

LAUNCHER is ShareWare from GreenHouse Software & Consulting.

=====
== LAUNCHER is ShareWare from GreenHouse Software & Consulting. ==
=====

Version 300: All executable files, ending with "n" and having file code 700, are native programs. Process_Create_ as well as NewProcess are supported now. The library supports code 100 as well as code 700 type programs.

Version 310: Some programs propagate a CPU number for the new process to be created. This can now be ignored by Launcher, allowing a better load distribution.
03Jul2000

*** To make this version run, the existing GHSLDATA file
*** has to be purged!

Version 400: In case Launcher has to start a PATHMON process, it does NOT change any CPU number: PATHMON gets its CPU and backup CPU always defined, and a load balance is not possible.

Version 500: To make the Round Robbin functionality more visible, a new dialog has been introduced in CPUMASK.

Version 510: Some programs, running as NonStop process pairs, get the backup CPU defined by an interaction via a command interpreter. A typical example is PATHMON, which gets the primary CPU defined when started, and PATHCOM, which tells the single PATHMON process where to create its backup CPU. LAUNCHER does not know about this, nor does it have a chance to get to this information, before the primary process is started.

To avoid conflicts, this version of LAUNCHER supports an EDIT type list named GHSLPROG, which has to reside in the same location as LAUNCHER. This file contains the program file names of those programs, which should not be controlled by LAUNCHER.

To make the file entries suitable for the two process creation procedure calls
- NewProcess and
- Process_Create_
the entries have to follow this rule:
- all points in a file name have to be replaced by asterisks.
- the node definition has to be replaced by an asterisk.

e.g.: \BEECH.\$SYSTEM.SYSTEM.PATHMON has to be defined as
\$SYSTEM\$SYSTEM*\$PATHMON

LAUNCHER check the last modification time stamp each time it is activated, and re-reads GHSLPROG, when necessary: A restart to activate a changed GHSLPROG is NOT required!

LAUNCHER takes the first 100 entries from GHSLPROG into account. In case you need more - which is very unlikely - please let me know.

Version 511: Program file names in GHSLPROG now can be in external format: [\node.].\$vol.subvol.file. Wildcard support is OK.

Version 512: Program file names in GHSLPROG now can have the prefix
04Jun2003 keyword: ALWAYS.
This causes the launcher library to ALWAYS optimize the process, created from the matching file name.

=====

Hi,

This is a library, that can be bound into programs, that create new processes, like TACL, TELSERV, LISTNER etc. It directs the program, to which it is bound, to automatically create new processes in various CPUs of the system, instead of simply using the CPU where the creator is running is.

Wouldn't it be nice to have a distribution mechanism within e.g. TELSERV, forcing it to use all available CPUs for starting the resources?

The same is true for a TACL: Each program, created by a TACL through the RUN-command, is started in the same CPU where TACL is running, except a CPU is specified in the RUN-command ([run] program /CPU n/), or \$CMON is involved.

The library supports three distribution mechanisms:

1. No distribution is done.
2. 'Wrap Around' method:
The processes are started on all configured CPUs in the system, independent of the current CPU load.
3. 'Least Busy' method:
A processes is started on the CPU with the smallest CPU busy value.

The CPUs, available to the distribution mechanisms, can be defined through a CPU mask.

Distribution method 1 (no distribution) does make sense for processes, that are started with the CPU parameter (EDIT/CPU 2,PRI 10/...), or where the backup CPU is supplied by a command interpreter interaction (PATHMON, PATHCOM).

Distribution method 2 (wrap around) does make sense for starting initial TACL processes through TELSERV. This enables a good basic distribution of the interactive users across the system.

Distribution method 3 (least CPU busy) does make sense for starting processes from an interactive TACL: Depending on the current CPU load, the new process is created in the CPU with the 'least busy' time.

To extract all files from LAUNCHER, follow these instructions:

- Upload LAUNCHER in binary mode onto your Tandem system into an empty location (\$vol.subvol), and name the file: LAUNCHER
- Change the file code from 0 to 100:
FUP ALTER LAUNCHER, CODE 100
- Extract the file LAUNCHER by simply running is:
[run] LAUNCHER
and it extracts its contents into the current location.

You'll get these files:

BUSYTST	100	Program to display the 'least busy CPU' of your system
BUSYTSTN	700	
CPUMASK	100	Program to set the CPU mask, and the sample time
CPUMASKN	700	
DOC	101	Documentation, basically this file
GHSLPROG	101	Configuration file
LIB	100	LAUNCHER library
LIBN	700	
LOAD	100	Test program to put load on a CPU
LOADN	700	

Program file code 100 defines a 'normal' object file, while file code 700 defines a native program.

Run the BUSYTST program to display the 'least busy CPU' of your system.

Run time parameters are:

```
[run] BUSYTST [sample time]
```

where <sample time> is an optional value of the time interval, used by BUSYTST, to find the least busy CPU.
When the parameter is omitted, 100 (= 1.00 seconds) is assumed.

A test run looks like this:

```
$GHS1 LAUNCHER 29> busytst
BUSYTST (200) - T9999G03.00 - (21Jan1998) System \BEECH, running NSK G02
Copyright (c) GreenHouse Software & Consulting 1998
*** Press BREAK to stop program ***
Sample Time is: 1.00 seconds
Least busy CPU: 1
Least busy CPU: 1
Least busy CPU: 1
```

```
$GHS1 LAUNCHER 30>
```

To stop the process, press the BREAK key.

Supplying a sample time directs BUSYTST, to check the busy state of the processors in the given time frame, e.g.:

```
$GHS1 LAUNCHER 30> busytst 35
BUSYTST (200) - T9999G03.00 - (21Jan1998) System \BEECH, running NSK G02
Copyright (c) GreenHouse Software & Consulting 1998
*** Press BREAK to stop program ***
Sample Time is: 0.35 seconds
Least busy CPU: 1
Least busy CPU: 1
Least busy CPU: 1
Least busy CPU: 1
Least busy CPU: 1
```

```
$GHS1 LAUNCHER 31>
```

Test the library.

Before you use the library, make sure, it works as expected. Here is a small 'cook book', describing the binding to a TACL program:

1. Put the files:

- CPUMASK
- GHSLPROG
- LIB

into the same subvol (this is to make life easier).

Secure CPUMASK, that only authorized persons can run it.

Secure LIB to "OONO" and give it to the same owner as CPUMASK.

Secure GHSLPROG to "NOOO".

2. Edit the GHSLPROG file, and add those program file names, which should be excluded from the optimization.
The file comes with a default entry of \$SYSTEM.SYSTEM.PATHMON.
Also add those programs, which ALWAYS should be optimized.
The default entry is: \$SYSTEM.SYS*.FTPSERV
3. Run the CPUMASK program and configure the CPUs, available to the distribution mechanism, optional the sample time, and the distribution method.

CPUMASK:

The CPU masks defines the CPUs, which can be used by LAUNCHER.
A one (1) in the mask defines an available CPU, while a zero(0) defines a not available CPU.

The default value is: 1111111111111111
 ^ ^
 | + = CPU 15
 + = CPU 0

IGNORECPU:

Some programs propagate a CPU for the process to be created.
Launcher can be directed to ignore this.
Default is: The propagated CPU number is used, LAUNCHER does NOT compute the CPU to be used.

SAMPLETIME:

The sample time specifies the time interval that is used by LAUNCHER, to find the CPU with the least busy CPU time value.
Sample time values are in 1/100 seconds.

Valid values are:

- 0 = No CPU busy check is performed, but a 'wrap around' method is enforced.
- 1 - 500 = Time interval between two measures to find the least busy CPU (1 = 0.01 seconds, 500 = 5.00 seconds)

The default value is: 0 = Wrap around

MODE:

When in distribution mode, LAUNCHER supports two methods:
- Round Robbin (default)
- Least Busy
To force LAUNCHER to use the Round Robbin method, MODE has to be set to "RR".
To force the Last Busy method, SAMPLETIME has to be > 0 and MODE has to be set to "LB".

In case GHSLDATA does NOT exist, it automatically becomes created!

Here is a sample CPUMASK session:

```
$GHS1 LAUNCHER 21> cpumask
CPUMASK (400) - T7172G06 - (10Jun2002) System \BEECH, running NSK G06
Copyright (c) GreenHouse Software & Consulting 1997-1998,2000,2002
File \BEECH.$GHS1.LAUNCHER.GHSLData does not exist
Do you want to create it (y/[N])? y
File \BEECH.$GHS1.LAUNCHER.GHSLData created
File \BEECH.$GHS1.LAUNCHER.GHSLData opened
File \BEECH.$GHS1.LAUNCHER.GHSLData initialized

Current CPU Mask: 1111111111111111
Do you want to change it (y/[N]): y          <<<---
CPU Mask in binary (e.g. 1100000000000000): 11  <<<---
```

```

CPU Mask is set to: 1100000000000000
OK (y/[N]): y <<<----
CPU Mask set to: 1100000000000000

Current ignore CPU: supplied CPU is used
Do you want to change it (y/[N]): n <<<----
Ignore CPU NOT changed

Current Sample Time: 0
Do you want to change it (y/[N]): y <<<----
CPU sample time in .01 seconds: 35 <<<----
Sample Time is set to: 35
OK (y/[N]): y <<<----
Sample Time set to: 35

Current Mode is: Round Robbin
Do you want to change it (y/[N]): y <<<----
Mode (LB/RR): lb <<<----
Optimization mode set to: Least Busy
OK (y/[N]): y <<<----
Optimization set to: Least Busy

```

\$GHS1 LAUNCHER 22>

The sample time is in milliseconds (1/100 of a second). It defines the time frame, used by LAUNCHER, to find the least busy CPU, defined by the CPU Mask.

A MODE of "LB" and a sample time value between 1 and 500 causes LAUNCHER to find the CPU with the smallest CPU-busy value. The bigger the sample time value, the more accurate the result, BUT: The longer it takes to start a new process!!!
A good value is between 10 and 100 (= 0.10 .. 1.00 seconds)
(GreenHouse uses values between 35 and 45).

The CPUMASK program can be used any time to change the mask, and/or to ignore a propagated CPU and/or to change the sample time value. A change takes effect immediately!

CPUMASK can be started with the following parameter:

```
[run] CPUMASK [\system.[$vol.[subvol]]]
```

where the start-up parameter defines the location of a GHSLData file to access.

4. Create a copy of TACL in the actual \$SYSTEM.SYSnn and name it e.g. TACLGHS
FUP DUP \$SYSTEM.SYSnn.TACL,\$SYSTEM.SYSnn.TACLGHS,SAVEALL
This ensures, that all TACL required files are where TACL expects them.
5. Start the copy of TACL with the following parameter:

```
[run] TACLGHS/LIB $vol.subvol.LIB/
```

You get a TACL prompt.
6. Logon to this TACL
7. At the TACL prompt, type in: PMSG ON. This causes TACL to display the PIN of newly created processes.

8. Create a new process, and watch the PIN:

```
$GHS1 LAUNCHER 4> pmsg on
$GHS1 LAUNCHER 5> edit
PID: 0,25 $SYSTEM.SYSTEM.EDIT<<<<<<<<
TEXT EDITOR - T9601D20 - (01JUN93)
*EOF!
```

- stop it, and re-create it:

```
STOPPED: 0,25
CPU time: 0:00:00.004
$GHS1 LAUNCHER 6> edit
PID: 1,39 $SYSTEM.SYSTEM.EDIT<<<<<<<
TEXT EDITOR - T9601D20 - (01JUN93)
*EOF!
STOPPED: 1,39
CPU time: 0:00:00.005
$GHS1 LAUNCHER 7>
```

9. In case you are satisfied with the mechanism, you can bind the library to your 'global' TACL, TELSERV, or other programs.

Here is an example for binding TACL in \$SYSTEM.SYSnn:

- a. Rename \$SYSTEM.SYSnn.TACL to \$SYSETM.SYSnn.TACLO
- b. FUP DUP \$SYSTEM.SYSnn.TACLO to \$SYSTEM.SYSnn.TACLN
- c. Bind the library to TACLN:
[run] [\$system.sysnn.]TACLN/LIB \$vol.subvol.LIB/
where \$vol.subvol is the location where the launcher library resides.
- d. Rename \$SYSTEM.SYSnn.TACLN to \$SYSTEM.SYSnn.TACL
Don't worry about all the renames - it is necessary, because TACL normally is OPEN, and you can't bind anything to an open object.

To un-bind the library, just run the program that has the library, with an 'empty' LIB start-up parameter, e.g.:

```
[run][$system.sysnn.]TACL/LIB/
```

Once the library is bound, it 'stays' there for ever.

A new binding has to be done, when the program, to which it is bound, is replaced with a new object.

Dependencies:

Keep the following files in the same location (\$vol.subvol):

- CPUMASK
- GHSLPROG
- LIB

The program CPUMASK creates and initializes a file names GHSLDATA, which is used, and modified, by the library, residing in the same location, at run time.

To run different libraries with different

- CPU masks
 - distribution mechanisms
 - = wrap around
 - = least CPU busy
 - programs to be excluded from the optimization
- create different locations with the three mentioned files.

Restrictions: The library runs on GUARDIAN Dxx and Gxx.

It replaces the procedure calls:

- Process_Create_
- NewProcess

The library can be bound to programs with file code 100 as well as file code 700.

To check programs for attached libraries, use the SHOWLIB FreeWare Tool.
To easily bind/de-bind a library, use the BINDLIB FreeWare Tool.
Both tools are available from www.GreenHouse.de.

GreenHouse Software & Consulting
Ingenieurbuero Karl-Heinz Weber
Heintoichstrasse 12
D-45711 Datteln/Horneburg
Germany

Phone: +49 2363 72566
FAX: +49 2363 66106
E-Mail: Carl.Weber@GreenHouse.de