Διασύνδεση συσκευών εισόδου-εξόδου (Ε/Ε)

Στις επόμενες παραγράφους θα βρείτε παραδείγματα και επεξηγήσεις για τα εξής θέματα:

- **Διασύνδεση ελεγκτών Ε/Ε**. Πώς βλέπει μια ΚΜΕ έναν ελεγκτή Ε/Ε και σε ποιον χώρο διευθύνσεων.
- **Εξυπηρέτηση αιτήσεων Ε/Ε**. Πώς θα καταλάβει η ΚΜΕ ότι έχει ολοκληρωθεί μια αίτηση Ε/Ε.
- **Μηχανισμός DMA**. Πώς μεταφέρονται τα δεδομένα από τη συσκευή Ε/Ε στην κύρια μνήμη του συστήματος και αντίστροφα.

Απλούστευση των παραδειγμάτων.
Τα παραδείγματα είναι τεχνικά απλουστευμένα, χωρίς να παύουν όμως να είναι κοντά στη σημερινή πραγματικότητα.

Διασύνδεση Ε/Ε και χώροι διευθύνσεων

Κάθε ΚΜΕ βλέπει τις συσκευές Ε/Ε (τους ελεγκτές Ε/Ε, για την ακρίβεια) ως θέσεις ανάγνωσης/εγγραφής, όπως ακριβώς βλέπει και τη μνήμη. Σε κάθε ελεγκτή Ε/Ε αντιστοιχεί ένας αριθμός θέσεων «μνήμης», στις οποίες η ΚΜΕ γράφει τις εντολές προς τον ελεγκτή και διαβάζει τα δεδομένα που επιστρέφονται από τη συσκευή.

Παράδειγμα

Υποθέστε ότι υπάρχει στο σύστημά σας συσκευή λήψης απομακρυσμένων μετρήσεων από δίκτυο 100 αισθητήρων. Η συνοπτική λειτουργία της συσκευής έχει ως εξής:

1. Δίνετε την εντολή έναρξης της μέτρησης στη συσκευή
2. Περιμένετε μέχρις ότου η συσκευή να μαζέψει τα δεδομένα.
3. Μεταφέρετε τα δεδομένα (100 μετρήσεις) από τη συσκευή στη μνήμη.
4. Το πρόγραμμα χρήστη συνεχίζει την επεξεργασία των μετρήσεων.

Η συσκευή συνδέεται με την ΚΜΕ μέσω ελεγκτή Ε/Ε, ο οποίος παρουσιάζει 4 θέσεις «μνήμης», του ενός byte η κάθε μία:

- Command (η ΚΜΕ γράφει την εντολή λειτουργίας)
- Status (η ΚΜΕ διαβάζει την κατάσταση της συσκευής)
- Data-in (η ΚΜΕ διαβάζει τα δεδομένα που έρχονται από τη συσκευή)
- Data-out (η ΚΜΕ γράφει δεδομένα προς τη συσκευή)

Τι διευθύνσεις θα δώσουμε σε αυτές τις 4 θέσεις; Υπάρχουν δύο τακτικές: η πρώτη τοποθετεί τις θέσεις των ελεγκτών Ε/Ε **μαζί με τις θέσεις μνήμης σε ενιαίο χώρο** (Ε/Ε απεικονισμένη στη μνήμη – memory mapped I/O) ενώ η δεύτερη τακτική χρησιμοποιεί **ξεχωριστό χώρο διευθύνσεων για τις συσκευές Ε/Ε** (απομονωμένη Ε/Ε – isolated I/O).

Ε/Ε απεικονισμένη στη μνήμη

Έστω ότι το σύστημά μας μπορεί να προσπελάσει έως 4GB μνήμης. Αυτό σημαίνει ότι οι διευθύνσεις μνήμης είναι μεταξύ 0 και 4294967295. Με την τακτική της **απεικονισμένης στη μνήμη Ε/Ε**, οι διευθύνσεις των ελεγκτών Ε/Ε βρίσκονται στον ίδιο, ενιαίο χώρο διευθύνσεων. Οι ακριβείς διευθύνσεις προσδιορίζονται από τη σχεδίαση του συστήματος.

Έστω ότι στο παράδειγμά μας με τη συσκευή απομακρυσμένης μέτρησης οι 4 θέσεις τοποθετούνται στις διευθύνσεις 0 έως 3.

Διεύθυνση	Θέση
0	Command
1	Status
2	Data-in
3	Data-out

Άμεση συνέπεια της τοποθέτησης αυτής είναι ότι **δεν μπορεί η ΚΜΕ να δει την κύρια μνήμη στις διευθύνσεις 0 έως 3**. Οι θέσεις αυτές είναι κατειλημμένες από τον ελεγκτή Ε/Ε. Αν δεν υπάρχουν άλλες συσκευές Ε/Ε στο σύστημα, οι θέσεις μνήμης μπορούν να έχουν διευθύνσεις από το 4 έως το 4294967295.

Η διεύθυνση προσδιορίζει με ποιο τμήμα επικοινωνεί η ΚΜΕ. Π.χ. η (ψεύδο[1])εντολή μηχανής:

```
read[1]
```

απευθύνεται στον ελεγκτή της συσκευής, ενώ η εντολή μηχανής:

```
read[4]
```

θα καταλήξει στην κύρια μνήμη του συστήματος.

Η επιλογή του στόχου γίνεται μέσω λογικών κυκλωμάτων που ελέγχουν **την τιμή της διεύθυνσης** ανάγνωσης ή εγγραφής.

Απομονωμένη Ε/Ε

Με τη δεύτερη πολιτική τοποθέτησης διευθύνσεων (**απομονωμένη Ε/Ε**) οι διευθύνσεις των ελεγκτών Ε/Ε **βρίσκονται σε ξεχωριστό χώρο από τις διευθύνσεις μνήμης**.

Έστω ότι το σύστημα του παραδείγματος ακολουθεί την τακτική αυτή και διαθέτει δύο ανεξάρτητους χώρους διευθύνσεων:

- Διευθύνσεις μνήμης: όπως και πριν, είναι μεταξύ 0 και 4294967295
- Διευθύνσεις ελεγκτών Ε/Ε: μεταξύ 0 και 65535

Η ύπαρξη διαφορετικών χώρων διευθύνσεων **επιτρέπει την εμφάνιση του ίδιου αριθμού διεύθυνσης τόσο στη μνήμη όσο και στους ελεγκτές Ε/Ε**. Π.χ. υπάρχει διεύθυνση μνήμης 0 αλλά και διεύθυνση ελεγκτή Ε/Ε 0. Πώς επιλέγει η ΚΜΕ σε ποιον χώρο απευθύνεται;

Υπάρχουν **διαφορετικές εντολές** για επικοινωνία με τη μνήμη και επικοινωνία με τους ελεγκτές Ε/Ε. Π.χ. η εντολή μηχανής:

```
ioread[0]
```

διαβάζει από τον χώρο των ελεγκτών Ε/Ε, ενώ η εντολή μηχανής:

```
memread[0]
```

απευθύνεται στην κύρια μνήμη.

Με βάση την εκτελούμενη εντολή (ioread/iowrite ή memread/memwrite) η ΚΜΕ παράγει πρόσθετο σήμα (ή σήματα) για να μπορεί η προσπέλαση να οδηγηθεί στο επιθυμητό τμήμα. Για παράδειγμα, μπορεί να υπάρχει ακροδέκτης που είναι 0 για προσπέλαση μνήμης και 1 για προσπέλαση συσκευών Ε/Ε.

1 Στο παράδειγμα δεν δίνεται κάποια αληθινή εντολή μηχανής· αυτό εξαρτάται από την εκάστοτε αρχιτεκτονική της ΚΜΕ. Δείτε το ως «ψευδοεντολή» μηχανής!

Γιατί υπάρχουν οι δύο τεχνικές;
Στο παρελθόν η ανάγκη επικοινωνίας της ΚΜΕ με τις πολύ αργές συσκευές Ε/Ε υπαγόρευε την ύπαρξη εντολών εισόδου-εξόδου με διαφορετικούς χρονισμούς από τις απλές αναγνώσεις/εγγραφές στη μνήμη. Στα σημερινά συστήματα η ΚΜΕ δεν διασυνδέεται απευθείας με τις συσκευές. Παρόλα αυτά υπάρχουν αρχιτεκτονικές που υποστηρίζουν απομονωμένη Ε/Ε, συχνά σε συνδυασμό με την απεικονισμένη στη μνήμη Ε/Ε.

Εξυπηρέτηση αιτήσεων Ε/Ε

Στο παράδειγμα της συσκευής απομακρυσμένης μέτρησης, έστω ότι ο χρήστης καλεί τη συνάρτηση get_readings() για να ανακτήσει τις μετρήσεις σε πίνακα m:

```
char m[100];
get_readings(m);  // &m[0]
```

Η get_readings() θα ανήκει στη βιβλιοθήκη κώδικα που συνοδεύει τη συσκευή και θα καλεί κώδικα του λειτουργικού συστήματος[2] για να επικοινωνήσει με τη συσκευή:

```
....
char buffer[100];

iowrite(0,READ_COMMAND);  // εδώ ξεκινάει η μέτρηση (στη συσκευή)

// τι μπαίνει εδώ;

// εδώ θα πρέπει να έχει τελειώσει η μέτρηση για να προχωρήσω
for (i=0;i<100;i++) {
  buffer[i] = ioread(2);
}
....
// στο σημείο αυτό το ΛΣ αντιγράφει το buffer m του χρήστη και
επιστρέφει.
```

Το πρόβλημα που προκύπτει είναι το πώς θα επιτευχθεί **η αναμονή** ολοκλήρωσης της αίτησης Ε/Ε (της μέτρησης). Όπως είναι ο προηγούμενος κώδικας, η ΚΜΕ θα προχωρήσει στο for της ανάγνωσης των τιμών της μέτρησης **πριν αυτή ολοκληρωθεί**. Ο χρόνος ολοκλήρωσης της μέτρησης είναι πολύ μεγάλος σε σχέση με τον κύκλο ρολογιού της ΚΜΕ: θα περάσουν εκατοντάδες χιλιάδες κύκλοι ρολογιού πριν η συσκευή να είναι σε θέση να δώσει τις νέες τιμές στην ΚΜΕ.

Διαδικασία polling

Polling ονομάζεται **η διαδικασία του επαναληπτικού ελέγχου** της κατάστασης της συσκευής. Στο παράδειγμά μας, υποθέστε ότι μετά την έναρξη της διαδικασίας μέτρησης, η θέση 1 (Status) του ελεγκτή Ε/Ε είναι 0 όσο η μέτρηση δεν έχει ολοκληρωθεί. Αμέσως μετά γίνεται 1. Την τιμή αυτή μπορεί να ελέγχει ο κώδικας του λειτουργικού συστήματος για να αποφασίσει εάν μπορεί να πάρει τις νέες τιμές:

```
....
char buffer[100];

iowrite(0,READ_COMMAND);  // εδώ ξεκινάει η μέτρηση (στη συσκευή)
```

2 Θυμηθείτε ότι μόνο το λειτουργικό σύστημα μπορεί να επικοινωνήσει με τις συσκευές εισόδου-εξόδου.

```
// κώδικας αναμονής
do {      // polling
  ready = ioread(1);
} while (ready==0);

// εδώ θα πρέπει να έχει τελειώσει η μέτρηση για να προχωρήσω
for (i=0;i<100;i++) {
  buffer[i] = ioread(2);
}
....
```

Το polling εγγυάται ότι η ΚΜΕ θα περιμένει ορθά για την ολοκλήρωση της μέτρησης, έχει όμως **ένα πολύ σοβαρό μειονέκτημα**: κατά την αναμονή η ΚΜΕ εκτελεί εκατοντάδες χιλιάδες άχρηστες εντολές ανάγνωσης γιατί η συσκευή είναι πολύ αρχή σε σχέση με την ΚΜΕ.

Πώς μπορούμε να απελευθερώσουμε την ΚΜΕ, έτσι ώστε κατά τον χρόνο ολοκλήρωσης της μέτρησης να εκτελείται κάποιο άλλο πρόγραμμα;

Η χρήση των διακοπών (interrupts)

Οι διακοπές (interrupts) είναι ένας μηχανισμός εξωτερικής ειδοποίησης της ΚΜΕ ότι συνέβη κάτι. Στο παράδειγμα με τη συσκευή μέτρησης, αν ο ελεγκτής Ε/Ε είναι εκείνος που ειδοποιεί την ΚΜΕ, τότε αντί για το polling μπορούμε να εκτελούμε κάτι άλλο μέχρι να ολοκληρωθεί η μέτρηση.

Ο κώδικας για τη μέτρηση έχει τώρα ως εξής:

```
....
char buffer[100];

iowrite(0,READ_COMMAND); // έναρξη μέτρησης

wait(...); // παράμετροι συνάρτησης = ποιος περιμένει τι
// εδώ το λειτουργικό βγάζει τον κώδικα εκτός εκτέλεσης[3] και
αρχίζει να εκτελείται κάποιο άλλο πρόγραμμα
...
```

Μόλις ολοκληρωθεί η μέτρηση, ο ελεγκτής Ε/Ε προκαλεί μια **διακοπή** και εκτελείται η ρουτίνα εξυπηρέτησης διακοπής (για τη συγκεκριμένη συσκευή):

```
ISR() {
  ...
  for (i=0;i<100;i++) {
    buffer[i] = ioread(2);
  }

  wake_up(..);  // το λειτουργικό σύστημα ξεκινά την εκτέλεση του
προγράμματος στο σημείο που είχε σταματήσει[4]
}
```

3 Η κατάσταση του προγράμματος (program counter και άλλοι καταχωρητές) αποθηκεύεται και το λειτουργικό σύστημα τοποθετεί τη διεργασία σε μια ουρά αναμονής. Στη συνέχεια μεταφέρει την εκτέλεση σε κάποιο άλλο πρόγραμμα που είναι έτοιμο να εκτελεστεί.

4 Στην ουσία, το αρχικό πρόγραμμα επιστρέφει από την κλήση της wait().

Μηχανισμός DMA

Στις προηγούμενες ενότητες, ο παρακάτω κώδικας μεταφέρει τα δεδομένα των μετρήσεων από τη συσκευή στην κύρια μνήμη:

```
  for (i=0;i<100;i++) {
    buffer[i] = ioread(2);
  }
```

Στα σύγχρονα υπολογιστικά συστήματα οι πηγές διακοπών είναι πολλές και οι ΚΜΕ θα έπρεπε να αφιερώνουν πολύ χρόνο για να μεταφέρουν τα δεδομένα λέξη-λέξη από τις συσκευές Ε/Ε στη μνήμη και αντίστροφα. Ο μηχανισμός DMA επιτρέπει στον ίδιο τον ελεγκτή Ε/Ε να κάνει τη μεταφορά στη μνήμη όταν ολοκληρωθεί η αίτηση Ε/Ε, απελευθερώνοντας τις ΚΜΕ από το καθήκον αυτό.

Τα βήματα ολοκλήρωσης μιας αίτησης Ε/Ε έχουν τώρα ως εξής:

1. Η ΚΜΕ δίνει την εντολή Ε/Ε στον κατάλληλο ελεγκτή Ε/Ε.
2. Μαζί με την εντολή, η ΚΜΕ δίνει στον ελεγκτή τις παραμέτρους μεταφοράς (διεύθυνση μνήμης και μήκος μεταφοράς).
3. Με την ολοκλήρωση της μεταφοράς, ο ελεγκτής Ε/Ε μεταφέρει τα δεδομένα από/στην κύρια μνήμη και ειδοποιεί μέσω διακοπής την ΚΜΕ ότι η αίτηση Ε/Ε ολοκληρώθηκε.

Στο υποθετικό παράδειγμά μας, ο κώδικας μπορεί να υλοποιηθεί ως εξής

```
....
char buffer[100];

iowrite(3,byte0(&buffer))  //  byte 0 διεύθυνσης
iowrite(3,byte1(&buffer))  //  byte 1 διεύθυνσης
iowrite(3,byte2(&buffer))  // κ.ο.κ. - υποθέστε διεύθυνση 32 bits
iowrite(3,byte3(&buffer))
iowrite(0,READ_COMMAND_DMA);

wait();
```

Ο ελεγκτής Ε/Ε θα προκαλέσει διακοπή αφού μεταφέρει ο ίδιος τα δεδομένα στη μνήμη (στη διεύθυνση του buffer). Στην ρουτίνα εξυπηρέτησης της διακοπής το λειτουργικό απλά θα ξυπνήσει το πρόγραμμα του χρήστη:

```
ISR() {
  ...

  wake_up();  // ξεκινά την εκτέλεση του προγράμματος στο σημείο
που είχε σταματήσει
}
```

Σύγχρονοι μηχανισμοί DMA
Η παρουσίαση του μηχανισμού DMA στο προηγούμενο παράδειγμα είναι απλουστευμένη. Σήμερα είναι πιο σύνθετος και καθοριστικός για την απόδοση ενός υπολογιστικού συστήματος: χρησιμοποιείται για τη γρήγορη μεταφορά δεδομένων ανάμεσα στις κύριες μνήμες των υποσυστημάτων του υπολογιστή (π.χ. μεταξύ της κύριας μνήμης και της μνήμης της κάρτας γραφικών).