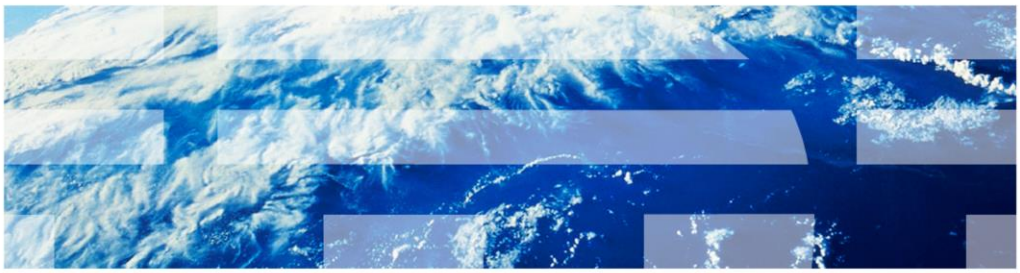
IBM

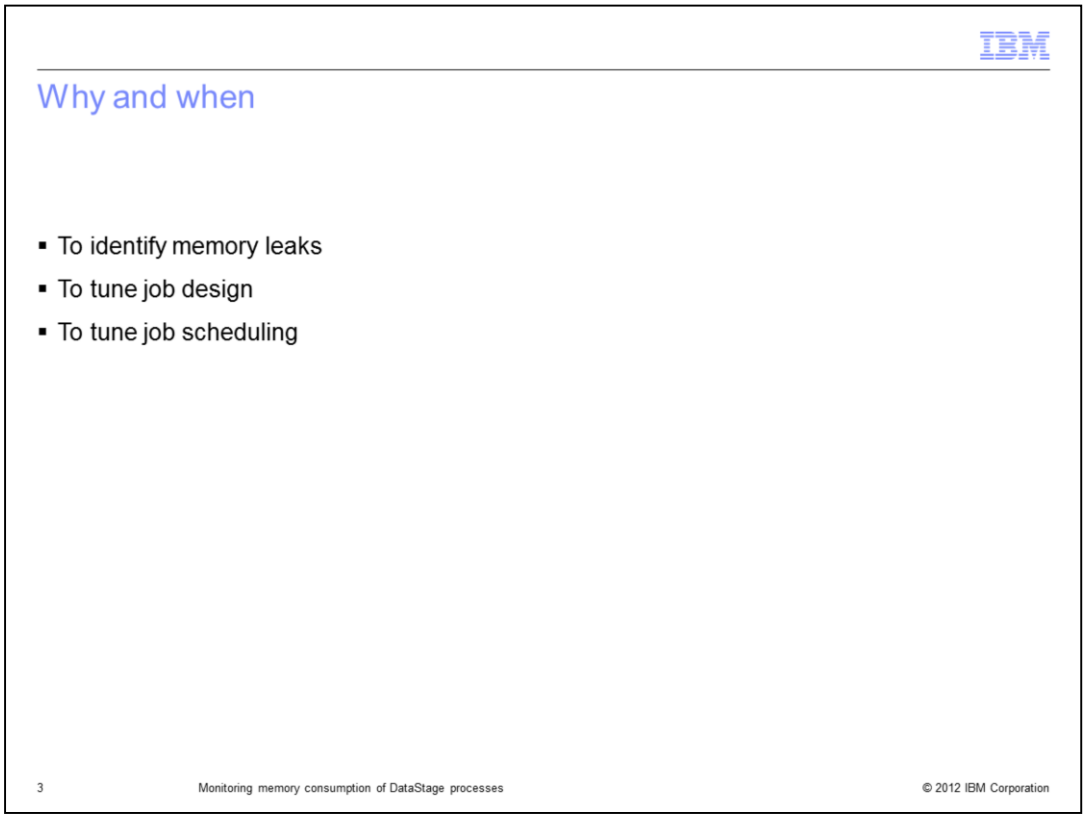
InfoSphere Information Server

Monitoring memory consumption of DataStage processes



This presentation explains how to monitor memory consumption of DataStage® processes during run time.

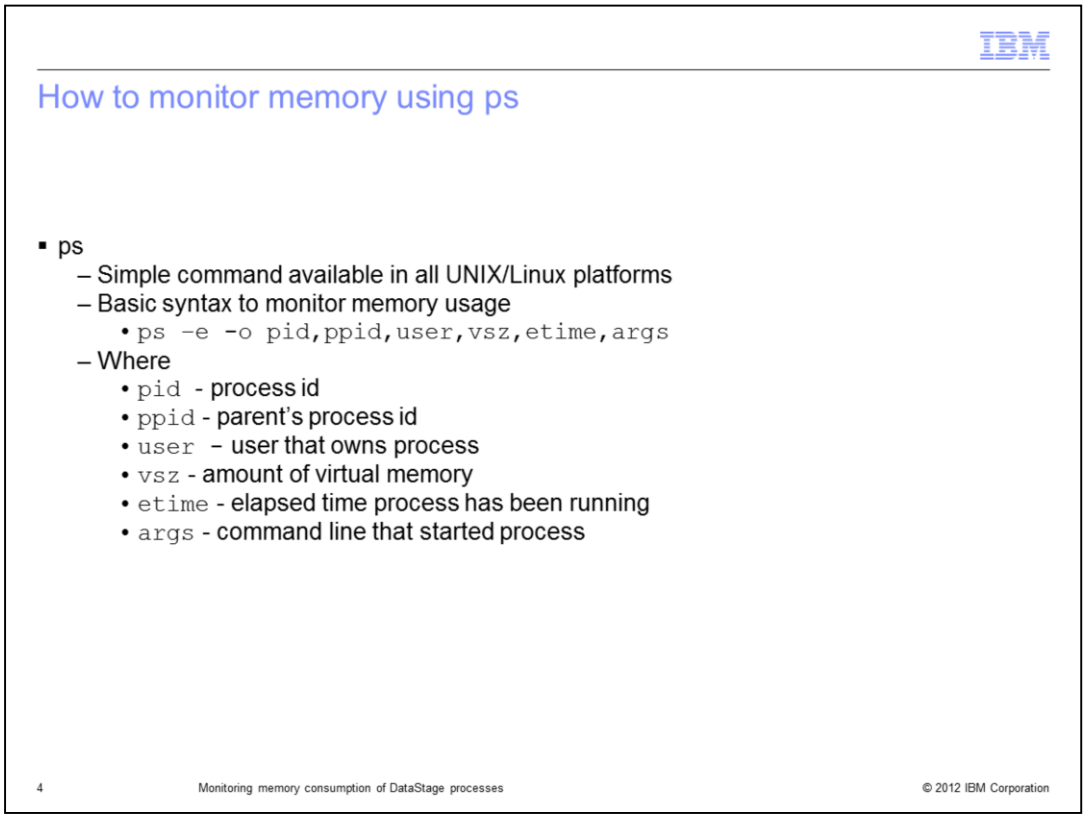
The objectives of this presentation are to explain why and when it is useful to monitor memory usage of DataStage processes and how to monitor this using the ps command. This presentation also explains how to perform this task under different scenarios.



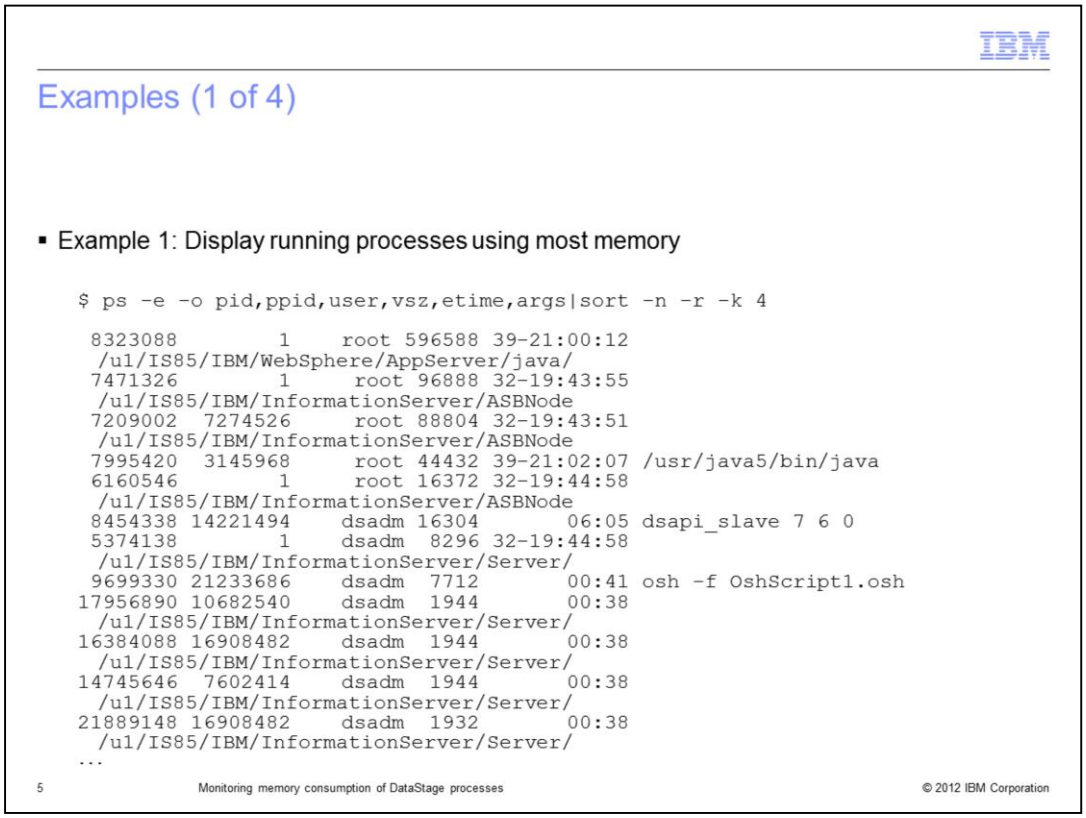
Analyzing memory usage can be useful in several scenarios. Some of the most common scenarios include identifying memory leaks. A memory leak is a type of bug that causes a program to keep increasing its memory usage indefinitely.

Another scenario is to tune a job design. Comparing the amount of memory different job designs consume can help you tune your designs to be more memory efficient.

The last scenario is to tune job scheduling. Collecting memory usage by processes over a period of time can help you organize job scheduling to prevent peaks of memory consumption.



Different operating systems provide different monitoring tools that can collect detailed information. This presentation will use the ps command because it is easy to use and is available on all UNIX and Linux platforms. The syntax to display basic information of all running processes is ps –e -o pid, ppid,user, vsz, etime, args. Where pid is the process id, ppid is the parent's process id, user is the user that owns the process, vsz is the amount of virtual memory used by the process, etime is the elapsed time the process has been running, and args is the command line that started the process.



The example displayed on this slide shows a list of all running processes and sorts them by memory consumption. To do this, use the ps command and pipe the output to the sort command. In the example output on this slide, the fourth column of the ps results shows the amount of virtual memory being used and the processes with the largest amount of memory are listed first. This output was produced on a machine running AIX® 6.1. The format of this output may vary in different operating systems.

Examples (2 of 4)

- Example 2: Continuously capturing memory usage of all osh processes

```
$ while :;do date;ps -e -o pid,ppid,user,vsz,time,args|grep -v grep|grep osh;sleep 1;done

    Tue Jul 31 14:05:07 EDT 2012
    16908508 21233686    dsadm  7312    00:00:00 osh -f OshScript1.osh

    Tue Jul 31 14:05:08 EDT 2012
    16908508 21233686    dsadm  7332    00:00:00 osh -f OshScript1.osh

    Tue Jul 31 14:05:09 EDT 2012
     9371766  9896138    dsadm  1384    00:00:00 /u1/IS85/IBM/InformationServer/Server/
     9896138        1    dsadm   744    00:00:00 /u1/IS85/IBM/InformationServer/Server/
    10813640 14745652    dsadm  1332    00:00:00 /u1/IS85/IBM/InformationServer/Server/
    12189920 14418118    dsadm  1388    00:00:00 /u1/IS85/IBM/InformationServer/Server/
    13369584        1    dsadm   688    00:00:00 /u1/IS85/IBM/InformationServer/Server/
    14352594  9896138    dsadm  1332    00:00:00 /u1/IS85/IBM/InformationServer/Server/
    14418118        1    dsadm   688    00:00:00 /u1/IS85/IBM/InformationServer/Server/
    ...
```

- Press Ctrl + c to stop command

The example displayed on this slide shows how to capture memory of all osh processes in a continuous loop. osh processes are processes created by DataStage parallel jobs. This command is used to monitor all osh processes over a period of time. In this example, the grep command is used to filter processes with the string “osh” but this can be modified if you want to filter processes by something else such as user ID or ppid. This loop is started every second but can also be modified by increasing the value after the sleep command. This command will continue to run until you press Ctrl + c.

Examples (3 of 4)

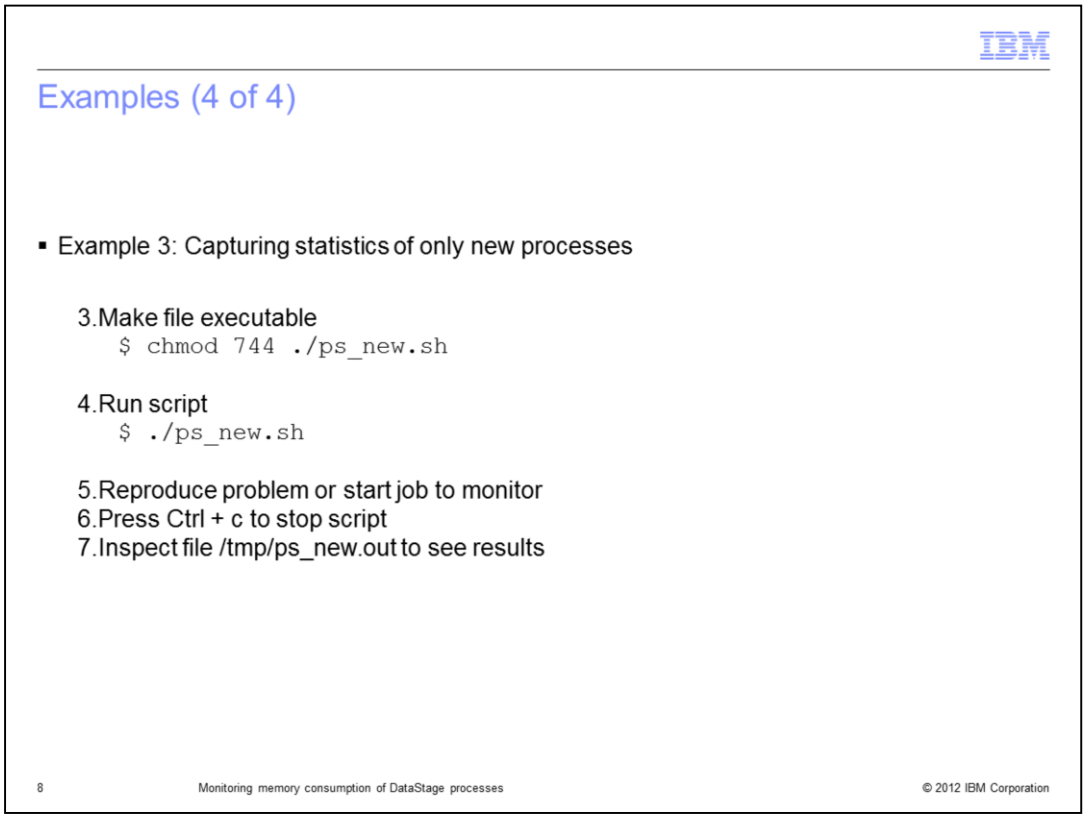
- Example 3: Capturing statistics of only new processes
  1. Create script on DataStage server

```
#!/bin/sh
OUTPUT_FILE=/tmp/ps.out # output file
SLEEP_TIME=1            # number of seconds between executions
rm aux1.log 2>/dev/null
rm aux2.log 2>/dev/null
rm aux3.log 2>/dev/null
rm $OUTPUT_FILE 2>/dev/null
ps -e -o pid,args|grep -v "ps -e -o"|grep -v grep| awk '{ print $1 }' >
  aux1.log 2> ps.err
while :
do
  ps -e -o pid,args|grep -v "ps -e -o"|grep -v grep|awk '{ print $1 }' >
  aux2.log 2> ps.err
  diff aux1.log aux2.log | grep \> | awk '{ print "^ *" $2}' >> aux3.log
  date >>  $OUTPUT_FILE
  ps -e -o pid,ppid,vsz,etime,args|egrep -f aux3.log >> $OUTPUT_FILE
  echo >>  $OUTPUT_FILE
  sleep $SLEEP_TIME
done
```

  2. Save file and call it ps_new.sh

The next example shows how to capture statistics for new processes. Sometimes it is difficult to find a common string to filter the processes you want to monitor. In those cases, and assuming that you can reproduce the problem or condition you want to analyze, you can use this script to keep track of all new processes. This script will ignore processes that existed before the script was executed. This is useful, for example, if you want to monitor all the processes of a job you are about to run. To use this script, open a text editor, create a new file and paste the lines included in this slide. You can edit the variable OUTPUT_FILE to indicate a different location for the output file and the variable SLEEP_TIME to indicate the delay between each execution.

Save the file and call it ps_new.sh.



Next, make the file executable using the chmod command and run the script. Reproduce the problem or start the job you want to monitor. Press Ctrl + c to stop the script. Inspect the output file to see the results. Monitor the size of the output file while the script is running as it may grow very quickly and take up too much disk space.

Trademarks, disclaimer, and copyright information