Επικοινωνία μεταξύ διεργασιών στα Windows Vista

Στην εργασία αυτή αναλύουμε όλους τους τρόπους διαδιεργασιακής επικοινωνίας στο λειτουργικό σύστημα Windows Vista. Παραθέτουμε επίσης δείγματα κώδικα που χρησιμοποιήθηκαν στην υλοποίησή τους καθώς επίσης και εκτελέσιμα που επιτελούν βασικές λειτουργίες κάθε μεθόδου επικοινωνίας.

Γιώργος Σταματελάτος 4132 – Δημήτρης Χρυσοστομίδης 4120
8/26/2007

Επιβλέπων Καθηγητής: Μηνάς Δασυγένης

Περιεχόμενα

Τρόποι διαδιεργασιακής επικοινωνίας (InterProcess Communication – IPC) στα Windows Vista 1
Πρόχειρο (Clipboard) 3
Αντιγραφή δεδομένων (Data Copy) 6
Δυναμική μεταφορά δεδομένων (Dynamic Data Exchange – DDE/DDX) 7
Κοινόχρηστα αρχεία (File mapping) 9
Ταχυδρομικές θυρίδες (Mailslots) 11
Διοχετεύσεις (Pipes) 14
Μακρινές κλήσεις διαδικασιών (Remote Procedure Call - RPC) 16
Υποδοχές (Sockets) 17

Τρόποι διαδιεργασιακής επικοινωνίας (InterProcess Communication – IPC) στα Windows Vista

Πρόχειρο (Clipboard): Το πρόχειρο ενεργεί ως κεντρικός χώρος καταθέσεων για τη διανομή στοιχείων μεταξύ των εφαρμογών. Όταν ένας χρήστης εκτελεί μια λειτουργία αποκοπής ή αντιγραφής σε μια εφαρμογή, η εφαρμογή αντιγράφει τα επιλεγμένα στοιχεία στο πρόχειρο με κάποια μορφή (format). Οποιαδήποτε άλλη εφαρμογή μπορεί έπειτα να ανακτήσει τα στοιχεία από το πρόχειρο. Το πρόχειρο είναι ένα πολύ αόριστα συνδεμένο μέσο ανταλλαγής, όπου η ανάγκη εφαρμογών συμφωνεί μόνο σχετικά με τη μορφή των δεδομένων. Οι εφαρμογές μπορούν να τρέξουν στον ίδιο υπολογιστή ή σε διαφορετικούς υπολογιστές σε ένα δίκτυο.

Αντιγραφή δεδομένων (Data Copy): Η μέθοδος αυτή επιτρέπει σε μια εφαρμογή να στείλει πληροφορίες σε μια άλλη εφαρμογή χρησιμοποιώντας το μήνυμα WM_COPYDATA. Αυτή η μέθοδος απαιτεί τη συνεργασία μεταξύ της εφαρμογής αποστολής και της λαμβάνουσας εφαρμογής. Η λαμβάνουσα εφαρμογή πρέπει να ξέρει τη μορφή των πληροφοριών και να είναι σε θέση να προσδιορίσει τον αποστολέα. Η εφαρμογή αποστολής δεν μπορεί να τροποποιήσει τη μνήμη που παραπέμπεται από οποιουσδήποτε δείκτες.

Δυναμική μεταφορά δεδομένων (Dynamic Data Exchange – DDE/DDX): Η μέθοδος αυτή είναι ένα πρωτόκολλο που επιτρέπει στις εφαρμογές να ανταλλάσουν στοιχεία με ποικίλα σχήματα. Οι μορφές των δεδομένων που χρησιμοποιούνται από το DDE είναι τα ίδια με εκείνους που χρησιμοποιούνται από το πρόχειρο. Οι πιο συνηθισμένες μορφές δεδομένων είναι κείμενο κωδικοποιημένο Unicode, εικόνες με τη μορφή jpeg αλλά υπάρχουν και πολλές άλλες γενικής χρήσης από τα Windows ή μορφές που αναγνωρίζονται μόνο από συγκεκριμένα προγράμματα. Το DDE μπορεί να θεωρηθεί ως επέκταση του μηχανισμού του προχείρου. Το πρόχειρο σχεδόν πάντα αντιδρά σε μια συγκεκριμένη εντολή του χρήστη, όπως η αντιγραφή ή η επικόλληση σε αυτό. Το DDE επίσης συνήθως αρχίζει από μια εντολή χρηστών, αλλά συνεχίζει συχνά να λειτουργεί χωρίς περαιτέρω αλληλεπίδραση χρηστών. Μπορεί κάποιος επίσης να καθορίσει ειδικούς (custom) τύπους δεδομένων για χρήση σε ειδικού τύπου IPC μεταξύ εφαρμογών με πιο υψηλές απαιτήσεις επικοινωνίας. Οι ανταλλαγές DDE μπορούν να

εμφανιστούν μεταξύ των εφαρμογών που τρέχουν στον ίδιο υπολογιστή ή σε διαφορετικούς υπολογιστές σε ένα δίκτυο IntraNet ή InterNet.

Κοινόχρηστα αρχεία (File mapping): Η χρήση κοινόχρηστων αρχείων επιτρέπει σε μια διεργασία να μεταχειριστεί το περιεχόμενο ενός αρχείου σαν ήταν ένα μπλοκ μνήμης στο διάστημα διευθύνσεων της διεργασίας. Η διεργασία μπορεί να χρησιμοποιήσει τις απλές διαδικασίες δεικτών για να εξετάσει και να τροποποιήσει το περιεχόμενο του αρχείου. Όταν δύο ή περισσότερες διαδικασίες έχουν πρόσβαση στο ίδιο χαρτογραφημένο αρχείο, κάθε διεργασία λαμβάνει έναν δείκτη στη μνήμη στο διάστημα διευθύνσεών της που μπορεί να χρησιμοποιήσει για να διαβάσει ή να τροποποιήσει το περιεχόμενο του αρχείου. Οι διαδικασίες πρέπει να χρησιμοποιήσουν ένα αντικείμενο συγχρονισμού, όπως ένας σηματοφόρος, για να αποτρέψουν έτσι τυχόν προβλήματα από το συναγωνισμό αυτών.

Μπορεί να χρησιμοποιηθεί μια πρόσθετη περίπτωση της χαρτογράφησης αρχείων για επώνυμη κοινή μνήμη μεταξύ των διαδικασιών. Εάν διευκρινίζετε το αρχείο σελιδοποίησης (swapping file) του συστήματος κατά δημιουργία ενός αντικειμένου αρχείου χαρτογράφησης, το αντικείμενο αυτό αντιμετωπίζεται ως κοινό μπλοκ μνήμης (shared memory). Άλλες διαδικασίες μπορούν να έχουν πρόσβαση στον ίδιο μπλοκ με το άνοιγμα του ίδιου mapped file.

Η χαρτογράφηση αρχείων είναι αρκετά αποδοτική και παρέχει επίσης τα χαρακτηριστικά ασφαλείας του λειτουργικού συστήματος που μπορούν να βοηθήσουν στο να αποτραπεί μη εξουσιοδοτημένη χρήση στοιχείων. Η χαρτογράφηση αρχείων μπορεί να χρησιμοποιηθεί μόνο μεταξύ των διαδικασιών σε έναν τοπικό υπολογιστή. Δεν μπορεί να χρησιμοποιηθεί πάνω σε ένα δίκτυο.

Ταχυδρομικές θυρίδες (Mailslots): Τα Mailslots παρέχουν τη μονόδρομη επικοινωνία. Οποιαδήποτε διαδικασία που δημιουργεί ένα mailslot είναι ένας κεντρικός υπολογιστής mailslot (Mailslot Server). Άλλες διαδικασίες, αποκαλούμενες mailslot πελάτες (Mailslot Clients), στέλνουν μηνύματα στο mailslot server με το γράψιμο ενός μηνύματος στη θύρα επικοινωνίας του mailslot. Τα εισερχόμενα μηνύματα επισυνάπτονται πάντα στο mailslot. Το mailslot σώζει τα μηνύματα έως ότου τα έχει διαβάσει ο κεντρικός υπολογιστής mailslot. Μια διαδικασία μπορεί να είναι και ένας διακομιστής mailslot και πελάτης mailslot, οπότε η διπλής κατεύθυνσης επικοινωνία είναι δυνατή χρησιμοποιώντας πολλαπλά mailslots.

Ένας πελάτης mailslot μπορεί να στείλει ένα μήνυμα σε ένα mailslot στον τοπικό υπολογιστή του, σε ένα mailslot σε έναν άλλο υπολογιστή, ή σε όλα τα mailslots με το ίδιο όνομα σε όλους τους υπολογιστές σε ένα συγκεκριμένο domain. Η μετάδοση μηνυμάτων σε όλα τα mailslots σε μια περιοχή δεν μπορεί να είναι περισσότερα από 400 bytes, ενώ τα μηνύματα που στέλνονται σε ένα συγκεκριμένο mailslot περιορίζονται μόνο από το μέγιστο μέγεθος μηνυμάτων που διευκρινίζεται από τον διακομιστήmailslot όταν δημιούργησε το mailslot.

Τα mailslots μοιάζουν πολύ με τα sockets που θα εξετάσουμε πιο κάτω.

Διοχετεύσεις (Pipes): Υπάρχουν δύο τύποι διοχετεύσεων για τη διπλής κατεύθυνσης επικοινωνία: ανώνυμα pipes και επώνυμα pipes. Τα ανώνυμα pipes επιτρέπουν σε σχετικές διεργασίες να ανταλλάσουν δεδομένα. Συνήθως, ένα ανώνυμο pipe χρησιμοποιείται για τον επαναπροσανατολισμό της πρότυπης εισόδου ή εξόδου στις διαδικασίας παιδιών (child process) έτσι ώστε μπορεί να ανταλλάξει τα στοιχεία με τη διαδικασία γονέων του. Για να ανταλλάσσονται τα δεδομένα και στις δύο κατευθύνσεις (duplex), πρέπει να δημιουργηθούν δύο ανώνυμοι σωλήνες. Η διαδικασία γονέα γράφει τα στοιχεία στο pipe με τη χρησιμοποίηση

δεικτών εγγραφής, ενώ η διαδικασία παιδιών διαβάζει τα στοιχεία από εκείνο το pipe χρησιμοποιώντας δείκτη ανάγνωσης. Ομοίως, η διαδικασία παιδιών γράφει τα στοιχεία στο άλλο pipe και η διαδικασία γονέα διαβάζει από αυτό. Οι ανώνυμες διοχετεύσεις δεν μπορούν να χρησιμοποιηθούν πέρα από έναν υπολογιστή, ούτε μπορούν να χρησιμοποιηθούν μεταξύ ανεξάρτητων διαδικασιών.

Οι επώνυμες διοχετεύσεις χρησιμοποιούνται για να μεταφέρουν δεδομένα μεταξύ διεργασιών που δεν είναι σχετικές και μεταξύ διεργασιών σε διαφορετικούς υπολογιστές. Συνήθως, ένας διακομιστής επώνυμου pipe δημιουργεί ένα pipe με ένα συγκεκριμένο όνομα που πρόκειται να κοινοποιηθεί στους πελάτες του και αποτελεί το χαρακτηριστικό της επικοινωνίας. Μια διεργασία πελάτη επώνυμου pipe που γνωρίζει το χαρακτηριστικό όνομα του σωλήνα μπορεί να ανοίξει το άλλο «άκρο» της, κάτω από τους περιορισμούς που έχει ορίσει ο διακομιστής. Αφότου και ο server και ο client έχουν συνδεθεί στο pipe μπορούν να ανταλλάξουν δεδομένα με λειτουργίες ανάγνωσης και εγγραφής στο pipe (όπως με τα ανώνυμα pipes).

Όπως και τα mailslots, τα pipes μοιάζουν και αυτά με τα sockets.

Μακρινές κλήσεις διαδικασιών (Remote Procedure Call - RPC): Το πρωτόκολλο RPC επιτρέπει στις εφαρμογές να καλούν συναρτήσεις απομακρυσμένα. Επομένως, το RPC καθιστά την διαδιεργασιακή επικοινωνία τόσο εύκολη όπως καλώντας μια συνάρτηση. Το RPC λειτουργεί μεταξύ των διεργασιών σε έναν υπολογιστή ή σε διαφορετικούς υπολογιστές σε ένα δίκτυο.

Το RPC που παρέχεται από τα Windows είναι συμβατό με το Open Software Foundation (OSF) Distributed Computing Environment (DCE). Αυτό σημαίνει ότι οι εφαρμογές που χρησιμοποιούν RPC είναι σε θέση να επικοινωνήσουν με τις εφαρμογές που τρέχουν σε άλλα λειτουργικά συστήματα συμβατά με DCE. Το RPC υποστηρίζει αυτόματη μετατροπή δεδομένων για να εξασφαλίσει συμβατότητα με τις διαφορετικές αρχιτεκτονικές υλικού και με τις διαφορετικές κωδικοποιήσεις μεταξύ περιβαλλόντων.

Οι πελάτες RPC και οι διακομιστές συνδέονται στενά διατηρώντας έτσι υψηλή απόδοση. Το σύστημα κάνει εκτενή χρήση RPC για να διευκολύνει μια σχέση πελατών/διακομιστών μεταξύ των διαφορετικών μερών του λειτουργικού συστήματος.

Υποδοχές (Sockets): Οι υποδοχές των Windows είναι μια διεπαφή ανεξάρτητη από το πρωτόκολλο επικοινωνίας. Εκμεταλλεύεται τις ικανότητες επικοινωνίας των επιμέρους πρωτοκόλλων. Στην δεύτερη έκδοση των sockets (Windows Sockets 2 – WinSock2), ένας δείκτης σε socket μπορεί να χρησιμοποιηθεί προαιρετικά σαν έναν δείκτη σε αρχείο με τις τυποποιημένες I/O λειτουργίες αρχείων.

Τα Windows Sockets είναι βασισμένες στην θεωρία των sockets που διαδόθηκαν πρώτα από τη διανομή λογισμικού του Berkeley (Berkeley Software Distribution - BSD). Μια εφαρμογή που χρησιμοποιεί τις Windows Sockets μπορεί να επικοινωνήσει με άλλη εφαρμογή υποδοχών σε άλλους τύπους συστημάτων. Παρόλ' αυτά, δεν υποστηρίζουν όλοι οι προμηθευτές υπηρεσιών μεταφοράς όλες τις διαθέσιμες επιλογές.

Πρόχειρο (Clipboard)

Οι μέθοδοι που μπορεί να χρησιμοποιήσεις κανείς σχετικά με το πρόχειρο είναι δύο: εντολή ανάγνωσης (διαβάζει τα περιεχόμενα του προχείρου) και εγγραφή (αντικαθιστά ή αλλάζει τα περιεχόμενα του προχείρου). Όπως είδαμε πιο πριν το πρόχειρο είναι ένα είδος

κοινής μνήμης για το λειτουργικό σύστημα συνεπώς κάθε διεργασία έχει πρόσβαση σε αυτό. Με αυτό τον τρόπο μπορούμε να πετύχουμε ένα είδος επικοινωνίας μεταξύ δύο ή περισσοτέρων διεργασιών.

Η μορφή των δεδομένων που βρίσκεται στο πρόχειρο μπορεί να είναι κάποια από τις προκαθορισμένες μορφές από τα Windows ή κάποια άλλη μορφή που ορίζεται από το χρήστη. Κάθε μορφή προσδιορίζεται από θετικό ακέραιο αριθμό (unsigned integer). Για τις τυποποιημένες (προκαθορισμένες) μορφές δεδομένων, αυτή η τιμή είναι μια σταθερά που καθορίζεται στο αρχείο Winuser.h. Για τις ορισμένες από το χρήστη μορφές (user-defined) είναι η επιστροφής τιμής της συνάρτησης RegisterClipboardFormat. Δεν είναι όλες οι μορφές δυνατόν να αναγνωριστούν από όλα τα προγράμματα/διεργασίες. Κάθε πρόγραμμα πρέπει να έχει το κομμάτι της αποκωδικοποίησης των δεδομένων ώστε να μετατραπούν αυτά σε μια πιο χρήσιμη μορφή (όπως αλφαριθμητικά, εικόνες κ.ά.). Οι πιο συνηθισμένες μορφές δεδομένων είναι οι παρακάτω: CF_BITMAP (εικόνα τύπου bitmap), CF_UNICODETEXT (κείμενο κωδικοποιημένο σε Unicode), CF_TEXT (κείμενο σε μορφή ANSI), CF_WAVE (PCM ασυμπίεστη μορφή ήχου). Σύνθετες εφαρμογές υποστηρίζουν εγγραφή και ανάγνωση από το πρόχειρο σε πολλά format, για παράδειγμα εγγραφή ενός αλφαριθμητικού τόσο σε μορφή Unicode όσο και σε ASCII.

Από προγραμματιστικής σκοπιάς, για να τοποθετήσει τα δεδομένα ή να ανακτήσει δεδομένα από το πρόχειρο, μία διεργασία πρέπει πρώτα να ανοίξει το πρόχειρο με τη χρήση της συνάρτησης API OpenClipboard. Η συνάρτηση αυτή θα επιστρέψει μεταβλητή τύπου bool υποδεικνύοντας έτσι αν η απόπειρα ήταν επιτυχής.

```
if (OpenClipboard())
{
    //Επεξεργασία προχείρου
}
```

Μόνο μια διεργασία μπορεί να έχει το πρόχειρο ανοικτό σε μία συγκεκριμένη χρονική στιγμή. Για να ανακαλύψει κανείς ποια διεργασία έχει ανοιχτό το πρόχειρο μπορεί να καλέσει τη συνάρτηση GetOpenClipboardWindow. Όταν η διαδικασία ανάγνωσης ή εγγραφής τελειώσει η διεργασία πρέπει να καλέσει τη συνάρτηση CloseClipboard ώστε να αποδεσμεύσει τυχόν πόρους και να επιτρέψει σε μια άλλη διεργασία πρόσβαση σε αυτό.

```
if (OpenClipboard())
{
    //Επεξεργασία προχείρου
    CloseClipboard();
}
```

Αφού το πρόχειρο έχει ανοίξει, η διεργασία πρέπει να καλέσει τη συνάρτηση EmptyClipboard.

```
if (OpenClipboard())
{
    EmptyClipboard();
    //Επεξεργασία προχείρου – διαδικασίες ανάγνωσης/εγγαφής
    CloseClipboard();
}
```

Αυτή η συνάρτηση στέλνει το μήνυμα WM_DESTROYCLIPBOARD στον προηγούμενο ιδιοκτήτη του προχείρου, ελευθερώνει τους πόρους που συνδέονται με τα στοιχεία του και ορίζει την ιδιοκτησία του στην διεργασία που έχει το πρόχειρο ανοιχτό.

Οι συναρτήσεις που διαβάζουν και γράφουν στο πρόχειρο είναι αντίστοιχα οι GetClipboardData και SetClipboardData.

Η συνάρτηση εγγραφής δέχεται δύο τυπικές παραμέτρους: ένα αναγνωριστικό της μορφής των δεδομένων (ένας unsigned integer όπως αναλύθηκε πιο πάνω) και έναν δείκτη σε θέση μνήμης όπου βρίσκονται τα δεδομένα προς εγγραφή.

```
HGLOBAL hClipboardData;
hClipboardData = GlobalAlloc(GMEM_DDESHARE,
                strData.GetLength()+1);
char * pchData;
pchData = (char*)GlobalLock(hClipboardData);
strcpy(pchData, LPCSTR(strData));
GlobalUnlock(hClipboardData);
SetClipboardData(CF_TEXT,hClipboardData);
```

Η μεταβλητή hClipboardData είναι ο δείκτης στην περιοχή που βρίσκονται τα δεδομένα στον οποίο αντιγράφουμε το αλφαριθμητικό strData και στη συνέχεια καλούμε την συνάρτηση εγγραφής με πρώτη παράμετρο CF_TEXT (κείμενο ANSI) και δεύτερη το δείκτη των δεδομένων. Μετά την κλήση σης συνάρτησης CloseClipboard το πρόχειρο θα περιέχει το αλφαριθμητικό σε μορφή ASCII.

Η συνάρτηση ανάγνωσης επιτελεί την ακριβώς αντίστροφη λειτουργία, δηλαδή δέχεται σαν παράμετρο το αναγνωριστικό της μορφής δεδομένων και επιστρέφει δείκτη στη θέση μνήμης των δεδομένων.

```
HANDLE hClipboardData = GetClipboardData(CF_TEXT);
char *pchData = (char*)GlobalLock(hClipboardData);
CString strFromClipboard = pchData;
GlobalUnlock(hClipboardData);
```

Στην πραγματικότητα οι συναρτήσεις του Windows API που σχετίζονται με το πρόχειρο είναι πολύ περισσότερες από τις συναρτήσεις ανοίγματος, ανάγνωσης, εγγραφής, αδειάσματος και κλεισίματος του προχείρου. Υπάρχουν συναρτήσεις που υλοποιούν πιο περίπλοκες λειτουργίες όπως οι: AddClipboardFormatListener (δημιουργεί ένα είδος δυναμικής επικοινωνίας με το πρόχειρο που εξυπηρετεί την άμεση ανταλλαγή δεδομένων χωρίς να χρειάζεται κάθε φορά κλήση στις συναρτήσεις ανάγνωσης και εγγραφής), CountClipboardFormats (επιστρέφει το πλήθος των διαφορετικών μορφών δεδομένων που βρίσκονται στο πρόχειρο), GetUpdatedClipboardFormats (επιστρέφει τα υποστηριζόμενα format). Επίσης το πρόχειρο υποστηρίζει «ειδοποιήσεις» στις διεργασίες που συνεργάζονται με αυτό. Τυπικά παραδείγματα ειδοποιήσεων (notifications) είναι ειδοποίηση όταν τα περιεχόμενα αλλάξουν, όταν κάποια διεργασία εισέλθει ή εξέλθει από τη λίστα των θεατών (viewer) του Clipboard κ.ά.

Το πρόχειρο είναι ένας αποτελεσματικός και εύκολος τρόπος διαδιεργασιακής επικοινωνίας τόσο μεταξύ διεργασιών που τρέχουν στο λειτουργικό τοπικά όσο και διεργασίες που τρέχουν σε διαφορετικούς υπολογιστές (υπάρχει δυνατότητα κλήσεων απομακρυσμένα). Παρόλ' αυτά όμως, επειδή το πρόχειρο εμπλέκει κλήσεις API που τρέχουν σε επίπεδο χρήστη καθιστά την άμεση προσπέλασή του από άλλα λειτουργικά συστήματα. Προϋποθέτει επίσης τη «γνώση» του format από κάθε διεργασία που πρόκειται να το αξιοποιήσει. Τέλος, εγκυμονεί κινδύνους ασφαλείας μιας και κάθε διεργασία μπορεί να έχει πρόσβαση σε αυτό, όχι μόνο αυτές που πραγματικά επικοινωνούν.

Αντιγραφή δεδομένων (Data Copy)

Με τη μέθοδο αυτή, μία διεργασία μπορεί να στείλει δεδομένα σε μια άλλη διεργασία χρησιμοποιώντας το μήνυμα WM_COPYDATA. Η διαδικασία αυτή είναι μια ειδικευμένη περίπτωση της γενικής ρουτίνας SendMessage του API των Windows. Η σύνταξη της συνάρτησης είναι η εξής:

```
LRESULT SendMessage(
   HWND hWnd,
   UINT Msg,
   WPARAM wParam,
   LPARAM lParam
);
```

Όπου hWnd είναι δείκτης στη διεργασία/προορισμό του μηνύματος, Msg είναι ένας θετικός ακέραιος που προσδιορίζει τον τύπο του μηνύματος και wParam και lParam επιπλέον πληροφορίες σχετικές με το μήνυμα. Να σημειώσουμε ότι η παράμετρος hWnd είναι δείκτης σε ένα παράθυρο που όμως συγχέεται με μια διεργασία. Το μήνυμα παραδίδεται ασύγχρονα στον προορισμό του και η διεργασία/παραλήπτης φροντίζει στην εξυπηρέτηση του και περεταίρω επεξεργασία.

Στη συγκεκριμένη περίπτωση της διαδιεργασιακής επικοινωνίας το μήνυμα προς αποστολή είναι το WM_COPYDATA. Συνεπώς η συνάρτηση παίρνει τη μορφή:

```
LRESULT SendMessage(hWnd, WM_COPYDATA, wParam, lParam);
```

Η μορφή των δεδομένων που «ταξιδεύουν» με τη συγκεκριμένη μέθοδο είναι συγκεκριμένη και ορίζεται ως μια δομή δεδομένων με όνομα COPYDATASTRUCT. Ο ορισμός της είναι ο εξής:

```
typedef struct tagCOPYDATASTRUCT {
   ULONG_PTR dwData;
   DWORD cbData;
   PVOID lpData;
} COPYDATASTRUCT
```

Όπου dwData είναι τα δεδομένα προς αποστολή, cbData είναι το μέγεθος των δεδομένων που ορίζει ο δείκτης lpData και LpData ο δείκτης αυτός.

Αφού η διεργασία/αποστολέας ορίσει μια μεταβλητή τύπου COPYDATASTRUCT, την περνάει σαν όρισμα στην SendMessage (συνήθως σαν την τέταρτη παράμετρο lParam) και στέλνει το μήνυμα. Αφού ο παραλήπτης λάβει το μήνυμα μπορεί να στείλει κι αυτός πάλι πίσω. Η διαδικασία παραλαβής του μηνύματος γίνεται με χρήση του γενικού υποθετικού βρόγχου για τα προγράμματα Win32.

Πώς όμως μπορούμε να βρούμε το δείκτη μιας διεργασίας ώστε να μπορούμε να στείλουμε το μήνυμα; Το API των Windows προσφέρει τη συνάρτηση FindWindow, η οποία επιστρέφει το δείκτη μιας διεργασίας που πληροί κάποιες προϋποθέσεις. Αυτές είναι το όνομα του παραθύρου που εκτελεί τη διεργασία και το είδος του. Ένα απλό παράδειγμα αποστολής ενός μηνύματος:

```
CWnd *pOtherWnd = CWnd::FindWindow(NULL, L"TargetName");
COPYDATASTRUCT cpd;
cpd.dwData = 0;
CString txt;
c_TxText.GetWindowText(txt);
cpd.cbData = txt.GetLength();
```

```
cpd.lpData = (void*)txt.GetBuffer(txt.GetLength());
copyDataResult = pOtherWnd->SendMessage(WM_COPYDATA,
            (WPARAM)AfxGetApp()->m_pMainWnd->GetSafeHwnd(),
            (LPARAM)&cpd);
txt.ReleaseBuffer();
```

Στο παράδειγμα αυτό, η SendMessage καλείται με έναν ειδικό τρόπο σαν μέλος του αντικειμένου pOtherWnd που εννοείται πως είναι η πρώτη παράμετρος της εντολής, δηλαδή ο δείκτης της διεργασίας/παραλήπτη.

Η μέθοδος της αντιγραφής δεδομένων σαν διαδιεργασιακή επικοινωνία θεωρείται εύκολη και απλή στην υλοποίηση. Δεν ορίζονται κίνδυνοι «κρυφακοής» μιας και τα μηνύματα παραδίδονται κατευθείαν στον προορισμό τους χωρίς άλλο μεσολαβητή (πλην του λειτουργικού φυσικά). Όμως, η μέθοδος αυτή αντιμετωπίζει μερικά προβλήματα. Διεργασίες που κατοικούν σε διαφορετικούς υπολογιστές δεν μπορούν να ανταλλάξουν τέτοιου είδους μηνύματα καθιστώντας έτσι τη επικοινωνία τοπική. Επίσης προϋποθέτει τη «συνεννόηση» μεταξύ αποστολέα και παραλήπτη σχετικά με τον τύπο των δεδομένων ώστε να μπορεί ο παραλήπτης να τα αποκωδικοποιήσει. Ακόμα και αν ο παραλήπτης μπορεί να αποκωδικοποιήσει τη μορφή χρειάζεται ρητές οδηγίες για το ποια είναι αυτή, σε αντίθεση με τη μέθοδο του προχείρου όπου η μορφή εξάγεται από αυτό.

Δυναμική μεταφορά δεδομένων (Dynamic Data Exchange – DDE/DDX)

Το πρωτόκολλο DDE είναι μία συλλογή μηνυμάτων και κατευθυντήριων οδηγιών. Στέλνει μηνύματα μεταξύ εφαρμογών που μοιράζονται δεδομένα και χρησιμοποιούν διαμοιραζόμενη μνήμη για να ανταλλάσσουν δεδομένα μεταξύ εφαρμογών. Οι εφαρμογές μπορούν να χρησιμοποιούν το πρωτόκολλο DDE για μιας φοράς μεταφορά δεδομένων και για συνεχείς ανταλλαγές στις οποίες οι εφαρμογές στέλνουν ενημερώσεις η μια στην άλλη μόλις νέα δεδομένα γίνουν διαθέσιμα.

Τα Windows επίσης υποστηρίζουν την βιβλιοθήκη διαχείρισης DDE (Dynamic Data Exchange Management Library, DDEML). Η DDEML είναι μια βιβλιοθήκη δυναμικής σύνδεσης (dynamic-link library, DLL) που μπορούν να χρησιμοποιούν οι εφαρμογές για να μοιράζονται δεδομένα. Η DDEML παρέχει συναρτήσεις και μηνύματα που απλοποιούν τη δουλειά της προσθήκης της δυνατότητας DDE σε μια εφαρμογή. Αντί της αποστολής, ανάρτησης, και επεξεργασίας μηνυμάτων DDE άμεσα, μια εφαρμογή χρησιμοποιεί τις συναρτήσεις DDEML για να διαχειριστούν τις συζητήσεις DDE.

Το DDEML επίσης παρέχει μια ευκολία για διαχείριση των δεδομένων που οι εφαρμογές DDE μοιράζονται. Αντί της χρήσης ατόμων και δεικτών προς αντικείμενα διαμοιραζόμενης μνήμης, οι εφαρμογές DDE δημιουργούν και ανταλλάσσουν λαβές σειρών δεδομένων (string), που αναγνωρίζουν τις σειρές δεδομένων, και λαβές δεδομένων, οι οποίες αναγνωρίζουν αντικείμενα μνήμης. Το DDEML επίσης καθιστά δυνατό για μία εφαρμογή εξυπηρετητή να εγγραφεί για τα ονόματα εξυπηρέτησης που υποστηρίζει. Τα ονόματα γνωστοποιούνται σε άλλες εφαρμογές στο σύστημα, οι οποίες μπορούν να χρησιμοποιήσουν τα ονόματα αυτά για να συνδεθούν στον εξυπηρετητή. Επιπλέον, το DDEML εγγυάται συμβατότητα μεταξύ εφαρμογών DDE αναγκάζοντάς τες να υλοποιήσουν το πρωτόκολλο DDE σε ένα συνεπή τρόπο συμπεριφοράς.

Υπάρχουσες εφαρμογές που χρησιμοποιούν το βασισμένο σε μηνύματα πρωτόκολλο DDE είναι πλήρως συμβατές με εκείνες που χρησιμοποιούν το DDEML. Αυτό επειδή μία εφαρμογή που χρησιμοποιεί το βασισμένο σε μηνύματα πρωτόκολλο DDE μπορεί να εγκαθιδρύσει συζητήσεις και να επιτελέσει συναλλαγές με εφαρμογές που χρησιμοποιούν το DDEML. Εξαιτίας των πολλών πλεονεκτημάτων του DDEML, οι νέες εφαρμογές θα πρέπει να το χρησιμοποιούν αντί των μηνυμάτων DDE. Για να χρησιμοποιηθούν τα στοιχεία API του DDEML, πρέπει να συμπεριληφθεί η βιβλιοθήκη του DDEML σε κάθε αρχείο πηγαίου κώδικα που την χρησιμοποιεί, να συνδεθεί με τη βιβλιοθήκη DDEML, και να εγγυηθεί πως η βιβλιοθήκη δυναμικής σύνδεσης DDEML είναι στο μονοπάτι (path) αναζήτησης του συστήματος.

Για να εγκαθιδρυθεί μια επικοινωνία δυναμικής ανταλλαγής δεδομένων (DDE conversation), ο πελάτης στέλνει ένα μήνυμα WM_DDE_INITIATE. Συνήθως, ο πελάτης εκπέμπει αυτό το μήνυμα καλώντας την SendMessage, με πρώτη παράμετρο –1. Αν η εφαρμογή έχει ήδη το δείκτη διεργασίας για την εφαρμογή του εξυπηρετητή, μπορεί να στείλει το μήνυμα απευθείας σε αυτό το παράθυρο. Ο πελάτης προετοιμάζει άτομα (atoms) για το όνομα της διεργασίας και το όνομα του θέματος καλώντας την GlobalAddAtom. Ο πελάτης μπορεί να συνδεθεί σε οποιονδήποτε δυναμικό εξυπηρετητή και για κάθε δυναμικό θέμα (topic) αποθέτοντας μηδενικά (NULL, wildcard) άτομα για την εφαρμογή και το θέμα.

```
SendMessage((HWND) HWND_BROADCAST, // broadcasts message
        WM_DDE_INITIATE,        // initiates conversation
        (WPARAM) hwndClientDDE, // handle to client DDE window
        MAKELONG(atomApplication,   // application-name atom
        atomTopic));            // topic-name atom
```

Η εφαρμογή πελάτη στέλνει ένα μήνυμα **WM_DDE_INITIATE** με αυτά τα δύο άτομα στην παράμετρο *lParam* του μηνύματος. Στην κλήση της συνάρτησης **SendMessage**, ο ειδικός δείκτης παραθύρου –1 οδηγεί το σύστημα να στείλει αυτό το μήνυμα σε όλες τις άλλες ενεργές εφαρμογές. Η **SendMessage** δεν επιστρέφει στην εφαρμογή πελάτη μέχρι όλες οι εφαρμογές που λαμβάνουν το μήνυμα να έχουν, διαδοχικά, επιστρέψει τον έλεγχο στο σύστημα. Αυτό σημαίνει πως όλα τα μηνύματα WM_DDE_ACK που αποστέλλονται για απάντηση από τις εφαρμογές εξυπηρετητή είναι εγγυημένο πως έχουν επεξεργαστεί από τον πελάτη μέχρι την επιστροφή της κλήσης **SendMessage**.

Για να ανακτηθεί ένα αντικείμενο από τον εξυπηρετητή, ο πελάτης στέλνει στον εξυπηρετητή ένα μήνυμα **WM_DDE_REQUEST** καθορίζοντας το αντικείμενο και τη μορφή (format) προς ανάκτηση

```
PostMessage(hwndServerDDE,
    WM_DDE_REQUEST,
    (WPARAM) hwndClientDDE,
    PackDDElParam(WM_DDE_REQUEST, CF_TEXT, atomItem));
```

Στο παράδειγμα αυτό, ο πελάτης καθορίζει τη μορφή (format) του προχείρου (clipboard) CF_TEXT ως το προτιμώμενο format για το αντικείμενο των δεδομένων που ζητήθηκαν.

Αν ο εξυπηρετητής έχει πρόσβαση στο ζητούμενο αντικείμενο και μπορεί να το αποδώσει στο ζητούμενο format, ο εξυπηρετητής αντιγράφει την τιμή του αντικειμένου ως ένα αντικείμενο διαμοιραζόμενης μνήμης και στέλνει στον πελάτη ένα μήνυμα WM_DDE_DATA.

```
PostMessage(hwndClientDDE,
        WM_DDE_DATA,
        (WPARAM) hwndServerDDE,
```

```
        lParam));
```

Ο πελάτης μπορεί να στείλει μία τιμή αντικειμένου στον εξυπηρετητή χρησιμοποιώντας ένα μήνυμα **WM_DDE_POKE**. Ο πελάτης αποδίδει το αντικείμενο προς αποστολή και στέλνει το μήνυμα **WM_DDE_POKE.**

```
PostMessage(hwndServerDDE,
    WM_DDE_POKE,
    (WPARAM) hwndClientDDE,
    PackDDElParam(WM_DDE_POKE,
    (UINT) hPokeData, atomItem));
```

Τα παραπάνω ήταν απλά παραδείγματα της χρήσης του DDE. Η μέθοδος αυτή όμως περιλαμβάνει προχωρημένες μεθόδους ενδοεπικοινωνίας, τις οποίες δεν θα αναφέρουμε.

Η τεχνική της δυναμικής ανταλλαγής δεδομένων δεν είναι τόσο αποδοτική όπως οι νεότερες τεχνολογίες. Παρόλ' αυτά, μπορούμε ακόμη να χρησιμοποιούμε το DDE αν οι άλλοι μηχανισμοί IPC δεν ταιριάζουν ή αν πρέπει να αλληλεπιδράσουμε με μια υπάρχουσα εφαρμογή που υποστηρίζει μόνο DDE.

Κοινόχρηστα αρχεία (File mapping)

Τα κοινόχρηστα αρχεία (file mapping) είναι ο συσχετισμός του περιεχομένου ενός αρχείου με ένα τμήμα του χώρου εικονικών διευθύνσεων μιας διεργασίας. Το σύστημα δημιουργεί ένα file mapping object (γνωστό και ως section object) για να υποστηρίξει αυτή τη σύνδεση. Το file view είναι ένα τμήμα του εικονικού χώρου διευθύνσεων που μια διεργασία χρησιμοποιεί για να προσπελάσει τα περιεχόμενα του αρχείου. Το file mapping επιτρέπει στη διεργασία να χρησιμοποιεί τόσο τυχαία Ε/Ε, όσο και σειριακή Ε/Ε. Επίσης επιτρέπει στη διεργασία να δουλεύει αποτελεσματικά με μεγάλα αρχεία δεδομένων, όπως βάσεις δεδομένων, χωρίς να χρειάζεται να αναρτήσει ολόκληρο το αρχείο στη μνήμη. Πολλαπλές διεργασίες μπορούν επίσης να χρησιμοποιούν αναρτημένα στη μνήμη αρχεία για να μοιράζονται δεδομένα.

Οι διεργασίες διαβάζουν από και γράφουν στο file view χρησιμοποιώντας δείκτες (pointers), όπως ακριβώς θα έκαναν με δυναμικά κατανεμημένη μνήμη. Η χρήση του file mapping βελτιώνει την απόδοση επειδή το αρχείο ανήκει στο δίσκο, αλλά το file view ανήκει στη μνήμη. Το αρχείο στο δίσκο μπορεί να είναι οποιοδήποτε αρχείο που θέλουμε να χαρτογραφήσουμε στη μνήμη, ή ακόμη και το αρχείο σελιδοποίησης του συστήματος. Το file mapping object μπορεί να περιέχει όλο ή μόνο κομμάτι από το αρχείο. Έχει την υποστήριξη του αρχείου στο δίσκο. Αυτό σημαίνει πως όταν το σύστημα «κάνει swap out» σελίδες του file mapping object, οποιεσδήποτε αλλαγές έγιναν στο file mapping object είναι γραμμένες στο αρχείο. Όταν οι σελίδες του file mapping object επιστρέφουν από την εναλλαγή, ανακτώνται από το αρχείο.

Ένα file view μπορεί να περιέχει όλο ή μέρος του file mapping object. Μία διεργασία χειρίζεται επιδέξια το αρχείο μέσω των file views. Μια διεργασία μπορεί να δημιουργήσει πολλαπλά views για ένα file mapping object. Τα file views που δημιουργήθηκαν από κάθε διεργασία ανήκουν στον χώρο εικονικών διευθύνσεων της συγκεκριμένης διεργασίας. Όταν η διεργασία χρειάζεται δεδομένα από ένα τμήμα του αρχείου που δεν είναι στο τρέχον file view, μπορεί να απο-χαρτογραφήσει το τρέχον file view, και μετά να δημιουργήσει ένα καινούριο.

Όταν πολλαπλές διεργασίες χρησιμοποιούν το ίδιο file mapping object για να δημιουργήσουν εικόνες για ένα τοπικό αρχείο, τα δεδομένα είναι συναφή. Αυτό επειδή οι εικόνες περιέχουν πανομοιότυπα αντίγραφα του αρχείου στο δίσκο. Το αρχείο δεν μπορεί να

ανήκει σε απομακρυσμένο υπολογιστή αν θέλουμε να μοιραστούμε μνήμη μεταξύ πολλαπλών διεργασιών.

Το πρώτο βήμα για τη χαρτογράφηση ενός αρχείου είναι να ανοίξουμε το αρχείο καλώντας τη συνάρτηση **CreateFile**. Για να εξασφαλίσουμε ότι οι άλλες διεργασίες δε θα μπορούν να γράψουν στο τμήμα του αρχείου που χαρτογραφήθηκε, θα πρέπει να ανοίξουμε το αρχείο με αποκλειστική πρόσβαση. Επιπλέον, ο χειρισμός του αρχείου θα πρέπει να παραμείνει ανοιχτός έως ότου η διεργασία δε χρειάζεται πια το file mapping object. Ένας εύκολος τρόπος να αποκομίσουμε αποκλειστική πρόσβαση είναι να καθορίσουμε μηδενισμό στην παράμετρο *fdwShareMode* της **CreateFile**. Η επιστροφή της **CreateFile** χρησιμοποιείται από την συνάρτηση **CreateFileMapping** για τη δημιουργία ενός file mapping object.

```
m_hSharedMemoryFile = CreateFileMapping((HANDLE)0xFFFFFFFF,
                NULL,
                PAGE_READWRITE,
                0/*dwMaximumSizeHigh*/,
                dwNumberOfBytesToMap/*dwMaximumSizeLow*/,
                csName.GetBuffer(0));
```

Η συνάρτηση **CreateFileMapping** επιστρέφει μια διεύθυνση στο file mapping object. Αυτή η διεύθυνση θα χρησιμοποιηθεί στη δημιουργία ενός file view κι έτσι θα μπορούμε να προσπελάσουμε τη διαμοιραζόμενη μνήμη. Όταν καλούμε την **CreateFileMapping**, ορίζουμε ένα όνομα αντικειμένου, τον αριθμό bytes που θα χαρτογραφηθούν από το αρχείο, και τον περιορισμό ανάγνωσης/εγγραφής για τη χαρτογραφημένη μνήμη. Η πρώτη διεργασία που καλεί την **CreateFileMapping** δημιουργεί το file mapping object. Μπορούμε να ξέρουμε αν μια επιτυχής κλήση της **CreateFileMapping** δημιούργησε ή άνοιξε το file mapping object ή όχι, καλώντας τη συνάρτηση **GetLastError**. Η **GetLastError** επιστρέφει NO_ERROR στην διεργασία δημιουργίας και ERROR_ALREADY_EXISTS στις επακολουθούσες διεργασίες.

Για να χαρτογραφήσουμε τα δεδομένα από ένα αρχείο στην εικονική μνήμη μιας διεργασίας, πρέπει να δημιουργήσουμε μία εικόνα (view) του αρχείου. Οι συναρτήσεις **MapViewOfFile** και **MapViewOfFileEx** χρησιμοποιούν το δείκτη της επιστρεφόμενης διεύθυνσης file mapping object από την **CreateFileMapping** για να δημιουργηθεί μια εικόνα του αρχείου ή ενός τμήματος του αρχείου στο χώρο εικονικών διευθύνσεων της εφαρμογής. Αυτές οι συναρτήσεις αποτυγχάνουν αν οι σημαίες πρόσβασης έχουν διένεξη με εκείνες που καθορίστηκαν όταν η **CreateFileMapping** δημιούργησε το file mapping object.

```
m_pwData = MapViewOfFile(m_hSharedMemoryFile,
                FILE_MAP_WRITE,
                0/*dwFileOffsetHigh*/,
                0/*dwFileOffsetLow*/,
                dwNumberOfBytesToMap);
```

Η συνάρτηση **MapViewOfFile** επιστρέφει έναν δείκτη για το file view. Με την αποχαρτογράφηση ενός δείκτη στον χώρο διευθύνσεων που καθορίστηκε στην **MapViewOfFile**, μια διεργασία μπορεί να διαβάσει να διαβάσει δεδομένα απ' το αρχείο και να γράψει δεδομένα στο αρχείο. Το γράψιμο στο file view οδηγεί σε αλλαγές στο file mapping object. Το πραγματικό γράψιμο στο αρχείο στο δίσκο το χειρίζεται το σύστημα. Τα δεδομένα δεν ταξιδεύουν ακριβώς τη στιγμή που το file mapping object εγγράφεται. Αντιθέτως, αρκετή από την Ε/Ε του αρχείου αποθηκεύεται στην cache για να βελτιώσει την γενική απόδοση του συστήματος. Οι εφαρμογές μπορούν να επανεγγράψουν αυτή τη συμπεριφορά καλώντας την συνάρτηση **FlushViewOfFile** για να εξαναγκάσουν το σύστημα να εκτελέσει την εγγραφή στο δίσκο αμέσως.

Για να διαβάσουμε από ένα file view, διαφοροποιούμε τον δείκτη που επιστρέφεται από την **MapViewOfFile**.

```
char * pwCData = (char *)m_pwData;
//Επεξεργασία του δείκτη εδώ – εγγραφή/ανάγνωση
```

Όταν μια διεργασία έχει τελειώσει με το file mapping object, θα πρέπει να καταστρέψει όλα τα file views σε αυτό τον χώρο διευθύνσεων χρησιμοποιώντας τη συνάρτηση **UnmapViewOfFile** για κάθε file view.

```
UnmapViewOfFile(m_pwData);
CloseHandle(m_hSharedMemoryFile);
```

Η αποχαρτογράφηση ενός file view ακυρώνει τον δείκτη προς τον εικονικό χώρο διευθύνσεων της διεργασίας. Αν οποιαδήποτε από τις σελίδες του file view έχει αλλάξει από τότε που χαρτογραφήθηκε το view, το σύστημα γράφει τις αλλαγμένες σελίδες του αρχείου στο δίσκο χρησιμοποιώντας caching. Για να δεσμευτούν όλα τα δεδομένα στο δίσκο προτού αποχαρτογραφήσει το file view, χρησιμοποιούμε τη συνάρτηση FlushViewOfFile. Μόλις η κάθε διεργασία τελειώσει χρησιμοποιώντας το file mapping object και έχει αποχαρτογραφήσει όλα τα views, θα πρέπει να κλείσει τον δείκτη του file mapping object και το αρχείο στο δίσκο, καλώντας την **CloseHandle**. Αυτές οι κλήσεις στην **CloseHandle** πέτυχαν ακόμη κι όταν υπάρχουν file views ακόμη ανοιχτά. Παρόλα αυτά, το να αφήνονται file views χαρτογραφημένα προκαλεί διαρροές μνήμης (memory leaks).

Η τεχνική του File Mapping είναι μια αποδοτική μέθοδος που χαρακτηρίζεται από ευκολία στην κατανόηση αφού οι διαδικασίες χρησιμοποιούν δείκτες εγγραφής και ανάγνωσης σε αρχείο για την επικοινωνία. Η μέθοδος όμως αυτή δεν μπορεί να χρησιμοποιηθεί από διεργασίες που φιλοξενούνται σε διαφορετικούς υπολογιστές.

Ταχυδρομικές θυρίδες (Mailslots)

Η ταχυδρομική θυρίδα (*mailslot)* είναι ένας μηχανισμός για μονόδρομη διαδιεργασιακή επικοινωνία (IPC). Οι διεργασίες μπορούν να αποθηκεύουν μηνύματα σε μια ταχυδρομική θυρίδα. Ο ιδιοκτήτης της ταχυδρομικής θυρίδας μπορεί να ανακτά τα μηνύματα που είναι αποθηκευμένα εκεί. Αυτά τα μηνύματα στέλνονται συνήθως μέσω ενός δικτύου είτε σε κάποιον συγκεκριμένο υπολογιστή, είτε σε όλους τους υπολογιστές σε ένα συγκεκριμένο τομέα. Ένας τομέας είναι μία ομάδα από σταθμούς εργασίας και εξυπηρετητές που μοιράζονται ένα όνομα ομάδας.

Μία ταχυδρομική θυρίδα είναι ένα ψευδο-αρχείο που ανήκει στη μνήμη, και χρησιμοποιούνται συγκεκριμένες συναρτήσεις αρχείων για να το προσπελάσουν. Τα δεδομένα σε ένα μήνυμα ταχυδρομικής θυρίδας μπορούν να είναι σε οποιαδήποτε μορφή, αλλά δε μπορούν να είναι μεγαλύτερα από 424 bytes όταν αποστέλλονται μεταξύ υπολογιστών. Σε αντίθεση με τα αρχεία δίσκων, οι ταχυδρομικές θυρίδες είναι προσωρινές. Όταν όλοι οι χειριστές μιας ταχυδρομικής θυρίδας κλείσουν, η ταχυδρομική θυρίδα και όλα τα δεδομένα που περιέχει σβήνονται.

Ένας εξυπηρετητής ταχυδρομικών θυρίδων είναι μια διεργασία που δημιουργεί και κατέχει ένα mailslot. Μόλις ο εξυπηρετητής δημιουργήσει μια ταχυδρομική θυρίδα, λαμβάνει ένα δείκτη/χειριστή ταχυδρομικής θυρίδας. Αυτός ο δείκτης πρέπει να χρησιμοποιείται όταν μια διεργασία διαβάζει μηνύματα από το mailslot. Μόνο η διεργασία που δημιουργεί μια ταχυδρομική θυρίδα ή που έχει αποκτήσει το δείκτη με κάποιον άλλο μηχανισμό (όπως η

κληρονομικότητα) μπορεί να διαβάσει από την ταχυδρομική θυρίδα. Όλες οι ταχυδρομικές θυρίδες είναι τοπικές στην διεργασία που τις δημιουργεί. Μια διεργασία δεν μπορεί να δημιουργήσει μια απομακρυσμένη ταχυδρομική θυρίδα.

Ένας πελάτης ταχυδρομικής θυρίδας είναι μια διεργασία που γράφει ένα μήνυμα σε μια ταχυδρομική θυρίδα. Οποιαδήποτε διεργασία έχει το όνομα μιας ταχυδρομικής θυρίδας μπορεί να τοποθετήσει εκεί ένα μήνυμα. Τα νέα μηνύματα ακολουθούν οποιαδήποτε υπάρχοντα μηνύματα στην ταχυδρομική θυρίδα.

Οι ταχυδρομικές θυρίδες μπορούν να εκπέμπουν μηνύματα μέσα σε έναν τομέα. Αν αρκετές διεργασίες σε έναν τομέα δημιουργήσουν (η καθεμιά) από μία ταχυδρομική θυρίδα χρησιμοποιώντας το ίδιο όνομα, κάθε μήνυμα που θα διευθύνεται σε αυτή την ταχυδρομική θυρίδα και αποστέλλεται στον τομέα, παραλαμβάνεται από τις συμμετέχουσες διεργασίες. Επειδή μια διεργασία μπορεί να ελέγχει τόσο ένα δείκτη ταχυδρομικής θυρίδας, όσο και τον ανακτημένο δείκτη του πελάτη όταν η ταχυδρομική θυρίδα είναι ανοιχτή για μια λειτουργία εγγραφής, οι διεργασίες μπορούν εύκολα να εκτελέσουν μία απλή χέρι-χέρι μεταφορά μηνύματος μέσα σε έναν τομέα.

Για την αποστολή μηνυμάτων μεγαλύτερων από 424 bytes μεταξύ υπολογιστών, χρησιμοποιούνται τα named pipes ή Windows Sockets εναλλακτικά.

Οι εξυπηρετητές ταχυδρομικής θυρίδας έχουν αποκλειστική χρήση τριών συναρτήσεων, όπως φαίνεται:

CreateMailslot: Δημιουργεί μία ταχυδρομική θυρίδα και επιστρέφει ένα χειριστή θυρίδας.

GetMailslotInfo: Ανακτά το μέγιστο μέγεθος μηνύματος, το μέγεθος ταχυδρομικής θυρίδας, το μέγεθος του επόμενου μηνύματος στην ταχυδρομική θυρίδα, τον αριθμό των μηνυμάτων στην ταχυδρομική θυρίδα, και το ποσό χρόνου που μπορεί να περιμένει μια εργασία για ένα μήνυμα.

SetMailslotInfo: Αλλάζει το χρονικό όριο ανάγνωσης μιας ταχυδρομικής θυρίδας.

Το παρακάτω παράδειγμα δείχνει ένα απλό τρόπο δημιουργίας ενός mailslot.

```
HANDLE hSlot;
LPTSTR Slot = TEXT("\\\\.\\mailslot\\sample_mailslot");

BOOL WINAPI MakeSlot(LPTSTR lpszSlotName)
{
   hSlot = CreateMailslot(lpszSlotName,
                0,   // no maximum message size
        MAILSLOT_WAIT_FOREVER,   // no time-out for operations
        (LPSECURITY_ATTRIBUTES) NULL); // default security

  if (hSlot == INVALID_HANDLE_VALUE) {
      printf("CreateMailslot failed with %d\n",
                              GetLastError());
      return FALSE;
  }
  else printf("Mailslot created successfully.\n");
  return TRUE;
}
```

```
void main() {
    MakeSlot(Slot);
}
```

Το να γράφουμε σε μια ταχυδρομική θυρίδα είναι παρόμοιο με το να γράφουμε σε ένα συγκεκριμένο αρχείο δίσκου. Ο παρακάτω κώδικας χρησιμοποιεί τις συναρτήσεις **CreateFile**, **WriteFile**, και **CloseHandle** για να τοποθετήσουν ένα μικρό μήνυμα στην ταχυδρομική θυρίδας. Το μήνυμα εκπέμπεται σε κάθε υπολογιστή στον πρωτεύοντα τομέα του συστήματος.

```
LPTSTR SlotName = TEXT("\\\\.\\mailslot\\sample_mailslot");

BOOL WriteSlot(HANDLE hSlot, LPTSTR lpszMessage)
{
  BOOL fResult;
  DWORD cbWritten;
  fResult = WriteFile(hSlot,
    lpszMessage,
    (DWORD) (lstrlen(lpszMessage)+1)*sizeof(TCHAR),
    &cbWritten,
    (LPOVERLAPPED) NULL);
  if (!fResult)
  {
    printf("WriteFile failed with %d.\n", GetLastError());
    return FALSE;
  }
  printf("Slot written to successfully.\n");
  return TRUE;
}

int main()
{
  HANDLE hFile;
  hFile = CreateFile(SlotName,
    GENERIC_WRITE,
    FILE_SHARE_READ,
    (LPSECURITY_ATTRIBUTES) NULL,
    OPEN_EXISTING,
    FILE_ATTRIBUTE_NORMAL,
    (HANDLE) NULL);
  if (hFile == INVALID_HANDLE_VALUE)
  {
    printf("CreateFile failed with %d.\n", GetLastError());
    return FALSE;
  }
  WriteSlot(hFile, TEXT("Message one for mailslot."));
  WriteSlot(hFile, TEXT("Message two for mailslot."));
  Sleep(5000);
  WriteSlot(hFile, TEXT("Message three for mailslot."));
```

```
  CloseHandle(hFile);
  return TRUE;
}
```

Η διεργασία που δημιουργεί μία ταχυδρομική θυρίδα μπορεί να διαβάζει μηνύματα από αυτήν χρησιμοποιώντας ο δείκτη ταχυδρομικής θυρίδας σε μια κλήση της συνάρτησης **ReadFile**. Το παρακάτω παράδειγμα καλεί τη συνάρτηση **GetMailslotInfo** για να αποφασίσουν αν υπάρχουν μηνύματα στην ταχυδρομική θυρίδα ή όχι.

Μια ταχυδρομική θυρίδα υπάρχει μέχρι την κλήση της συνάρτησης **CloseHandle** για όλες τις λαβές του εξυπηρετητή, ή μέχρι όλες οι διεργασίες εξυπηρετητή που κατέχουν μία ταχυδρομική θυρίδα χειριστούν την έξοδο. Σε κάθε περίπτωση, όλα τα αδιάβαστα μηνύματα διαγράφονται από την ταχυδρομική θυρίδα, όλοι οι δείκτες πελάτη για την ταχυδρομική θυρίδα κλείνουν, και η ίδια η ταχυδρομική θυρίδα διαγράφεται από τη μνήμη.

```
HANDLE hSlot;
LPTSTR lpszBuffer;
DWORD cbMessage, cbRead;
OVERLAPPED ov;
fResult = ReadFile(hSlot, lpszBuffer, cbMessage, &cbRead, &ov);
```

Συνοψίζοντας, τα mailslots μοιάζουν πολύ τα sockets σε πιο ειδικό επίπεδο. Συνδυάζουν την ασφάλεια, ευχρηστία, αποδοτικότητα και παράλληλα εξασφαλίζουν επικοινωνία διεργασιών σε επίπεδο δικτύου.

Διοχετεύσεις (Pipes)

Μια διοχέτευση (pipe) είναι ένα τμήμα διαμοιραζόμενης μνήμης που χρησιμοποιούν οι διεργασίες για επικοινωνία. Η διεργασία που δημιουργεί μια διοχέτευση είναι ο εξυπηρετητής διοχετεύσεων *(pipe server)*. Μια διεργασία που συνδέεται σε μία διοχέτευση είναι ένας πελάτης διοχέτευσης *(pipe client)*. Μια διεργασία γράφει πληροφορίες στη διοχέτευση, και μετά η άλλη διεργασία διαβάζει τις πληροφορίες από τη διοχέτευση.

Υπάρχουν δύο τύποι διοχετεύσεων: ανώνυμες διοχετεύσεις και επώνυμες διοχετεύσεις. Ανώνυμες διοχετεύσεις απαιτούν μικρότερες επικεφαλίδες από τις ονομαστικές διοχετεύσεις, αλλά προσφέρουν περιορισμένες υπηρεσίες.

Ο όρος διοχέτευση, όπως χρησιμοποιείται εδώ, συνεπάγεται πως μια διοχέτευση χρησιμοποιείται ως ένα κανάλι μεταφοράς πληροφοριών. Θεμελιωδώς, μία διοχέτευση έχει δύο άκρα. Μία μονόδρομη διοχέτευση επιτρέπει τη διεργασία στο ένα άκρο να γράφει στη διοχέτευση, και επιτρέπει τη διεργασία στο άλλο άκρο να διαβάζει από τη διοχέτευση. Μία αμφίδρομη (ή διπλή) διοχέτευση επιτρέπει σε μια διεργασία να διαβάζει και να γράφει από το άκρο της στη διοχέτευση.

Μία ανώνυμη διοχέτευση είναι μία χωρίς όνομα, μονόδρομη διοχέτευση που συνήθως μεταφέρει δεδομένα μεταξύ μιας γονικής διεργασίας και μιας θυγατρικής διεργασίας. Οι ανώνυμες διοχετεύσεις είναι πάντοτε τοπικές. Δεν μπορούν να χρησιμοποιηθούν για επικοινωνία σε ένα δίκτυο.

Μία ονομαστική διοχέτευση είναι μία ονομασμένη, μονόδρομη ή διπλή διοχέτευση για επικοινωνία μεταξύ του εξυπηρετητή διοχέτευσης και ενός ή περισσοτέρων πελατών διοχέτευσης. Όλα τα τεκμήρια μιας ονομαστικής διοχέτευσης μοιράζονται το ίδιο όνομα διοχέτευσης, αλλά κάθε τεκμήριο έχει τις δικές του ουρές και χειριστές, και παρέχει ένα

διακριτό κανάλι μεταφοράς για επικοινωνία πελάτη/εξυπηρετητή. Η χρήση τεκμηρίων επιτρέπει σε πολλαπλούς πελάτες διοχετεύσεων να χρησιμοποιήσουν την ίδια ονομαστική διοχέτευση ταυτοχρόνως.

Οποιαδήποτε διεργασία μπορεί να προσπελάσει ονομαστικές διοχετεύσεις, υποκείμενη σε ελέγχους ασφάλειας, καθιστώντας τις ονομαστικές διοχετεύσεις ένα εύκολο καλούπι επικοινωνίας μεταξύ συγγενικές ή μη διεργασίες.

Κάθε διεργασία μπορεί να δράσει ως εξυπηρετητής και ως πελάτης, καθιστώντας την πόρτα-σε-πόρτα επικοινωνία δυνατή. Όπως χρησιμοποιείται εδώ, ο όρος εξυπηρετητής διοχέτευσης αναφέρεται σε μια διεργασία που δημιουργεί μία ονομαστική διοχέτευση, και ο όρος πελάτης παροχέτευσης αναφέρεται σε μια διεργασία που συνδέεται σε ένα τεκμήριο μιας ονομαστικής διοχέτευσης.

Οι ονομαστικές διοχετεύσεις μπορούν να χρησιμοποιηθούν για να παρέχουν επικοινωνία μεταξύ διεργασιών στον ίδιο υπολογιστή ή μεταξύ διεργασιών σε διαφορετικούς υπολογιστές κατά μήκος ενός δικτύου. Αν η εξυπηρέτηση του εξυπηρετητή τρέχει, όλες οι ονομαστικές διοχετεύσεις είναι προσβάσιμες απομακρυσμένα.

Κάθε ονομαστική διοχέτευση έχει ένα μοναδικό όνομα που την ξεχωρίζει από άλλες ονομαστικές διοχετεύσεις στη λίστα συστήματος ονομαστικών διοχετεύσεων. Ένας εξυπηρετητής διοχέτευσης καθορίζει ένα όνομα για τη διοχέτευση μόλις καλεί τη συνάρτηση **CreateNamedPipe** για να δημιουργήσει ένα ή περισσότερα τεκμήρια μιας ονομαστικής διοχέτευσης. Οι εξυπηρετητές διοχετεύσεων ορίζουν το όνομα της διοχέτευσης μόλις καλέσουν τη συνάρτηση **CreateFile** ή **CallNamedPipe** για να συνδεθούν σε ένα τεκμήριο της ονομαστικής διοχέτευσης.

Διαδικασία δημιουργίας pipe από τον server:

```
hPipe = CreateNamedPipe(
        lpszPipename,           // pipe name
        PIPE_ACCESS_DUPLEX,     // read/write access
        PIPE_TYPE_MESSAGE |     // message type pipe
        PIPE_READMODE_MESSAGE |  // message-read mode
        PIPE_WAIT,              // blocking mode
        PIPE_UNLIMITED_INSTANCES, // max. instances
        BUFSIZE,                // output buffer size
        BUFSIZE,                // input buffer size
        0,                      // client time-out
        NULL);
```

Άνοιγμα και χρησιμοποίηση Pipe από τον client:

```
hPipe = CreateFile(
        lpszPipename,   // pipe name
        GENERIC_READ |  // read and write access
        GENERIC_WRITE,
        0,              // no sharing
        NULL,           // default security attributes
        OPEN_EXISTING,  // opens existing pipe
        0,              // default attributes
        NULL);          // no template file
```

Οι συναρτήσεις **ReadFile**, **WriteFile**, **TransactNamedPipe**, και **ConnectNamedPipe** μπορούν να επιτελέσουν λειτουργίες εισόδου και εξόδου σε μια διοχέτευση είτε σύγχρονα είτε ασύγχρονα. Όταν μια συνάρτηση τρέχει σύγχρονα, δεν επιστρέφει έως ότου η λειτουργία που επιτελεί τελειώσει. Αυτό σημαίνει πως η εκτέλεση του νήματος κλήσης μπορεί να εμποδιστεί για μια απροσδιόριστη περίοδο όσο περιμένει για μια χρονοβόρα λειτουργία να ολοκληρωθεί. Όταν μια συνάρτηση τρέχει ασύγχρονα, επιστρέφει άμεσα, ακόμη κι εάν η λειτουργία δεν έχει ολοκληρωθεί. Αυτό επιτρέπει σε μια χρονοβόρα λειτουργία να εκτελεστεί στο παρασκήνιο ενώ το νήμα κλήσης είναι ελεύθερο να επιτελέσει άλλες λειτουργίες.

```
do {
        // Read from the pipe.
        fSuccess = ReadFile(
                hPipe,    // pipe handle
                chBuf,    // buffer to receive reply
                BUFSIZE*sizeof(TCHAR),  // size of buffer
                &cbRead,  // number of bytes read
                NULL);    // not overlapped
} while (!fSuccess);  // repeat loop if ERROR_MORE_DATA
```

Παράδειγμα ανάγνωσης με μέθοδο polling (σύγχρονα).

```
fSuccess = WriteFile(
          hPipe,        // handle to pipe
      chReply,      // buffer to write from
      cbReplyBytes, // number of bytes to write
      &cbWritten,   // number of bytes written
      NULL);        // not overlapped I/O
```

Μακρινές κλήσεις διαδικασιών (Remote Procedure Call - RPC)

Η απομακρυσμένη διαδικασία κλήσης (Remote procedure call, RPC) είναι μια τεχνολογία που επιτρέπει ένα πρόγραμμα υπολογιστή να προκαλέσει μια υπορουτίνα ή διαδικασία για να εκτελέσει σε έναν άλλο χώρο διευθύνσεων (συνήθως σε έναν άλλο υπολογιστή σε ένα διαμοιραζόμενο δίκτυο) χωρίς τη σαφήνεια του προγραμματιστή να κωδικεύει τις λεπτομέρειες για αυτή την απομακρυσμένη αλληλεπίδραση. Αυτό επειδή ο προγραμματιστής θα έγραφε ουσιωδώς τον ίδιο κώδικα είτε η υπορουτίνα είναι τοπική στο εκτελούμενο πρόγραμμα, είτε απομακρυσμένη. Όταν το λογισμικό στην αίτηση είναι γραμμένο χρησιμοποιώντας αντικειμενοστραφείς αρχές, το RPC μπορεί να αναφερθεί ως απομακρυσμένη επίκληση ή μέθοδος απομακρυσμένης επίκλησης.

Το RPC περιλαμβάνει τα ακόλουθα κύρια συστατικά:

- Μεταγλωττιστής MIDL
- Βιβλιοθήκες και αρχεία header πραγματικού χρόνου
- Παροχέας εξυπηρέτησης ονομάτων (μερικές φορές αναφερόμενος ως ο Locator)
- Endpoint mapper (μερικές φορές αναφερόμενος ως ο port mapper)

Στο μοντέλο RPC, μπορούμε τυπικά να καθορίσουμε μία διασύνδεση με τις απομακρυσμένες διαδικασίες χρησιμοποιώντας μια γλώσσα σχεδιασμένη για αυτό το σκοπό. Αυτή η γλώσσα καλείται Interface Definition Language, ή IDL. Η υλοποίηση της Microsoft για αυτή τη γλώσσα καλείται Microsoft Interface Definition Language, ή MIDL.

Αφού δημιουργήσουμε μία διασύνδεση, πρέπει να το περάσουμε μέσω του μεταγλωττιστή MIDL. Αυτός ο μεταγλωττιστής δημιουργεί τα στελέχη που ερμηνεύουν τις τοπικές κλήσεις διαδικασιών σε απομακρυσμένες κλήσεις διαδικασιών. Τα στελέχη είναι συναρτήσεις που κρατούν τις θέσεις και πραγματοποιούν τις κλήσεις στις συναρτήσεις βιβλιοθήκης πραγματικού χρόνου, οι οποίες διαχειρίζονται την απομακρυσμένη κλήση διαδικασιών. Το πλεονέκτημα αυτής της προσέγγισης είναι ότι το δίκτυο καθίσταται σχεδόν ολοκληρωτικά διαφανές στην κατανεμημένη εφαρμογή σας. Το πρόγραμμα πελάτης καλεί αυτές που φαίνεται να είναι τοπικές διαδικασίες. Η δουλειά για τη μετατροπή τους σε απομακρυσμένες κλήσεις γίνεται για εμάς αυτόματα. Όλος ο κώδικας που ερμηνεύει δεδομένα, προσπελαύνει το δίκτυο, και ανακτά αποτελέσματα δημιουργείται για σας από τον μεταγλωττιστή MIDL και είναι αόρατος για την εφαρμογή μας.

Υποδοχές (Sockets)

Όπως είπαμε πιο πάνω, τα Windows Sockets είναι βασισμένες στην θεωρία των sockets που διαδόθηκαν πρώτα από τη διανομή λογισμικού του Berkeley (Berkeley Software Distribution - BSD). Οι διοχετεύσεις είναι μια τεχνολογία ευρέως διαδεδομένη που χρησιμοποιείται σήμερα πάνω στο layer TCP/IP υλοποιώντας διαδιεργασιακή επικοινωνία μέσω των πρωτοκόλλων TPC ή UDP. Είναι μια πλήρης παραμετροποιήσιμη και ευέλικτη μέθοδος επικοινωνίας πάνω στην οποία έχουν στηριχτεί πολλές διαδικτυακές εφαρμογές τύπου server/client αλλά ακόμα και άλλες μέθοδοι διαδιεργασιακής επικοινωνίας.

Τα Sockets χρησιμοποιούν μεθόδους αποστολής, παραλαβής όπως με την μέθοδο των pipes αλλά σε πιο γενικό επίπεδο.

Ένα Internet socket αποτελείται από τα ακόλουθα:

- Protocol
- Local IP address
- Local port
- Remote IP address
- Remote port

Στην περίπτωση που ένα socket πρόκειται να χρησιμοποιηθεί σαν διακομιστής συνδέσεων (listener/client) δεν είναι απαραίτητο να εισάγουμε Remote IP ή Address μιας και το μόνο που θα κάνει είναι να λειτουργήσει σαν περιοχή σύνδεσης άλλων sockets.

Τα Λειτουργικά Συστήματα συνδυάζουν τα sockets με μια ενεργή διεργασία ή διεργασίες (οι οποίες χρησιμοποιούν το socket για να στείλουν και να παραλάβουν δεδομένα στο δίκτυο), και ένα πρωτόκολλο μεταφοράς (π.χ. TCP ή UDP) με το οποίο η/οι διεργασία/ες επικοινωνούν με τον απομακρυσμένο host.

Συνήθως τα sockets εκτελούνται πάνω στο TCP αλλά αυτό δεν είναι απαραίτητο. Μπορούν να εκτελεστούν πάνω από οποιοδήποτε πρωτόκολλο μεταφοράς, όπως το SNA. Η αρχή ενός socket είναι μια οντότητα που υλοποιεί ένα API, αδιαφορώντας για την υλοποίηση.

Δύο ευρέως χρησιμοποιούμενοι τύποι Internet socket είναι:

1. Datagram sockets, ο οποίος χρησιμοποιεί UDP
2. Stream Sockets, ο οποίος χρησιμοποιεί TCP

Παρακάτω θα εξετάσουμε μια βασική υλοποίηση των Sockets στην πλατφόρμα .NET 2.0. Το namespace στο οποίο βρίσκονται οι μέθοδοι που θα χρησιμοποιήσουμε είναι το System.Net.Sockets και συγκεκριμένα το class Socket.

Το **Socket** class παρέχει μια πλούσια συλλογή μεθόδων και ιδιοτήτων για επικοινωνίες δικτύου. Το **Socket** class μας επιτρέπει να επιτελούμε τόσο σύγχρονη όσο και ασύγχρονη

μεταφορά δεδομένων χρησιμοποιώντας οποιοδήποτε από τα πρωτόκολλα επικοινωνιών που καταγράφονται στην απαρίθμηση ProtocolType. Οι πιο συχνά χρησιμοποιούμενες μέθοδοι είναι οι TCP και UDP. Υπάρχουν όμως και πολλά άλλα όπως Icmp, Ggp, Ipx κ.ά. Το **Socket** class ακολουθεί το .NET Framework ονομαστικό πρότυπο για ασύγχρονες μεθόδους. Για παράδειγμα, η σύγχρονη μέθοδος Receive (παραλαβής) ανταποκρίνεται στις ασύγχρονους μεθόδους BeginReceive και EndReceive.

Αν μια εφαρμογή απαιτεί μόνο ένα νήμα κατά τη διάρκεια της εκτέλεσης, χρησιμοποιούνται οι ακόλουθες μέθοδοι, οι οποίες είναι σχεδιασμένες για κατάσταση σύγχρονης λειτουργίας.

- Αν χρησιμοποιείται ένα πρωτόκολλο προσανατολισμένο στη σύνδεση όπως το TCP, ο διακομιστής μπορεί να ακούει για συνδέσεις χρησιμοποιώντας την μέθοδο Listen. Η μέθοδος Accept επεξεργάζεται οποιαδήποτε ζήτηση εισερχόμενης σύνδεσης και επιστρέφει ένα **Socket** που μπορεί να χρησιμοποιηθεί για να επικοινωνήσετε με δεδομένα με τον απομακρυσμένο host. Χρησιμοποιούμε αυτό το επιστραφέν **Socket** για να καλέσουμε τη μέθοδο Send ή Receive. Καλούμε τη μέθοδο Bind πριν την κλήση της μεθόδου **Listen** αν θέλουμε να καθορίσουμε την τοπική διεύθυνση IP και ο αριθμός port. Χρησιμοποιούμε έναν μηδενικό αριθμό port αν θέλουμε τον θεμελιώδη παροχέα εξυπηρέτησης να αναθέσει ένα ελεύθερο port. Αν θέλουμε να συνδεθούμε σε έναν listener, μπορούμε να καλέσουμε τη μέθοδο Connect. Για να στείλουμε ή να λάβουμε δεδομένα καλούμε τη μέθοδο **Send** ή **Receive**.
- Αν χρησιμοποιήσουμε ένα πρωτόκολλο χωρίς σύνδεση όπως το UDP, δε χρειάζεται να παρακολουθούμε καθόλου για συνδέσεις. Καλώντας τη μέθοδο ReceiveFrom η διεργασία λαμβάνει οποιοδήποτε εισερχόμενο datagram. Χρησιμοποιούμε τη μέθοδο SendTo για να στείλουμε datagram σε έναν απομακρυσμένο host.

Για να επεξεργαστούμε επικοινωνίες χρησιμοποιώντας ξεχωριστά νήματα κατά τη διάρκεια της εκτέλεσης, χρησιμοποιούμε τις ακόλουθες μεθόδους, οι οποίες είναι σχεδιασμένες για κατάσταση ασύγχρονης λειτουργίας.

- Αν χρησιμοποιούμε ένα πρωτόκολλο προσανατολισμένο στη σύνδεση όπως το TCP, χρησιμοποιούμε τις μεθόδους BeginConnect και EndConnect για να συνδεθούμε με ένα host ακρόασης. Χρησιμοποιούμε τις μεθόδους BeginSend και EndSend ή BeginReceive και EndReceive για να επικοινωνήσουμε με δεδομένα ασύγχρονα. Αιτήματα για εισερχόμενες συνδέσεις μπορούν να επεξεργαστούν χρησιμοποιώντας τις BeginAccept και EndAccept.
- Αν χρησιμοποιείται ένα πρωτόκολλο χωρίς σύνδεση όπως το UDP, μπορούμε να χρησιμοποιήσουμε τις BeginSendTo και EndSendTo για να στείλουμε datagram, και τις BeginReceiveFrom και EndReceiveFrom για να λάβετε datagram.

Αν επιτελούνται πολλαπλές ασύγχρονες λειτουργίες σε ένα socket, δεν ολοκληρώνονται κατ' ανάγκην στη σειρά με την οποία ξεκίνησαν.

Όταν τελειώσουμε την αποστολή και παραλαβή δεδομένων, χρησιμοποιούμε τη μέθοδο Shutdown για να απενεργοποιήσουμε το **Socket**. Αφού καλέσουμε την **Shutdown**, καλούμε τη μέθοδο Close για να απελευθερωθούν όλοι οι πόροι που συνδέονται με το **Socket**.

Παράδειγμα ενός ασύγχρονου TCP listener

```
public void Start() {
        ServerSocket = new Socket(AddressFamily.InterNetwork,
                        SocketType.Stream,
           ProtocolType.Tcp); //Αρχικοποίηση του socket
    ServerSocket.Bind(new IPEndPoint(IPAddress.Any, Port));
    ServerSocket.Listen(5); //Λειτουργία ως διακομιστής στο
```

```
                        //Port
    ServerSocket.BeginAccept(new syncCallback(OnClientConnect),
                null); //Ασύγχρονη αποδοχή - ξεχωριστό νήμα
    //Κλήση μεθόδου OnClientConnect όταν επιτευχθει σύνδεση
}

private void OnClientConnect(IAsyncResult asyn) {
    Socket TempSocket = ServerSocket.EndAccept(asyn);
   //Τέλος ασύγχρονης αποδοχής και αποθήκευση καινούριου socket
   ClientSockets.Add(TempSocket); //Προσθήκη του socket στη
                            //λίστα συνδεδεμένων client
   WaitForData(TempSocket); //Καταγραφή δεδομένων από το
                        //socket/πελάτη
   ServerSocket.BeginAccept(new AsyncCallback(OnClientConnect),
            null); // Ξεκίνημα νέου thread για σποδοχή πελάτη
}

private void WaitForData(Socket ClientSocket) {
    Packet TempPacket = new Packet(ClientSocket,
                            new byte[10240]);
    pfnCallBack = new AsyncCallback(OnDataReceived);
    //Αρχή ασύγχρονης αποδοχής δεδομένων από το συγκεκριμένο
    //socket-πελάτη
    ClientSocket.BeginReceive(TempPacket.Buffer, 0,
            TempPacket.Buffer.Length, SocketFlags.None,
            pfnCallBack, TempPacket);
}

private void OnDataReceived(IAsyncResult asyn) {
    int DataLength =
      (Packet)asyn.AsyncState).StateSocket.EndReceive(asyn);
    //Επεξεργασία δεδομένων
    WaitForData(((Packet)asyn.AsyncState).StateSocket);
    //Συνέχιση ασύγχρονης αποδοχής από τον ίδιο πελάτη
}
```

Παράδειγμα ενός ασύγχρονου client που συνδέεται στον παραπάνω server:

```
public void Connect() {
        //Αρχικοποίηση socket
        ClientSocket = new Socket(AddressFamily.InterNetwork,
              SocketType.Stream, ProtocolType.Tcp);
        //Ασύγχρονη σύνδεση στον server
        ClientSocket.BeginConnect(new IPEndPoint(remoteIP,
                Port), new AsyncCallback(OnClientConnect),
                null);
}
```

```
private void OnClientConnect(IAsyncResult asyn) {
        //Τέλος σύνδεσης και
        ClientSocket.EndConnect(asyn);
        //Ξεκίνημα thread για παραλαβή δεδομένων
        WaitForData();
}

private void WaitForData() {
        Packet TempPacket = new Packet(ClientSocket,
                        new byte[10240]);
        pfnCallBack = new AsyncCallback(OnDataReceived);
}

private void OnDataReceived(IAsyncResult asyn) {
        int DataLength = 0;
        DataLength =
      ((Packet)asyn.AsyncState).StateSocket.EndReceive(asyn);
        //Επεξεργασία δεδομένων εδώ
        WaitForData();
}
```

Η διαδικασία της αποστολής δεδομένων μπορεί να υλοποιηθεί είτε στον server είτε στον πελάτη είτε και στους δύο ανάλογα με τη μέθοδο επικοινωνίας.

```
public void Send(byte[] buffer) {
      Packet p = new Packet(ClientSocket, buffer);
      ClientSocket.BeginSend(p.Buffer, 0, p.Buffer.Length, SocketFlags.None,
                        new AsyncCallback(OnDataSent), p);
}

private void OnDataSent(IAsyncResult asyn) {
      ClientSocket.EndSend(asyn);
}
```