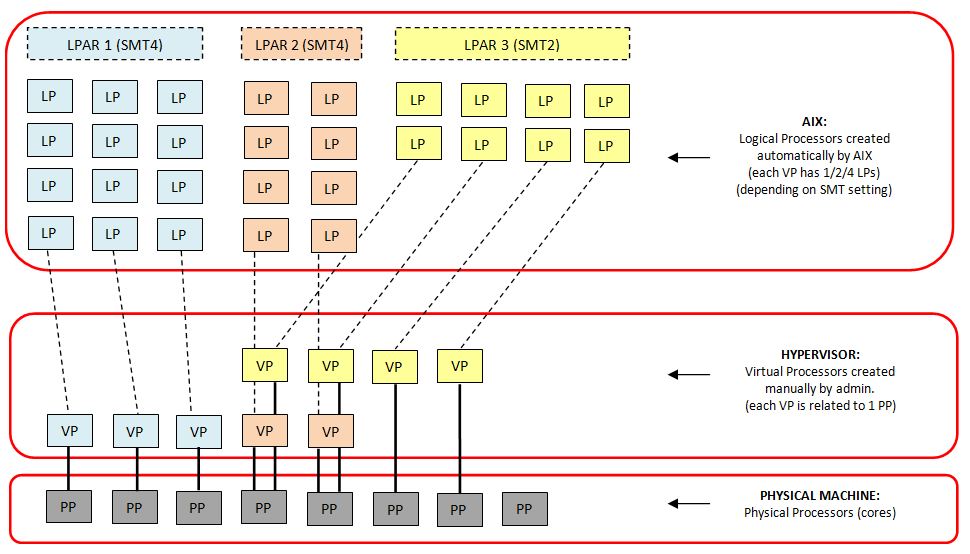
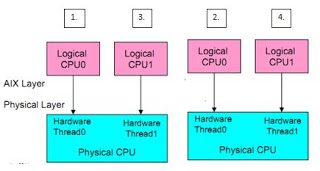
**CPU - PROCESSES:**  
  
**Physical - Virtual - Logical CPU:**  
  
Physical Processors are cores in the machine. Virtual Processors are assigned to an LPAR manually when LPAR is created. Logical Processors are created automatically by AIX, depending on the SMT setting.

[](http://2.bp.blogspot.com/-6DHbWEJX3LI/UWHNiZHKohI/AAAAAAAAAKU/Y-9vv_MNsak/s1600/Capture.JPG)

------------------------  
  
**Simultaneous Multi-Threading (SMT)**  
SMT is that feature of a Power Processor, when multiple hardware threads can run on one physical processor at the same time (a processor appears as 2 or 4 logical CPU). Within a CPU (core/cpu/processor are the same thing) there are multiple execution units. For example: floating point arithmetic unit, load and store execution units... A single thread would use only 1 or 2 of those units at any point in time. So most of the executional units within a core will not be utilized. With the ability of multi-threading 2 (or 4) threads could be running in a core at same time. One of them will use the floating processor while the other doing load and store ...(If there are collisions, one of them would be delayed but it happens no too often.)  
  
**How threads will be dispatched to multiple cores:**  
First thread will be dispatched to the primary hw thread of a physical cpu. If we have another CPU then next thread will be dispatched there (to avoid collision)

[](http://1.bp.blogspot.com/-TxV0eqN2-JI/Tn3S7m615aI/AAAAAAAAADA/g8kYeLAI4TY/s1600/1.JPG)

**IMPORTANT:**  
  
**- Prior AIX6 TL4:** if only a single hw thread was busy, processor reported as 100% utilized (this is an error because all the secondary threads were not utlized)  
  
**- AIX6 TL4 and later:** potential capacity of unused hw threads are from TL4 reported as idle time for the processor (it measures the capacity of the unused hw threads.)  
  
------------------------  
  
**SMT behaviour and intelligent SMT threads:**  
  
AIX default behaviour is to use all the VPs for maximun performance. If workload grows it will use up all VPs (CPU cores) quickly, but AIX first uses SMT thread 1 on all CPU cores before allocating work to the 2nd, 3rd and 4th SMT threads.  
  
SMT threads can be seen as Logical CPUs on AIX. If SMT=4 then 1 VP shows up as 4 Logical CPU. From Power7 there is a thing called "intelligent SMT threads". If there are not enough processes to run on all SMT threads (official mode is SMT=4) it will be dynamically switched to 2 or 1.  
  
**mpstat or topas -L is showing it:**  
  
At the column "lpa" the sign "-" will show turned off SMT threads  
  
**# mpstat 2**  
  
cpu  min  maj  mpc  int   cs  ics   rq  mig lpa sysc us sy wa id   pc  %ec  lcs  
  0    0    0    0  265   35   24    2    0 100   64 100  0  0  0 0.63 31.7   99  
  1    0    0    0   12   12    0    0    0 100    9  0  0  0 100 0.12  6.1   22  
  2    0    0    0    9    0    0    0    0   -    0  0  0  0 100 0.12  6.1   20  <--this SMT thread is turned off  
  3    0    0    0    9    0    0    0    0   -    0  0  0  0 100 0.12  6.1   19  <--this SMT thread is turned off  
  4    0    0    0  100   12    8    1    0 100    0 100  0  0  0 0.64 31.8   99  
  5    0    0    0   19   59    0    0    0 100    9  0  0  0 100 0.12  6.1   69  
  6    0    0    0    9    0    0    0    0   -    0  0  0  0 100 0.12  6.1    9  <--this SMT thread is turned off  
  7    0    0    0    9    0    0    0    0   -    0  0  0  0 100 0.12  6.1    9  <--this SMT thread is turned off  
ALL    0    0    0  432  118   32    3    0   0   82 63  0  0 37 2.00 999.8  346  
--------------------------------------------------------------------------------  
  
------------------------  
  
**Context Switch:**  
It is inherent in any multiprocessing operating system. Different appl. threads are sharing a CPU. Every time 1 thread is leaving a CPU and a new thread  is dispatched to the CPU, a context switch occurs. The environment of the leaving one has to be saved and new environment ha to be reestablished for the new process. High context switch rates can cause many work (overhead) for the CPU, which can be a problem.  
  
------------------------  
  
  
**PROCESS:**  
  
You use commands to tell the operating system what task you want it to perform. When commands are entered, they are recognized by a command interpreter (also known as a shell), and the task is processed.  
  
A program or command that is actually running on the computer is referred to as a process.  
  
**The commom types of processes:**  
  
**Foreground processes**  
Processes that require a user to start them or to interact with. Programs and commands run as foreground processes by default.  
  
**Background processes**  
Processes that are run  independently of a user. To run a process in the background, type the name of the command with the appropriate parameters and flags, followed by an ampersand (&). When a process is running in the background, you can perform additional tasks by entering other commands at the command prompt. Most processes direct their output to standard output (stdout), even when they run in the background. Because the output from a background process can interfere with your other work on the system, it is usually good practice to redirect the output of a background process to a file.  
  
**Daemon processes**  
Daemons are processes that run unattended. They are constantly in the background and are available at all times. Daemons are started usually when the system starts, and they run until the system stops. A daemon process typically performs system services. For example qdaemon (provides access to system resources such as printers) and sendmail are daemons.  
  
**Zombie processes**  
A zombie process is a dead process that is no longer executing but is still recognized in the process table (in other words, it has a PID number). Zombie processe have been killed or have exited and continue to exist in the process table until the parent process dies or the system is shut down and restarted. Zombie processes display as <defunct> when listed by the ps command. The only way to remove zombies is to reboot the system.  
  
**Thread**  
Each process is made up of one or more kernel threads. A thread is a single sequential flow of control. Rather than duplicating the environment of a parent process, as done via fork, all threads within a process use the same address space and can communicate with each other through variables.  
  
------------------------------  
  
**Process priority**  
  
A priority is a number assigned to a thread. The kernel maintains a priority value (0-255). A smaller priority value indicates a more important thread. Real time thread priorities are lower than 40.  
  
**Nice value**  
A nice value is a priority adjustment factor added to the base user priority of 40 (for non-fixed priority threads). The nice value is used by the system to calculate the current priority of a running process. The first process in the system (init) has a nice value of 20, and therefore an effective priority of 60. (PRI heading in the below output) A foreground process has a nice value of 20 (24 for a background process).  
  
***ps -el***           shows process priorities  
***ps -ekl***               shows process priorities including kernel processes  
***ps -kmo THREAD***         shows processes with their threads priorities  
  
  
root@aix31: / #***ps -el***  <--shows the nice values under the NI heading (-- means it is running with fixed prio.  
       F S UID    PID   PPID   C PRI NI ADDR    SZ    WCHAN    TTY  TIME CMD  
  200003 A   0      1      0   0  60 20 7037000   784               -  0:39 init  
  200103 A   0 311326 352456   0  24 -- 81d8400  4676               -  1:15 xmtopas  
  
The nice value can be set at process creation time by using the nice command. If the process already created the renice command is used.  
(ksh will add automatically 4 to the default nice value (20) if a process is started in the background (&))  
  
The nice value can be ranged from 0 to 39, with 39 being the lowest priority.  
***nice -10 <command>***       add 10 to current nice value (lower priority)  
***nice --10 <command>***        subtract 10 from current nice value (higher priority)  
  
The renice value can be -20 to 20. (1-20: lowers the priority, 0:sets to the base scheduling priority, -20 to -1:highers the priority)  
***renice 10 -p <pid>***        add 10 to the default nice value (20) (lower priority)  
***renice -n 10 -p <pid>***     add 10 to current nice value  (lower priority)  
***renice -10 -p <pid>***       subtract 10 from the default nice value (20) (higher priority)  
***renice -n -10 -p <pid>***     subtract 10 from current nice value (higher priority)  
                           (-n: incerment is added to the current nice value, not default)  
  
---------------------------   
  
**CPU infos:**  
  
***lscfg | grep proc***           shows how many (virtual) processors we have (lsdev -Cc processor, shows also how many virt. proc we have)  
***bindprocessor -q***             shows how many logical (SMT) processors we have  
***lsattr -El procX***           shows the processor settings  
***pmcycles -m***               shows the processors speed (if smt is enabled it will show for all the logical processors)  
***smtctl***                   it will show how many processor we have (if smt is turned on or not)  
  
---------------------------  
  
**Process handling:**  
  
***Ctrl-C or Ctrl-Backspace***  cancels a foreground process  
  
***ps***                     lists processes (by default lists only processes started from the current terminal)  
   ***-e***                   every process runnning on the system  
  ***-f***                 full listing (username, PPID...)  
   ***-L <pid>***               lists all processes which PPID is <pid>  
   ***-u <user>***              lists all processes running under <user>  
  ***-T***                   lists the tree of a given process (shows the children of a given process)  
  
***ps -elmo THREAD***            lists processes and its threads (shows pids and the threads (tid) which belong to a given process)  
  
***proctree <pid>***           displays the process tree of the specified process  
  
***kill <pid>***                notification to the process to terminate (it is using the default, 15, signal)  
***kill -9 <pid>***              kills the process without notification  
***kill -1 <pid>***            restarts the process (rereads the config files as well) (HUP - hangup)  
                           (when a background process is running and you log off a hangup signal is sent)  
***kill -2 <pid>***             interrupt signal (same as ctrl+c)  
***kill -l***           lists all the signals supported by kill (cat /usr/include/sys/signal.h will show as well, with details)  
  
***ls -R / > ls.out &***        starts ls in the background (standard output is ls.out)  
***nohup ls -R / > ls.out &***   nohup allows a background process to continue after logging off the system   
                           (if output isn't redirected, it will create nohup.out)  
***echo "<command>" | at now***  this also starts in the background (and you can log off)  
***jobs***                 lists which processes are running in the background  
  
***nohup alt\_disk\_copy -d hdisk1 -B &*** can't be hanged up and in backgound (kill command can stop it)  
  
**Restarting a stopped foreground process (jobs command):**  
***1. Ctrl-Z***                 stops a foreground process, its PID is still in the process table (it goes to background)  
***2. jobs***                  this will list stopped processes  
[1] + Stopped (SIGTSTP)        ./myscript    <--you will see a line like this (here #1 is the job id)  
***3. fg %1***                  put given job into foregeound (bg %1 puts into background)  
  
  
**Restarting a stopped foreground process (ps -ef <pid>):**  
***1.Ctrl-Z***                stops a foreground process, its PID is still in the process table (it goes to background)  
***2.ps -ef | grep <PROC.NAME>***    find the process ID (PID)  
***3.fg <PID>***            restarts that stopped proces (it will go to foreground)  
  
**Removing a background process:**  
***1.find / -type f > output &***    run the find command in the background  
***2.ps***                    lists the PID numbers  
***3.kill <PID>***             cancel the process  
  
------------------------------  
  
The operating system allows you to manipulate the input and output (I/O) of data to and from your system. For example you can specify to read input entered on the keyboard (standard input) or to read input from a file. Or you can specify to write output data to the screen (standard output) or to write it to a file.  
  
When a command begins running, it usually expects that the following files are already open: standard input, standard output and standard error. A number, called a file descriptor, is associated with each of these files:  
  
0     represents standard input (stdin)  
1     represents standard output (stdout)  
2     represents standard error (stderr)  
  
**The redirection symbols and their meanings:**  
<     redirects input (stdin) (< filename is added to the end of the command)  
>     redirects output (stdout) (> filename is added to the end of the command)  
>>    appends output  
<<    inline input (see pg. 574)  
2>    redirects output (stderr)  
1>&2  redirects stdout to stderr  
2>&1  redirects stderr to stdout  
  
  
***mail denise < letter1***       sends the file letter1 to user denise with the mail command  
***echo $PATH > path1***           saves the value of the PATH variable on the file path1  
***cat file2 >> file1***          append file2 to file1 (the cat commands can concatenate not only display files)  
***ls -l file1 2> list1***        save the stderr to file list1 (if file1 does not exist)  
***ls \*.dat \*.txt > files.out 2> files.err***    (files.out: stdout, file.err: stderr)  
***command > output 2>&1***       saves all the output (stdout and stderr) in one single file