

- **sendData:** Ara el cas finestra zero es tracta separat al cas no finestra zero. Si la finestra no es zero es fa el mateix d'abans amb la diferencia que despres d'enviar el paquet es crida startRTO, que fins a no rebre el ACK del missatge estara periodicament enviant el segment de dades. El cas de finestra zero crida startRTO.
- **segmentize:** Funciona igual que a la practica 5.
- **timeout:** Es reenvia el segment de sondeig i es torna a crida RTO.
- **startRTO:** S'espera un temps determinat i es crida timeout.
- **stopRTO:** Para el bucle de crides per enviar segments de sondeig un cop s'ha rebut el ACK del paquet.
- **sendAck:** Funciona igual, el que ara afegim el numero de segment per a saber quin hem reconegut.
- **processReceivedSegment:** Si es ACK actualitza el seguent a ser reconegut, para el RTO i actualitza la finestra disponible. Si no es ACK mira que el segment no s'hagi rebut ja, si es el cas envia el ACK i retorna. Si no lha rebut, augmenta el nextReceive i guarda el paquet a la cua de recepció.