Plan

Chapter 4: Linux – Managing Processors

1. Introduction
2. Overview
3. Process Launching Modes
4. Shell commands for Process Manipulation
5. Redirections
6. Interprocess Communication: Pipes
7. Process Grouping

1. Introduction

- A computer appears to perform multiple activities simultaneously, such as
  - displaying a clock while using a word processor;
  - or printing a document while continuing to write.
- Despite having only one processor, it manages these tasks by efficiently switching between them.
- Processes exchange information, such as a word processor periodically saving the written text to the disk.

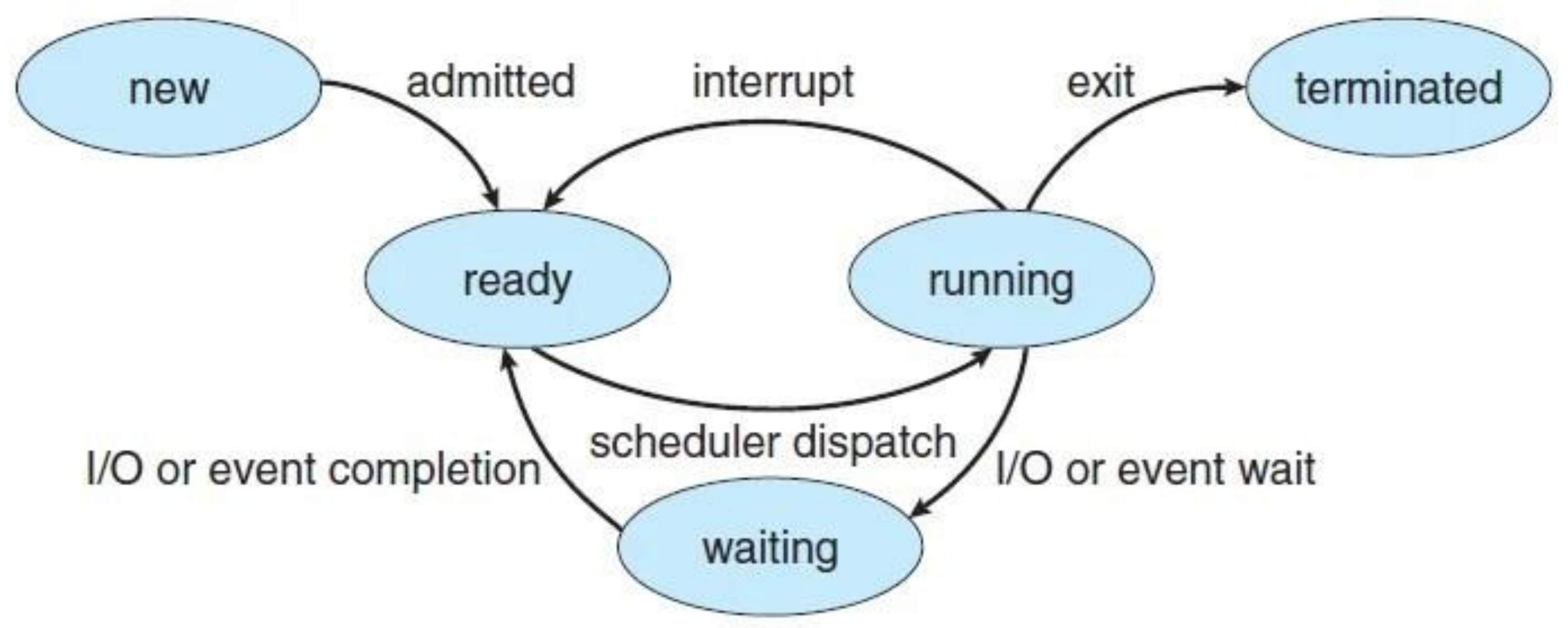
2. Overview

- A process is a dynamic entity representing the execution of a sequence of instructions. It includes a running program, its data, stack, program counter, stack pointer, resources, and other necessary register contents for execution
- It has 5 states and different transitions

2. Overview

- **Process states:**
- **New State**: When a process is created, it enters this state. The OS performs admission control, allocates resources, and initiates a process control block.
- **Ready State**: The process is ready to execute but is not currently running on the CPU. It waits for the scheduler to move it to the running state.
- **Running State**: The process is currently executing on the CPU. It can be interrupted, causing a context switch, or it can initiate a longer operation, moving it to the waiting state.
- **Waiting State**: The process is waiting for an event to occur or an I/O operation to complete. When the event occurs or the operation completes, it becomes ready again.
- **Terminated State**: The process has finished all operations or encountered an error, and it returns an exit code (success or error).
- **Transitions between States:**
- A process can move from the new state to the ready state if **admission control is successful**.
- A process can move from the ready state to the running state when **the scheduler schedules it on the CPU (Scheduler dispatch)**.
- A process can move from the running state to the ready state **if it is interrupted** and context-switched.
- A process can move from the running state to the waiting state if **it initiates a longer operation**.
- A process can move from the waiting state to the ready state when **the event occurs or the I/O operation completes**.
- A process can move from the running state to the terminated state if **it finishes all operations or encounters an error**.

2. Overview



https://i0.wp.com/zitoc.com/wp-content/uploads/2019/02/process-state.png?resize=720%2C285&ssl=1

2. Overview

- **<u>Difference between processor and thread</u>**

  - Processors (CPUs) are the heart of a computer, while **processor cores** are components of processors.

  - A processor can have one or more cores, while a core can have one or more threads.

  - Threads are virtual sequences of instructions given to a CPU.

  - Cores are physical processing units.

  - Multithreading allows for better utilization of available system resources by dividing tasks into separate threads and running them in parallel.

  - Hyperthreading further increases performance by allowing processors to execute two threads concurrently.

3. Process Launching Modes

- **Foreground Process:**

  When a command is executed, the shell must wait until the command finishes executing before it can execute a second command and return to the prompt. This is called a foreground process.

- **Background Process:**

  By adding an **"&"** after a command, it can be launched as a background process. The shell no longer waits for its execution to finish. When a user starts a background process, the shell displays the job number and process ID (PID) in brackets. The first background process started will have a job number of 1.

```
cat &
[1]   32124
```

3. Process Launching Modes

- **Sequential Chaining of Processes:**

  Commands can be chained sequentially using a semicolon.

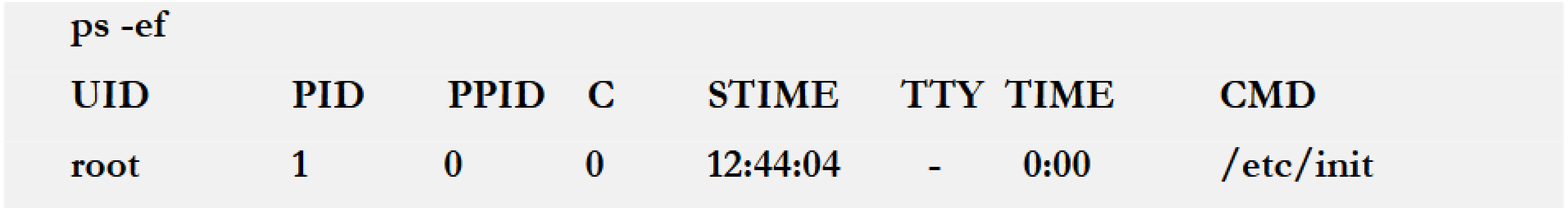
  For example, consider the following command line:

```
echo « bjr » ; cd /etc ; pwd
```

4. Shell Commands for Process Manipulation

- The **ps** (process) command lists the user's processes. The -e option displays all running processes on a computer, and the -l and -f options provide detailed information.

  For example, the command produces output similar to:



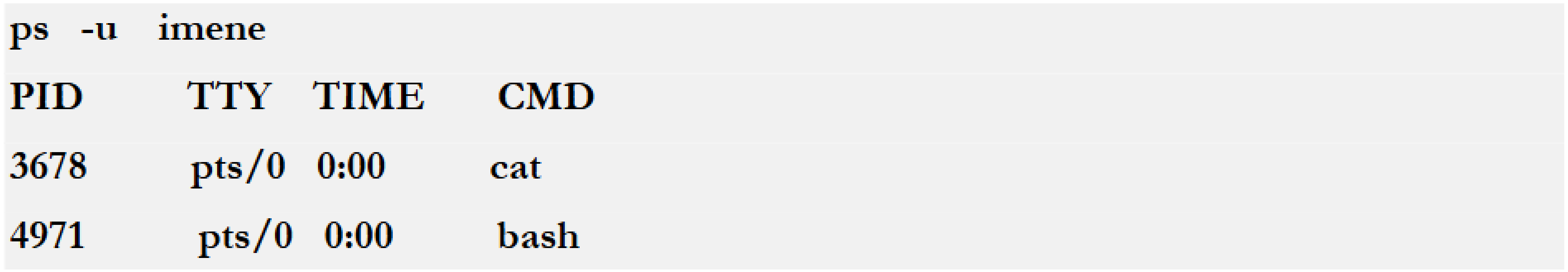
```
ps -ef
UID        PID    PPID   C     STIME      TTY   TIME      CMD
root       1      0      0     12:44:04    -    0:00      /etc/init
```

4. Shell Commands for Process Manipulation

- **UID**: The username of the person who launched the process.
- **PID**: The process ID number.
- **PPID**: The parent process ID number.
- **C**: The priority factor: the higher the value, the higher the priority of the process.
- **STIME**: The start time of the process.
- **TTY**: The name of the terminal.
- **TIME**: The amount of CPU time the process has used.
- **COMMAND**: The name of the process

4. Shell Commands for Process Manipulation

- The “**u**” option allows displaying the processes started by a specific user



```
ps   -u   imene
PID        TTY   TIME     CMD
3678        pts/0  0:00        cat
4971         pts/0  0:00        bash
```

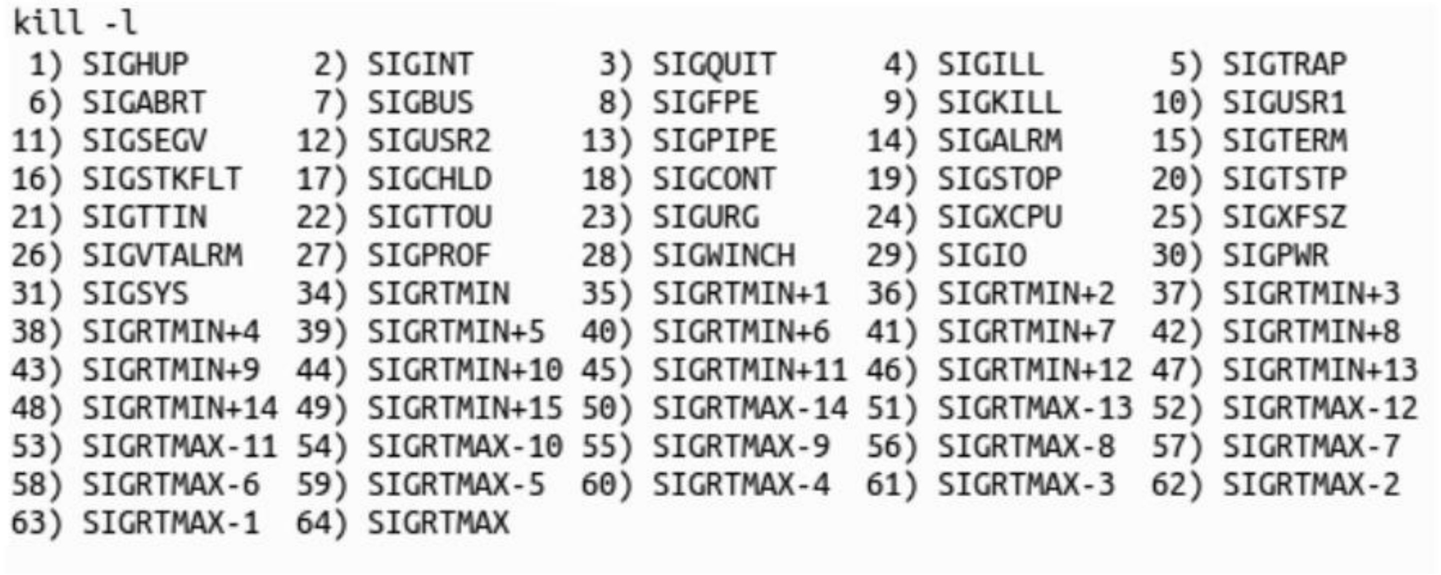
4. Shell Commands for Process Manipulation

- A process can receive various signals generated by the **kill** command, user keystrokes, or hardware and software errors.
- A user can kill a foreground process by pressing CTRL+C or suspend it by pressing CTRL+Z.
- The kill command sends a signal to a process, identified by a number. Despite its name, kill does not necessarily terminate a process but can send different signals to it.

```
kill [-l] -Num_signal PID1 [PID2...]
```

4. Shell Commands for Process Manipulation

- Signals are a means of communication between processes. When a signal is sent to a process, it must intercept and react to it accordingly. Some signals can be ignored, while others cannot.



```
kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

- To kill a specific processor

```
kill  -9  <PID>
```

4. Shell Commands for Process Manipulation

- Below are the numbers, names, and descriptions of some signals that a process can receive:

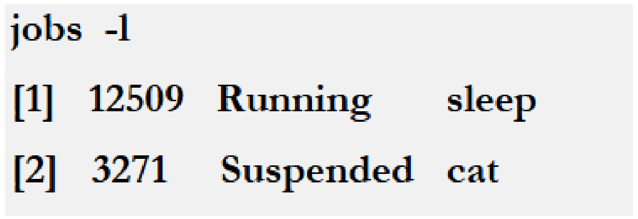
- Number: 9
- Name: KILL
- Description: A signal that will forcefully terminate the receiving process.

- Number: 15
- Name: TERM (Terminate)
- Description: Sent to a process to gracefully terminate it if possible.

- Number: 19
- Name: STOP
- Description: This signal suspends a process.

4. Shell Commands for Process Manipulation

- The **jobs** command displays the list of tasks in the current shell (suspended or running in the background) along with their task numbers, PIDs, and process states. Using jobs -l provides more detailed information.

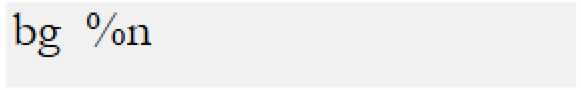


```
jobs -l
[1]  12509  Running    sleep
[2]  3271   Suspended  cat
```

- The **fg** command restarts the execution of a background process as a foreground process. fg %nn refers to the task number.

```
fg %n
```

- The **bg** command restarts a suspended process as a background process. bg %nn refers to the task number.



```
bg %n
```

- The **top** command continuously displays system activity information, particularly useful for monitoring process resource usage (such as RAM quantity, CPU percentage, and process duration since startup). Press "q" to exit the top utility.
- The **time** command measures the execution duration of a command, providing three values:

  real: total execution duration of the command,

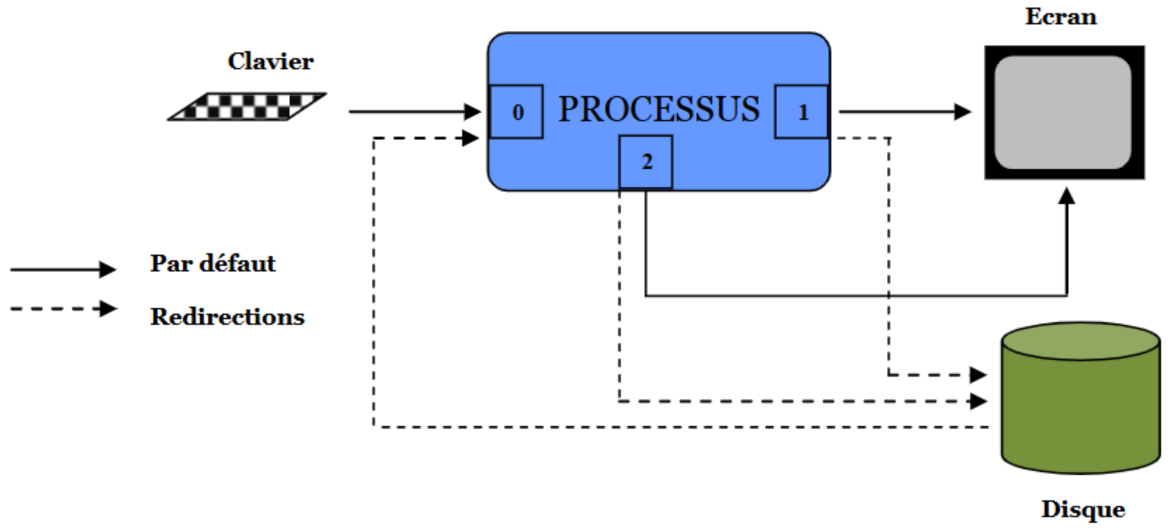
  user: CPU time required to execute the program, and

  system: CPU time required to execute OS-related commands (system calls within a program).

-> If the result is less than 10, the machine has good performance; beyond 20, the machine's load is too heavy (low performance).

5. Redirections

- Each process communicates with the outside world through three special files that are constantly open:
  - **0: Standard input**, where the process reads the data it needs (by default, the keyboard).
  - **1: Standard output**, where the process writes the result to the user (by default, the screen).
  - **2: Standard error output**, where any error messages are written (by default, the screen).



5. Redirections

- By default, these files are linked to the terminal, representing the keyboard (input) and the screen (output).
- When creating a process, we can redirect its standard input and output, connecting these standard files to other entities such as files, pipes, or devices.
- Output redirection can overwrite or append to a file, depending on whether the file exists. For input redirection, the file must exist.
- The **<** character followed by a filename indicates redirection of standard input from that file.
- The **>** character followed by a filename indicates redirection of standard output to that file. The file will be overwritten if it already exists, unless the boolean variable noclobber is initialized.