جامعة الأمير مقرن بن عبد العزيز
University of Prince Mugrin

Lab 3

Introduction to C Programming & Managing Processes in Linux

Overview

جامعة الأمير مقرن بن عبد العزيز
University of Prince Mugrin

- Introduction
- Text editor, Compiler, C library
- Demo: Code1, Code2, Code3
- Managing Processes in Linux

Introduction

- **In order to create a C program, you need :**
  - Text editor
  - Compiler
  - C standard library.

Text editor

- A **text editor** is all that is needed to create the **source code** for a program in C or in any other language.
- A **text editor** is a program for writing and editing plain text.
- **C programs** can be written using any of the many text editors that are available for Linux, such as **vi**, **gedit**, **kedit** or **nano**.

Compiler

- A **compiler** is a specialized program that converts source code into machine language(also called object code or machine code) so that it can be understood directly by a CPU (central processing unit).
- An **excellent C compiler** is included in the GNU Compiler Collection (GCC), one of the most important components of most modern Linux distributions.
- All that is necessary to see if the GCC is already installed and ready to use is to type the following command and press the ENTER key: **gcc**
- **gcc** and **g++** are **command line compilers**; that is; they do not have a GUI
- You can also check the version using the following command: **gcc --version**

C library

جامعة الأمير مقرن بن عبد العزيز
University of Prince Mugrin

- A **library** is a collection of subprograms that any programmer can employ to reduce the amount of complex and repetitive source code that has to be written for individual programs.
- **C standard library** consists of built-in functions, macros and constants useful for C programming.

- Data definitions and prototypes of these functions are present in the **header files** (.h extension)
- Below are 3 important header files. Other examples: **<string.h>, <math.h>, <signal.h>, <time.h>, <ctype.h>** and many more.

```
#include <stdio.h>
```

Standard input/output & file operation functions.
E.g. printf (), scanf (), fprintf (), fscanf ()

```
#include <stdlib.h>
```

General utilities such as memory operation, random number generation etc.
E.g. malloc (), free (), rand ()

```
#include <unistd.h>
```

POSIX standard functions for interacting with OS
E.g. fork (), exec ()

Demo

1) Write the code using Text editor.

2) Compile the program using gcc command.

3) Run the program using **a.out** or specified **object code**.

Code 1

جامعة الأمير مقرن بن عبد العزيز
University of Prince Mugrin

```c
#include <stdio.h>

int main()
{
  printf ("Welcome to Madinah! \n");
  return 0;
}
```

- Compile then run executable `a.out`:

  gcc code1.c

  ./a.out

- Compile and run the specified object code :

  gcc -o code1 code1.c

  ./code1

```
~$ gcc code1.c
~$ ./a.out
Welcome to Madinah!
~$
~$
~$ gcc -o code1 code1.c
~$ ./code1
Welcome to Madinah!
~$
```

Code 2 : Read from keyboard and file

code2a.c

```c
#include <stdio.h>

int main()
{
  int num;

  printf ("Please enter a number : ");
  scanf ("%d", &num);
  printf ("The number is : %d \n", num);
  return 0;
}
```

```
~$ gcc code2.c
~$ ./a.out
Please enter a number : 33
The number is : 33
~$
```

code2b.c

```c
#include <stdio.h>

int main()
{
  int num;
  FILE *fd;

  fd = fopen ("int_data.txt","r");

  if (fd == NULL) {
    printf("\nFile does not exist.");
    return 1;
  }

  fscanf (fd, "%d", &num);
  printf ("The number is : %d \n", num);

  fclose (fd);
  return 0;
}
```

int_data.txt

```
1   987
```

```
~$ gcc code2-num-file.c
~$ ./a.out
The number is : 987
```

Code 3 : Array and for loop

```c
#include <stdio.h>
#define MAX_NUM 3

int main()
{
  int i, num[MAX_NUM];


  for (i = 0; i < MAX_NUM; i++) {
    printf ("Please enter a number : ");
    scanf ("%d", &num[i]);
  }

 for (i = 0; i < MAX_NUM; i++) {
    printf ("The number is : %d \n", num[i]);
  }

  return 0;
}
```

```
~$ ./a.out
Please enter a number : 7
Please enter a number : 2
Please enter a number : 5
The number is : 7
The number is : 2
The number is : 5
~$
```

Code 4 : Additional (struct and pointer)

```c
#include <stdio.h>

struct Personal_info {
  char name[25];
  int  age;
  float salary;
};

int main () {
  struct Personal_info employee, *emp_ptr;

  emp_ptr = &employee;

  printf ("\nEnter your full name : ");
  fgets (employee.name, sizeof(employee.name),stdin);
  // Use fgets() or  scanf ("%[^\n]",employee.name);
  printf ("\nEnter your age : ");
  scanf("%d", &employee.age);
  printf ("\nEnter your salary : ");
  scanf ("%f", &emp_ptr->salary);

  printf ("\n  PERSONAL INFORMATION");
  printf("\nName : %s ", employee.name);
  printf("\nAge : %d ", employee.age);
  printf("\nSalary : %.2f ", emp_ptr->salary);

  return 0;
}
```

```
~$ ./a.out

Enter your full name : Hanan Hassan

Enter your age : 26

Enter your salary : 850

  PERSONAL INFORMATION
Name : Hanan Hassan

Age : 26
Salary : 850.00
~$
```

جامعة الأمير مقرن بن عبد العزيز
University of Prince Mugrin

Managing Processes in Linux

How To View Running Processes in Linux

جامعة الأمير مقرن بن عبد العزيز
University of Prince Mugrin

- The easiest way to find out what are the processes that are running on your server is to run the **top command:**

command

```
top
```

Output

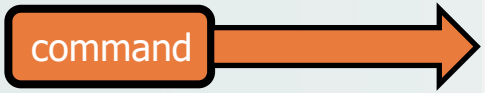
```
top - 15:14:40 up 46 min,  1 user,  load average: 0.00, 0.01, 0.05
Tasks:  56 total,   1 running,  55 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   1019600k total,   316576k used,   703024k free,     7652k buffers
Swap:        0k total,        0k used,        0k free,   258976k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    1 root      20   0 24188 2120 1300 S  0.0  0.2   0:00.56 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:00.07 ksoftirqd/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:00.03 watchdog/0
    8 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 cpuset
    9 root       0 -20     0    0    0 S  0.0  0.0   0:00.00 khelper
   10 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kdevtmpfs
```

While top is running, try:

- **?** to get help
- q to quit.

How To Use ps to List Processes

- This output shows all of the processes associated with the current user and terminal session.
- This makes sense because we are only running **bash** and **ps** with this terminal currently.

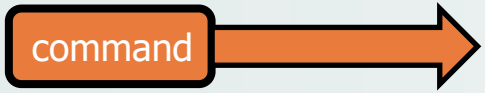



```
ps
```


```
  PID TTY          TIME CMD
 1017 pts/0    00:00:00 bash
 1262 pts/0    00:00:00 ps
```

How To Use ps to List Processes

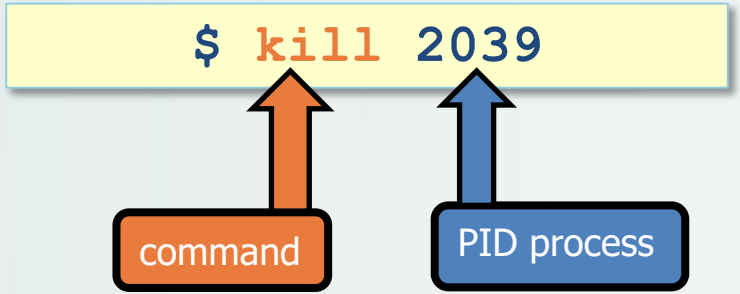
- To get a **more complete picture of the processes on this system**, we can run (**ps aux**).
- These options tell **ps** to show processes owned by all users (regardless of their terminal association) in a user-friendly format.




```
ps aux
```


```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2  24188  2120 ?        Ss   14:28   0:00 /sbin/init
root         2  0.0  0.0      0     0 ?        S    14:28   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    14:28   0:00 [ksoftirqd/0]
root         6  0.0  0.0      0     0 ?        S    14:28   0:00 [migration/0]
root         7  0.0  0.0      0     0 ?        S    14:28   0:00 [watchdog/0]
root         8  0.0  0.0      0     0 ?        S<   14:28   0:00 [cpuset]
root         9  0.0  0.0      0     0 ?        S<   14:28   0:00 [khelper]
. . .
```

How To Kill Processes by PID



Question?