Die Bergung der Kursk

Ingenieurgemeinschaft IgH

Essen

Januar 2002

Dieser Text[1] ist in der Zeitschrift „Linux-Magazin“, Ausgabe 01/02, erschienen.
Die Wiedergabe erfolgt mit freundlicher Genehmigung des Verlages.

Ingenieurgemeinschaft IgH
Heinz-Bäcker-Str. 34
D-45356 Essen
Tel.: +49-201-619931
Fax.: +49-201-619836
E-mail: msk@igh-essen.com

[1] `kurskbergung`, Revision: 1.2 , Stand: Date: 2002/03/28 17:15:40

Die Bergung der Kursk
Tux hilft bei der Bergung des U-Bootes

Zusammenfassung

Die Bergung der Kursk war eine gewaltige technische Herausforderung. Dieses einmalige Projekt beeindruckte bereits durch die schieren Ausmaße und den enormen Zeitdruck. Es gab kaum Erfahrungen über Bergeaktionen diesen Umfangs. Die Mission wurde dennoch ein Erfolg, weil alle Beteiligten in gewaltiger Anstrengung mithalfen. Alle verfügbaren Mittel kamen zum Einsatz, unter anderem auch das Betriebssystem Linux. Im Leitstand des Bergeschiffes waren fünf hochwertige aber typische Industrie-PC vorhanden, die allesamt unter Linux gefahren werden konnten. Die jahrelangen guten Erfahrungen mit Linux hatten den Ausschlag gegeben, es zum Fundament entscheidend wichtiger Arbeiten zu machen. Nach dem Erfolg der Mission läßt sich eindeutig sagen: Linux hat sich einmal mehr bestens bewährt.

1 Wie die „Kursk“ unterging...

Am 12. August 2000 ging in der Barentsee nahe Murmansk das russische Atom-U-Boot „Kursk“ nach mehreren Explosionen unter. Von den 118 Mann Besatzung überlebte niemand dieses Unglück. Das Wrack, 158 Meter lang, 16 Meter breit, 11 Meter hoch und 13000 Tonnen schwer lag anschließend über ein Jahr in 108 Metern Tiefe.

Bereits im Herbst 2000 begannen die Ingenieure des russischen Konstruktionsbüros Rubin, die die Kursk gebaut hatten, mit den Vorbereitungen zur Bergung. Erklärtes Ziel war es, das U-Boot noch 2001 zu heben. Den Auftrag für dieses Projekt erhielt das niederländische Konsortium Mammoet Smit International. Die hier beteiligten Firmen befassen sich mit Schwerlast- und Off-Shore-Transport. Doch auch für sie war dieses Projekt aufgrund der schieren Größe einmalig. Außerdem ist aus der Klimabeobachtung des Nordmeeres bekannt, dass ab Anfang Oktober sehr schwer vorhersagbares und oft stürmisches Wetter einsetzt, die Aktion mußte also unter enormem Zeitdruck ablaufen.

2 ... und wie sie geborgen werden sollte

Wer ein U-Boot heben will, kann dies auf sehr verschiedene Arten tun. Hier entschloss man sich sehr früh, eine besondere Form der Krantechnik einzusetzen. Der Plan sah vor, 26 Bündel von je 54 Stahltrossen mit Greifern zu versehen, die in Löchern in der Außenwand des Wracks befestigt werden sollten. Auf dem Ponton „Giant 4“, einem riesigen schwimmfähigen Kasten, waren entsprechend 26 Hubeinrichtungen montiert, welche die Trossenbündel mit der Last daran heben sollten. Der Ponton schwimmt in den Wellen der Meeresoberfläche, während das Wrack am Seeboden liegt. Würde man nun die Trossen straff spannen, dann käme es bei jeder Wellenhebung zu einer starken ruckartigen Belastung, die auch die stärksten Stähle brechen läßt. Aus diesem Grund mussten die Heber auf dem Ponton gefedert werden, um den erwarteten Seegang von etwa 3 Metern auszugleichen. Wie sieht eine Feder aus, auf der 40 voll beladene Sattelschlepper um drei Meter auf und nieder wippen können? Auf der „Giant 4“ setzten die Ingenieure 104 Gasdruckzylinder ein. Je vier von diesen je 4 Meter langen und halbmeterweiten Rohren tragen auf dicken Stempeln eine Plattform, die wiederum den eigentlichen hydraulisch arbeitenden Hubzylinder hält. Die Gaszylinder sind mit Stickstoff unter Drücken von bis zu 160 bar gefüllt. Um ihre Elastizität zu erhöhen, hängt an jeder Vierergruppe von Gasfedern noch ein Container mit 128 Druckflaschen von je 50 Litern Fassungsvermögen. Die gesamte Anlage hält etwa 200.000 Liter Stickstoff unter Druck und sucht damit weltweit ihresgleichen.

Zunächst wurden die Gasfedern, die nach ihrem Zweck auch Wellenkompensatoren genannt wurden, idealisiert betrachtet; sie sollten während des Eindrückens in entsprechender Weise ihre Kraft erhöhen. Diese Näherung musste verfeinert werden, denn ein Gas setzt seiner Kompression einen überverhältnismäßig hohen Wiederstand entgegen. Außerdem erhitzt es sich bei der Kompression und gibt die entstehende Wärme über die Behälterwandungen ab, was wiederum den Druck abfallen läßt. Der Autor dieses Artikels und seine Kollegen befassen sich seit längerem mit der Berechnung des Verhaltens von Stickstoff und anderen Gasen unter hohen Drücken. So behandelten die ersten Aufgaben, mit denen ihre Ingeniergemeinschaft IgH betraut wurde, das Verhalten der Gasdruckzylinder. So kam wieder einmal die Programmsammlung zur Stickstoffmodellierung zum Einsatz. Diese Programme sind mit dem Numeriksystem „octave“, das unter `http://www.octave.org` erhältlich ist, entwickelt worden, eine Software, die unter der GNU General Public License steht und ein außerordentlich wertvolles Ingenieurs-Werkzeug darstellt. Das gilt insbesondere dann, wenn es in seiner vertrauten Umgebung

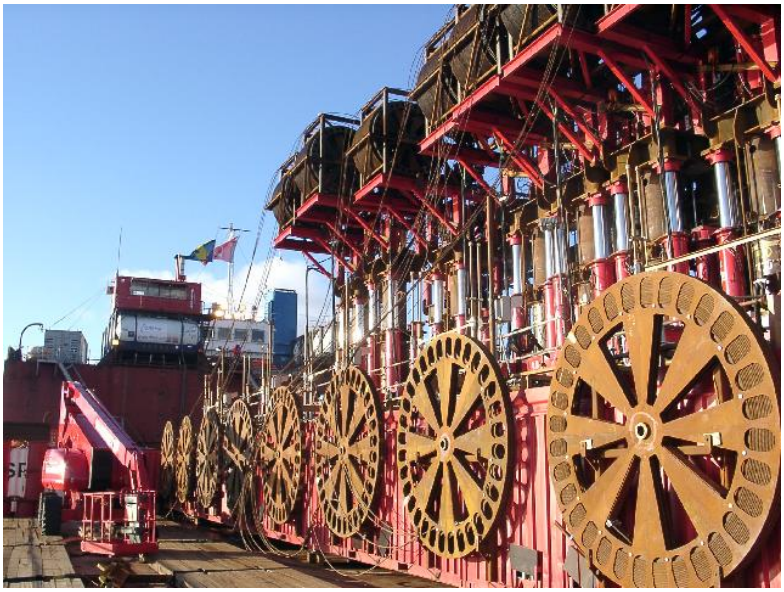

Abbildung 1: Die gewaltigen Wellenkompensatoren

eingesetzt wird, und die heißt Linux. Die IgH verwendet als Dienstleistungs- und Entwicklungsfirma im Maschinenbau dieses Betriebssystem bereits seit der Gründung 1995 mit großem Erfolg sowohl als Server- wie auch als Desktop-Betriebssystem.

Abbildung 2: Arbeiten an den Wellenkompensatoren

Die zahllosen technischen Details des Bergungsprojektes stellten über die Fragen zu den Gasdruckfedern hinaus erhebliche weitere Herausforderungen für die IgH-Ingenieure dar, denn deren Rat war nun auch in weiteren technisch schwierigen Fragen begehrt. Es zeigte sich bald, dass es Probleme von ungleich schwierigerer Berechenbarkeit gab als das Stickstoffverhalten der Gasfedern. Diese Federn stellten aus konstruktiven Gründen ein äußerst sensibles Element der Mission dar. Ihre verfügbare Arbeitslänge von drei Metern mußte unter allen Umständen eingehalten werden. Die Kolben durften in den Zylinderrohren keinesfalls anschlagen. Wäre nämlich ein Kolben vollends eingetaucht, dann hätten die Trossen eine schlagartige Belastung erlitten, die sie nicht ertragen hätten. Hätte andererseits der Kolben versucht, das Zylinderrohr zu verlassen, dann hätte er sich in einer keilartigen Sicherungskonstruktion festgefressen. Diese war notwendig, weil die Kolben wie Projektile eines Luftgewehrs hochgeschleudert worden wären, wenn etwa das Seilbündel gerissen wäre. Die Spannkraft des Gases hätte ausgereicht, eine 60-Tonnen-Plattform etwa 50 Meter hoch zu katapultieren. Bei diesen harten Randbedingungen erhielt nun die Kenntnis des Bewegungsablaufes von Ponton und Wrack entscheidende Bedeutung. Das ganze System bestand aus riesigen Massen, die elastisch miteinander verbunden waren. Derartige Systeme sind schwingungsfähig, insbesondere dann, wenn sie etwa durch periodische Wellenbewegungen angeregt werden. Tatsächlich zeigten erste überschlägige Rechnungen, dass die Kursk Pendel- und Taumelbewegungen in allen Richtungen des Raumes ausführen konnte, ohne sich durch die umgebenden Wassermassen nennenswert bremsen zu lassen.

3 Kontrolle ist besser

Zur Kontrolle des Bergungssystems waren über 400 Sensoren für Drücke, Temperaturen, Wege und so weiter eingebaut. Die Steuerung erfolgte über mehr als 700 Ventile, die teils automatisch und teils manuell betätigt wurden. Das gesamte System musste schließlich auf Anhieb funktionieren. Also wurde die IgH beauftragt, ein Simulationssystem zu erstellen, das vor allem diese Zwecke zu erfüllen hatte:

- das Steuerungssystem entwickeln und testen,
- die Bedienungsmannschaft trainieren,
- ein Handbuch mit Verfahrensanweidungen für den Hebevorgang entwickeln,

- während des Hebevorgangs Verfahrensvarianten durchspielen.

Die Simulationssoftware wurde mit Matlab/Simulink von der Firma Mathworks entwickelt. Es handelt sich hier um eine Kombination aus Numerik- und Simulationssoftware, die einen Kommandointerpreter, eine grafische Programmieroberfläche und Schnittstellen zu den klassischen Compilerhochsprachen wie C, C++ und Fortran anbietet. Dieses Werkzeug verbindet also Entwicklungseffizienz mit Recheneffizienz. Diese war dringend erforderlich, um das Differentialgleichungssystem mit über 300 Integratoren in mehrfacher realzeitlicher Geschwindigkeit zu lösen, denn es sollten in der knappen Zeit möglichst viele Hebeaktionen vorab durchgespielt werden, die kürzer dauern sollten als die real geplanten 12 Stunden. Die numerisch aufwendigen Funktionen, die zum Beispiel das Bewegungsverhalten beschrieben, wurden also in C programmiert. Matlab/Simulink und C sind auf den verschiedensten Betriebssystemen verfügbar. In diesem Projekt aber wurde Linux gewählt. Neben der Robustheit gaben weitere Eigenschaften den Ausschlag:

- die hervorragende Eignung für Serverdienste im Netzwerk,
- die Verfügbarkeit exzellenter Software (emacs, gcc, gdb, octave, perl und andere mehr) in ihrer angestammten Umgebung,
- die elegante SYS V Inter Process Communication (IPC).

Dieser Artikel soll die hervorgehobenen Linux-Eigenschaften näher beleuchten.

Die Bedienung der Gasdrucksysteme erfolgte über einzelne Speicherprogrammierbare Steuerungen (SPS), die über redundante Busse an die Leitzentrale angebunden waren. Als Protokoll kam Modbus Plus der Firma Modicon zum Einsatz. Dieses Protokoll steht darüber hinaus in einer Variante zur Verfügung, die in TCP/IP eingebettet ist und daher „Open Modbus/TCP“ heißt. Die Protokolldocumentation ist unter `http://www.modicon.com/openmbus/standards/openmbus.htm` erhältlich. Für die Entwicklung der Steuerungsanlage war dies von großem Vorteil, denn die Firma Raster, die diese Aufgabe übernommen hatte, arbeitet in den Niederlanden. Raster entwickelte die Leitstandsoberfläche unter Windows 2000 und war nun darauf angewiesen, möglichst rasch Tests am Simulationssystem durchzuführen, das in Essen entwickelt wurde. Die Anbindung erfolgte über das Internet. Es ist dabei durchaus erwähnenswert, dass dabei die Datenpakete per Portforwarding durch Firewalls geschleust wurden, die - natürlich - unter Linux arbeiten.

Die Leitstandsrechner der Firma Raster verhielten sich den dezentralen Steuerungen gegenüber wie Clients, die eine Vielzahl von Servern abfragen. Diese liefern Messdaten und nehmen Stelldaten entgegen. Das Protokoll ist binär und vergleichsweise einfach aufgebaut. Für die Kursk-Bergung wurden nur die einfachsten Protokollfunktionen, nämlich Lesen und Schreiben benötigt. Jede Steuerung ist am Bus über eine Unit-Id gekennzeichnet und verfügt über einen Speicherbereich von 1024 Registern zu je zwei Byte in Big-Endian-Kodierung. Das Busprotokoll erlaubt nun, mit einer Unit Registerinhalte auszutauschen. Die Kodierung der einfachen hier verwendeten Funktionen ist in Abschnitt B(„Open Modbus/TCP“) zusammengestellt.

4 Schein und Wirklichkeit - oft dicht beieinander

Das Simulationssystem bildete diese Registerstrukturen nach. An allen simulierten Wellenkompensatoren wurden fortwährend die aktuellen Zustandsdaten hinterlegt und die Requests von seiten des Leitrechners abgefragt und bearbeitet. Das Entwicklungskonzept der Simulationssoftware sah jedoch die Konzentration auf die eigentlichen Rechenaufgaben vor. Auf die Implementierung eines eigenen Öpen Modbus/TCPServers sollte verzichtet werden. Diese Aufgabe übernahm ein kleines Script in Perl, das für derartige Anwendungen geradezu prädestiniert ist. Der Datenaustausch zwischen dem Simulationssystem erfolgte via Sys V Inter Process Communication (IPC). Dabei greifen mehrere Prozesse auf einen gemeinsamen Speicherbereich zu. Normalerweise verhindert das Betriebssystem den Zugriff eines Prozesses auf die Datenbereiche anderer Prozesse strengstens. Eine Gruppe von Betriebssystemfunktionen jedoch erlaubt den wohldefinierten Zugriff auf gemeinsam genutzte Speicherbereiche.

Sollen mehrere Prozesse gemeinsamen Speicher benutzen, also also Shared Memory IPC betreiben, dann muß ein erster Prozeß beim Betriebssystem ein Speichersegment einer bestimmten Größe anfordern und dessen Zugriffsberechtigungen festlegen, die analog zu Dateizugriffsrechten formuliert werden. Das Betriebssystem liefert eine Nummer zurück, über die das Speichersegment identifiziert wird. Diese Shared-Memory-Id muß anderen Prozessen bekanntgegeben werden, die ebenfalls auf dieses Segment zugreifen sollen. Der Einfachheit halber schreibt der erste Prozeß diese ID in eine kleine Datei. Jeder Prozeß, der nun zugreifen möchte, schaltet sich auf das Segment auf. Das Betriebssystem liefert dafür einen Zeiger auf den Adreßbereich zurück,

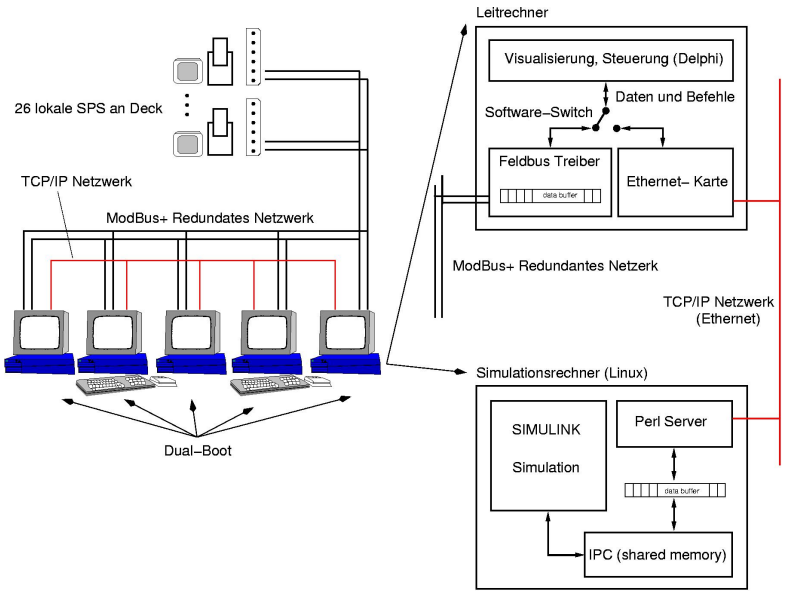

Abbildung 3: Steuer- und Regelungskonzept der „Kursk“-Hebemechanik

in dem der Prozeß nun arbeiten kann. Will der Prozeß den Zugriff auf das Speichersegment beenden, dann teilt er dies wiederum dem Betriebssystem mit. Die Verwendung von Shared Memory ist damit unwesentlich komplizierter als die dynamische Speicherallozierung. Optimal ist die Sys-V IPC immer dann, wenn ein Prozeß schreibt und mehrere andere lesen sollen, ohne sich untereinander zu stören. Neben den Systemaufrufen stellt Linux noch einige einfache Kommandos zur Handhabung der Speichersegmente bereit. Im Abschnitt A („Sys V Inter Process Communication“) sind die wichtigsten Systemaufrufe und Kommandos zusammengestellt.

5 Kleine, aber feine Helfer

Die Simulation arbeitete nach erheblichem Entwicklungsaufwand, der in gut acht Wochen etwa 12.000 Zeilen C hervorgebracht hatte, auf dem Computeserver der IgH unter voller Last etwa fünfmal schneller als die Realität. Das parallel entwickelte Steuerungssystem besaß eine Umschaltoption. Entweder wurde es über den Feldbus mit den realen Steuerungsstationen verbunden, die aber noch keine Daten lieferten, oder über TCP/IP mit dem Simulationsrechner. Dort arbeitete ein kleiner Perlserver, der dem Steuerungsrechner ein komplettes Liftingsystem vorgaukelte, indem er simulierte Meßda-

ten lieferte und Anweisungen entgegennahm. Das Listing ist in Abschnitt C („Perl Modbus Server“) wiedergegeben. Der Server arbeitet im sogenannten Single-Thread. Das heißt, alle Requests werden direkt vom Server bearbeitet. Dies ist sinnvoll, weil der Datenumfang jedes Requests und jedes Replies nur sehr gering ist, also sehr schnell abgearbeitet werden kann. Anders als bei Webservern etwa kann man auf den Start von Subprozessen hier verzichten. Die Kommunikation erfolgt binär, es gibt also keine definierten Zeichen, an denen der Datenstromzustand erkannt werden könnte, etwa ein Zeilenende. Daher bearbeitet der Server jeweils genau einen Request, der nach endlich vielen Zeichen abgehandelt ist und schließt dann von sich aus die Verbindung zum Client. Damit befindet er sich auf jeden Fall wieder in einer definierten Situation. In guter Daemonensitte läuft auch dieser Server im Hintergrund, weshalb nach Programstart ein `fork` aufgerufen wird. Praktischerweise hinterlegt der Server seine Prozessnummer in einer Datei `perlserver.pid`, so dass der Aufruf des Kommandos

```
kill -TERM ‘cat perlserver.pid‘
```

immer zielsicher den richtigen Prozeß meuchelt. Das System wird unter Administratorenrechten gefahren, weil der Modbus/TCP-Port 502 privilegiert arbeitet. Daher sind kleine Vorsichtsmaßnahmen sehr angebracht. Andererseits arbeitet die gesamte EDV an Bord in einer sehr vertrauenswürdigen Umgebung, so dass hochentwickelte Sicherheitskonzepte entbehrlich waren. Schlielich ist es auch noch die Aufgabe des Perl-Servers, das Shared-Memory-Segment beim Betriebssystem anzufordern und dessen Id in einer Datei zu hinterlegen, die vom Simulationssystem gelesen wird. Auf einer ordentlichen Maschine mit einem Pentium III Prozessor bei 1 GHz arbeitete der Server auch bei 40.000 Requests pro Sekunde völlig stabil und ohne merkliche Störung der sonstigen Prozesse.

Die Reisezeit der Giant 4 in die Barentsee wurde zu umfangreichen Trainings genutzt, während derer die Mannschaft der Operatoren sich mit dem Bergungssystem und mit den Abläufen vertraut machte. Am Simulationssystem spielte eine Operator sozusagen Schicksal und war Herr über Wind, Wetter und Wellen. Die Operatoren an den Leitrechnern mußten aus der vorgegebenen Situation nun jeweils das beste machen. Aus einer Vielzahl von beispielhaften Szenarien und möglichen Abläufen wurden die voraussichtlich günstigsten Maßnahmen entwickelt und handbuchartig formuliert. So entstand ein wichtiges Hilfsmittel für die eigentliche Prozedur, die im Ernstfall wenig Zeit zum gründlichen Nachdenken gelassen hätte.

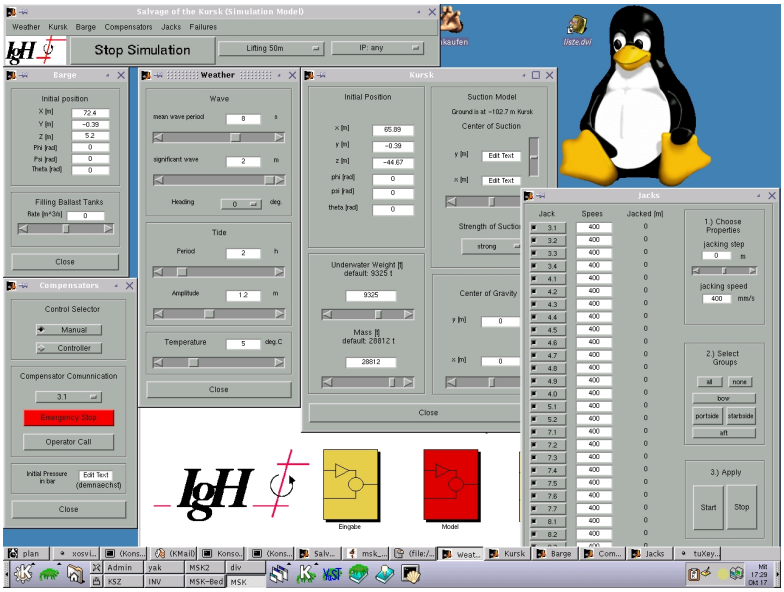

Abbildung 4: Oberfläche des Simulationssystems

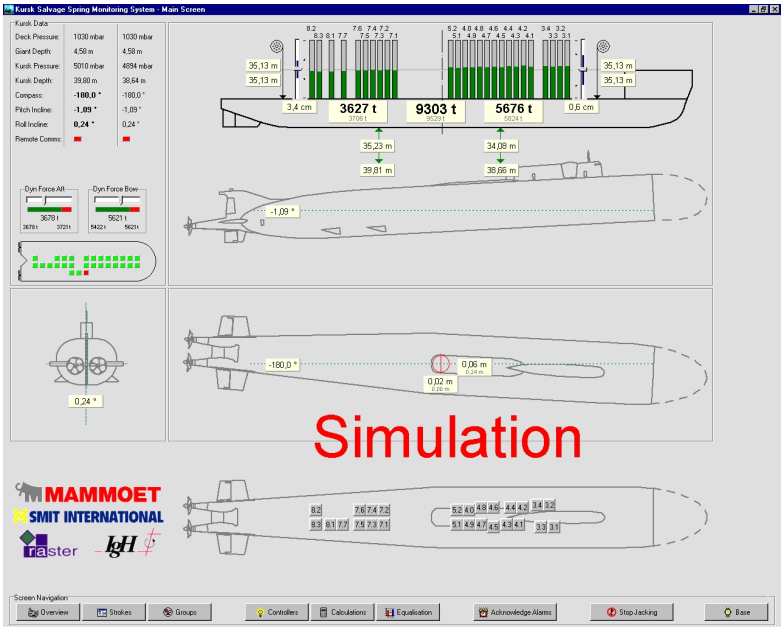

Abbildung 5: Oberfläche des Steuerungssystems

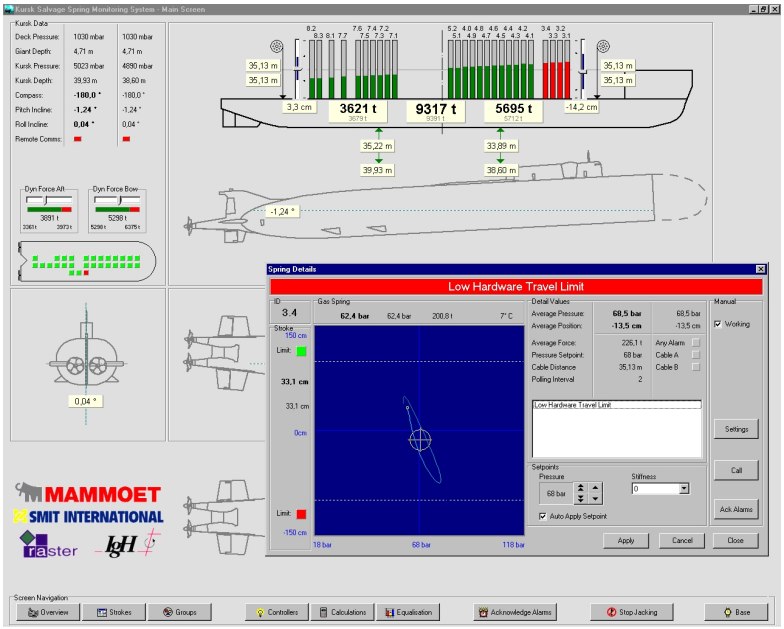

Abbildung 6: Oberfläche des Steuerungssystems mit Druck/Hub-Verlauf

6 Das U-Boot am Haken

Beim eigentlichen Lifting befanden sich aus Redundanzgründen fünf Rechner im Leitstand, die für das Wellenkompensatorensystem zuständig waren. Einer diente als Leitstation für die Kompensatorensteuerung, ein weiterer präsentierte laufende Systemdaten der Hebeaktion und stellte diverse Tools zur Verfügung, beispielsweise zur optimalen Lastverteilung der einzelnen Heberpunkte. Ein dritter protokollierte kontinuierlich den Hebevorgang in allen erdenklichen Daten. Die übrigen Systeme wurden zur Online-Simulation verwendet. Die fünf Maschinen waren äquivalent aufgesetzt und konnten über Dual-Boot entweder Linux oder Windows 2000 fahren. Die Linux-Systeme fügten sich via Samba problemlos in das übrige Netzwerk ein. Das heterogene Netzwerk stellte also auch für dieses System, das definitiv missionskritisch war, nicht nur kein Problem dar, sondern erlaubte die gezielte Nutzung der jeweiligen Systemvorteile.

Das Heben des Wracks erfolgte schließlich in der Nacht vom 7. auf den 8. Oktober. Dabei waren zwei Kollegen des Autors als Mitglieder eines Teams von über 50 Mitarbeitern aus vielen europäischen Ländern maßgeblich beteiligt. Obwohl das Projekt noch kurz vor dem Ende in stürmischer See zu scheitern drohte, gelang die Bergung so, wie von der Simulation vorher beschrieben. Die Kursk wurde unter den Rumpf der Giant 4 gezogen und nach Rosljakovo nahe Murmansk in ein Trockendock geschleppt.

7 Zum Gedenken

Bei allem Erfolg der Bergung gedenken wir der 118 Mitglieder der Kurskbesatzung, die am 12. August 2000 Opfer der Tragödie in der Barentsee wurden. Unser tiefes Mitgefühl gilt ihren Angehörigen.

A Sys V Inter Process Communication

Die IPC-Systemaufrufe und Kommandos sind unter Linux in den Manual-Pages technisch detailliert beschrieben. Die wichtigsten von ihnen lassen sich mit einfachen typischen Aufrufen beispielhaft beschreiben. Hier ist eine Umsetzung in C beschrieben. Die Programme müssen die entsprechenden Systemheaderdateien einbinden:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

FILE sid_file;
int shm_size, shm_flags, shm_id;
char *data;

...
shm_size = 1024;   /* Segmentumfang in Bytes       */
shm_flags = 0666;  /* Zugriffsberechtigungen
                      analog zu chmod (octal!)     */

...
/* Anforderung eines Speichersegmentes */
shm_id = shmget(IPC_PRIVATE, shm_size, shm_flags);
             /* IPC_PRIVATE ist in shm.h definiert */
...

/* Abspeichern der Shared Memory Id in eine Datei  */
sid_file = fopen("sid-info-file.txt", "w");
fprintf(sid_file, "%d\n", shm_id);
fclose(sid_file);
...

/* Aufschalten auf das Speichersegment             */
/* auf die gleiche Weise erhalten alle anderen
   interessierten Prozesse Zugriff                 */
data = shmat(shm_id, 0, 0);
...

/* Speichersegment abhaengen                       */
shmdt(data);
...
```

An der Kommandozeile stehen kleine Tools zur IPC-Verwaltung zur Verfügung. Das Kommando `ipcs` liefert Informationen über IPC-Ressourcen, beispielsweise die Id, die Größe und die Anzahl der angehängten Prozesse. Der Aufruf `ipcrm shm 123` schließlich würde das Shared-Memory-Segment mit der Id 123 löschen, sofern der Benutzer die Berechtigung dazu besitzt.

B Open Modbus/TCP

Das Modbus Plus Protokoll ist binär wortorientiert in Big-Endian-Notation. Ein Wort umfaßt zwei Bytes. Es kennt mehrere Klassen von Funktionen, von denen hier nur die Klasse 0 verwendet wird. Sie enthält die Funktionen Lesen(3) und Schreiben(16). Die Einbettung des Protokolls in TCP erfolgt über einen zusätzlichen Header. Die ersten beiden Worte des Headers sind in der Regel 0. Es folgt ein Wort, das die Zahl der folgenden Bytes angibt. Das nächste Byte gibt die Id der Serverstation an, gefolgt von einem weiteren Byte mit dem Funktionskode.

Das Protokoll verwendet über TCP den well known port 502. Ein Server für dieses Protokoll muß daher mit privilegierten Rechten laufen, de facto als root!

Beispiele:

In den Registern 4 und 5 der Station 13 stehen die Werte 47 und 11. Will der Client diese Register lesen, dann sendet er den Request

```
0 0 0 0 0 6 13 3 0 4 0 2
```

Die Station (der Server) antwortet mit

```
0 0 0 0 0 7 13 3 4 0 47 0 11
```

Hinter dem Funktionskode(3) wird die Anzahl der folgenden Bytes angegeben. Die Werte folgen wieder in Wortnotation.

In die Register 7 und 8 der Station 5 soll die Werte 21 und 37 eingetragen werden. Der Client sendet hinter dem Funktionskode(16) die erste Registeradresse und die Anzahl der zu schreibenden Worte. Skurrilerweise folgt dann die Anzahl der folgenden Bytes in einem(!) Byte und schließlich die Werte selbst.

```
0 0 0 0 0 11 5 16 0 7 0 2 4 0 21 0 37
```

Die Station bestätigt die Registeradresse und die Anzahl der geschriebenen Worte:

```
0 0 0 0 0 6 5 16 0 7 0 2
```

C Perl Modbus Server

In vielen technischen Prozessen und Berechnungen fallen Datenströme von großer Dichte und unvorhersagbarer Dauer an. Oft ist es dabei interessant, diese Daten via Netzwerk zu übertragen und weiterzuverarbeiten. Dabei gibt es Applikationen, die als kontinuierliche Datenquellen oder -senken arbeiten. Es bietet sich an, die Datenübertragung durch einfache Serverprogramme zu realisieren. Ein Beispiel für ein solche Anwendung ist der in Perl geschriebene Modbus-Server, der bei der Bergung der Kursk verwendet wurde.

Das Programm ist hier vereinfacht wiedergegeben und soll als Anregung für die Anwendung verschiedener nützlicher Techniken dienen:

- Zugriff auf Shared Memory
- Signalhandler
- servertypischer Fork
- Socket-Verbindung via TCP/IP
- Binärdatenwandlung mit Pack (etwa Big- in Little-Endian Konvertierung)

Statements zur Fehlerbehandlung sind weitgehend entfernt.

```
#!/usr/bin/perl -w

# Singlethreaded Server
# In Kombination mit SysV-IPC (ShMem)

use Socket;
use IPC::SysV qw(IPC_RMID IPC_PRIVATE);
```

```
# Process Id und Shared-Memory Id in Dateien
# ablegen, die den Servernamen tragen
$myself ( $0 =~ s/\.pl$// );
$pidfile = "$myself.pid";
$sidfile = "$myself.sid";

$shm_flags = 0666;  # Zugriffsrechte: rw-rw-rw-
$tcpmodbus = 502;   # Modbus well known port (privilegiert!)

$max_unit = 28;     # Anzahl der Steuerungsstationen (SPS)
$max_ref = 1024;    # letztes Register der SPS
$unit_size = 1024;  # Registeranzahl
$word_size = 2;     # Bytes pro Wort

# Platz fuer Shared Memory
$shm_size = 2 * $max_unit * $unit_size * $word_size; # array twice!

$MODBUS_READ  = 3;    # Modbus Funktion: Lesen
$MODBUS_WRITE = 0x10; # Modbus Funktion: Schreiben

# Unterprogramm zum Programmende (ausgeloest durch kill -TERM)
sub getout {
  shmctl ($sid, IPC_RMID, 0);     # Shared Memory freigeben
  unlink $pidfile, $sidfile;
  exit 0;
}

# Forken und im Hintergrund weiterarbeiten.
if ($pid = fork) {
  exit 0;
}

# Signalhandler fuer kill -TERM bereitstellen und
# an alle Kindprozesse vererben.
$SIG{TERM} = \&getout;

# Process Id vermerken.
open (PID, ">$pidfile");
print PID "$$\n";
close PID;

# Shared Memory anlegen
```

```
$sid = shmget(IPC_PRIVATE, $shm_size, $shm_flags);

# Shared Memory Id vermerken
open (SID, ">$sidfile");
print SID "$sid\n";
close SID;

# Server Port und Protokoll
my $port = $tcpmodbus;
my $proto = getprotobyname('tcp');

# Server socket erstellen,
# Hostadresse binden und
# auf Requests warten
socket(Server, PF_INET, SOCK_STREAM, $proto);
setsockopt(Server, SOL_SOCKET, SO_REUSEADDR, pack("l", 1));
bind (Server, sockaddr_in($port, INADDR_ANY));
listen (Server, SOMAXCONN);

my $paddr;
# Endlos Verbindungen akzeptieren, bearbeiten und
# wieder schließen.
for ( ; $paddr = accept(Client, Server); close Client ) {
  my ($port, $iaddr) = sockaddr_in($paddr);
  my ($ta_id, $prot_id, $ta_len, $unit_id, $mb_fc, $bc,
      $ref, $count, $data, @data, $got, $line, $header,
      $req, $sent, $string);

  $req = 12; # Die ersten 12 bytes lesen
  recv Client, $line, $req, 0;
  # und in ihre Bestandteile zerlegen
  ($ta_id, $prot_id, $ta_len, $unit_id, $mb_fc, $ref, $count) =
    unpack "nnnCCnn", $line;
  if ( $mb_fc == $MODBUS_READ ) {
    # mehr lesen
    shmread $sid, $line, 2*$ref, 2*$count;
    $line = pack('n*', unpack 'S*', $line);
    $header = pack 'nnnCCC',
    $ta_id, $prot_id, 2*$count+3, $unit_id, $mb_fc, 0xff;
    $string = $header . $line;
    # und antworten
    send(Client, $string, 0);
  } elsif ( $mb_fc == $MODBUS_WRITE ) {
```

```
    # oder schreiben
    $req = 2*$count+1;
    recv Client, $line, $req, 0;
    ($bc, @data) = unpack 'Cn*', $line;
    shmwrite ($sid, pack ('S*', @data),
              2*(($unit_id-1)*$unit_size+$ref), 2*$count);
    $header = pack 'nnnCCCnn',
    $ta_id, $prot_id, 5, $unit_id, $mb_fc, $ref, $count;
    send (Client, $header, 0);
  }
}
```

D Perl Modbus Client

Bevor man einen Server aus einer hochkomplexen Applikation heraus anspricht, schreibt man sich einen kleinen Test-Client. Auch dafür ist Perl bestens ggeignet. Hier ist das Gegenstück zum Modbus-Server wiedergegeben, um eine Anregung für Client-Server Programmierung zu geben.

```
#!/usr/bin/perl -w

use Socket;
use Getopt::Std;

use vars qw($opt_u $opt_r $opt_c $opt_g $opt_t $opt_h $opt_s $opt_p);

my ($remote, $port, $iaddr, $paddr, $proto, $line, $ans);
my ($ta_id, $prot_id, $unit_id, $mb_fc, $ref, $count, @data);

# Der Client wird flexibel durch Optionen, die es erlauben,
# ihn wie ein klassisches Unix-Tool mit vielen Parametern
# aufzurufen
$unit_id = $opt_u = 1;         # SPS-Station
$ref =     $opt_r = 0;         # Register darauf
$count =   $opt_c = 16;        # Anzahl übertragener Register
           $opt_g = 0;         # Anforderung zum Lesen
           $opt_t = 0;         # Anforderung zum Schreiben
           $opt_h = 0;         # Anforderung der Hilfe
$remote =  $opt_s = 'server';  # IP-Name des Servers
$port =    $opt_p = 502;       # Port-Nummer
```

```
getopts('u:r:c:gths:p:'); # Abfrage der Parameter

if ($opt_h)  {
# Netterweise eine Gebrauchsanweisung
  print "\n usage: $0 [-u unit(1)] [-r register(0)] [-c count(16)]\n",
          "         [(-g et)|-t ransmit] [-h elp]\n",
          "         [-s server(yak)] [-p port(502)]\n\n";
  exit;
}

# Welche Optionen sind eingegeben worden?
$unit_id = $opt_u;
$ref =     $opt_r;
$count =   $opt_c;
$remote =  $opt_s;
$port =    $opt_p;

$mb_fc = 3;
if ($opt_t) {
  unless ($opt_g) {
#   wenn nicht lesen, dann schreiben
    $mb_fc = 16;
    @data = @ARGV;
    $count = $#data + 1;
  }
}

$ta_id = 1234;   # beliebig
$prot_id = 502;  #    " - aber in Anlehung an die Portnummer

# Verbindungsdaten festlegen
if ($port =~ /\D/) { $port = getservbyname($port, 'tcp') }
die "No Port" unless $port;
$iaddr = inet_aton($remote) or die "No Host: $remote";
$paddr = sockaddr_in($port, $iaddr);
$proto = getprotobyname('tcp');

while ( 1 ) {
# dieser Client hält eine dauernde Verbindung zum Server
# dies erlaubt die kontinuierliche Beobachtung der Simulation
  socket(SOCK, PF_INET, SOCK_STREAM, $proto) or die "socket: $!";
  connect(SOCK, $paddr) or die "connect: $!";
```

```
# Netzwerkverbindung und Terminalausgabe ungepuffert.
  select SOCK;
  $| = 1;
  select STDOUT;
  $| = 1;

# Request formulieren (binär!)
  $line = pack "nnnCCnn", $ta_id, $prot_id, 6, $unit_id, $mb_fc, $ref, $count;

  if ( $mb_fc == 0x10 ) {
#   binär kodieren - Big Endian
    $line .= pack 'Cn*', 2*$count, @data;
  }

# und absenden
  send SOCK, $line, 0;

# ein wenig Geduld zeigen - hier 100 msec
  select(undef, undef, undef, 0.1);
# und auf Antwort warten
  next unless defined(recv SOCK, $ans, 6+3+2*$count, 0);
# Wenn der Server die Verbindung nicht schließt, dann tun wir das
  close (SOCK);
# und bereiten die Ausgabe vor
  my $header = substr($ans, 0, 6);
  my ($tid, $prid, $hilen, $lolen) = unpack 'nnCC', $header;
  my ($unit, $fc, $bc) = unpack 'C*', substr($ans, 6, 3);
# binär dekodieren
  @data = unpack 'n*', substr($ans, 9);
  my $len = 0x100 * $hilen + $lolen;
  print "Unit $unit(Ref $ref): ";
  foreach (@data) {
    printf "%5d ", $_;
  }
  print "\n";
  if ( $mb_fc == 0x10 ) {
#   wenn wir nur geschrieben haben, Schluß
    exit;
  }
}
exit;
# That's it
```

E Die Autoren

Der Autor des Artikels ist:

Dr.-Ing. Torsten Finke

An der Bergung beteiligt waren:

Dr.-Ing. Siegfried Rotthäuser Dr.-Ing. Wilhelm Hagemeister

Neben diesen zeichnet für wesentliche Teile der Software verantwortlich:

Dipl.-Ing. Jörg Essmann

Alle sind Mitarbeiter der Ingenieurgemeinschaft IgH in Essen.

F Links:

`www.igh-essen.com`

`www.kursksalvage.com`

`www.mammoet.com`

`www.modicon.com`

`www.raster-ia.nl`

`www.mathworks.de`

`www.octave.org`