To measure the memory used by *Code_Aster*

Summary:
One presents some tools allowing to measure the memory used in the routines FORTRAN or the Python sources.

Contents

1 Important parameters

1.1 Parameters of ASTK

```
Box "Total Memory (Mo)": MTOT
Box "Of which Aster     (Mo)": MJEV
```

MJEV

MJEV is the memory which will be provided to JEVEUX for the allowance of the "dynamic" objects (a dynamic object is an object whose size is higher than the limit fixed in DEBUT/MEMOIRE/DYNAMIQUE=xxx).

MTOT

MTOT is the total memory which will be allocated with the process. In interactive, this parameter is without effect. On the other hand, on the waiter aster, the resource adminitrator (RMS) control which the process does not exceed this limit.

MTOT must be > MJEV

Notice concerning RMS

The memory used by a process is "multiple": data of the user, buffers of input-output, achievable and libraries dynamic,…
The experiment showed that for launching Code_Aster on the waiter aster (version NEW9 beginning 2008), the "assumption of responsibility" of the job: loading of achievable, installation of the environment of execution,… consumes several hundreds of Mo of memory before even being able to start to treat the data of the user.
This is why, the manager of batch (LSF) request with RMS to authorize the process to be consumed:

MEM_RMS = MTOT + 300 Mo

Significance of MTOT

MTOT is thus the quantity of memory allocated with the process for the user's needs:

- A calculation "2 times larger" will require in theory one MTOT double.
- A very small calculation must be able to pass with MTOT | 0

For what is used MTOT ?

It was seen that the memory JEVEUX MJEV was taken on MTOT. The rest (MTOT − MJEV) is available for:

- The space of the variables python command set
- Dynamic allocations made in the code: use of Mumps, PETSc, …

1.2 Parameters of the command line

On the command line of Aster, one can find 2 parameters concerning the management of the memory:

- − mem_jeveux mjevs (MW)
- − mxmemdy mjevd (MW)

These 2 parameters are expressed in Méga- “word” (MW). The “word” being the memory capacity necessary to the storage of one INTEGER FORTRAN (8 bytes on the machines 64 bits like the waiter aster, 4 bytes on the machines 32 bits).

- memjeveux mjevs

  mjevs is obligatory. It is used to dimension the “dynamic” memory JEVEUX. The value transmitted by ASTK is that deduced from MJEV (by changing unit just).

- mxmemdy mjevd

  mjevd is optional. It is used to limit the “dynamic” memory JEVEUX. The dynamic allocation JEVEUX will stop in error <S> as soon as the cumulated length of the dynamic objects present in memory exceeds this limit. For the moment, this value can be transmitted to Code_Aster only by adding the parameter - mxmemdy on the command line. One can do it with ASTK in the panel "STUDY" with the bottom of the window in the zone of seizure "Arguments".
  For example, so that dynamic storage of IWANT that is to say limited to 100 Mo, on a machine "I8" like Bull, it is necessary to write:

```
 - mxmemdy 12.5
```

2 What can one measure? with which tools?

2.1 Figures given by Aster

At the beginning of a calculation, JEVEUX print in the file “message” several information concerning the management of the memory:

```
 MEMORY IMPOSEE FOR JEVEUX:         3145728 BYTES (     3,000 MEGABYTES)
 MEMORY GIVEN BY "MEMDIS":         3145728 BYTES
 MEMORY TAKEN               :         3145728 BYTES (     3,000 MEGABYTES)
 DYNAMIC LIMIT STORAGE   :     1258291200 BYTES (  1200.000 MEGABYTES)
```

In this example, important information is with the first and the last line: static storage JEVEUX is limited to 3 Mo and the dynamic storage is limited to 1200 Mo. By default, one allocates all the objects in dynamic storage and the static storage is limited to 1 Mo. Thus if one asks for 100 Mo of total memory in ASTK, one has 1 Mo on the first line and 99 Mo on the last.

At the end of calculation, JEVEUX print other information concerning the use of the dynamic storage:

```
STATISTICS CONCERNING THE DYNAMIC ALLOCATION:
     MAXIMUM SIZE CUMULEE                    577  Mo,
     OF WHICH                        3  Mo FOR ZONE GEREE BY JEVEUX.
     SIZE CUMULEE LIBEREE                   1173  Mo.
     FULL NUMBER OF ALLOWANCES  :                    487.
     FULL NUMBER OF RELEASES:                   487.
                          0  CALLS TO THE MECHANISM OF RELEASE.
     SIZE MEMORY CUMULEE RECUPEREE:                      0  Mo.
```

In the preceding example, one can realize (a posteriori) that dynamic storage allotted to JEVEUX (1200 Mo) is oversize: With 577 Mo, calculation would have gone to the end without starting the “mechanism of release”. It is probable that this calculation can be made with less memory than 577 Mo.

2.2 The Unix order “top”

The order Unix top gives information (refreshed with regular interval) about the course of a process.

3 columns VIRT, LMBO, SHR relate to the use of the memory.

VIRT : total of the virtual memory used by the process.
LMBO : memory “resident” (not-swapped)
SHR : “shared” memory

But of other parameters are accessible: DATED, SWAP,…

For more details, to consult top man

2.2.1 The filing system /proc

The system UNIX create for each process of number xxxxx, a repertoire (temporary) of name /proc/xxxxx.

In this repertoire, one finds in particular the file status. This file, updated periodically by the system gives information about the use of the memory by the process. The interesting items have a name starting with VM (Virtual Memory).

One will find for example:

```
VmPeak:   174912 KB
VmSize:   168384 KB
VmLck:         0 KB
VmHWM:     33728 KB
VmRSS:     33664 KB
VmData:   159424 KB
VmStk:      1344 KB
VmExe:       384 KB
VmLib:      3520 KB
VmPTE:       704 KB
```

A small experiment on the machine Bull showed that 3 parameters:

```
VmSize
VmRSS
VmData
```

incremented themselves (and désincrémentaient themselves) after the ALLOCATE/DEALLOCATE of FORTRAN 90.
They can thus be used “to follow” in time the sum of the variables allocated dynamically by the program.

For more information: http://okki666.free.fr/docmaster/articles/linux070.htm

2.3 To measure the use of the memory in a Fortan program

Since version 9.3.11, one knows after each order the instantaneous consumption of JEVEUX, but also the peak reaches up to that point:

```
#  USE OF MEMORY JEVEUX
#  - DYNAMIC STORAGE CONSOMMEE: 112.76 Mo (MAXIMUM REACHED: 558.45 Mo)
```

In addition, an indication interesting to look at the end of the execution is the necessary minimal memory to make turn calculation (value MAXIMUM REACHED MEMORY UTILISEE) :

```
#     - MEMORY UTILISEE:    7.54 Mo (MAXIMUM REACHED:   110.47 Mo)
```

2.4 To measure the use of the memory in a program Python

On the Web, a small piece of code Python was found (see below) which exploits the /proc/xxxxx/status file presented to the preceding paragraph.

In a program Python, if one wishes to know the consumption of memory of a piece of code, one can make:

```
mav=memory ()
…
end of code to be instrumented
…
map=memory ()
print "Accroissment of the memory used (in bytes): ", map - mav


#-------------------------------------------------------------------------------
# Script Python to measure the core use:
#-------------------------------------------------------------------------------
importation bone

_proc_status = '/proc/%d/status' % os.getpid ()

_scale = {'KB': 1024.0, 'mB': 1024.0*1024.0,
          'KB': 1024.0, 'MB': 1024.0*1024.0}

def _VmB (VmKey):
    ''' Private.
    '''
    total _proc_status, _scale
     # get pseudonym slips by  /proc/ <pid>/status
    try:
        T = open (_proc_status)
        v = t.read ()
        t.close ()
    except:
        return 0.0  # not-Linux?
     # get VmKey line e.g. 'VmRSS:  9999  KB \ N… '
    I = v.index (VmKey)
    v = v [I:] .split (Nun, 3)  # whitespace
    yew len (v) < 3:
        return 0.0  # invalid format?
```

```
     # been worth convert VM to bytes
    return float (v [1]) * _scale [v [2]]


def memory (since=0.0):
    ''' Return memory use in bytes.
    '''
    return _VmB ('VmSize: ') - since


def resident (since=0.0):
    ''' Return resides memory use in bytes.
    '''
    return _VmB ('VmRSS: ') - since


def stacksize (since=0.0):
    ''' Return stack size in bytes.
    '''
    return _VmB ('VmStk: ') - since
#--------------------------------------------------------------------------------
```