- Bei *socket* lässt sich *SOCK_DGRAM* als zweiter Parameter angeben.
- Der Netzwerkdienst kann dann wie gewohnt *setsockopt* und *bind* aufrufen. Der Systemaufruf *listen* fällt weg, da dieser nur bei verbindungsorientierten Sockets Anwendung findet.
- Eingehende Pakete können dann mit *recvfrom* empfangen werden, das (in Ergänzung zu *read*) auch die Absenderadresse mitliefert. Mit *sendto* ist eine Antwort an eine gegebene Adresse möglich.
- Der Klient verwendet wie gewohnt *connect* und kann dann *read* und *write* verwenden, wobei hier (falls die Buffergröße groß genug ist) vollständige Pakete gelesen und verschickt werden.

timeserver.c

```
struct sockaddr_in address = {0};
address.sin_family = AF_INET;
address.sin_addr.s_addr = htonl(INADDR_ANY);
address.sin_port = htons(PORT);
int sfd = socket(PF_INET, SOCK_DGRAM, 0);
int optval = 1;
if (sfd < 0 ||
     setsockopt(sfd, SOL_SOCKET, SO_REUSEADDR,
                &optval, sizeof optval) < 0 ||
     bind(sfd, (struct sockaddr *) &address,
          sizeof address) < 0) {
   perror("socket"); exit(1);
}
ssize_t nbytes; char buf[BUFSIZ];
struct sockaddr sender; socklen_t sender_len = sizeof(sender);
while ((nbytes = recvfrom(sfd, buf, sizeof buf, 0,
         &sender, &sender_len)) >= 0) {
   char timebuf[32]; time_t clock; time(&clock);
   ctime_r(&clock, timebuf, sizeof timebuf);
   sendto(sfd, timebuf, strlen(timebuf), 0,
      &sender, sender_len);
}
```

timeclient.c

```
int fd;
if ((fd = socket(PF_INET, SOCK_DGRAM, 0)) < 0) {
   perror("socket"); exit(1);
}
if (connect(fd, (struct sockaddr *) &addr, sizeof addr) < 0) {
   perror("connect"); exit(1);
}
char buffer[BUFSIZ]; ssize_t nbytes;
/* send an empty packet */
if (write(fd, buffer, 0) < 0) {
   perror("write"); exit(1);
}
/* receive response */
if ((nbytes = read(fd, buffer, sizeof buffer)) > 0) {
   write(1, buffer, nbytes);
} else {
   perror("read"); exit(1);
}
```

- Bei TCP bzw. *SOCK_STREAM* erfolgte implizit bereits ein Austausch von Paketen durch die Systemaufrufe *listen* und *connect*.
- Bei UDP bzw. *SOCK_DGRAM* fällt dies weg. Der Systemaufruf *connect* hat hier nur die Funktion, dass eine Socket fest mit einer Adresse verbunden wird, d.h. es kann anschließend wie gewohnt *read* und *write* verwendet werden ohne eine weitere Spezifikation der Adresse.
- Wenn *connect* wegfällt, muss die Adresse beim Paketversand immer angegeben werden.
- Da *connect* bei *SOCK_DGRAM* noch nicht zum Austausch von Paketen führt, kann zu diesem Zeitpunkt noch nicht festgestellt werden, ob die Gegenseite den Dienst überhaupt anbietet. Das stellt sich erst (mit etwas Glück) bei einem anschließenden *write* heraus.
- Anders als bei TCP bzw. *SOCK_STREAM* kann das Versenden leerer Pakete sinnvoll sein.

timeclient2.c

```
/* receive response */
struct pollfd pollfds[1] = { {.fd = fd, .events = POLLIN} };
unsigned int attempts = 0;
while (attempts < 10 &&
       poll(pollfds, 1, 100 /* milliseconds */) == 0) {
   ++attempts;
   /* resend package */
   if (send(fd, buffer, 0, 0) < 0) {
      perror("send"); exit(1);
   }
}
if (attempts < 10) {
   if ((nbytes = recv(fd, buffer, sizeof buffer, 0)) > 0) {
      write(1, buffer, nbytes);
   } else {
      perror("recv");
   }
}
```

- UDP-Pakete können verloren gehen. Entsprechend sollte damit gerechnet werden, dass keine Antwort auf eine Anfrage eingeht. Entsprechend ist es sinnvoll, mehrere Versuche einzuplanen.