Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Systèmes Embarqués 1 & 2

p.03 - Entrées - Sorties

Classes T-2/I-2 // 2018-2019

Daniel Gachet | HEIA-FR/TIC

p.03 | 27.03.2019

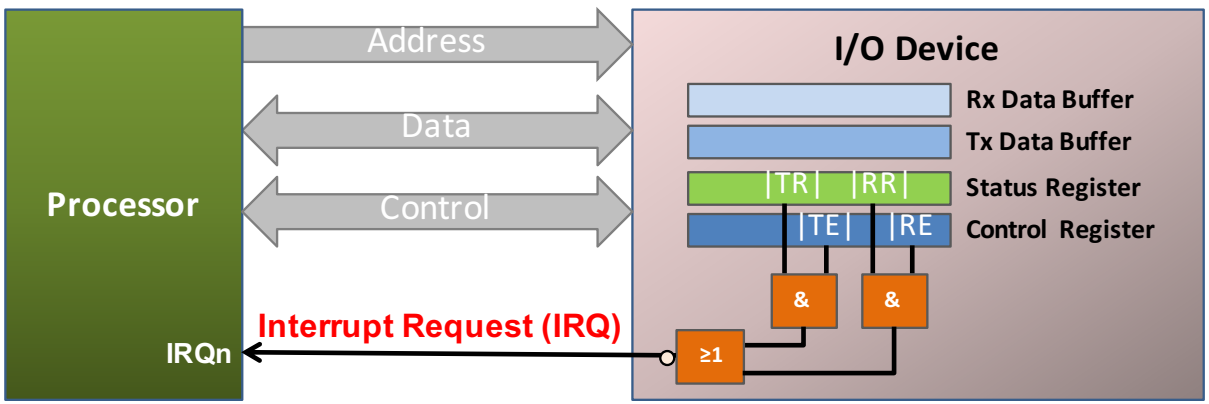
Hes·so
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz

Contenu

- Concept d'entrées/sorties
- Modes de traitement
- Exemples

Concept

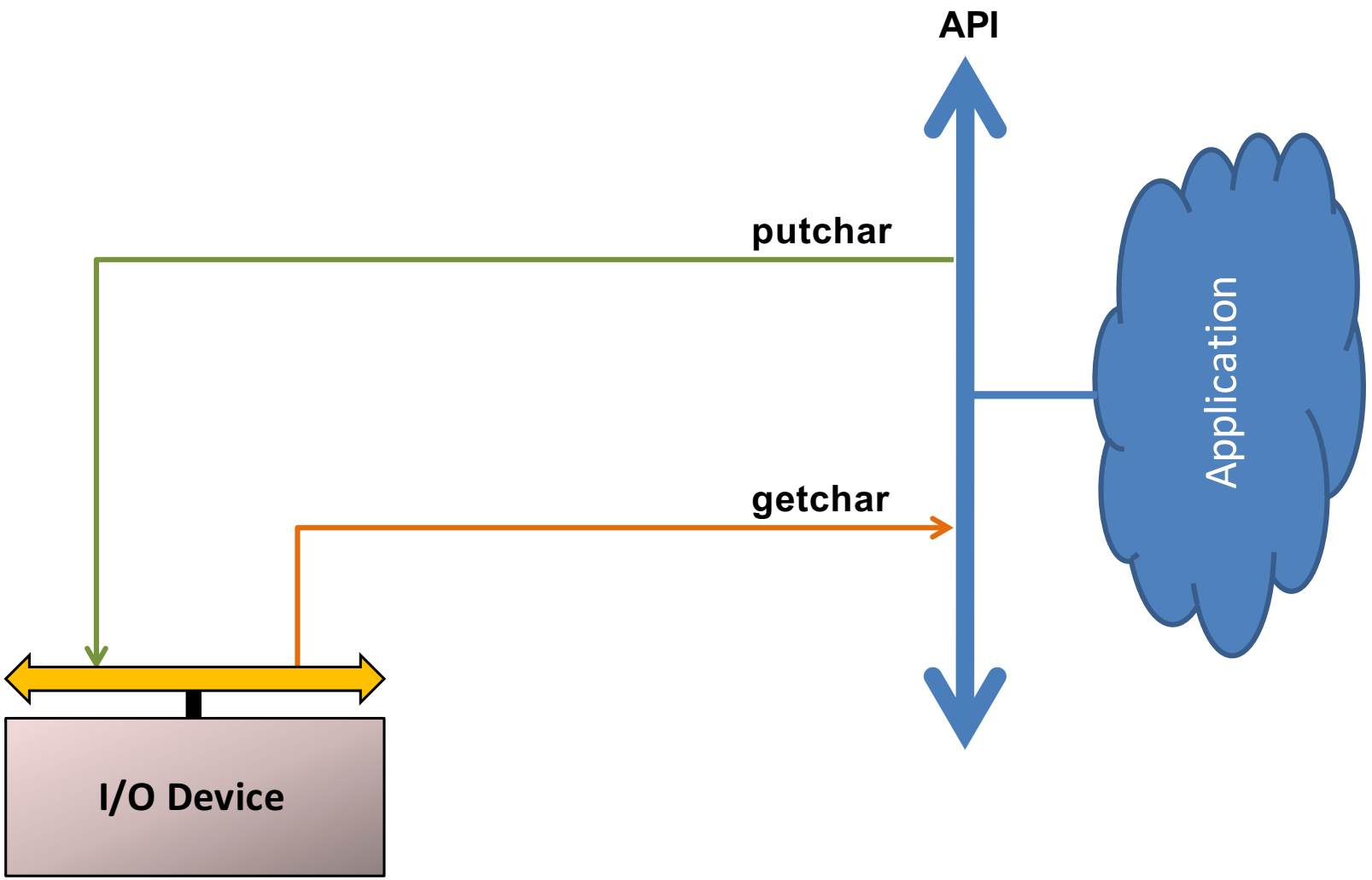
- Les périphériques d'entrées/sorties implémentent généralement une interface se composant de
  - Un buffer pour la réception des données *(Rx Data Buffer)*
  - Un buffer pour la transmission des données *(Tx Data Buffer)*
  - Un registre de statut indiquant l'état du périphérique
    (RR : data received, TR : transmit ready)
  - Un registre de contrôle permettant la configuration et l'opération du périphérique
    (RE : receive interrupt enabled, TE : transmit interrupt enabled)
  - Une logique d'interruptions



Techniques

- Quelques techniques pour piloter des périphériques d'entrées/sorties
  - Scrutation continue
  - Scrutation périodique
  - Interruption
  - Interruption et traitement par thread

Scrutation continue



Scrutation continue - Principe

- Principe *(polling mode)*
  - On lit continuellement l'état du périphérique pour déterminer si celui-ci est prêt à émettre des données ou s'il en a reçu
  - Si le buffer d'émission n'est pas plein, on écrit les données
  - Si des données sont disponibles, on les lit
- \+ Avantages
  - Implémentation très simple
  - Bon temps de réaction lors de changement d'état du périphérique
- \- Désavantages
  - Pertes de performance du µP pour des périphériques lents
  - Aucune autre tâche ne peut être exécutée
- Contraintes
  - Pertes d'information si la fréquence de scrutation est inférieure à la fréquence de réception des données

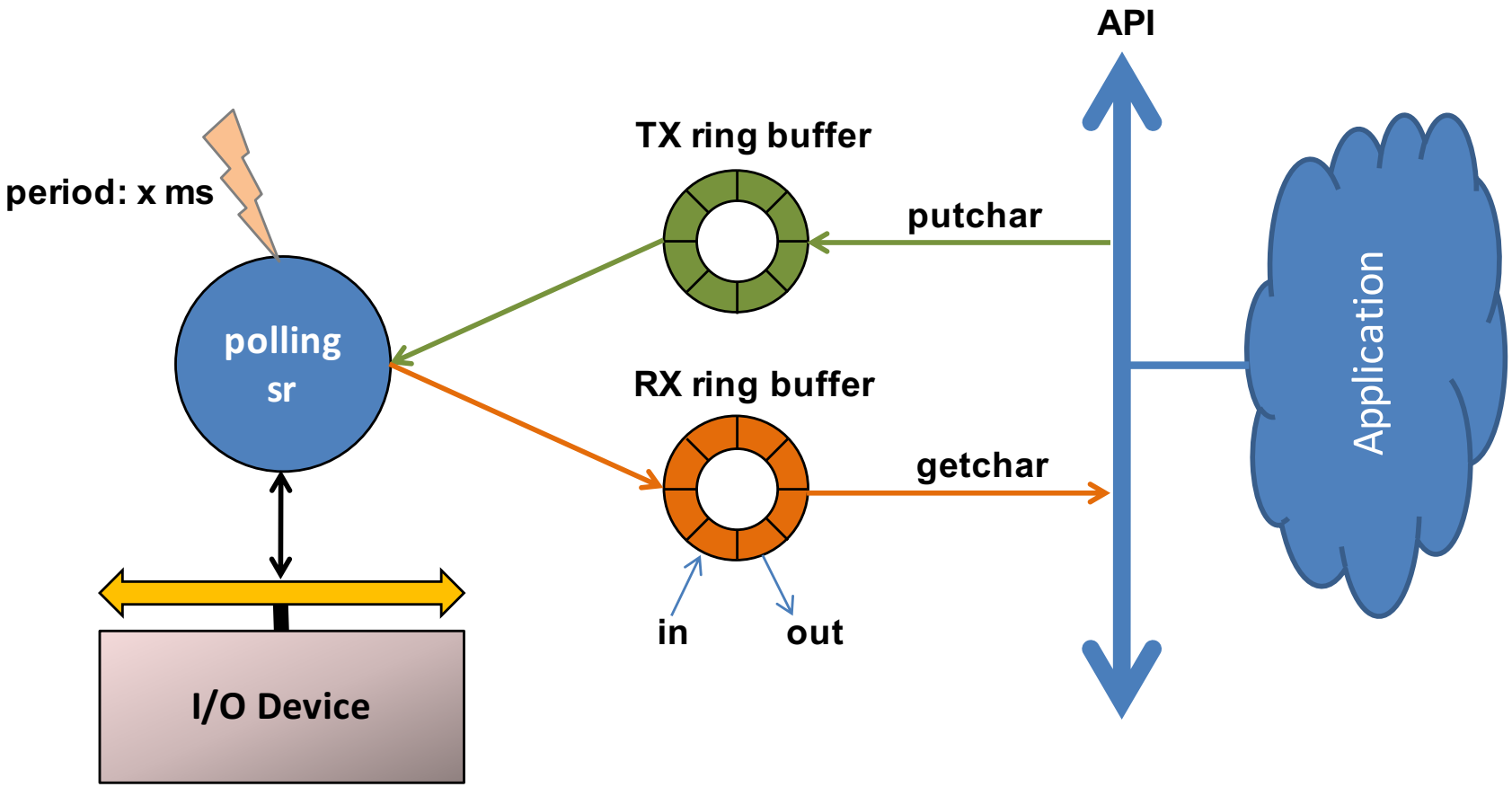
Scrutation continue - Exemple

```
#define STATUS_RR   (1<<0)  // data received
#define STATUS_TR   (1<<1)  // transmit ready

#define CONTROL_RE  (1<<0)  // receive interrupt enabled
#define CONTROL_TE  (1<<1)  // transmit interrupt enabled

struct device_ctrl {
    uint8_t rx;          // receive buffer
    uint8_t tx;          // transmit buffer
    uint8_t status;      // status register
    uint8_t control;     // control register
}

static volatile struct device_ctrl* io_dev = (struct device_ctrl*) 0x10001000;
```

Scrutation continue - Exemple (II)

```
// put a character to the serial interface
void putchar (char data) {
    // wait until transmitter is ready for new data
    while ((io_dev->status & STATUS_TR) == 0);
    io_dev->tx = data;
}


// get a character from the serial interface
char getchar() {
    // wait until a character has been received
    while ((io_dev->status & STATUS_RR) == 0);
    return io_dev->rx;
}
```

Scrutation périodique



Ring buffer conditions:

if (in == out) → buffer empty

if (out == ((in + 1) % SIZE)) → buffer full

Scrutation périodique - Principe

- Principe *(periodic polling mode)*
  - L'utilisation d'un timer permet d'éliminer une partie des carences du mode par scrutation
  - La scrutation ne s'effectue plus continuellement, mais à intervalle régulier
  - Si des données ont été reçues, celles-ci seront placées dans le buffer de réception
  - Si des données peuvent être transmises, on les prendra du buffer d'émission pour les transmettre

putchar

- Si le buffer d'émission n'est pas plein, on place les données dans le buffer, sinon on attend

getchar

- Si des données sont disponibles dans le buffer de réception, on les lit, sinon on attend

Scrutation périodique - Principe (II)

+ Avantages
  - µP est moins chargé et peut traiter d'autres informations entre deux scrutations
  - Bonne maîtrise des surcharges *(overload management)*
  - Traitement de périphériques ne disposant pas de logique d'interruption
- Désavantages
  - Mauvais temps de réaction si la période de scrutation est longue (latence)
  - µP est utilisé/chargé même si aucune donnée n'est à échanger
  - Complexité du code
- Contraintes
  - Fréquence de scrutation doit être supérieure à la fréquence de réception des données

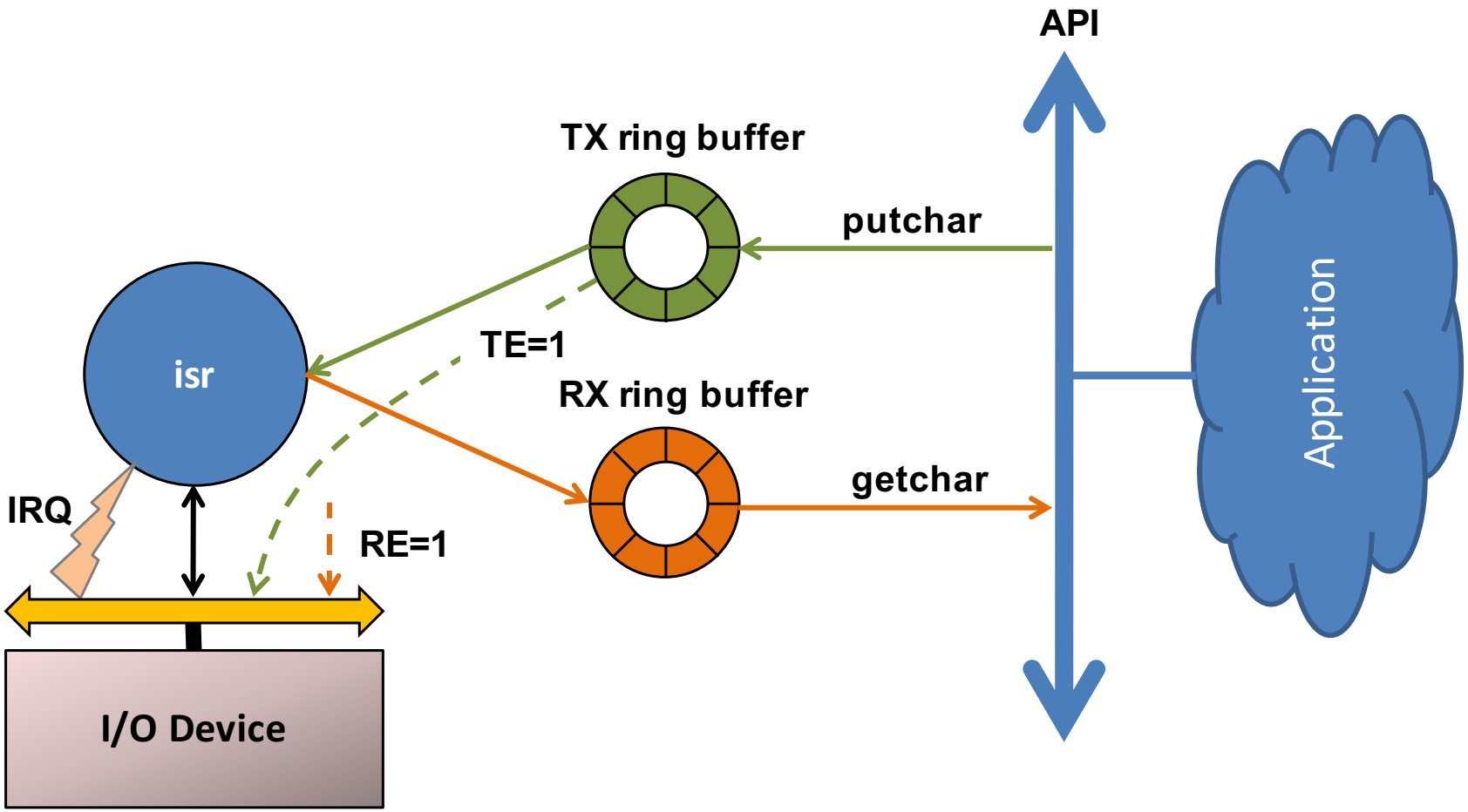
Scrutation périodique - Exemple

```
#define SIZE 256
struct ring_buffer {
  unsigned in;
  unsigned out;
  char buffer[SIZE];
} tx, rx;

void putchar (char data) {
  while (tx.out == ((tx.in+1) % SIZE));
  tx.buffer[tx.in] = data;
  tx.in = (tx.in + 1) % SIZE;
}

char getchar() {
  while (rx.in == rx.out);
  char data = rx.buffer[rx.out];
  rx.out = (rx.out + 1) % SIZE;
  return data;
}
```

Scrutation périodique - Exemple (II)

```
void polling_service_routine() {
  // check if data have been received
  if ((io_dev->status & STATUS_RR) != 0) {
     char data = io_dev->rx;
     if (rx.out != ((rx.in+1) % SIZE)) {
       rx.buffer[rx.in] = data;
       rx.in = (rx.in + 1) % SIZE;
     }
  }

  // check if data could be sent
  if ((io_dev->status & STATUS_TR) != 0) {
     if (tx.in != tx.out) {
       io_dev->tx = tx.buffer[tx.out];
       tx.out = (tx.out + 1) % SIZE;
     }
  }
}
```

Interruption



Interruption - Principe

- Principe *(interrupt mode)*
  - Si des données ont été reçues ou peuvent être transmises, le périphérique signale l'évènement en activant sa ligne d'interruption
  - La routine d'interruption contrôle si des données ont été reçues et les placera dans le buffer de réception
  - Elle contrôlera également si des données peuvent être transmises,dans ce cas elle les prendra du buffer d'émission pour les transmettre

putchar

- Si le buffer d'émission n'est pas plein, on place les données dans le buffer, sinon on attend

getchar

- Si des données sont disponibles dans le buffer de réception, on les lit, sinon on attend

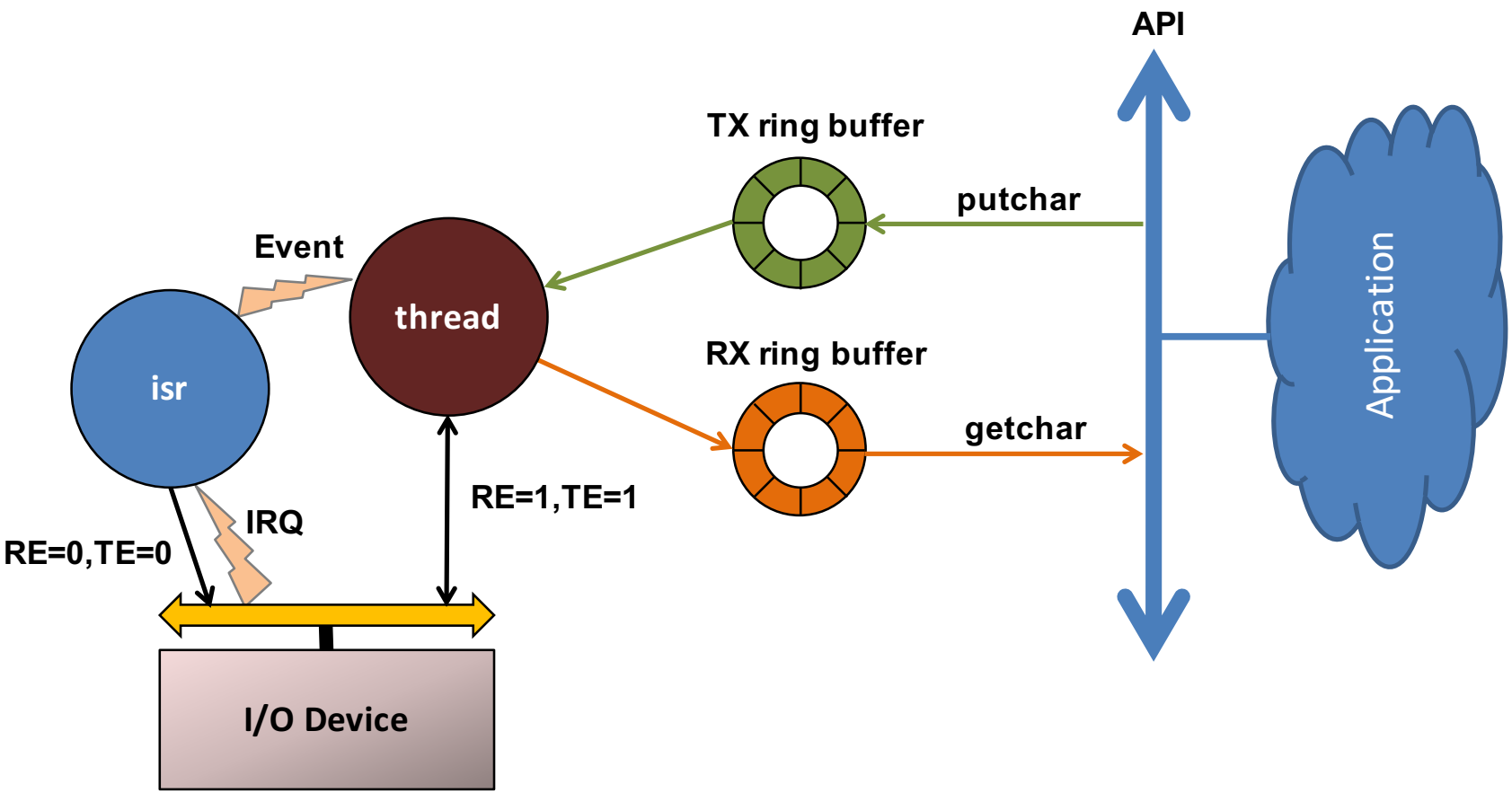
Interruption - Principe (II)

+ Avantages
  - Processeur est utilisé seulement lorsque des informations sont à traiter
  - Bon temps de réaction (attention à la gigue lors de traitements temps réel)
- Désavantages
  - Complexité du code
  - Temps de traitement peut-être exagérément long pour une interruption
  - Difficulté de maîtriser les surcharges (overload management)
  - Coûts matériels
- Contraintes
  - Nécessité de disposer d'une logique d'interruption

Interruption - Exemple

```
void interrupt_service_routine() {
  // data reception: no adaption...
  // data transmission: modifications
  if ((io_dev->status & STATUS_TR) != 0) {
    if (tx.in != tx.out) { // no adaption... }
    // disable interrupt if no more data to be sent
    if (tx.out == tx.in) {
      ctrl->control &= ~CONTROL_TE;
    }
  }
}

void putchar (char data) {
  while (tx.out == ((tx.in+1) % SIZE));
  tx.buffer[tx.in] = data;
  tx.in = (tx.in + 1) % SIZE;
  // enable transmit interrupt generation
  io_dev->control |= CONTROL_TE;
}
```

Interruption et traitement par thread



Interruption et traitement par thread - Principe

- Principe *(interrupt & threading mode)*
  - Si des données ont été reçues ou peuvent être transmises, le périphérique signale l'évènement en activant sa ligne d'interruption
  - La routine d'interruption désactive les interruptions et signale l'évènement à un thread
  - Le thread traite les informations et réactive les interruptions
- \+ Avantages
  - Temps de traitement minimal dans la routine d'interruption
  - Traitement des informations exécuté dans un thread
  - Bonne maîtrise des surcharges et de la latence
  - Bonne réactivité du système
- \- Désavantages
  - Délai entre l'apparition de l'évènement et son traitement
  - Complexité du code
- Contraintes
  - Nécessite l'utilisation d'un système d'exploitation (un noyau)