acadèmica

Concurrència i sistemes distribuïts

Francisco Daniel Muñoz Escoí | Estefanía Argente Villaplana
Agustín Rafael Espinosa Minguet | Pablo Galdámez Saiz
Ana García-Fornes | Rubén de Juan Marín | Juan Salvador Sendra Roig

EDITORIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CONCURRÈNCIA I SISTEMES DISTRIBUÏTS

Francisco Daniel Muñoz Escoí (Coord.)

Estefanía Argente Villaplana

Agustín Rafael Espinosa Minguet

Pablo Galdámez Saiz

Ana García-Fornés

Rubén de Juan Marín

Juan Salvador Sendra Roig

EDITORIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Colecció Acadèmica

Revisió lingüística: Àrea de Promoció i Normalització Lingüística de la UPV

"La publicació d'aquest llibre ha rebut una ajuda de l'Àrea de Promoció i Normalització Lingüística de la Universitat Politècnica de València per a la redacció de manuals universitaris en valencià"

Per a referenciar esta publicació utilitze la següent cita: MUÑOZ ESCOÍ, F.D. [et al] (2013). Concurrència y sistemes distribuïts. València : Editorial Universitat Politècnica

Primera edició, 2013

Editorial Universitat Politècnica de València
Tel. 96 387 70 12 / www.editorial.upv.es / ref. Ed 863

Distribució: pedidos@editorial.upv.es
Tel. 96 387 70 12

Imprimeix: By print percom sl

ISBN: 978-84-9048-010-6
Imprés sota demanda

Imprés a Espanya

Pròleg

Aquest llibre va dirigit als estudiants de l'assignatura Concurrència i Sistemes Distribuïts de la titulació del grau en Enginyeria Informàtica de la Universitat Politècnica de València (UPV), encara que s'ha estructurat de manera que qualsevol persona interessada en aspectes de programació concurrent i sistemes distribuïts puga obtenir profit de la seua lectura.

Els autors d'aquest llibre som professors de la citada assignatura i, a l'hora de confeccionar una relació bibliogràfica útil per al nostre alumnat, ens vam adonar que en el mercat hi ha un gran nombre de llibres que aborden aspectes tals com concurrència, sincronització, sistemes distribuïts, sistemes de temps real, administració de sistemes, però no trobem cap llibre que tractara tots aquests continguts en conjunt. Aquest llibre pretén donar una visió integradora de les aplicacions concurrents i els sistemes distribuïts, oferint també una primera aproximació a les tasques d'administració de sistemes (necessàries en sistemes distribuïts).

Convé destacar que s'ha emprat el llenguatge Java per a detallar exemples de programes concurrents. L'elecció d'aquest llenguatge de programació es justifica pels avantatges que aporta per a la implementació de programes concurrents i pel seu elevat grau d'utilització, tant en l'àmbit professional com acadèmic. Per això, s'ha inclòs un capítol específic sobre les utilitats que ofereix Java per a la implementació de programes concurrents.

Grau en Enginyeria Informàtica

El títol de Graduat en Enginyeria Informàtica per la UPV substitueix als títols d'enginyer en Informàtica, enginyer tècnic en Informàtica de Gestió i enginyer tècnic en Informàtica de Sistemes, com a resultat de l'adaptació d'aquestes titulacions al marc de l'Espai Europeu d'Educació Superior. Des de gener de 2010 compta amb l'avaluació favorable per part de l'ANECA per a la seua implantació, iniciada al setembre de 2010.

El Pla d'estudis del títol de grau en Enginyeria Informàtica de la UPV està organitzat en quatre cursos de 60 ECTS cadascun. Cada ECTS suposa 10 hores de docència presencial i entre 15 i 20 hores per a la resta del treball de l'alumne, inclosa l'avaluació. El Pla d'Estudis s'estructura en mòduls i matèries. Cada matèria es descompon, al seu torn, en una o més assignatures de 4.5, 6 o 9 ECTS.

Matèria de Sistemes Operatius

Dins del Pla d'Estudis del Grau d'Enginyeria Informàtica de la UPV, l'assignatura Concurrència i Sistemes Distribuïts pertany al mòdul de Matèries Obligatòries i, dins d'aquest mòdul, a la matèria de Sistemes Operatius. Aquesta matèria comprèn l'estudi de les característiques, funcionalitats i estructura dels sistemes operatius així com l'estudi dels sistemes distribuïts.

El sistema operatiu és un programa que actua d'interfície entre els usuaris i el maquinari del computador, abstraient els components del sistema informàtic. D'aquesta manera oculta la complexitat del sistema als usuaris i aplicacions, i s'encarrega de la gestió dels recursos, es maximitza el rendiment i s'incrementa la seua productivitat. Per tant, el sistema operatiu s'encarrega de la gestió dels recursos del maquinari (CPU, memòria, emmagatzematge secundari, dispositius d'entrada/eixida, sistema de fitxers), i de controlar l'execució dels processos d'usuari que necessiten accedir als recursos del maquinari.

Per la seua banda, un sistema distribuït és una col·lecció d'ordinadors independents que ofereixen als seus usuaris la imatge d'un sistema coherent únic. De manera anàloga a com actua un sistema operatiu en una màquina, els sistemes distribuïts s'han d'encarregar d'ocultar a l'usuari les diferències entre les màquines i la complexitat dels mecanismes de comunicació. Els sistemes distribuïts han de facilitar l'accés dels usuaris als recursos remots, donar transparència de distribució (ocultant el fet que els processos i recursos estan físicament distribuïts en diferents ordinadors), oferir estàndards per a facilitar la interoperabilitat i portabilitat de les aplicacions, així com oferir escalabilitat.

En el Pla d'estudis del grau d'Enginyeria Informàtica de la UPV, la matèria Sistemes Operatius és de caràcter obligatori, amb una assignació de 12 crèdits ECTS i s'imparteix en segon curs. Les assignatures que componen aquesta matèria són dues: Fonaments de Sistemes Operatius, de 6 ECTS, que s'imparteix en el primer quadrimestre de segon curs; i Concurrència i Sistemes Distribuïts, de 6 ECTS, que s'imparteix en el segon quadrimestre d'aquest mateix curs.

En l'assignatura Fonaments de Sistemes Operatius (FSO) s'introdueix el concepte de sistema operatiu, la seua evolució històrica, així com el funcionament i els serveis que proporcionen, detallant la gestió dels processos, la gestió de la memòria i la gestió d'entrada/eixida i fitxers. També s'introdueixen els conceptes de sistemes

de temps real i la programació de sistemes, detallant el concepte de crida a sistema i la programació bàsica de sistemes.

Per la seua banda, l'assignatura Concurrència i Sistemes Distribuïts (CSD) se centra a descriure els principis i tècniques bàsiques de la programació concurrent, així com les característiques rellevants dels sistemes distribuïts. D'aquesta manera, l'assignatura de Concurrència i Sistemes Distribuïts permetrà que l'alumne identifique i resolga els problemes plantejats per l'execució de múltiples activitats concurrents en una aplicació informàtica. Aquests problemes apareixen a l'hora d'accedir a recursos compartits i poden generar resultats o estats inconsistents en aquests recursos. Per a evitar-los s'han d'emprar certs mecanismes de sincronització que l'alumne aprendrà a utilitzar de manera adequada. A més, en els sistemes de temps real apareixen altres restriccions sobre l'ús d'aquests mecanismes que l'alumne identificarà i gestionarà correctament.

D'altra banda, l'alumne coneixerà les diferències que comporta el disseny i desenvolupament d'una aplicació distribuïda, així com els mecanismes necessaris per a proporcionar diferents tipus de transparència (replicació, concurrència, fallades, ubicació, migració, etc.) en aquest tipus d'aplicacions. Utilitzarà el model client-servidor per a estructurar aquestes aplicacions, pels avantatges que això comporta a l'hora de gestionar els recursos. Coneixerà els avantatges que aporta el disseny d'algorismes distribuïts descentralitzats, tant pel que fa a la gestió de les fallades com a l'escalabilitat de les aplicacions resultants. Finalment, coneixerà les limitacions existents a l'hora de resoldre problemes de sincronització en entorns distribuïts, que requerirà l'ús d'una ordenació lògica dels esdeveniments en la majoria dels casos.

A més, s'iniciarà l'alumne en les tasques d'administració de sistemes, utilitzant el Directori Actiu de Windows Server per a aquesta fi.

Ampliació dels continguts de Sistemes Operatius

Els continguts de la matèria Sistemes Operatius s'amplien en altres assignatures en 3r i 4t curs: Tecnologies dels Sistemes d'Informació en la Xarxa (obligatòria), Administració de Sistemes (del mòdul de Tecnologies de la Informació), Disseny de Sistemes Operatius, i Disseny i Aplicacions dels Sistemes Distribuïts (ambdues del mòdul d'Enginyeria de Computadors).

Tecnologies dels Sistemes d'Informació en la Xarxa (obligatòria de 3r curs) introdueix les tecnologies i estàndards existents per a dissenyar, desenvolupar, desplegar i utilitzar aplicacions distribuïdes. Amb això s'estendrà la formació rebuda en CSD sobre sistemes distribuïts, desenvolupant aplicacions distribuïdes de certa complexitat, utilitzant el model client-servidor i identificant diferents aproximacions per a implantar cadascun dels components d'aquestes aplicacions.

Administració de Sistemes (de 3r curs) desenvolupa conceptes i habilitats relacionades amb les tasques d'administració de sistemes més habituals en les organitzacions actuals, entre les quals s'inclouen les habilitats de: instal·lació, configuració i manteniment de sistemes operatius; la gestió de serveis (impressió, fitxers, protocols); el disseny, planificació i implantació d'una política de sistemes (versions, funcionalitats, llicències, etc.); l'administració de dominis de sistema (usuaris, grups, seguretat); el manteniment de còpies de seguretat i recuperació davant desastres; l'automatització (amb l'ús de *scripts*, polítiques, etc.); i la formació i suport a l'usuari.

Disseny de Sistemes Operatius (de 3r curs) introdueix els aspectes d'implementació i disseny de sistemes operatius. Es tracta d'una assignatura fonamentalment pràctica, en la qual es treballarà amb un sistema operatiu real, en concret amb el sistema operatiu Linux, veient amb deteniment els mecanismes que ofereix el seu nucli. D'aquesta manera, s'analitzaran les crides al sistema que ofereix Linux, la gestió d'interrupcions del maquinari i la gestió de processos. La comprensió del funcionament del sistema operatiu permetrà a l'alumne prendre decisions de disseny (d'aplicacions) que optimitzen l'ús dels recursos de computador.

Finalment, Disseny i Aplicacions dels Sistemes Distribuïts (de 4t curs) aprofundeix en els aspectes de comunicació en sistemes distribuïts (models client-servidor, models orientats a objectes, models de grups), en els conceptes de seguretat, així com en les tecnologies per a la integració d'aplicacions en la web i els aspectes bàsics de disseny d'aplicacions distribuïdes, tals com a replicació i *caching*.

Coneixements recomanats

Respecte als coneixements recomanats per al seguiment adequat de l'assignatura de Concurrència i Sistemes Distribuïts, la primera part de les seues unitats didàctiques se centra en els problemes de concurrència i sincronització en un model de memòria compartida. Per a això, els alumnes han de fer ús dels conceptes de fil i procés estudiats en Fonaments de Sistemes Operatius.

D'altra banda, per a la realització de les pràctiques, així com per a l'estudi, comprensió i resolució dels problemes i casos d'estudi, és recomanable que els alumnes tinguen un acceptable nivell de programació utilitzant Java. Aquest nivell s'ha d'haver aconseguit en les assignatures de programació de primer curs (Introducció a la Informàtica i a la Programació; Programació; i Fonaments de Computadors) i la corresponent de primer quadrimestre de segon curs (Llenguatges, Tecnologies i Paradigmes de la Programació).

Índex general

ÍNDEX GENERAL V

ÍNDEX DE FIGURES XI

ÍNDEX DE TAULES XVII

1 PROGRAMACIÓ CONCURRENT 1

1.1 Introducció 1
1.2 Definició 1
1.3 Aplicacions concurrents 4
1.3.1 Avantatges i inconvenients 4
1.3.2 Aplicacions reals 6
1.3.3 Problemes clàssics 9
1.4 Tecnologia Java 14
1.4.1 Concurrència en Java 14
1.4.2 Gestió de fils d'execució 15
1.5 Resum 17

2 COOPERACIÓ ENTRE FILS 19

2.1 Introducció 19
2.2 Fils d'execució 20
2.2.1 Cicle de vida dels fils Java 21

2.3 Cooperació entre fils 24
2.3.1 Comunicació 25
2.3.2 Sincronització 26
2.4 Model d'execució 26
2.5 Determinisme 30
2.6 Secció crítica 34
2.6.1 Gestió mitjançant *locks* 37
2.7 Sincronització condicional 39
2.8 Resum 40

3 PRIMITIVES DE SINCRONITZACIÓ 43
3.1 Introducció 43
3.2 Monitors 46
3.2.1 Motivació 46
3.2.2 Definició 49
3.2.3 Exemples 50
3.2.4 Monitors en Java 53
3.3 Variants 56
3.3.1 Variant de Brinch Hansen 58
3.3.2 Variant d'Hoare 60
3.3.3 Variant de Lampson i Redell 63
3.4 Invocacions niades 66
3.5 Resum 68

4 INTERBLOQUEJOS 71
4.1 Introducció 71
4.2 Definició d'interbloqueig 72
4.2.1 Condicions de Coffman 74
4.2.2 Exemples 74
4.3 Representació gràfica 76
4.3.1 Algorisme de reducció de grafs 78
4.4 Solucions 82
4.4.1 Prevenció 83
4.4.2 Evitació 87

4.4.3 Detecció i recuperació 88
4.4.4 Exemples de solucions 89
4.5 Resum 91

5 BIBLIOTECA java.util.concurrent 93
5.1 Introducció 93
5.2 Inconvenients de les primitives bàsiques de Java 94
5.3 La biblioteca java.util.concurrent. 95
5.3.1 Locks 95
5.3.2 'Condition' i monitors 99
5.3.3 Col·leccions concurrents 101
5.3.4 Variables atòmiques 104
5.3.5 Semàfors 109
5.3.6 Barreres 113
5.3.7 Execució de fils 118
5.3.8 Temporització precisa 119
5.4 Resum 120

6 SISTEMES DE TEMPS REAL 123
6.1 Introducció 123
6.2 Anàlisi bàsica. 125
6.3 Exemple de càlcul dels temps de resposta 128
6.4 Compartició de recursos. 129
6.4.1 Protocol de sostre de prioritat immediat. 132
6.4.2 Propietats 134
6.4.3 Càlcul dels factors de bloqueig. 135
6.4.4 Càlcul del temps de resposta amb factors de bloqueig. 137
6.5 Resum 139

7 SISTEMES DISTRIBUÏTS 141
7.1 Introducció 141
7.2 Definició de sistema distribuït. 142
7.3 Objectius dels sistemes distribuïts 145
7.3.1 Accés a recursos remots 145
7.3.2 Transparència de distribució 146

7.3.3 Sistemes oberts 152
7.3.4 Sistemes escalables 154
7.4 Dificultats en el desenvolupament de sistemes distribuïts 162
7.5 Resum 163

8 COMUNICACIONS 165
8.1 Introducció 165
8.2 Arquitectura en nivells (TCP/IP) 166
8.3 Crida a procediment remot (RPC) 168
8.3.1 Passos en una RPC 170
8.3.2 Pas d'arguments 172
8.3.3 Tipus d'RPC 173
8.4 Invocació a objecte remot 175
8.4.1 Visió d'usuari 179
8.4.2 Creació i registre d'objectes 179
8.4.3 Detalls de la invocació remota 181
8.4.4 Altres aspectes 187
8.4.5 RMI (Remote Method Invocation) 188
8.5 Comunicació basada en missatges 193
8.5.1 Sincronització 193
8.5.2 Persistència 195
8.6 Resum 197

9 SINCRONITZACIÓ DISTRIBUÏDA 199
9.1 Introducció 199
9.2 Rellotges 201
9.2.1 Algorismes de sincronització 201
9.2.2 Rellotges lògics 205
9.2.3 Rellotges vectorials 209
9.3 Estat global 211
9.3.1 Tall o imatge global 212
9.3.2 Algorisme de Chandy i Lamport 214
9.4 Elecció de líder 217
9.4.1 Algorisme *Bully* 218
9.4.2 Algorisme per a anells 219

9.5 Exclusió mútua 220
9.5.1 Algorisme centralitzat 220
9.5.2 Algorisme distribuït 221
9.5.3 Algorisme per a anells 222
9.5.4 Comparativa 223
9.6 Resum 224

10 GESTIÓ DE RECURSOS 227
10.1 Introducció 227
10.2 Anomenament 228
10.2.1 Conceptes bàsics 228
10.2.2 Espais de noms 230
10.3 Serveis de localització 234
10.3.1 Localització per difusions 236
10.3.2 Punters avant 236
10.4 Serveis de noms jeràrquics: DNS 239
10.5 Serveis basats en atributs: LDAP 244
10.6 Resum 247

11 ARQUITECTURES DISTRIBUÏDES 249
11.1 Introducció 249
11.2 Arquitectures de programari 250
11.3 Arquitectures de sistema 253
11.3.1 Arquitectures centralitzades 255
11.3.2 Arquitectures descentralitzades 259
11.4 Resum 269

12 DIRECTORI ACTIU 271
12.1 Introducció 271
12.2 Serveis de domini 271
12.2.1 Controlador de domini 272
12.2.2 Arbres i boscos 273
12.2.3 Principals protocols emprats 275

12.3 Serveis de directori 275
12.3.1 Arquitectura i components 276
12.3.2 Objectes del directori 277
12.4 Gestió de permisos 280
12.4.1 Permisos per a fitxers 281
12.4.2 Permisos per a carpetes 282
12.4.3 Llistes de control d'accés 284
12.4.4 Ús de les ACL 286
12.4.5 Disseny d'ACL 289
12.5 Recursos compartits 290
12.5.1 Creació del recurs compartit 291
12.5.2 Assignació d'un identificador d'unitat 291
12.5.3 Autenticació i autorització 292
12.6 Resum 293

Bibliografia 297

ÍNDEX ALFABÈTIC 307

Llista de figures

1.1. Ús d'un *buffer* de capacitat limitada. 10
1.2. Problema dels cinc filòsofs. 13
1.3. Exemple de creació de fils. 17

2.1. Diagrama d'estats dels fils en Java. 22
2.2. Classe Node. 27
2.3. Classe Queue. 28
2.4. Comptador no determinista. 31
2.5. Protocols d'entrada i eixida a una secció crítica. 34
2.6. Comptador determinista. 39

3.1. Exemple de monitor. 51
3.2. Monitor del simulador. 52
3.3. Segon monitor del simulador. 53
3.4. Monitor *Buffer* en Java. 55
3.5. Monitor Java per al simulador de la colònia de formigues. 57
3.6. Exemple de monitor amb reactivació en cascada. 58
3.7. Exemple de monitor amb tots dos tipus de reactivació. 59
3.8. Monitor que implanta el tipus o classe `Semaphore`. 60

3.9. Monitor `BoundedBuffer` (variant de Brinch Hansen). 61

3.10. Monitor `BoundedBuffer` (variant d'Hoare). 62

3.11. Monitor `SynchronousLink` (variant d'Hoare). 64

3.12. Monitor `BoundedBuffer` en Java. 66

3.13. Monitor `BoundedBuffer` incorrecte en Java. 67

4.1. Graf d'assignació de recursos. 77

4.2. Algorisme de reducció de grafs. 78

4.3. Traça de l'algorisme de reducció (inicial). 79

4.4. Traça de l'algorisme de reducció (iteració 1). 79

4.5. Traça de l'algorisme de reducció (iteració 2). 80

4.6. Traça de l'algorisme de reducció (iteració 3). 81

4.7. Segona traça de reducció (estat inicial). 81

4.8. Segona traça de reducció (iteració 1). 82

4.9. Problema dels cinc filòsofs. 90

5.1. Secció crítica protegida per un `ReentrantLock`. 98

5.2. Monitor `BoundedBuffer`. 100

5.3. Gestió d'un *buffer* de capacitat limitada. 104

5.4. Comptador concurrent. 108

5.5. Solució amb semàfors al problema de productors-consumidors. . . . 111

5.6. Sincronització amb `CyclicBarrier` (incorrecta). 114

5.7. Sincronització amb `CyclicBarrier` (correcta). 115

5.8. Sincronització amb `CountDownLatch`. 117

6.1. Representació gràfica dels atributs de les tasques. 126

6.2. Cronograma d'execució. 128

6.3. Exemple d'inversió de prioritats. 130

6.4. Exemple d'interbloqueig. 131

6.5. Protocol del sostre de prioritat immediat. 133

6.6. Evitació d'interbloqueig amb el protocol del sostre de prioritat immediat. 134

6.7. Sistema format per tres tasques i dos semàfors. 137

7.1. Arquitectura d'un sistema distribuït. 144

7.2. Transparència d'ubicació en una RPC. 148

8.1. Propagació de missatges (arquitectura TCP/IP). 167

8.2. Visió basada en protocols de l'arquitectura TCP/IP. 168

8.3. Ubicació del *middleware* en una arquitectura de comunicacions. . . 168

8.4. Passos en una crida a procediment remot. 171

8.5. RPC convencional. 173

8.6. RPC asincrònica. 174

8.7. RPC asincrònica diferida. 174

8.8. Invocació local i remota. 176

8.9. Funcionament bàsic d'una ROI. 178

8.10. Creació d'objectes remots a sol·licitud del client. 180

8.11. Creació d'objectes remots per part del servidor. 180

8.12. Detalls d'una invocació remota. 182

8.13. Referència directa. 183

8.14. Referència indirecta. 184

8.15. Referència a través d'un localizador. 184

8.16. Pas d'objectes per valor. 185

8.17. Pas per referència d'un objecte local. 186

8.18. Pas per referència d'un objecte remot. 186
8.19. Exemple de pas d'arguments. 187
8.20. Utilització del registre en RMI. 189
8.21. Diagrama d'execució de l'exemple. 192
8.22. Exemple de comunicació asincrònica. 193
8.23. Comunicació sincrònica basada en enviament. 194
8.24. Comunicació sincrònica basada en lliurament. 195
8.25. Comunicació sincrònica basada en processament. 195

9.1. Sincronització amb l'algorisme de Cristian. 203
9.2. Rellotges lògics de Lamport. 207
9.3. Rellotges lògics de Lamport generant un ordre total. 209
9.4. Rellotges vectorials. 210
9.5. Tall precís d'una execució. 212
9.6. Tall consistent d'una execució. 213
9.7. Tall inconsistent d'una execució. 213
9.8. Traça de l'algorisme de Chandy i Lamport (passos 1 a 3). 215
9.9. Traça de l'algorisme de Chandy i Lamport (passos 4 a 6). 216
9.10. Traça de l'algorisme de Chandy i Lamport (passos 7 a 9). 216
9.11. Traça de l'algorisme de Chandy i Lamport (estat global). 217

10.1. Directoris i entitats en un espai de noms. 231
10.2. Nivells en un espai de noms jeràrquic. 232
10.3. Inserció d'un punter avant. 237
10.4. Retallada d'una cadena de punters avant. 238
10.5. Exemple de fitxer de zona DNS. 243

11.1. Arquitectura de programari en nivells. 250
11.2. Arquitectura de programari basada en esdeveniments. 253
11.3. Rols client i servidor en diferents invocacions. 256
11.4. Seqüencia petició-resposta en una interacció client-servidor. 256
11.5. Cinc exemples de desplegament d'una arquitectura en capes. . . . 257
11.6. Distribució horitzontal en un servidor web replicat. 260
11.7. Graus de centralització en un sistema P2P. 262
11.8. Ubicació de recursos en el sistema Chord. 265

12.1. Elements en un domini. 272
12.2. Exemple de bosc de dominis. 274
12.3. Arquitectura dels serveis de directori en un DC. 276
12.4. Relacions entre els diferents tipus de permisos (per a fitxers). . . . 282
12.5. Relacions entre els diferents tipus de permisos (per a carpetes). . . 283
12.6. Recurs compartit. 290

Llista de taules

1.1. Variants de la definició de fils. 15

3.1. Taula de mètodes. 51
3.2. Taula de mètodes del monitor `Formigues`. 52
3.3. Característiques de les variants de monitor. 58
3.4. Traça amb la variant d'Hoare. 63

5.1. Mètodes de la classe `ReentrantLock`. 97
5.2. Mètodes de la interfície `Condition`. 99
5.3. Mètodes de la interfície `BlockingQueue<E>`. 102
5.4. Classes del *package* `java.util.concurrent.atomic`. 106
5.5. Mètodes de la classe `AtomicInteger`. 107

7.1. Tipus de transparència. 146

9.1. Comparativa d'algorismes d'exclusió mútua. 223

11.1. Classes de sistemes P2P. 265

12.1. Exemple d'ACL amb entrades explícites i heretades. 285
12.2. Exemple d'ACL amb desactivació d'herència. 285

12.3. Exemple d'ACL. 287

Unitat 1

PROGRAMACIÓ CONCURRENT

1.1 Introducció

Aquesta unitat didàctica presenta el concepte de *programació concurrent*, i proporciona la seua definició informal en la secció 1.2. La secció 1.3 discuteix els principals avantatges i inconvenients que ofereix aquest tipus de programació quan és comparada amb la programació seqüencial. A més, es presenten alguns exemples d'aplicacions que típicament seran concurrents i es discuteix per què resulta avantatjós programar-les d'aquesta manera. Finalment, la secció 1.4 ofereix una primera introducció als mecanismes incorporats en el llenguatge Java per a suportar la programació concurrent.

1.2 Definició

Un *programa seqüencial* [Dij71] és aquell en el qual les instruccions es van executant en sèrie, una després d'una altra, des de l'inici fins al final. Per tant, en un programa seqüencial solament existeix una única activitat o fil d'execució. És la manera tradicional de programar, ja que els primers ordinadors només tenien un processador i els primers sistemes operatius no van proporcionar suport per a múltiples activitats dins d'un mateix procés.

Per contra, en un *programa concurrent* [Dij68] s'arribaran a crear múltiples activitats que progressaran de manera simultània. Cadascuna d'aquestes activitats serà seqüencial però ara el programa no avançarà seguint una única seqüència perquè

cada activitat podrà seguir una sèrie d'instruccions diferent i totes les activitats avancen de manera simultània (o, almenys, aquesta és la imatge que ofereixen a l'usuari). Perquè en un determinat procés hi haja múltiples activitats, cadascuna d'aquestes activitats estarà suportada per un *fil d'execució*. Perquè un conjunt d'activitats constituïsca un programa concurrent, totes aquestes activitats han de cooperar entre si per a dur a terme una tasca comuna.

Per a il·lustrar el concepte de programa o procés concurrent, podríem citar un processador de textos. N'hi ha molts (MS Word, LibreOffice Writer, Apple Pages...). La majoria poden dur a terme múltiples tasques en segon pla, mentre s'escriu (per a fer això es necessita que hi haja un fil d'execució que es dedique a cadascuna d'aquestes accions, mentre un altre atenga l'entrada proporcionada per l'usuari). Alguns exemples de tasques realitzades per aquests fils són: revisió gramatical del contingut del document, enregistrament del contingut en disc de manera periòdica per a evitar les pèrdues de text en cas de tall involuntari d'alimentació, actualització de la finestra en la qual es mostra el text introduït, etc. Com s'observa en aquest exemple, en aquest programa (format per un sol fitxer executable) hi ha múltiples fils d'execució i tots cooperen entre si i comparteixen la informació gestionada (el document que estiguem editant).

Presentem seguidament un exemple d'una execució no concurrent. Assumim que en una sessió de laboratori obrim diverses consoles de text en Linux per a realitzar les accions que ens demane algun enunciat i provar-les. En aquest segon cas, seguiríem utilitzant un únic programa (un intèrpret d'ordres; p. ex., el `bash`) i tindríem múltiples activitats (és a dir, hi hauria diverses activitats executant aquest únic programa), però aquestes activitats no col·laborarien entre si. Cada consola oberta la utilitzaríem per a llançar una sèrie diferent d'ordres i les ordres llançades des de diferents consoles no interactuarien entre si. Serien múltiples instàncies d'un mateix programa en execució, que generen un conjunt de processos *independents*. S'executarien simultàniament, però cadascun d'aquests no dependria dels altres. Això no és una aplicació concurrent: *si volem que hi haja concurrència ha d'haver-hi cooperació entre les activitats*. Quelcom similar ocorreria si en comptes de llançar múltiples consoles haguérem llançat múltiples instàncies del navegador. Cada finestra oberta la utilitzaríem per a accedir a una pàgina diferent i no tindria per què haver-hi cap tipus d'interacció o cooperació entre tots els processos navegadors generats.

En aquesta secció s'ha assumit fins al moment que per a generar una aplicació informàtica n'hi ha prou amb un únic programa. Això no serà sempre així. De fet, algunes aplicacions poden estructurar-se com un conjunt de components i cadascun d'aquests components pot estar implantat mitjançant un programa diferent. En general, quan parlem d'una *aplicació concurrent* assumirem que està composta per un conjunt d'activitats que cooperen entre si per a dur a terme una tasca comuna. Aquestes activitats podran ser: múltiples fils en un mateix procés, múltiples processos en una mateixa màquina o fins i tot múltiples processos en màquines

diferents; però en tots els casos hi haurà cooperació entre les activitats. Per contra, en una *aplicació seqüencial* únicament existirà un únic procés i en tal procés només hi haurà un únic fil d'execució.

Per a suportar la concurrència s'ha d'executar aquest conjunt d'activitats de manera simultània o paral·lela, per a la qual cosa hi ha dos mecanismes:

- *Paral·lelisme real.* Es donarà quan tinguem disponibles múltiples processadors, de manera que situarem a cada activitat en un processador diferent. En les arquitectures modernes, els processadors poden tenir múltiples nuclis. En aquest cas, n'hi ha prou amb assignar un nucli diferent a cada activitat. Totes elles podran avançar simultàniament sense cap problema.

  Si només disposem d'ordinadors amb un processador i un únic nucli, podrem encara aconseguir la concurrència real utilitzant múltiples ordinadors. En aquest cas l'aplicació concurrent estarà formada per múltiples processos i aquests cooperaran entre si intercanviant missatges per a comunicar-se.

- *Paral·lelisme lògic.* Com els processadors actuals són ràpids i les operacions d'E/S suspenen temporalment l'avanç del fil d'execució que les haja programades, es pot intercalar l'execució de múltiples activitats i oferir la imatge que aquestes progressen simultàniament. Així, mentre se serveix una operació d'E/S, el processador queda lliure i es pot triar una altra activitat preparada per a executar-la durant aquest interval. Aquest és el principi seguit per a proporcionar *multiprogramació* (tècnica de multiplexació que permet l'execució simultània de múltiples processos en un únic processador, de manera que sembla que tots els processos s'estan executant alhora, encara que hi ha un únic procés en el processador en cada instant de temps). Per això, el fet que únicament es dispose d'un processador en un determinat ordinador no suposa cap restricció que impedisca implantar aplicacions concurrents.

  L'usuari serà incapaç de distingir entre un sistema informàtic amb un processador i un únic nucli d'un altre que tinga múltiples processadors, ja que els sistemes operatius oculten aquests detalls. En qualsevol dels dos casos, l'usuari percep la imatge que múltiples activitats avancen simultàniament.

Tots dos tipus de paral·lelisme poden combinar-se sense cap dificultat. Per això, si en un determinat ordinador tenim un processador amb dos nuclis, el nombre d'activitats que podrem suportar en una aplicació concurrent que hi executem no s'ha de limitar a dues. En combinar el paral·lelisme real amb el lògic podrem utilitzar tants fils com siga necessari en aquesta aplicació.

1.3 Aplicacions concurrents

Aquesta secció estudia amb més deteniment alguns aspectes complementaris de la programació concurrent, com són els seus principals avantatges i inconvenients, així com altres exemples d'aplicacions, tant reals com acadèmics. Aquests últims il·lustren algunes de les dificultats que trobarem a l'hora de desenvolupar una aplicació concurrent.

1.3.1 Avantatges i inconvenients

La programació concurrent proporciona els avantatges següents:

- *Eficiència.* El fet de disposar de múltiples activitats dins de l'aplicació permet que aquesta acabe el seu treball de manera més ràpida. El fet que alguna activitat se suspenga en utilitzar (p. ex., en accedir al disc per a llegir part d'un fitxer) o en sol·licitar (p. ex., en utilitzar l'operació P() sobre un semàfor) algun recurs no suspèn tota l'aplicació i amb això aquesta última podrà seguir avançant. En una aplicació concurrent resulta senzill que s'utilitzen múltiples recursos simultàniament (fitxers, semàfors, memòria, processador, xarxa...).

- *Escalabilitat*. Si una determinada aplicació pot dissenyar-se i implantar-se com un conjunt de components que interactuen i col·laboren entre si, generant almenys una activitat per cada component, es facilitarà la distribució de l'aplicació sobre diferents ordinadors. Per a això hi ha prou amb instal·lar cada component en un ordinador diferent, i generar així una *aplicació distribuïda*. D'aquesta manera, la quantitat de recursos de còmput que es podran utilitzar simultàniament s'incrementarà. A més, si el nombre d'usuaris d'aquesta aplicació també s'incrementara significativament, es podrien utilitzar tècniques de *replicació* [Sch90, BMST92], incrementant la seua capacitat de servei. Així millora l'escalabilitat d'una aplicació.

  En general, es diu que un *sistema* és *escalable* [Bon00] quan és capaç de gestionar adequadament càrregues creixents de treball o quan té la capacitat per a créixer i adaptar-se a aquestes càrregues. En el cas de les aplicacions concurrents ambdues interpretacions es poden complir combinant la distribució i la replicació de components.

- *Gestió de les comunicacions*: L'ús eficient dels recursos, ja comentat en el primer avantatge citat en aquest apartat, permet que aquells recursos relacionats amb la comunicació entre activitats siguen explotats de manera senzilla en una aplicació concurrent. Si la comunicació està basada en l'intercanvi de missatges a través de la xarxa, aquesta comunicació no implicarà que tota l'aplicació es detinga esperant alguna resposta. Per això, l'ús de mecanismes sincrònics de comunicació entre activitats es pot integrar de manera natural

en una aplicació concurrent. En la unitat 8 es detallaran diferents tipus de mecanismes de comunicació.

- *Flexibilitat.* Les aplicacions concurrents solen utilitzar un disseny modular i estructurat, en el qual es presta especial atenció a què és el que ha de fer cada activitat, quins recursos requerirà i quins mòduls del programa necessitarà executar. Si els requisits de l'aplicació canviaren, resultaria relativament senzill identificar quins mòduls haurien de ser adaptats per a incorporar aquestes variacions en l'especificació i a quines activitats implicarien. Com a resultat d'aquest disseny estructurat i acurat, es generaran aplicacions flexibles i adaptables.
- *Menor buit semàntic.* Un bon nombre d'aplicacions informàtiques requereixen l'ús de diverses activitats simultànies (per exemple, en un videojoc sol utilitzar-se un fil per cada element mòbil que hi haja a la pantalla; en un full de càlcul tenim un fil per a gestionar l'entrada de dades, un altre per a recalcular, un altre per a la gestió dels menús, un altre per a l'actualització de les cel·les visibles...). Si aquestes aplicacions s'implanten com a programes concurrents resulta senzill dissenyar-les. Per contra, si s'intentara implantar-les com un sol programa seqüencial la coordinació entre aquestes diferents funcionalitats resultaria difícil.

No obstant això, la programació concurrent també presenta alguns inconvenients que convé tenir en compte:

- *Programació delicada.* Durant el desenvolupament d'aplicacions concurrents poden arribar a sorgir alguns problemes. El més important es coneix com a *condició de carrera* (que s'estudiarà en la unitat 2) i pot generar inconsistències imprevistes en el valor de les variables o atributs compartits entre les activitats, quan aquestes els modifiquen. Un segon problema són els *interbloquejos* [CES71], que s'analitzaran en la unitat 4. Per això, s'han de conèixer els problemes potencials que comporta la programació concurrent i s'han de prendre les precaucions oportunes per a evitar que apareguen.
- *Depuració complexa.* Una aplicació concurrent, com que està composta per múltiples activitats, pot intercalar de diferents maneres en cada execució les sentències que executen cadascuna de les seues activitats. Encara que es proporcionen les mateixes entrades a l'aplicació, els resultats que aquesta genere poden arribar a ser diferents en diferents execucions. Si algun d'aquests resultats fóra incorrecte, resultaria difícil la reproducció d'aquesta execució. A més, com que per a depurar qualsevol aplicació informàtica se solen utilitzar eines especialitzades (diferents tipus de depuradors), la pròpia gestió del depurador podria evitar que es donara la traça que va provocar l'error. Això il·lustra que les aplicacions concurrents no tenen una depuració senzilla i que els mecanismes que s'usen en els seus depuradors no sempre seran idèntics als utilitzats sobre aplicacions seqüencials.

Malgrat aquests inconvenients hi ha un interès creixent en el desenvolupament d'aplicacions concurrents. Una bona part de les aplicacions actuals s'estructuren en múltiples components que poden ser desplegats en ordinadors diferents, és a dir, són aplicacions concurrents distribuïdes els components de les quals necessiten intercomunicar-se a través de la xarxa. També estan apareixent múltiples dispositius mòbils amb processadors acceptables i amb discos durs o memòries flaix amb suficient capacitat d'emmagatzematge. Aquests elements (tabletes, *netbooks*, *ultrabooks*, telèfons intel·ligents...) interactuen tant amb el seu entorn com amb altres dispositius mòbils i és habitual que utilitzen múltiples activitats en les seues aplicacions, tal com suggereix un model de programació concurrent. Al seu torn, no és rar que els processadors utilitzats tant en els ordinadors personals tradicionals com en aquest nou conjunt d'ordinadors *lleugers* disposen de múltiples nuclis. Amb això, avui dia no suposa cap inconvenient l'execució paral·lela de múltiples activitats, tant de manera lògica com real.

1.3.2 Aplicacions reals

Passem a veure alguns exemples d'aplicacions concurrents reals. Aquests exemples il·lustren els conceptes presentats fins al moment en aquesta unitat, però també proporcionen la base per a entendre alguns dels aspectes analitzats en les unitats posteriors en les quals presentem els problemes que planteja la programació concurrent així com els mecanismes necessaris per a solucionar-los:

1. *Programa de control de l'accelerador lineal Therac-25* [LT93]. Els acceleradors lineals s'utilitzen en els tractaments de radioteràpia per a alguns tipus de càncer, eliminant mitjançant unes determinades classes de radiació (radiació d'electrons, rajos X i rajos gamma, principalment) les cèl·lules cancerígenes. El Therac-25 va ser un accelerador lineal desenvolupat entre 1976 i 1982, prenent com a base els dissenys d'altres acceleradors més antics (el Therac-6 i el Therac-20). En el Therac-25 es podien utilitzar dos tipus de radiació (d'electrons i gamma) que abastaven un ampli interval d'intensitats. En els models anteriors, l'accelerador disposava de sistemes de control per maquinari que evitaven qualsevol tipus de disfunció (parant l'accelerador en cas d'error). En el Therac-25 aquesta gestió es va integrar en el programari.

   El tractament es realitzava en una sala aïllada. L'operador del sistema configurava adequadament l'accelerador una vegada el pacient estiguera preparat. Per a aconseguir-ho s'havia d'especificar el tipus de radiació a utilitzar, la potència a emprar, la durada de la sessió, etc. El sistema tardava alguns segons a respondre a aquesta configuració. Una vegada transcorria aquest temps, començava el tractament. El programa de control s'encarregava de monitorizar tots els components del sistema: posició i orientació de l'accelerador, potència de la radiació, posició dels escuts d'atenuació (a utilitzar en els tipus de radiació més intensa), etc. Si es detectava algun error lleu (com puga

ser un lleuger desenfocament), el sistema parava i s'avisava l'operador que podia reprendre el tractament una vegada corregida la situació d'error. En cas d'error greu (per exemple, una sobredosi de radiació) el sistema parava per complet i havia de ser reiniciat.

Aquest programa de control era una aplicació concurrent, composta per múltiples activitats, cadascuna de les quals gestionava un conjunt diferent d'elements del sistema. Entre les activitats més importants caldria distingir:

- *Gestió d'entrada.* Aquesta activitat monitoritzava la consola i arreplegava tota la informació de configuració establida per l'operador. Es deixava indicat en una variable global si l'operador havia finalitzat la seua introducció de dades.
- *Gestió dels imants.* Els components interns necessaris per a la generació de la radiació necessitaven aproximadament vuit segons per a posicionar-se i acceptar la nova especificació del tipus de radiació i la seua intensitat. Com el Therac-25 admetia dos tipus de radiació, en un d'aquests s'utilitzaven escuts d'atenuació i en l'altre no.
- *Pausa.* Hi havia una tercera activitat que suspenia la gestió d'entrada mentre s'executava la gestió dels imants. Amb això s'intentava que la gestió dels imants finalitzara el més prompte possible. Se suposava que qualsevol intent de reconfigurar l'equip durant aquesta pausa seria atès quan la pausa finalitzara, just abans d'iniciar el tractament, i en aquest punt s'obligava a reiniciar la gestió dels imants.

Lamentablement, hi havia un error de programació en l'aplicació concurrent i, a més de suspendre temporalment la gestió d'entrada, quan finalitzava l'activitat de *pausa* no s'arribava a consultar si l'operador havia sol·licitat una modificació de la configuració del sistema. Així, l'operador observava que la nova configuració apareixia en pantalla (i suposava que havia sigut acceptada), però el sistema seguia tenint els seus imants adaptats a la configuració inicialment introduïda. S'havia donat una *condició de carrera* i l'estat real del sistema no concordava amb el que apareixia en la pantalla de l'operador. Això va arribar a tenir conseqüències fatídiques en alguns tractaments, en els quals es va arribar a donar la radiació de major intensitat sense la presència dels escuts d'atenuació: entre 1985 i 1987 es van donar sis accidents d'aquest tipus, que van causar tres morts [LT93].

Aquest és un dels exemples més rellevants dels problemes que pot arribar a causar una aplicació concurrent amb errors en el seu disseny.

2. *Servidor web Apache* [Apa12]. Apache és un servidor web que empra programació concurrent. L'objectiu d'un servidor web és la gestió de cert conjunt de pàgines web situades en aquest servidor. Aquestes pàgines resulten accessibles utilitzant el protocol HTTP (o Hypertext Transfer Protocol [FGM+99]). Els navegadors han de fer peticions a aquests servidors per a obtenir el contingut de les pàgines que mostren en les seues finestres.

En un servidor d'aquest tipus interessa tenir múltiples fils d'execució, de manera que cadascun d'ells puga atendre a un client diferent, paral·lelitzant així la gestió de múltiples clients. Perquè la pròpia gestió dels fils no supose un alt esforç, Apache utilitza un conjunt de fils (o *thread pool*) en espera, coordinats per un fil addicional que atén inicialment les peticions que vagen arribant, associant cadascuna d'aquestes als fils del conjunt que queden disponibles. En tenir aquest conjunt de fils ja generats i preparats per a la seua assignació es redueix el temps necessari per a iniciar el servei de cada petició. Si el nombre de peticions rebudes durant un cert interval de temps supera la grandària d'aquest conjunt, les peticions que no poden servir-se immediatament es mantenen en una cua d'entrada. L'objectiu d'aquest model de gestió és la reducció del temps necessari per a gestionar els fils que servisquen les peticions rebudes. De fet, els fils ja estan creats abans d'iniciar l'atenció de les peticions i no es destrueixen quan finalitzen l'atenció de cada petició, sinó que tornen al *pool* i poden ser reutilitzats posteriorment. Amb això, en lloc de crear fils en rebre les peticions o destruir-los en finalitzar el seu servei (dues operacions que requereixen bastant temps) n'hi ha prou amb assignar-los o retornar-los al *pool* (operacions molt més ràpides que les anteriors).

3. *Videojocs.* La majoria dels videojocs actuals (tant per a ordinadors personals com per a algunes consoles) s'estructuren com a aplicacions concurrents compostes per múltiples fils d'execució. Amb aquesta finalitat se sol utilitzar un *motor de videojocs* que s'encarrega de proporcionar cert suport bàsic per a la renderització, la detecció de col·lisions, el so, les animacions, etc. Els motors permeten que múltiples fils s'encarreguen d'aquestes facetes. Així, el rendiment dels videojocs pot augmentar en cas de disposar d'un equip el processador del qual dispose de múltiples nuclis.

   Aquests motors també faciliten la portabilitat del joc, ja que proporcionen una interfície que no depèn del processador ni de la targeta gràfica. Si el motor ha arribat a implantar-se sobre múltiples plataformes, els jocs desenvolupats amb aquest motor es poden migrar a les plataformes sense excessives dificultats. Per exemple, la detecció de col·lisions en un motor de videojocs sol ser responsabilitat d'un component anomenat *motor de física* (*physics engine*). En el joc FIFA d'Electronic Arts, aquest motor (que en aquest cas es diu *Player Impact Engine*) es va renovar en la segona meitat de 2011. El motor de videojocs del qual forma part ha facilitat la seua portabilitat a algunes de les plataformes en les quals el joc FIFA 12 ha estat disponible: Xbox 360, Wii, Nintendo 3DS, PC, PlayStation 3, PlayStation 2, PlayStation Portable, PlayStation Vita, Mac, iPhone, iPad, iPod i Android.

4. *Navegador web.* Quan Google va presentar el seu navegador Chrome (al desembre de 2008), la seua arquitectura [McC08] era molt diferent a la de la resta de navegadors web (Microsoft Internet Explorer, Mozilla Firefox,

Opera...). En Chrome, cada pestanya oberta està suportada per un procés independent. D'aquesta manera, si alguna de les pestanyes falla, la resta pot seguir sense problemes. Per a millorar-ne el rendiment es va optimitzar el seu suport per a Javascript, compilant i mantenint el codi generat en lloc d'interpretar-ho cada vegada. A més, en cada pestanya s'utilitzen múltiples fils per a obtenir cadascun dels elements de la pàgina que haja de visualitzar-se. Amb això, la càrrega d'una pàgina es podia fer de manera molt més ràpida que en altres navegadors. Actualment la major part dels navegadors web han adoptat una arquitectura similar a la de Chrome pels avantatges en rendiment i robustesa que proporciona.

1.3.3 Problemes clàssics

Existeix una sèrie de problemes clàssics (o acadèmics) de la programació concurrent que il·lustren una sèrie de situacions en les quals múltiples activitats han de seguir certes regles per a gestionar un recurs compartit de manera adequada.

Encara que s'incloguen en aquesta secció, els exemples que descriurem seguidament s'utilitzaran en les pròximes unitats per a il·lustrar alguns dels problemes que es plantegen a l'hora de desenvolupar una aplicació concurrent. Per això, no és necessari que s'estudien amb deteniment dins d'aquesta primera unitat didàctica, però sí que es revisen més tard en una segona lectura en analitzar aquestes unitats didàctiques posteriors.

Productor-consumidor amb buffer acotat

En aquest problema [Hoa74] s'assumeix que existeixen dos tipus d'activitats o processos: els productors i els consumidors. La interacció entre els dos rols es duu a terme utilitzant un *buffer* de capacitat limitada, en el qual els processos *productors* aniran dipositant elements i d'on els processos *consumidors* els extrauran. Aquest *buffer* gestiona els seus elements amb una estratègia FIFO (First In, First Out), és a dir, els seus elements s'extrauran en l'ordre en què van ser inserits.

Perquè l'ús del *buffer* siga correcte, haurà de respectar una sèrie de restriccions que discutirem en funció de l'exemple mostrat en la figura 1.1. En aquesta figura s'observa un *buffer* amb capacitat per a set elements, construït com un vector les components del qual s'identifiquen amb els nombres naturals entre 0 i 6, tots dos inclusivament. Per a gestionar aquest *buffer* utilitzaríem (en qualsevol llenguatge de programació orientat a objectes) aquests atributs:

- `in`. Indicaria la component en la qual algun procés productor inseriria el pròxim element en el *buffer*. En l'estat mostrat en la figura 1.1, el seu valor seria 5.

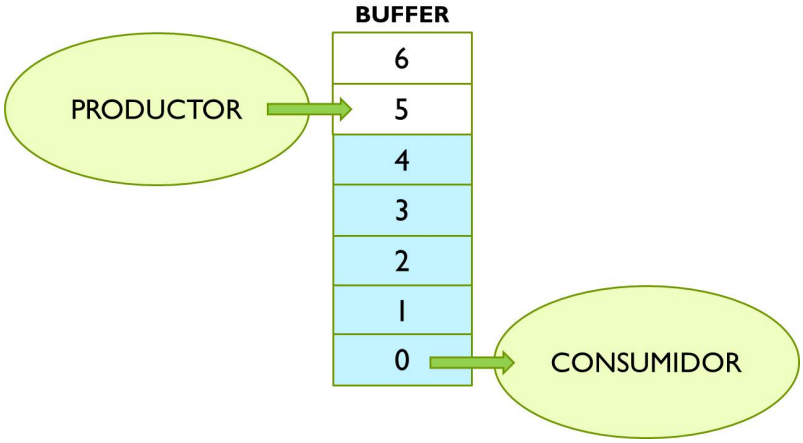

Figura 1.1: Ús d'un *buffer* de capacitat limitada.

- `out`. Indicaria quina component manté l'element que algun consumidor extraurà a continuació. En la figura 1.1 el seu valor seria 0.
- `size`. Mantindria la grandària del *buffer*. En aquest exemple seria 7.
- `numElems`. Mantindria el nombre actual d'elements en el *buffer*. En la figura 1.1 el seu valor seria 5.
- `store`. Vector en el qual s'emmagatzemen els elements del *buffer*.

Perquè un productor inserisca un element en el *buffer*, s'haurà de seguir un algorisme similar al següent:

1. Inserir l'element en la posició `in` del vector `store`.
2. Incrementar el valor de `in` de manera circular. És a dir: `in` ← `(in + 1) MOD size`.
3. Incrementar el nombre d'elements en el *buffer*. És a dir: `numElems++`.

Per la seua banda, perquè un consumidor extraga un element del *buffer*, s'utilitzaria aquest algorisme:

1. Retornar l'element situat en la posició `out` del vector `store`.
2. Incrementar el valor de `out` de manera circular. És a dir: `out` ← `(out + 1) MOD size`.
3. Decrementar el nombre d'elements en el *buffer*. És a dir: `numElems--`.

Aquests algorismes bàsics podrien ocasionar problemes en ser executats de manera concurrent. Vegem-ne alguns exemples:

- Si en la situació mostrada en la figura 1.1 dos processos productors iniciaren alhora la inserció d'un nou element en el *buffer*, tots dos observarien en el primer pas del seu algorisme el mateix valor per a l'atribut `in`. A causa d'això, els dos inseririen els seus elements en la posició 5 i així el segon element inserit sobreescriuria el primer. Posteriorment, tots dos incrementarien el valor de `in` (que passaria primer a valdre 6 i després a valdre 0), així com el valor de `numElems`, que arribaria a 7. No obstant això, no hi hauria cap element en la component 6 del vector `store`.

- De manera similar, si en aquesta mateixa situació inicial dos processos consumidors tractaren d'extraure de manera simultània un element del *buffer*, probablement tots dos obtindrien l'element contingut en la component 0 del vector `store`. Posteriorment s'incrementaria el valor de `out` (fins a arribar a 2) i es decrementaria el valor de `numElems`, que passaria a valdre 3. Cadascun d'ells estaria convençut d'haver obtingut un element diferent, però en lloc d'això, cadascun estaria processant pel seu compte el mateix element. Per contra, l'element mantingut en la component 1 no seria utilitzat mai per cap consumidor i seria sobreescrit posteriorment per algun element inserit per un productor.

- Si a partir de la situació mostrada en la figura 1.1 tres processos productors arribaren a inserir elements sense que cap consumidor n'extraguera cap mentrestant, la capacitat del *buffer* es desbordaria. A causa d'això, la tercera inserció sobreescriuria l'element mantingut en la posició 0 del vector `store`.

- Si a partir de la situació de la figura 1.1 sis processos consumidors extragueren elements del *buffer*, sense que cap productor inserira algun nou element, l'últim d'ells trobaria un *buffer* buit. No obstant això, no en seria conscient i retornaria el contingut de la posició 5 del vector.

Per a evitar aquestes situacions incorrectes, haurien d'imposar-se i respectar-se les següents restriccions:

PC1 Per a evitar la primera i la segona situació d'error s'hauria d'impedir que els mètodes d'accés al *buffer* foren executats per més d'una activitat simultàniament. És a dir, quan alguna activitat haja iniciat l'execució d'algun mètode, cap activitat més podrà executar algun dels mètodes d'accés al *buffer*.

PC2 Per a evitar que es desbordara el *buffer* s'hauria d'impedir que els productors executaren el mètode d'inserció quan el valor dels atributs `size` i `numElems` foren iguals.

Para seguir leyendo haga click aquí