Programmation réseau et concurrente

Benoît Barbot

Département Informatique, Université Paris-Est Créteil, M1

Jeudi 17 janvier 2019, Cours 2

UPEC
Connaissance - Action
UNIVERSITÉ
PARIS-EST CRÉTEIL
VAL DE MARNE

Corrigé TP1

I/O bloquantes/non bloquantes

I/O bloquantes (par default)

- *accept()*, *read()*, *write()* peuvent bloquer
- Interruption du *thread* appelant
- Déblocage quand l'action peut avoir lieu (des données à lire, écrire,...)
- Plusieurs *threads*, problème de concurrence

I/O bloquantes/non bloquantes

I/O bloquantes (par default)

- *accept(), read(), write()* peuvent bloquer
- Interruption du *thread* appelant
- Déblocage quand l'action peut avoir lieu (des données à lire, écrire,...)
- Plusieurs *threads*, problème de concurrence

I/O non bloquantes

- *accept(), read(), write()* retournent immédiatement
- Utilisation de *select()* (ou *poll(), epoll()*)
- Un seul *thread* peut gérer toutes les I/O
- Pas d'exécution parallèle, car 1 seul *thread*

IO non bloquantes : boucle *select(), poll()*

en pseudo-code :

```
openConnections();
Set s = {all socket waiting for I/O}
while(true){
   Set s2 = select(s,timeout);
   for( c : s2 ){
      readWriteAccept(c);
   }
}
```

Boucle *select* en C

```
int retval;
fd_set clientset;
int maxclient;
int nbsocket;
int clients[32];
// Initialisation.
while(1){
  FD_ZERO(&clientset);
  for(int i=0; i<nbsocket ; i++)
    FD_SET(clients[i],&clientset);
  retval = select(maxclient, &clientset,
                                  NULL, NULL, NULL);
  if(retval== -1)error_handling("select");
  for(int i =0; i< nbsocket; i++)
    if(FD_ISSET(clients[i],&clientset)){
          // IO sur client[i]
        }
}
```

Passage au Java, utilisation de NIO

Des classes pour chaque concepts

- DatagramChannel -> socket UDP
- ServerSocketChannel -> socket TCP, pour accepter
- SocketChannel -> socket TCP, pour lire/ecrire
- sockaddr_in(6) -> InetSocketAddress

Passage au Java, utilisation de NIO

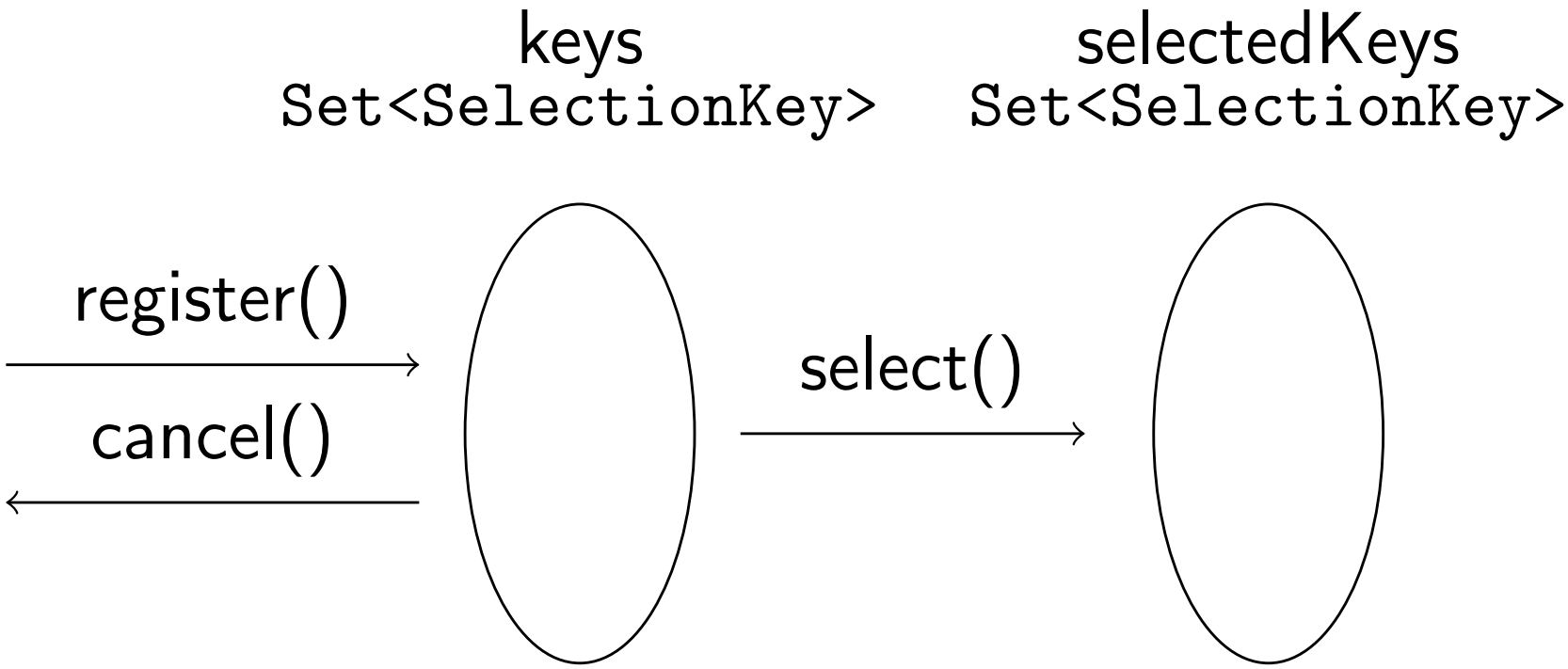
Des classes pour chaque concepts

- DatagramChannel -> socket UDP
- ServerSocketChannel -> socket TCP, pour accepter
- SocketChannel -> socket TCP, pour lire/ecrire
- sockaddr_in(6) -> InetSocketAddress

Select

- Implémenté par Selector.
- Socket marqué comme non bloquant explicitement
- Les sockets s'enregistrent (register) au Selector

Selector


keys
Set<SelectionKey>
selectedKeys
Set<SelectionKey>
register()
cancel()
select()

Java nio buffer

Buffer pour les I/O

- Implémente des I/O efficaces proches du système
- Adapté aux échanges de données binaires
- Sous classé pour chacun des types primitifs
- Gestion propre des encodages

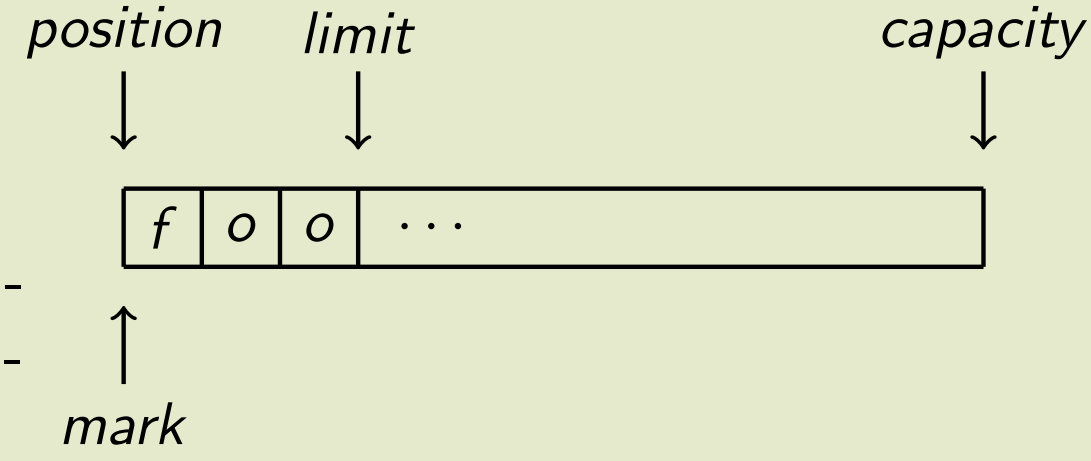
Java nio buffer

Buffer pour les I/O

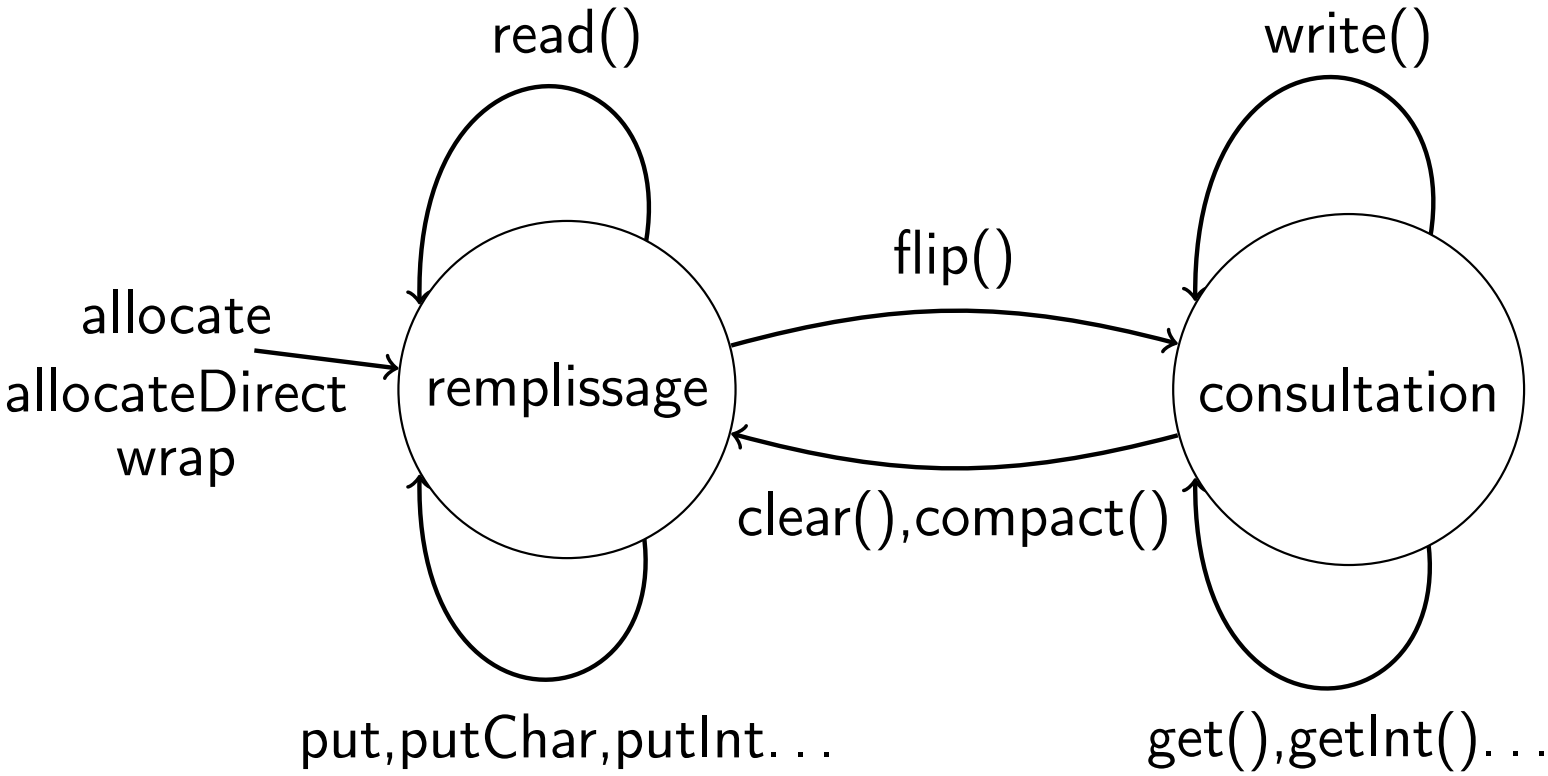
- Implémente des I/O efficaces proches du système
- Adapté aux échanges de données binaires
- Sous classé pour chacun des types primitifs
- Gestion propre des encodages

Description

- Un tableau
- 4 pointeurs
- Deux modes : remplissage et consultation



Utilisation



Encodage

Charset

- Spécifie l'encodage
- ex de charset US-ASCII, ISO646-US, ISO-8859-1,UTF-8,UTF-16

Encodage

Charset

- Spécifie l'encodage
- ex de charset US-ASCII, ISO646-US, ISO-8859-1,UTF-8,UTF-16

String -> ByteBuffer

```
Charset c = Charset.forName("UTF-8");
ByteBuffer bb = c.encode("test UTF-8");
```

Encodage

Charset

- Spécifie l'encodage
- ex de charset US-ASCII, ISO646-US, ISO-8859-1,UTF-8,UTF-16

String -> ByteBuffer

```
Charset c = Charset.forName("UTF-8");
ByteBuffer bb = c.encode("test UTF-8");
```

ByteBuffer -> String

```
Charset c = Charset.forName("UTF-8");
CharBuffer cb = c.decode(bb);
String s = cb.toString();
```