1A05
64-bit Adressierung unter Alpha OpenVMS

Mit Compaq C und Compaq C++

64 Bit Adressierung

Raymund Grägel
SWG GmbH, Sandhausen
OpenVMS seit 1980, VMS2.5
Dienstleistung und Softwareentwicklung

Hintergrund: Xetra Observer

Unterlagen und Beispiele:
www.swg-gmbh.de/decus
kontakt@swg-gmbh.de
Tel 06224-80315

Agenda

- Motivation
- 32-bit Welt in den VMS Sprachen historisch betrachtet
- Programmierung
- 64-bit Referenzen in Argumenten von System Services
- 64-bit Pointer in Item-Lists bei System Services
- Herstellung der Adressräume
- Vorhandene Bibliotheken und Umstellung eines Softwarepakets

Was ist das Problem?

SS$_VASFULL – virtual adress space is full

```
%SYSTEM-F-VASFULL, virtual address space is full
$ HELP /MESSAGE VASFULL
VASFULL,  virtual address space is full
  Facility:     SYSTEM, System Services
  Explanation:  The virtual address space is full.
  User Action:  Reboot and increase the amount of allowable
virtual address space. If this message is associated with a
vector disabled (VECDIS) status code, insufficient process
virtual address space exists to allow the current process's
mainline vector state to be saved.
```

Warum gibt es das Problem ?

- Die Portierung von VAX auf Alpha muss sourcekompatibel sein.
- Deshalb 32-bit orienterte Sprachen.
- speziell System Services und Runtimelibrary (RTL)

Was muss man tun?

- Den Compiler auf 64-bit Adressen umschalten
- Datenstrukturen, die Pointer enthalten, ändern ihren Aufbau (Descriptoren, Item-Lists)
- Argumente von System Service mit Argumenten by reference prüfen.

Compaq C-Compiler

$ CC /POINTER=LONG

```
$ CC SYS$INPUT /OBJ=TMP.OBJ
#include <stdio.h>
main () {printf ("%d\n", 8*sizeof (void*) ) ;}
$ link tmp
$ run tmp
32
$
```

Normalerweise sind die Pointer 32 Bits lang

Aber:

```
$ CC SYS$INPUT /OBJ=TMP.OBJ /POINTER=64
#include <stdio.h>
main () {printf ("%d\n", 8*sizeof (void*) );}
$ link tmp
$ run tmp
64
$
```


Compaq C, Mittel

$ CC /POINTER

```
$ CC /POINTER=32
$ CC /POINTER=short
$ CC /POINTER=64
$ CC /POINTER=long
```

Direktive __INITIAL_POINTER_SIZE

```
#if __INITIAL_POINTER_SIZE == 64
```

/NOPOINTER (default)	0
/POINTER=32	32
/POINTER=SHORT	32
/POINTER=64	64
/POINTER=LONG	64

Pragma __REQUIRED_POINTER_SIZE

```
#pragma __required_pointer_size __save
#pragma __required_pointer_size 32
#pragma __required_pointer_size __short
#pragma __required_pointer_size 64
#pragma __required_pointer_size __long
#pragma __required_pointer_size __restore
```

Compaq C, Pragma

Beides, Short- und Long-Pointer

```
#pragma __required_pointer_size __save
#pragma __required_pointer_size __long
typedef char* CHAR_PQ ;
#pragma __required_pointer_size __short
Typedef __int64* INT64_PS ;
#pragma __required_pointer_size __restore
   char a ; __int64 i
   CHAR_PQ pq_a ;
   INT64_PS ps_i ;
```


Compaq C++ Compiler

Verfügbar ab Version 6.3, Überprüfung mit

```
$ cxx /version nl:
Compaq C++ V6.3-048 for OpenVMS Alpha V7.2-1H1
```

Command Qualifier /MODEL=ANSI impliziert /POINTER=LONG und C++ 64-bit Classlibrary

```
$ CXX /MODEL=ANSI
```

Direktive __INITIAL_POINTER_SIZE

Qualifier	`__initial_pointer_size`
(kein Qualifier)	Undefiniert
`/MODEL=ANSI`	64

Compaq C++, Linker

Man muss CXXLINK benutzen

```
$ CXXLINK /MODEL=ANSI /LOG
```

- Alle C++ Module eines Images sollten mit /MODEL=ANSI kompiliert sein.
- Dieser Befehl ist zu benutzen wenn das Image mindestens ein C++ Modul enthält
- Auch wenn das Modul mit dem Hauptprogramm kein C oder C++ Programm ist

System Services

- STARLET.H und __NEW_STARLET
- Unterscheidung System Services
- Argumente

System Services – STARLET.H

__NEW_STARLET und GENERIC_64

Auszug aus STARLET.H:

```
#ifdef __NEW_STARLET
int sys$gettim(struct _generic_64 *timadr);
#else   /* __OLD_STARLET */
int sys$gettim( __unknown_params);
#endif  /* #ifdef __NEW_STARLET */
```

Auszug aus DEMO_GETTIM.C (vgl. GEN64DEF.H)

```
GENERIC_64 Zpkt;
Zpkt.gen64$r_quad_overlay.gen64$q_quadword = 0 ;
sys$gettime (&Zpkt) ;
```

Deshalb:

```
$ CC  /DEFINE=(__NEW_STARLET=1)/POINTER=LONG
$ CXX /DEFINE=(__NEW_STARLET=1)/MODEL=ANSI
```


System Services - Unterscheidung

- 32-Bit System Service
- 64-Bit Friendly Interface
- 64-Bit System Service
- Der Suffix _64

Aufruf von System Services

- Argument by value – keine Änderung
- Argument by reference
- Argument by descriptor
- Argument mit Itemlist

Referenzargument

Automatische Adresserweiterung von 32 auf 64 Bits

32-bit Beispiel Adresse im rufenden Programm	64-bit Empfangsadresse im Unterprogramm
`0X 7E008064`	`0X 00000000 7E008064`
`0X 80E08064`	`0X FFFFFFFF 80E08064`

„sign extented Address“

Vgl. DEMO_ARG.C

System Services - Descriptor

„by descriptor“
struct dsc$descriptor_s

vgl.
DESCRIP.H

31	15	0
class	dtype	Pufferlänge
Stringadresse (32 bits)		

“by 32- or 64-bit descriptor”

struct dsc64$descriptor_s

31	15	0
Class	dtype	+1 (16 bits)
-1 (32 immer gesetzte bits)		
Pufferlänge (64 bits)		
Stringadresse (64 bits)		

System Services - Descriptor

Auszug aus DEMO_DSC.C

```
#include <descrip.h>
main ()
{
    unsigned short int timlen ;
    char timbuf[40+1];
    $DESCRIPTOR64 (dsc_timbuf, timbuf) ;
    sys$asctim (&timlen, &dsc_timbuf, 0, 0);
}
```

Item-List Entries (ILE)

Item-List Entries (ILE)

Open VMS usage: item_list_2

Open VMS usage: item_list_3

Open VMS usage: 32-bit item_list_2 or 64-bit item_list_64a

Open VMS usage: 32-bit item_list_3 or 64-bit item_list_64b

Vgl. ILEDEF.H

Deklariere mit `ILE2`,

`ILE3`,

`ILEA_64`,

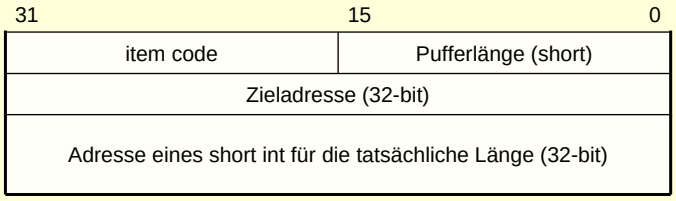
`ILEB_64`

Item-List Entry, 32 bit

Open VMS usage: item_list_3

Der 32-bit ILE

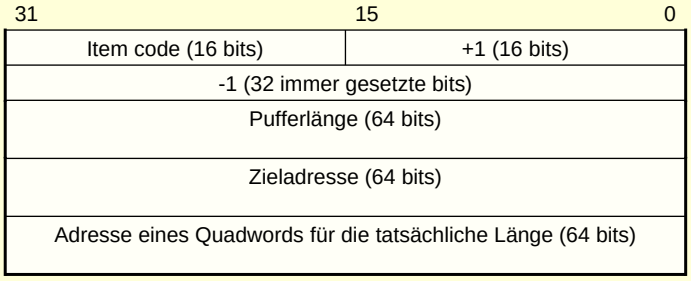
struct _ile3 in ILEDEF.H ILE3



Item-List Entry, 64 bit

Open VMS usage: 64-bit item_list_64b

struct _ileb_64 in ILEDEF.H ILEB_64



vgl. DEMO_ILE.C

Item-List, Beispiel

Auszug aus DEMO_ILE.C

```
struct
{
    SWG_ILEB_64  pid_entry ;
    SWG_ILEB_64  image_entry ;
    SWG_ILEB_64  user_entry ;
    long terminator ;
}   itemlist ;

itemlist.pid_entry.ileb_64$q_length = sizeof (pid) ;
itemlist.pid_entry.ileb_64$w_code = JPI$_PID ;
itemlist.pid_entry.ileb_64$pq_bufaddr = (void*)&pid ;
itemlist.pid_entry.ileb_64$pq_retlen_addr = &lenOfPid ;
itemlist.terminator = 0 ;
status = sys$getjpiw (0,0,0,&itemlist, &userIosb,0,0);
```

Global Sections - Übersicht

- Fab anlegen und File öffnen
- Neu: Region anlegen
- vorhandene Section mappen
- Alternativ: Section anlegen
- Dienstprogramm SWG_GSEC_64.C unter http://www.swg-gmbh.de/decus

Global Section – die Region

```
systemConditionValue = sys$create_region_64
(   bytecount,      // quadword read by value, size of the region
    region_prot,    // uns.long read by value, protection
    flags,          // uns.long read by value, flag mask
    & modgbl.region_id_64 ,
                    // uns.quadw. written by ref, the region id
    & modgbl.region_va_64 ,
                    // pointer written by ref, lowest address of the
                    // region
    & modgbl.region_length_64 ,
                    // uns. quadw written by ref, natural length
    0               // pointer read by value, start address or 0
)   ;
```

existierende Global Section

```
systemConditionValue = sys$mgblsc_64
(
    &dsc_secnam ,  // read by desc, name of the global section
    &ident_64 ,    // read by ref, identification (not really used)
    &modgbl.region_id_64 , // read by ref, the  region id
    0 ,            // read by val, offset
    0 ,            // read by val, the size of the section
    0 ,            // read by val, the access mode
    flagmask,      // read by val, Flag mask specifying options for
                   // the operation
    & pointerToSection ,
                   // written by ref, the process virtual address
    & naturalLengthOfSection ,
                  // written by ref, the natural size of the section
    0             // read by val, a starting address, must not be
                  // given here
)
```


Global Sections – Neuanlage

```
rmsConsitionValue = sys$create (&fab);
systemConditionValue = sys$crmpsc_gfile_64
(
    &dsc_secnam ,  // read by desc, name of the global section
    &ident_64 ,    // read by ref, identification (not really used)
    0 ,            // read by val, offset
    requestedLengthOfSection,  // read by val, size of section
    fab.fab$l_stv, // read by val, Number of channel on which the
                   // file has been accessed
    &modgbl.region_id_64 ,     // read by ref, the region id
    0 ,            // read by val, offset
    0 ,            // read by val, the access mode
    flagmask,      // read by val, Flag mask specifying options
    & pointerToSection , // written by ref, the virtual address
    & naturalLengthOfSection) // written by ref, the size of section
```

64 Bit Programmierung - Tipps

- in einem Projekt (abgeschlossenen Entwicklungsteil) grundsätzlich 64-Bit Adressen verwenden
- Ausnahmen sind Argumente für runtime library routinen.
- Deklariere 32-bit-pointer mit typedef etwa so:

```
#   pragma __required_pointer_size save
#   pragma __required_pointer_size short
typedef long* LONG_PS
typedef void* VOID_PS
#   pragma __required_pointer_size restore
```


Lock Status block (LKSB)

Enthält keine Pointer. Deklaration in LKSBDEF.H:

```
LKSB privatelock;
```

Siehe SYS$ENQ

IO-status-block (IOSB)

Enthält keine Pointer. Deklaration in IOSBDEF.H:

```
IOSB privateOutput;
```

Siehe Beispiel DEMO_GETJPI

Übersicht über die Beispiele

- DEMO_ARG zu Referenzargumenten
- DEMO_DSC zu IOSB und DSC
- DEMO_GETTIM zu GENERIC_64
- DEMO_ILE zu ILE
- SWG_GSEC_64 mit DEMO_GSEC_64 zur Programmierung von global sections

Aus www.swg-gmbh.de/decus

Kontakt@swg-gmbh.de

Tel 06224-80315

Literaturhinweise

OpenVMS Alpha Guide to 64-Bit Addressing and VLM Features LINK
Guidelines for Writing Clean 64-Bit Code LINK
Using Compaq C++ for OpenVMS Alpha; Chapter 9, Using 64-bit Address Space LINK
OpenVMS System Services Reference Manual, Preface, System Services Support for OpenVMS Alpha 64-bit Addressing LINK

LINK zu Pragma POINTER_SIZE

LINK zu Pragma REQUIRED_POINTER_SIZE