

Lab 3

Linux Process Management

ITSC205: Operating Systems Internals

NAME: Ahmed Almass

Table of Contents

[Lab Outcome(s) 2](#_Toc472385055)

[Reading 2](#_Toc472385056)

[Introduction 2](#_Toc472385057)

[1.0 Trace Processes 3](#_Toc472385058)

[2.0 PROC – Pseudo File System 4](#_Toc472385059)

[3.0 Process System Calls 6](#_Toc472385060)

[4.0 Processes and Signals handling 8](#_Toc472385061)

**EVALUATION**:

|  |  |  |
| --- | --- | --- |
| Trace Processes | 10 |  |
| Explore PROC – Pseudo File system structure | 10 |  |
| Analyze process System calls | 15 |  |
| Run and analyze process and Signals Handling | 15 |  |
| TOTAL MARK | 50 |  |

Lab Outcome(s)

* Trace processes running on Linux systems
* Apply system calls to create and terminate processes
* Run and modify c programs to analyze processes system calls
* Send signals to processes using Linux utilities and system calls.

Reading

* Textbook chapter 3, sections 3.3.1 Process Creation and 3.3.2 Process Termination. Chapter 20 (Linux system ) section 20.4 Process Management

Introduction

A process is a program in execution. A Process requires system resources to complete its jobs such as: CPU time, memory space (memory areas to store image (executable) and to store global variables and stack for local variables and functions arguments), open files (file descriptor) and other handles.

In UNIX/Linux operating systems processes stablish parent /child relationship. Processes can be in different states such as: Running on CPU or waiting for resources in a specific queue. A signal can be sent to a process to suspend it, terminate it or change its behaviour.

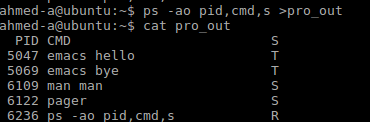
To find information about signals use the command: **info signal**

To create or clone a process UNIX/Linux operating systems implements **fork( )** system call which generates an integer value to differentiate between a parent and child process (value=0). While the child implements **exec( )** system call to execute the program, the parent will invoke the system call wait() to wait for the child to complete its job –(**wait(&childPID)** waits on child PID). After child process completes its job the signal **(SIGCHLD)** is sent to the parent process which will invoke the wait( ) or waitpid() system call and releases resources used by child process. If the parent does not exist the child process may become an orphan (child of init process or a thread) or a zombie that will hold resources affecting system performance.

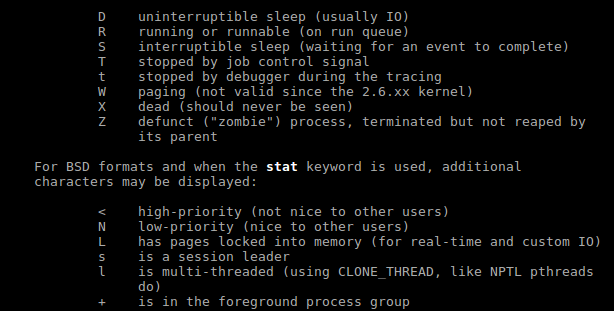
1. Trace Processes \_\_\_/10

Unix/Linux systems support many utilities/commands that can be used to manage processes. ***ps*** and ***top*** commandsarecommon commands to manage processes. Use Linux commands to manage processes and threads as follows:

1. Use the man command to learn different options that can be used with ***ps*** command. Write down the command and syntax that redirects to a file called **pro\_out** the Process Name (cmd), PID, and STATE of **ALL** processes on the system.



1. Use the ***man ps*** command to read the manual on ps. At the last line of the manual type **/PROCESS STATE CODES** to search the manual and find information about process state. Read the topic and write down the different processes states code and the respective description.



1. Use the command **top** to trace dynamically the current processes. Analyze top results, find and write down:
   1. the total amount of processes or tasks:



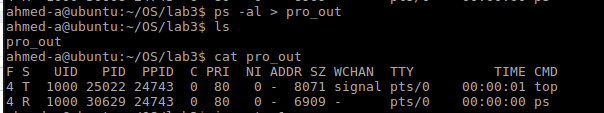
* 1. the amount of running processes:



* 1. the amount of sleeping processes:

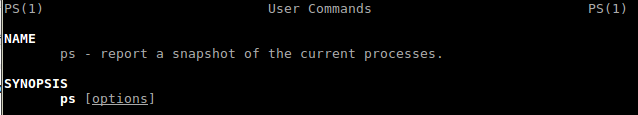


1. Demo to the instructor the following:
   1. Open the file you created called **pro\_out** and demo the results to the instructor



* 1. Create two processes. Suspend the processes (you can use Ctrl-z or &) and use the respective command(s) to display **ONLY** the properties or attributes of the suspended process. Demo the results



1. Create a Zombie process as follows :
   1. Start, and keep running, man by typing: **man ps**
   2. Suspend the process by pressing ctrl-z



* 1. Find the status of man and its child processes: **ps -l**

****

* 1. Terminate a parent process: **kill -9 *PID\_of\_pager***

******

* 1. Find the status of man and its child processes: **ps -l**

****

**trace is the status**

6. Demonstrate the results to the instructor

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

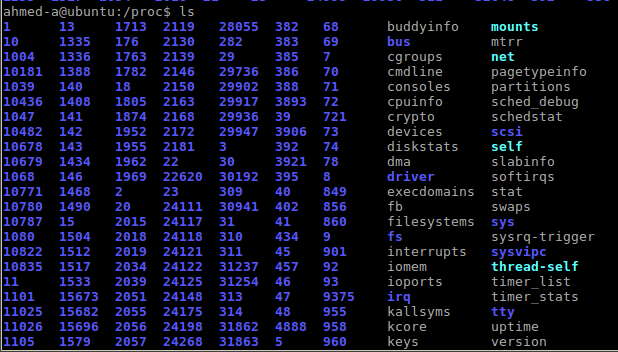
2.0 /PROC pseudo file-system Structure \_\_\_\_/10

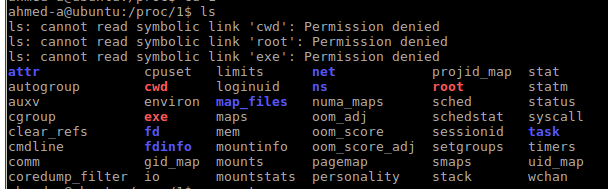
The **/proc pseudo file-system** is the root source of information about the processes within a GNU/Linux system. Within **/proc** directory, you'll find a set of directories with numbered filenames. These numbers represent the **process IDs** (pids) of active processes within the system.

Each ***pid*** directory presents a hierarchy of information about that process (process data structure) including the command line that started it, a symbolic link to the root file system, a symbolic link to the directory (current working directory) where the process was started, and other **process attributes** and configurations. Use **man proc** command for more information.

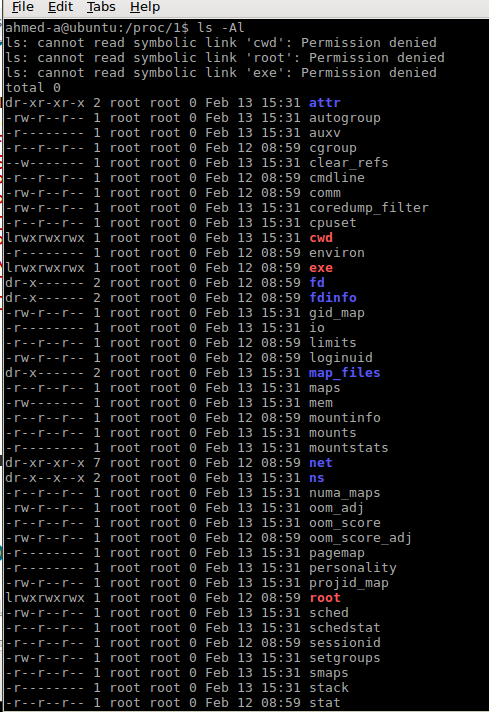
1. Explore the **/proc** directory and identify the process id (PID) of ***init*** or ***systemd*** process.

Pid : 1

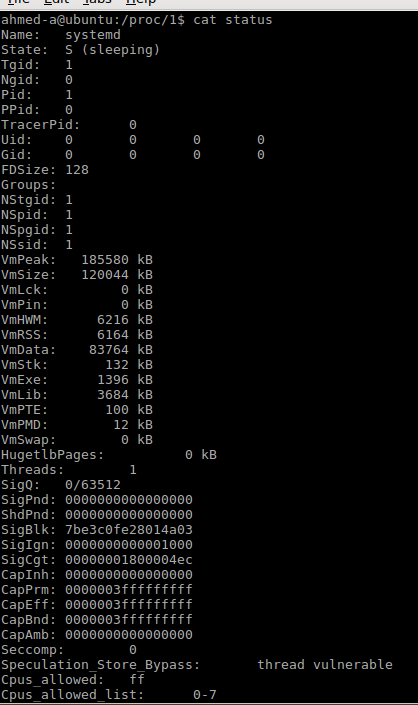




2. Access the PID directory of **init** or **systemd** process and explore **its** attributes



3. Use **cat command to display the content of the status** file for **init or systemd** process and answer the following questions:



1. What is the state of the process?

* s(sleeping)

1. What is its PID and PPID?

* PID: 1 PPID: 0

1. How many threads are running in this process?

* Threads 1:

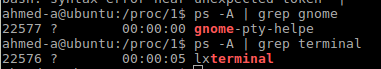
1. What is **context switch**?

* the process of storing the state of a process or of a thread, so that it can be restored and execution resumed from the same point later.

1. How many voluntary context switches are needed by this process?

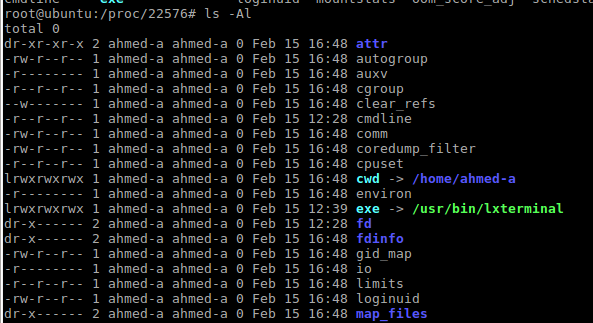


4. Use **ps –A** command to identify the PID of process **gnome –terminal**.



Instead of gnome terminal its lxterminal because im using ubuntu lxde desktop format.

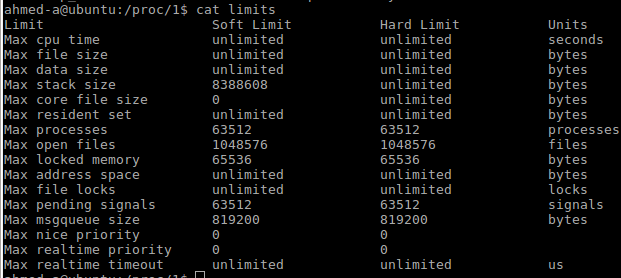
5. Access **/proc** directory and access **gnome –terminal** process PID directory and explore **its** attributes. How many threads are used by this process?



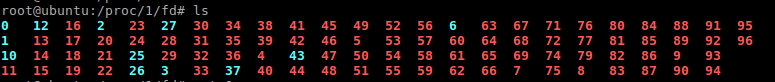


6. Use **man proc**. Explore **/proc/pid** information, demo and explain to the instructor the purpose of the following:

* 1. Limits
  + From man(This file displays the soft limit, hard limits and units of measurements for each of the process resources limits.)
  + Those are numbers used to identify the limits to which certain resources can be allocated to different users. Those resources include processor, memory and disk resources, e.g. number of child processes a user can have, percentage disk quota allowed for the user, number of open terminals, etc.(online)
  + A hard limit is the upper limit that the user can never, ever go beyond. Say you set a hard limit of 255 processes per user. No one of the users can exceed that limit, ever.
  + The soft limit, on the other hand, is a “warning” limit. It tells the user and the system admin that you are close to reach the danger level, which is the hard limit. Users are allowed to go over the soft limit, unlike the hard limit (it is an absolute no-no to go over hard limits).

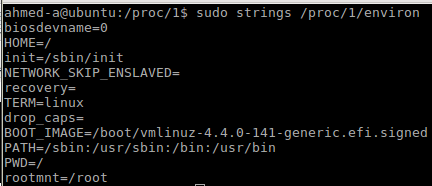


* 1. fd (File Descriptor)
* When a process makes a successful request to open a file, the kernel returns a file descriptor which points to an entry in the kernel's global file table. The file table entry contains information such as the inode of the file, byte offset, and the access restrictions for that data stream (read-only, write-only, etc.). (from online)



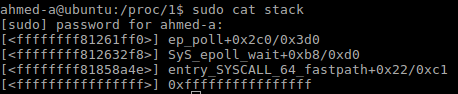
* 1. environ

This file contains the environment for the process. The entries are separated by null bytes ('\0'), and there may be a null byte at the end.



* 1. Stack

This file provides a symbolic trace of the function calls in this process's kernel stack.

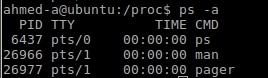


1. Use **man proc** to find and briefly describe the purpose of pid\_max file located in **/proc/sys/kernel**

* maximum available processes in your system

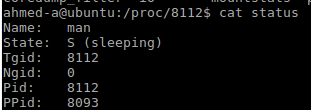
****

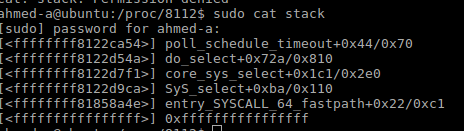
1. Open a new terminal
2. Create a new process (e.g. man, bash , gedit, or browser)
3. Use ps command to find the PID of process created



1. Access /PROC pseudo file system and demo to the instructor the process’ data structure created such as: **process status, stack, file descriptor, and other process information**.

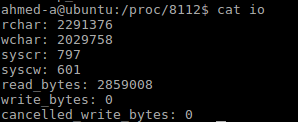
PID: 8112







IO: Contains I/O statistics for the process:



|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

**3.0 Process system calls \_\_\_\_\_\_/15**

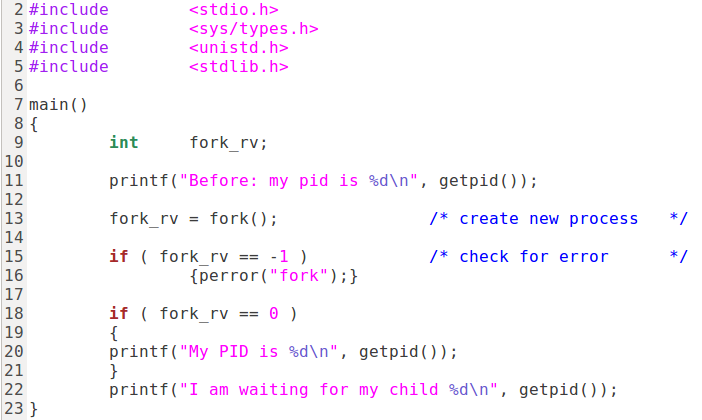
**Fork( ), Execvp( ) and wait( ) system calls**

1. Compile and run the following C code
   1. What is the purpose of this code?

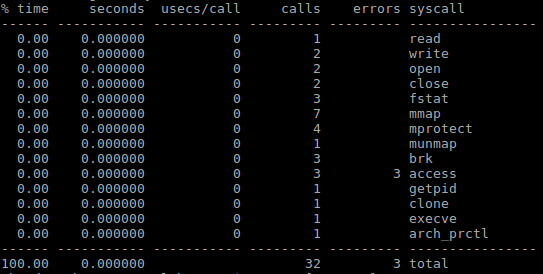
First off: reads and prints off the PID when the process runs

2nd: creates a new child process and reads the PID and prints out the parents pid and the child pid.

* 1. What are the values of fork( ) function?
* 0: Returns a value of 0 when making a child process
* -1: Returns a value of -1 when no child processes is created
* Positivevalue**:** Returned to parent or caller. The value contains process ID of newly created child process.

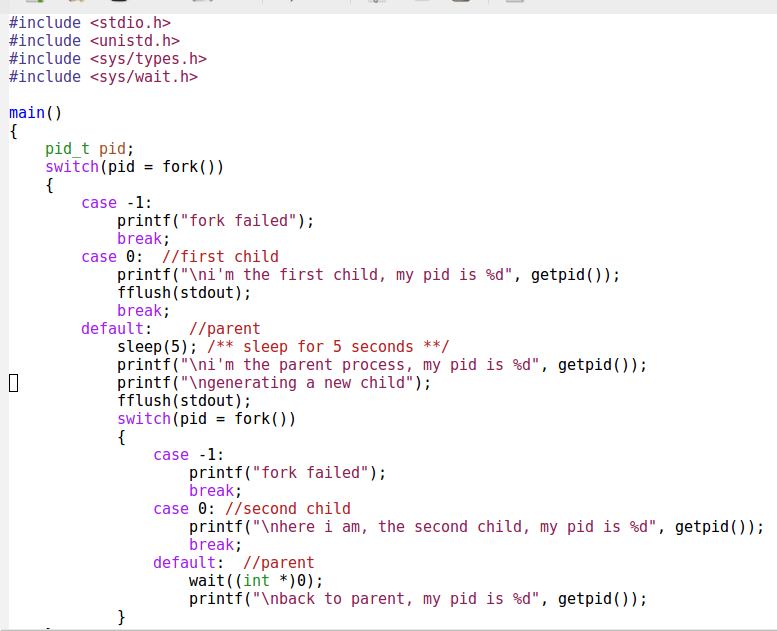


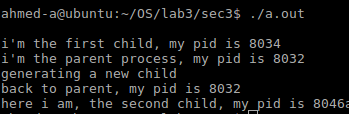
1. Use **strace –c** to run the program and identify the system calls used. Explain to the instructor **fork( )** purpose and output.



Purpose of the the fork() is to create child process in which makes an exact copy of the parents address space and give it to the child. The parent and child process have the separate address space.

1. Identify child process and parent process and modify the program using **sleep()** and **wait()** system calls so that the child process sleeps for 5 seconds and the parent waits for the child to complete its task. Demo modified code to the instructor.

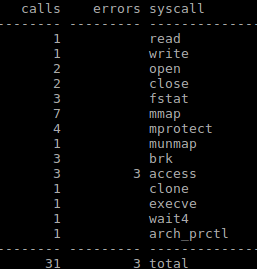




|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

1. Compile and run the following C code
   1. What is the purpose of this code?

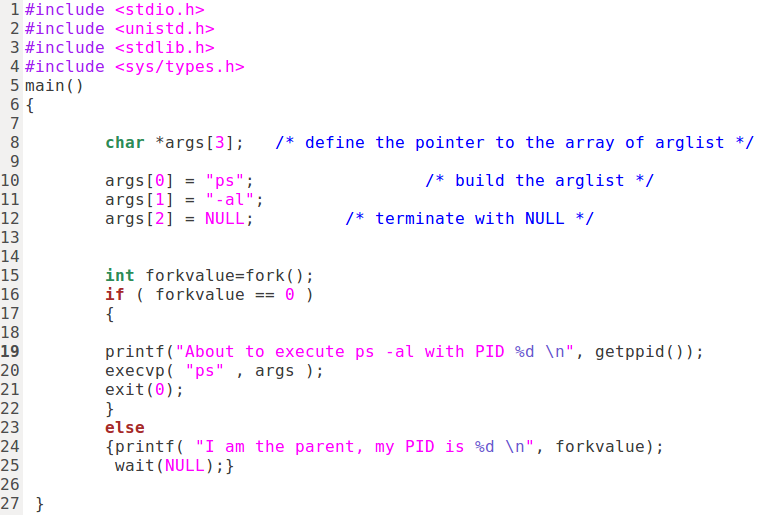
* Showing that once executing the c program it creates a parents process executing arg; using fork() creating a and printing new pid
  1. How many processes were cloned in this program?
  + one
  1. Identify the system calls in this program



* 1. Which process executes ps command (child or parent process)?

Parent process

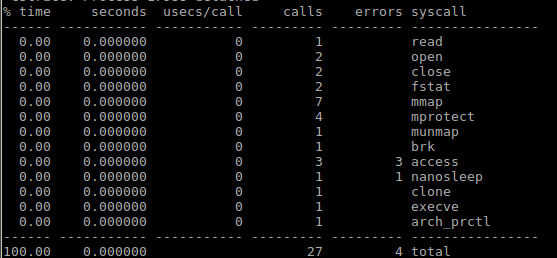




* 1. Modify the program to create a zombie by making the parent process to **sleep(120).**
  2. Compile and run the program. While the program is running open a second terminal and use the command: **ps – Al | grep Z** to identify the zombie.



* 1. Use strace program to verify the system calls use by this program and identify the signal send to the process.





SIGCHILD:

Child process has stopped or exited, changed (POSIX)

* 1. Demo and explain the results to the instructor

|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

4.0 Processes and Signal handles \_\_\_\_/15

Signals are messages sent to a process. Requests for signals come from users, kernel or processes. Signals generated by programs are synchronous signals ( trap) . Signals generated by user (hardware) are asynchronous signals. Signals numbers and their symbolic names are in /usr/src/include/signal.h .

1. The command **kill -l** displays signals supported by Linux . How many signals are supported by Linux? 64

2. Use the command ***kill –l*** to identify the code for the following signals:

a. SIGKILL 4

b. SIGTERM 15

c. SIGSTOP 19

d. SIGHUP 1

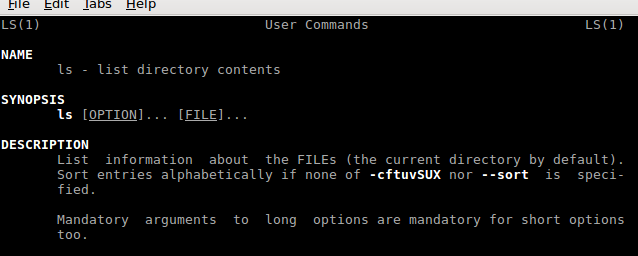
e. SIGCHLD 17

1. What is the difference among SIGKILL, SIGTERM and SIGSTOP?

SIGKILL: Kill(can't be caught or ignored) (POSIX)

SIGTERM: Termination (ANSI)

SIGSTOP: Stop executing(can't be caught or ignored) (POSIX)

1. Type ***man ls*** to start a process
2. Open a second terminal and use ***ps*** command with respective options to find the PID of ***man***  process



1. Use the ***kill command*** with respective code to send the SIGSTOP to **man** process

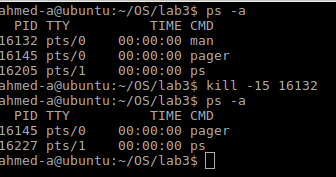


1. Use the kill command with the respective code to send SITERM signal to a process. Use ***ps***





7.verify the results. If the process was terminated why is the process still listed by ***ps***?



|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |

**Signals handling**

Compile and run the following Signals program:

/\*\* sigsample.c demonstrates how signals can terminate a process,

\* be ignored by a process, or be caught by a process \*\*/

#include <stdio.h>

#include <signal.h>

void catcher() /\* a function to run when a signal is caught \*/

{

printf(" Ouch! \n");

system("who");

}

main()

{

int i;

printf("Type ^C during either Case 1, Case 2, or Case 3");

printf("Case 1: no special arrangements..");

for (i=0;i<10;i++){

putchar('\*');fflush(stdout);

sleep(1);

}

putchar('\n');

signal(SIGINT, SIG\_IGN); /\* ignore INTerrupts \*/

printf("Case 2: ignoring interrupts..");

for (i=0;i<10;i++){

putchar('\*');fflush(stdout);

sleep(1);

}

putchar('\n');

signal(SIGINT, catcher); /\* handle interruptions \*/

printf("Case 3: catching interrupts..");

for (i=0;i<10;i++){

putchar('\*');fflush(stdout);

sleep(1);

}

putchar('\n');

}

1. Research and briefly explain the main system calls of the program: putchar(), fflush(), and signal().

* putchar() : writes a character specified by the argument char to stdout. Printing out char ‘\*’
* fflush(): forces a write of all user-space buffered data for given output. Printing out what ever in the buffer.
* Signal(): is a system call that returns the address of a function that takes an integer argument no return value.

1. Specifically, what does signal(SIGINT, SIG\_IGN) do?

* First argument of signal will handle the signal SIGINT, when detected the second argument is called SIG\_IGN which ignores the signal

1. Specifically, what does signal(SIGINT, catcher) do?

* first argument of signal will handle the signal SIGINT, when detected the second argument is called catcher which is a user function that prints out “Ouch!” and calls system(“who”) command.

1. Use **man kill** command and differentiate signals default actions. What is the difference among **exit**, **stop** and **core** actions?

* core: signal for stopping a process and telling it to create a core dump file
* Stop: Default action is to stop the process
* exit: terminate the calling process

1. What information is in a "core dump"?

* The processor's state
* The processor register's contents
* Memory management information
* The program's counter and stack pointer
* Operating system and processor information and flags

1. Why SIGKILL and SIGSTOP cannot be blocked?

* because they are designed to not to be handled to make sure a program can be killed.

1. **Demonstrate** and **explain** the output of the program to the instructor.

First while loop exit after doing the signal.

  
Second while loop ignores the kill command

3rd while loop receives signal and uses other argument which prints system who command .



|  |  |
| --- | --- |
| **ASK THE INSTRUCTOR TO SIGN OFF** |  |