



IBM Advanced Technical Support

# Managing Workloads on a Uniprocessor

Linda August  
March 8, 2007  
NCACMG

© 2007 IBM Corporation

# Trademarks

AIX*	IBM eServer	z/VM*
CICS*	IBM logo*	zSeries*
DB2*	IMS	
DB2 Connect	On Demand Business logo	
DB2 Universal Database	Parallel Sysplex*	
DRDA*	System z	
FICON*	System z9	
GDPS*	WebSphere*	
HiperSockets	z/Architecture	
IBM*	z/OS*	

\* Registered trademarks of IBM Corporation

The following are trademarks or registered trademarks of other companies

Intel is a trademark of the Intel Corporation in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Java and all Java-related trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries.

Microsoft, Windows and Windows NT are registered trademarks of Microsoft Corporation.

UNIX is a registered trademark of The Open Group in the United States and other countries.

\* All other products may be trademarks or registered trademarks of their respective companies.

## Agenda



Well, the engines on our new processor are so big that some of our LPARs turn out to have short CPs based on their weights.

But I heard that we should not run a uni LPAR. Can we manage it?



- **Changes in ready dispatcher to help SRM run work more efficiently in general**
- **WLM policy and MVS settings to help manage uniprocessor environment**
- **Managing CPU-intensive work**
- **z/OS functions which override WLM assigned dispatching priorities**

# "Small LPARs" and "short engines"

- A "small LPAR" is one which is guaranteed less than 50% engine based on its assigned weight when processor runs at capacity
- "Short engines" can occur when
  - ▶ Processor is very busy and
  - ▶ Number of logical CPs assigned is more than the number allowed based on an LPAR's weight

LPAR	WEIGHT	#CPs ASSIGNED	#CPs ALLOWED	% LP on CP
SYSA	400	7	4	57%
SYSB	350	4	3.5	87.5%
SYSC	200	4	2	50%
SYSD	50	4	.5	12.5%

2094-710

SHOULD ASSIGN  
1 LOGICAL CP  
BASED ON WEIGHT

- Results in 2 dispatchers managing work on LPAR
  - ▶ MVS dispatcher based on the WLM policy goals
    - It is not aware that PR/SM is taking an engine away
  - ▶ PR/SM dispatcher as it spreads the CPU allowed across all logical CPs assigned (%LP on CP)
    - It is not aware of importance of work running when it takes an engine away
- RMF CPU Activity report
  - ▶ Monitor LPAR versus MVS Busy
- Impacts
  - ▶ Single tasking workloads
  - ▶ Work on larger LPARs that need resources held by small LPARs

# Changes in MVS that positively impact work on a uniprocessor: reduced preemption

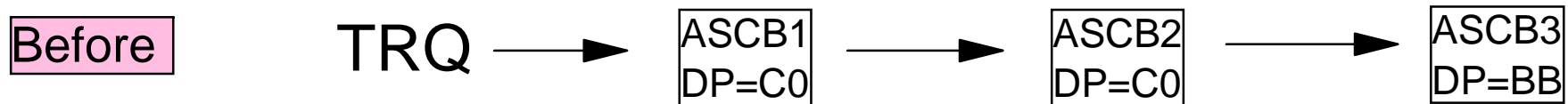
## ■ Prior to MVS/SP 3.1

- ▶ When an event occurred that would cause work to be made ready, MVS compared the priority of the newly ready address space against the priority of all interruptible address spaces on all processors and preempted the lowest priority process.

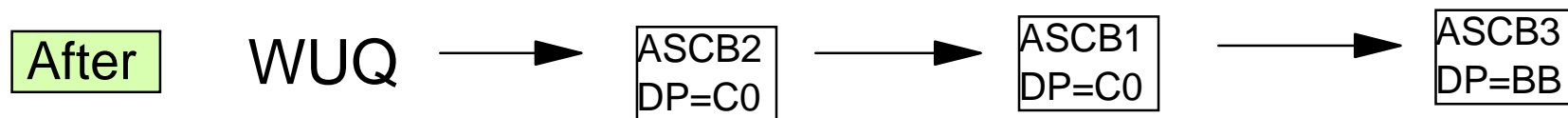
## ■ Reduced preemption in MVS/SP 3.1

- ▶ Works on basis that it is better to allow work to progress until it gives up an engine or reaches the end of its time slice than to immediately interrupt
- ▶ Avoids interrupting running tasks
- ▶ Expects normal work to release CPU soon
- ▶ Employs timed preemptions just in case...

## Changes in MVS that positively impact work on a uniprocessor: fair share dispatching



1. Dispatcher chooses ASCB1, first element on TRQ
2. Task loops so eventually incurs time slice interrupt but
3. ASCB1 still ready so requeued to front of the TRQ and
4. Selected again as long as it remains ready so ASCB2 has same DP as ASCB1 but gets no CPU



With MVS/SP 5.1

1. Dispatcher chooses ASCB1, first element on WUQ (Work Element Q)
2. Task loops so eventually incurs time slice interrupt
3. ASCB1 still ready but
4. Requeued to back of the WUQ within group of address spaces having same DP
5. Dispatcher selects first element on WUQ, ASCB2

## WLM goal mode changed the way work is managed

- **WLM dynamically manages workload priorities**
  - ▶ Replaces IEAIPS PARMLIB member and the DP= parameter
  - ▶ Only SYSTEM work runs at specific dispatching priority
- **Manage by importance and goals**
- **Remember: all work in same service class period runs at the same DP**
- **Assumes you understand the relative importance of your business goals when resources are constrained**

**And dispatching priority is key to managing a uniprocessor environment**

# WLM policy recommendations to manage a uniprocessor environment

- **Use of importance values**
  - ▶ Need to distinguish who gets the CP first when the system is busy
  - ▶ Define a clear priority order using all the importance levels
- **Achievable realistic goals**
  - ▶ Use PIs at peak intervals as basis for setting policy goals
  - ▶ Want PIs for important work to be close to 1
- **Classify all your work**
  - ▶ Define unique default service classes and report classes for all subsystem types
    - Monitor for any activity and then classify the new work
  - ▶ Use report classes to easily identify the CPU "hogs"
- **Watch for total amount of CPU for work in SYSTEM and SYSSTC service classes**
  - ▶ Move large consumers to managed service class if possible
- **Watch what runs in discretionary**
  - ▶ It can hold locks/latches and not get CPU cycles
  - ▶ Classify any work you find running in SYSOTHER
- **SMF 99 records show WLM DP decisions**

## Managing CPU-intensive work

- **Applications looping due to programming errors**
- **High priority work in SYSTEM and SYSSTC running at fixed DP of FF and FE have some functions which can become CPU intensive**
- **Other goal-managed high importance work**
  - ▶ DB2 query work, JAVA garbage collection, CICS/VSAM

## What we tested to show WLM managing looping applications

- **z/OS 1.7 system on z9**
- **Varied all engines off except 1**
- **Reset WLM policy before each run**
  - ▶ Wanted WLM to collect history from scratch each time
- **Dumped and switched SMF data sets before each run**
  - ▶ Kept data collection cleaner
- **Set RMF and SMF intervals to 1 minute in hopes of capturing some data for post processing**
  - ▶ Also relied on SDSF screen captures
- **Defined soaker jobs**
  - ▶ Mostly CPU plus some rummaging through storage and a little IO
- **Created job streams of multiple jobs in various service classes to see how they ran**

# WLM service policy for test scenarios

SCLASS	IMPORTANCE	VELOCITY GOAL
BAT_HI	2	40
BAT_MED	3	30
BAT_LO	4	20
BAT_VLOW	5	5
BATDISC		Discretionary

## INITIATORS

ID	Status	Classes
1	INACTIVE	HMLABJ
2	INACTIVE	HMLABJ
3	INACTIVE	HMLABJ
4	INACTIVE	HMLABJ
5	INACTIVE	HMLABJ
6	INACTIVE	HMLABJ
7	INACTIVE	HMLABJ
8	INACTIVE	HMLABJ
9	INACTIVE	HML
10	INACTIVE	HML
11	INACTIVE	HM
12	INACTIVE	HM
13	INACTIVE	HM
14	INACTIVE	HM
15	INACTIVE	H
16	INACTIVE	H

```

Subsystem Type JES - JES classification rules
Classification:
  Default service class is BAT_MED
  There is no default report class.
    Qualifier  Qualifier  Service
    # type     name         Class
    - - - - - - - - - - - - - - - - - - - -
    1 TC       L           BAT_VLOW
    1 TC       A           BATDISC
    1 TC       J           BAT_LO
    1 TC       M           BAT_MED
    1 TC       H           BAT_HI
    
```

## "Looping" a.k.a. CPU-intensive work

- **Ran multiple one-step soaker jobs at the same time on a uniprocessor**
  - ▶ All in same service class then
  - ▶ Some in high importance and some in low importance
  
- **SOAKER**
  - ▶ GETMAINS 10 1 Mb chunks, then loops 600 times and each time does
    - "ripple" through the tables then opens a file, writes one record, closes file
  
- **SOAK100**
  - ▶ GETMAINS 20 1 Mb chunks, then loops 100 times and each time does
    - "ripple" through the tables
    - opens a file, writes one record, closes file 10 times
  
- **BIGSOAK**
  - ▶ Very tight loop that runs forever
  
- **RMFPP**
  - ▶ Runs the RMF post processor ALL, CPU, CHAN, DEVICE reports

# Looping Results (1 of 2)

6 SOAKERS all in same service class: 5 identical, LGASN was RMFPP

SDSF	DA	SYSC	SYSC	PAG	0	CPU/L/Z	100/	99/	0	LINE	1-6
JOBNAME	StepName	SrvClass	DP	C	Workload	JobID	CPU%	SIO			
LGAS1	HIIMP	BAT_HI	F9	H	BAT_WKL	JOB21231	19.18	56.61			
LGASN	HIIMP	BAT_HI	F9	H	BAT_WKL	JOB21232	3.05	653.56			
LGAS2	HIIMP	BAT_HI	F9	H	BAT_WKL	JOB21233	19.14	56.61			
LGAS3	HIIMP	BAT_HI	F9	H	BAT_WKL	JOB21234	18.95	61.75			
LGAS4	HIIMP	BAT_HI	F9	H	BAT_WKL	JOB21235	19.05	56.61			
LGAS5	HIIMP	BAT_HI	F9	H	BAT_WKL	JOB21236	19.14	56.61			

HI IMPORTANCE  
HAD EQUAL  
ACCESS  
TO CPU

7 jobs: 6 identical soakers, LGAHIS1 had a bit more IO

SDSF	DA	SYSC	SYSC	PAG	0	CPU/L/Z	100/	99/	0	LINE	1-7
JOBNAME	StepName	SrvClass	DP	C	Workload	JobID	CPU%	SIO			
LGLS1	VLOW1	BAT_VLOW	F1	L	BAT_WKL	JOB21268	0.00	0.00			
LGLS2	VLOW2	BAT_VLOW	F1	L	BAT_WKL	JOB21269	0.00	0.00			
LGLS3	LOW1	BAT_LO	F3	J	BAT_WKL	JOB21270	0.00	0.00			
LGHIS1	HIIMP1	BAT_HI	F9	H	BAT_WKL	JOB21271	21.87	50.97			
LGHIS2	HIIMP2	BAT_HI	F9	H	BAT_WKL	JOB21272	24.98	20.39			
LGHIS3	HIIMP3	BAT_HI	F9	H	BAT_WKL	JOB21273	25.49	19.11			
LGHIS4	HIIMP4	BAT_HI	F9	H	BAT_WKL	JOB21274	24.20	20.39			

LO IMPORTANCE  
HAD NO  
ACCESS  
TO CPU

HI IMPORTANCE  
HAD EQUAL  
ACCESS  
TO CPU

# Looping Results (2 of 2)

Introduced a tight looper, BIGSOAK, in importance 5 BAT\_VLOW

SDSF DA SYSC SYSC PAG 0 CPU/L/Z 100/ 99/ 0 LINE 1-8	JOBNAME	StepName	SrvClass	DP	C	Workload	JobID	CPU%	SIO
	LGLS1	BIGSOAK	BAT_VLOW	EF	L	BAT_WKL	JOB21282	0.00	0.00
	LGLS2	SOAKVL1	BAT_VLOW	EF	L	BAT_WKL	JOB21283	0.00	0.00
	LGLS3	SOAKL1	BAT_LO	F5	J	BAT_WKL	JOB21284	0.00	0.00
	LGMS1	SOAKM1	BAT_MED	F7	M	BAT_WKL	JOB21285	0.12	0.00
	LGHIS1	HIIMP1	BAT_HI	F9	H	BAT_WKL	JOB21286	32.16	47.02
	LGHIS2	HIIMP2	BAT_HI	F9	H	BAT_WKL	JOB21287	32.52	28.21
	LGHIS3	HIIMP3	BAT_HI	F9	H	BAT_WKL	JOB21288	32.64	28.21
	LGMR1	RMFPPM	BAT_MED	F7	M	BAT_WKL	JOB21289	0.12	0.00
	LGLS1	BIGSOAK	BAT_VLOW	EF	L	BAT_WKL	JOB21282	0.00	0.00
	LGLS2	SOAKVL1	BAT_VLOW	EF	L	BAT_WKL	JOB21283	0.00	0.00
	LGLS3	SOAKL1	BAT_LO	F5	J	BAT_WKL	JOB21284	0.33	0.00
	LGMS1	SOAKM1	BAT_MED	F7	M	BAT_WKL	JOB21285	4.98	11.11
	LGHIS1	HIIMP1	BAT_HI	F9	H	BAT_WKL	JOB21286	89.78	78.82
	LGMR1	RMFPPM	BAT_MED	F7	M	BAT_WKL	JOB21289	1.43	49.44
	LGLS1	BIGSOAK	BAT_VLOW	EF	L	BAT_WKL	JOB21282	0.06	0.00
	LGLS2	SOAKVL1	BAT_VLOW	EF	L	BAT_WKL	JOB21283	0.06	0.00
	LGLS3	SOAKL1	BAT_LO	F5	J	BAT_WKL	JOB21284	2.05	1.19
	LGMS1	SOAKM1	BAT_MED	F7	M	BAT_WKL	JOB21285	64.95	50.36
	LGMR1	RMFPPM	BAT_MED	F7	M	BAT_WKL	JOB21289	28.83	314.62

NO IMPACT TO ONLINE

NO IMPACT TO ONLINE

ONLINE FINISHED  
BIGSOAK FINALLY  
GOT SOME  
CPU CYCLES

## Impact of looping work on other work in the system

- **Observation: Looping jobs only impacted lower priority work**
  - ▶ If in high importance sclass then it dominated the engine
  - ▶ If in low importance sclass then had difficulty getting cycles
    - Even tight loop shut out when higher importance work wanted engine
- **Observation: Equal priority work shared the CPU capacity**
- **Recommendations to manage CPU-intensive swappable work**
  - ▶ Use IEFUJV JES exit to limit CPU (Set TIME=)
  - ▶ Use strong JES class standards
  - ▶ Limit number of initiators for each JES class
  - ▶ Use the reset command E jobname,QUIESCE
  - ▶ Define multi-period service classes so CPU-intensive work will drop to service class period with lower importance or discretionary goal
- **Recommendations to manage CPU-intensive non-swappable work**
  - ▶ Define "limit" service class with discretionary goal and assign to resource group defined with minimum of 0 and maximum of 1
  - ▶ Use E jobname,SRVCLASS=LIMIT to assign looping work to service class LIMIT
- **Recommendation: Define a special TSO userid to the SYSSTC service class to get access to SDSF to cancel looping work**

# SYSTEM and high importance goal-managed CPU-intensive work

- **SYSTEM runs at FF and SYSSTC runs at FE**
- **Some system functions can become CPU intensive**
  - ▶ Job initiation functions
  - ▶ High volume SRBs
  - ▶ Contention for spin locks
  - ▶ System and network automation
    - Starting with Netview 1.4 you can separate management of Netview system automation tasks from network tasks via exploitation of enclaves
  - ▶ Dumps
  - ▶ Large CPU consumer incorrectly classified to SYSSTC
  - ▶ Page stealing
  - ▶ IRLM processing
- **Other goal-managed high importance work such as**
  - ▶ DB2 query work, JAVA garbage collection, CICS/VSAM
- **Minimize time these functions need the CP**
  - ▶ Depends on the subsystem or application
- **Note that even batch work which normally runs at lower importance than online can impact online response times**
  - ▶ Some job initiation functions run at SYSSTC dispatching priority

## z/OS 1.5 introduced importance-based Initiator dispatch priority control

### ■ Prior to z/OS 1.5

- ▶ work running under the Initiator prior to SYSEVENT JOBSELECT processing ran in SYSSTC DP of 254

### ■ What runs at the initiator DP?

- ▶ EXIT routines for SAF, SMS ACS, data set allocation, job accounting (IEFUJI), and starting and stopping of WLM-managed initiators
- ▶ Applies to JES, APPC, and OMVS initiators

### ■ Especially important to a uniprocessor

- ▶ What if hundreds of jobs were dumped into the system at once?
- ▶ How efficient are the ACS routines? How many lines are in them? How many volumes in a storage group?
- ▶ How many DD statements in the job?
- ▶ How extensive is RACF profile checking?

### ■ WLM controls changed in z/OS 1.5

- ▶ New IEAOPT parameter INITIMP to control DP of initiators before job execution
- ▶ Goal: ensure job initiation does not impact online workloads

## INITIMP parameter in IEAOPTxxx PARMLIB member

- **Values: 0, 1, 2, 3, E**
  - ▶ Default 0
- **How it works**
  - ▶ **0 value**- works the same as before with DP of 254 (SYSSTC)
  - ▶ **1,2,3 value** - Initiator DP has to be lower than DP for CPU critical work with same or higher importance level. If no service class with the CPU critical attribute and same or higher importance level is defined in the WLM policy, the DP is calculated in same way as for INITIMP=E
  - ▶ **E value** - initiator DP will be calculated in same way as ENQ promotion DP. The DP is calculated dynamically to ensure access to the processor. It should not impact high importance work, however no guarantee that CPU critical work will always have a higher DP
- **Recommended setting for uniprocessors: INITIMP=E**
- **Monitor SMF30ICU (TCB) and SMF30ISB (SRB) time in type 30 subtype 4 step end records**
  - ▶ CPU time spent running under the initiator

## Functions which override the WLM policy

- **Overrides let lower importance work run for some period and can impact higher importance work**
- **Areas**
  - ▶ ENQ contention
    - Lower importance work not getting CPU and holding an ENQ on a resource needed by higher importance work
  - ▶ Discretionary goal management (DGM)
    - WLM raising the DP of discretionary work above higher importance work
  - ▶ Resource groups
  - ▶ Application design
    - High importance work that is processing a queue of some kind and goes CPU-intensive at times
      - Can design be changed so the application can process the queue in chunks and take a breather?

# ENQ management

- **What changed in z/OS 1.3 to manage ENQ contention**
  - ▶ Objective: Ensure that critical work not blocked by resources because blocker is swapped out or not receiving services
  - ▶ A/S or enclave is promoted in terms of DP when another A/S space or enclave wants that resource in hopes that the holder will finish and release the resource faster than if it had the lower DP
  - ▶ A/S will not get swapped out while being ENQ promoted
  - ▶ Length of time it runs at higher DP governed by ERV value
  - ▶ ENQ promotion DP dynamically set by WLM every 10 seconds to give access to CPU where 40% of the capacity is used
  - ▶ A/S is promoted again by SRM for every contention indication by resource manager
- **ERV (enqueue residence value) parameter in IEAOPTxx**
  - ▶ CPU service units that an A/S can accumulate when it is promoted before being set back to lower DP
  - ▶ Default is 500 (small when the SU/Sec on 2084-301 is over 21,000)
  - ▶ **Recommendation: Increase ERV value to allow a lower priority unit of work holding an ENQ to receive some CPU service so hopefully it will complete and release the ENQ**
- **SMF type 30 SMF30CEP and SMF30CEPI fields contain the CPU time for A/S or job while ENQ promoted**

## ENQ management in action

### ■ What we tested - GRS-managed ENQs, DEQs

- ▶ Same set up as with looping test
- ▶ Increased ERV value so we could see impact of ENQ management
- ▶ Submitted a very low priority job, HOLDER1, that allocated data sets with DISP=OLD and executed SOAKER so it held the ENQs
- ▶ Then submitted high priority jobs that wanted the data sets (WAITERx)
- ▶ Watched and waited

### ■ Results

- ▶ At one point observed the DP of the importance 5 velocity 5 work raised to same DP as importance 2 velocity 40 work with CPU%>0 in SDSF

# Results of adjusting ERV value

DA	SYSC	SYSC	PAG	0	CPU/L/Z	100/	99/	0	LINE	1-10	(10)
JOBNAME	StepName	SrvClass	DP	C	Workload	JobID			CPU%	SIO	
LGGH1	HOLDER1	BAT_VLOW	F9	L	BAT_WKL	JOB17486			13.26	80.97	
LGGH2	HOLDER2	BAT_VLOW	F9	L	BAT_WKL	JOB17487			13.71	80.97	
LGGO1	ONLINE1	BAT_HI	F9	H	BAT_WKL	JOB17488			34.59	24.23	
LGGO2	ONLINE2	BAT_HI	F9	H	BAT_WKL	JOB17489			35.41	16.15	
LGGO3	ONLINE3	BAT_MED	F7	M	BAT_WKL	JOB17490			0.00	0.00	
LGGSM1	MEDIMP	BAT_MED	F7	M	BAT_WKL	JOB17491			0.00	0.00	
LGGSM2	MEDIMP	BAT_MED	F7	M	BAT_WKL	JOB17492			0.00	0.00	
LGGW1	WAITER1	BAT_HI	FF	H	BAT_WKL	JOB17493			0.00	0.00	
LGGW2	WAITER2	BAT_HI	FF	H	BAT_WKL	JOB17494			0.00	0.00	
LGGW3	WAITER3	BAT_HI	FF	H	BAT_WKL	JOB17495			0.00	0.00	

DA	SYSC	SYSC	PAG	0	CPU/L/Z	100/	99/	0	LINE	1-9	(9)
JOBNAME	StepName	SrvClass	DP	C	Workload	JobID			CPU%	SIO	
LGGH1	HOLDER1	BAT_VLOW	F3	L	BAT_WKL	JOB17486			0.00	0.00	
LGGH2	HOLDER2	BAT_VLOW	F3	L	BAT_WKL	JOB17487			0.00	0.00	
LGGO1	ONLINE1	BAT_HI	F9	H	BAT_WKL	JOB17488			87.78	66.31	
LGGO3	ONLINE3	BAT_MED	F7	M	BAT_WKL	JOB17490			4.94	0.00	
LGGSM1	MEDIMP	BAT_MED	F7	M	BAT_WKL	JOB17491			1.10	22.66	
LGGSM2	MEDIMP	BAT_MED	F7	M	BAT_WKL	JOB17492			1.45	19.34	
LGGW1	WAITER1	BAT_HI	FF	H	BAT_WKL	JOB17493			0.00	0.00	
LGGW2	WAITER2	BAT_HI	FF	H	BAT_WKL	JOB17494			0.00	0.00	
LGGW3	WAITER3	BAT_HI	FF	H	BAT_WKL	JOB17495			0.00	0.00	

HOLDERS DP raised to F9, same as BAT\_HI SCLASS

Once each HOLDER got its ENQ promoted CPU, their DP was lowered and the high importance work that was not waiting got the CPU.

# Discretionary Goal Management

## ■ What is it

- ▶ Allows discretionary work to get CPU service when other non-discretionary goal work is overachieving its goal
  - Caps the overachieving work
  - May not be desirable when only one engine
- ▶ Receiver service class period has a discretionary goal
- ▶ Potential donor service class period:
  - Velocity goal  $\leq 30$  or RT goal  $> 1$  minute
  - Any importance
  - PI  $< .7$

## ■ How to control it

- ▶ Define a resource group with null min and max values and assign the potential overachieving service classes to the group
- ▶ Change the goal (velocity  $> 30$  or RT  $< 1$  minute)
- ▶ Tighten your goals so service classes are not overachieving

## Resource Groups can impact higher importance work but can also be used to manage CPU-intensive work

### ■ A resource group defined with a Minimum means:

- ▶ If a resource group is not meeting its minimum capacity ***and*** work in that resource group is not meeting its goals, workload management will attempt to give CPU resource to that work,  
even if it causes more important work to miss its goals.
- ▶ Dispatching priority is increased to give access to CPU resources
- ▶ The minimum capacity setting has no effect when work in the resource group is meeting its goals.
- ▶ WLM will only help discretionary work in a resource group that is not meeting its minimum capacity if it does not cause other work to miss its goals

### ■ A resource group defined with a Maximum can be used to control CPU-intensive non-swappable work

- ▶ Define a limit service class LIMIT with a discretionary goal and assign to resource group with a min of 0 and max of 1
  - Use the command E jobname,SRVCLASS=LIMIT to assign the service class to the job

## Other z/OS controls that impact uniprocessor environments

### ■ **z/OS 1.8 GRS enhancements**

- ▶ SETGRS support to identify LPAR which will be the GRS Contention Notification System
  - Rolled back to z/OS 1.7 in APAR OA11382
- ▶ Ensure that a uniprocessor LPAR member of a sysplex is not selected as the one to control notifications

### ■ **SRM - OA16097**

- ▶ Swap in of very large (6 Gb) logically swapped address space required UIC update to be called to process the address space's frames
- ▶ Max amount of time to stay disabled hard-coded as x'7FFFFFFF' and caused SRM lock to be held for more than 9 seconds
- ▶ Can also add the undocumented IEAOPT parm UICFRAMESCHECKTIME which tells UIC update how many thousands of frames to process in frame queue before quitting. For example, a value of 2 would tell SRM to process 2000 frames. Used where a job owns a lot of dataspace storage.
- ▶ PTF available November 2006

## Other z/OS controls that impact uniprocessor environments

### ■ **OA19677 - JES2 (new - February 1)**

- ▶ NETSERV loops trying to connect on a uniprocessor if its DP is higher than JES2
  - Initial signon "I" record sent to adjacent node
  - NETSERV loops on MTST\_SEND\_ECB
  - Response signon "J" record received must be processed by JES2 before ECB is cleared
- ▶ If JES2 DP lower than NETSERV, JES2 not dispatched so NETSERV loops on a uni
  - MTST\_SEND\_ECB not cleared when send completes
- ▶ **RECOMMENDATION:** Do not run NETSERV address space above JES
  - NJE data transmission is CPU-intensive and can lock out other work running at lower priority

## Other z/OS controls that impact uniprocessor environments

### ■ **New command to assist in diagnosis of DB2 thread hangs**

- ▶ PQ83649 - DB2 - HIPER
- ▶ Little information available to when DB2 hangs without forcing a console dump. No DB2 interface to help DBAs in diagnosis of problem
- ▶ New keyword 'service' added to the display thread command

`-dis thd(*) service(wait)`

- ▶ Command will display all threads suspended for twice the IRLM timeout limit or minimum of 60 seconds
- ▶ Will also try to dynamically boost priority for any workunits holding latches that appear stuck
- ▶ Intended for IBM real-time hang diagnosis

## Use the z/OS Health Checker

- **Goal: ensure settings are correct for z/OS**
  - ▶ Identify potential problems that could impact availability
  - ▶ Avoid outages
  - ▶ Program integrated into z/OS 1.7 base
- **Current checks cover:**
  - ▶ Consoles, GRS, RACF, RRS, RSM, VSM, ASM, Logger, SVC dumps, XCF
  - ▶ New in 1.8: TCP/IP and VTAM
  - ▶ These tasks run at high dispatching priorities
- **New checks continually being added**

# Summary

- **It takes work**
  - ▶ Detailed understanding of workload behavior
  - ▶ Well-tuned WLM policy
  - ▶ IEAOPT parameter settings for ERV and INITIMP
  - ▶ Use of the z/OS Health Checker
  - ▶ Ability to monitor workloads to identify abusers and take action
  - ▶ Emergency TSO userid
- **There are some things you cannot control**
  - ▶ Minimize their likelihood of occurring
- **And you also need to consider**
  - ▶ Impact of "short engines" on workloads
  - ▶ Availability considerations
  - ▶ Impact of small LPARs in big sysplexes
  - ▶ LPAR consolidation - could small LPARs be consolidated into a larger LPAR with a weight guaranteeing at least one engine?
  - ▶ .....

## References

- **IBM White paper WP100925, "Managing CPU-Intensive Work on Uniprocessor LPARs"**
- **SG24-6472 Redbook: System Programmer's Guide to: Workload Manager** revised February, 2007 with chapter on uniprocessors
- **"WLM Knowns & Unknowns" by Jim McCoy, SHARE Boston 2005 Session 2545**
- **IBM Health Checker for z/OS User's Guide: SA22-7994**
- **"MVS Workload Manager Velocity Goals: What You Don't Know Can Hurt You" by John Arwe**  
[www-03.ibm.com/servers/eserver/zseries/zos/wlm/documents/velocity/velocity.html](http://www-03.ibm.com/servers/eserver/zseries/zos/wlm/documents/velocity/velocity.html)