au réseau Ethernet. Il est alors possible d'utiliser la pile TCP embarquée dans le SocketModem pour pouvoir se connecter à un serveur distant. Nous précisons alors l'adresse IP de notre serveur et demandons de nous connecter au port 80. Pour les premiers essais, le mieux est encore de recompiler l'application *tip* pour notre cible ARM7 qui nous permettra de dialoguer avec le modem au travers d'un terminal lancé depuis la cible.

Listing 5. Traitement des commandes AT destinées au modem

```
fd = fopen(filename, "rw");
do {
   ret = fgets(lineFromIni, sizeof(lineFromIni), fd);
   if (ret == NULL)
      break;
   if (lineFromIni[0] == '>') {
      len = strlen(lineFromIni);
      lineFromIni[len-1] = 0x0d;
      lineFromIni[len] = 0x0a;
      lineFromIni[len+1] = 0x00;
      if (write(targetfd, &lineFromIni[1], len-1) < 0) {
         printf("Unable to send request %s\n", &lineFromIni[1]);
         fclose(fd);
         return (-1);
      }
      rr = read(targetfd, lineFromTarget, len-1);
      if (strncmp(&lineFromTarget[0], &lineFromIni[1], len-1)) {
         printf("target don't send expected response %s \n", &lineFromIni[1]);
         fclose(fd);
         return (-1);
      }
      continue;
   }
   if (lineFromIni[0] == '<') {
      retry = 10;
      while(retry) {
         rr = read(targetfd, lineFromTarget, 50);
         if (rr) {
            lineFromTarget[rr] = 0;
            printf("<=%s", &lineFromTarget[2]);
         }
         if (!strncmp(&lineFromIni[1], "xxx.xxx.xxx.xxx", 15)) {
            GetDynamicIPAddr(&lineFromTarget[2]);
            break;
         }
         if (strncmp(&lineFromTarget[2], &lineFromIni[1], 2)) {
            retry--;
            sleep(1);
         }
         else {
            tcflush(targetfd, TCIFLUSH);
            break;
         }
      }
      if (!retry) {
         printf("target does not send expected response %s (%s)\n",
               &lineFromIni[1], &lineFromTarget[2]);
         fclose(fd);
         return (-1);
      }
   }
} while(1);
```

Organisation de l'application

Nous avons donc établi 2 périphériques distinctes, le module GPS et le modem GPRS. Plutôt que de réaliser une application constituée de 2 threads, nous allons créer 2 processus qui dialogueront ensemble au travers d'un pipe nommé, c'est-à-dire un fichier un peu particulier utilisé pour envoyer une série de données à un autre processus.

Comme nous sommes partis pour écrire 2 programmes, ne nous arrêtons pas là et écrivons en un de plus. Ce dernier sera en charge de créer les pipes nommés sous */home*, un répertoire qui se trouve sur la première partition de notre flash NAND (16Mo). Ce programme contient essentiellement la ligne suivante :

```
mkfifo("/home/GPS_to_GPRS.pipe",
       S_IRUSR | S_IXUSR | S_IWUSR);
```

Ensuite, nous allons créer 2 processus, le premier étant destiné à faire les acquisitions des données GPS.

Programme d'acquisition des coordonnées GPS :

Dans le programme ci-dessous, vous remarquerez que nous commençons par ouvrir le port com série connecté au module GPS. Dans notre cas nous utilisons un module connecté au bus USB tout en émulant un port série, notre interface sera donc */dev/ttyUSB0*.

Ensuite, nous utilisons la fonction *select* pour déléguer au système d'exploitation la gestion des trames séries reçues et réveiller notre processus en attente lorsqu'une trame est reçue (line feed détecté en fin de chaîne). Nous pouvons associer un délai à cette fonction, ici de 4 secondes, largement suffisant, le module GPS transmettant une ou des trames série toutes les secondes.

Ensuite, il est nécessaire de vérifier le checksum de la trame afin de garantir son con-

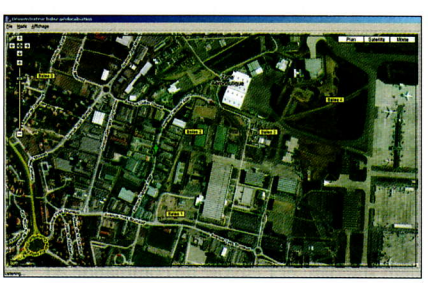

Figure 5. Exemple de réalisation sur un serveur distant