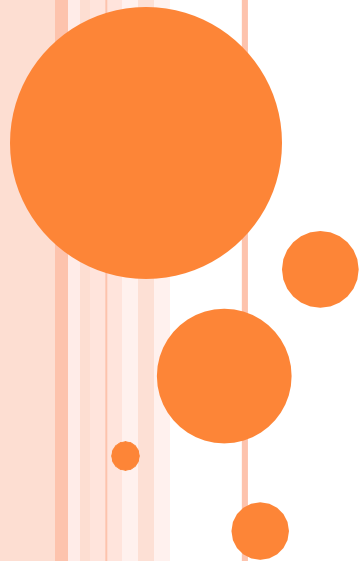


# C PROGRAMMING

Lab 4

Marin Iuliana





# C PROGRAMMING

Memory management using Makefiles Virtual memory

File system (fat, ntfs, ext3)

Virtualization

Security

# MEMORY MANAGEMENT USING MAKEFILES

- **Make** is a utility that allows automation and efficiency of tasks.
- Particularly used to automate program compilation.
- For obtaining an executable from multiple sources it is inefficient to compile each file of each file and then link-editing.
- Each file is compiled separately, and only one modified file will be recompiled.
- The **make** utility uses a configuration file called **Makefile** that contains rules and automation commands.



# MEMORY MANAGEMENT USING MAKEFILES

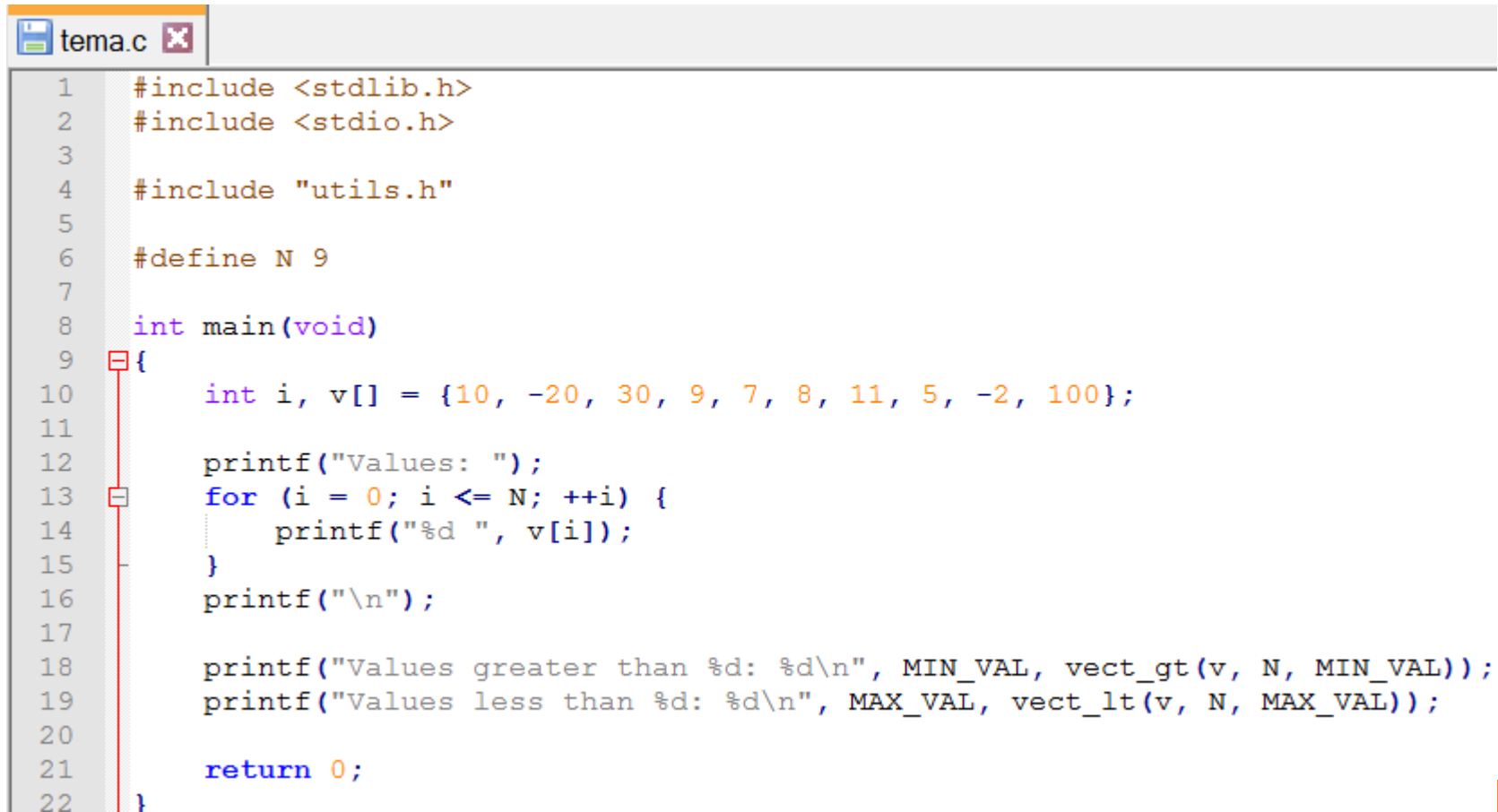
```
utils.h
```

```
1  #define MIN_VAL 5
2  #define MAX_VAL 3
3
4  int vect_gt(int*, int, int);
5  int vect_lt(int*, int, int);
```

```
utils.c
```

```
1  #include <stdlib.h>
2
3  int vect_gt(int *v, int n, int val)
4  {
5      if (n < 0)
6          return 0;
7      return v[n] > val ? 1 + vect_gt(v, n - 1, val) : vect_gt(v, n - 1, val);
8  }
9
10 int vect_lt(int *v, int n, int val)
11 {
12     if (n < 0)
13         return 0;
14     return v[n] < val ? 1 + vect_lt(v, n - 1, val) : vect_lt(v, n - 1, val);
15 }
```

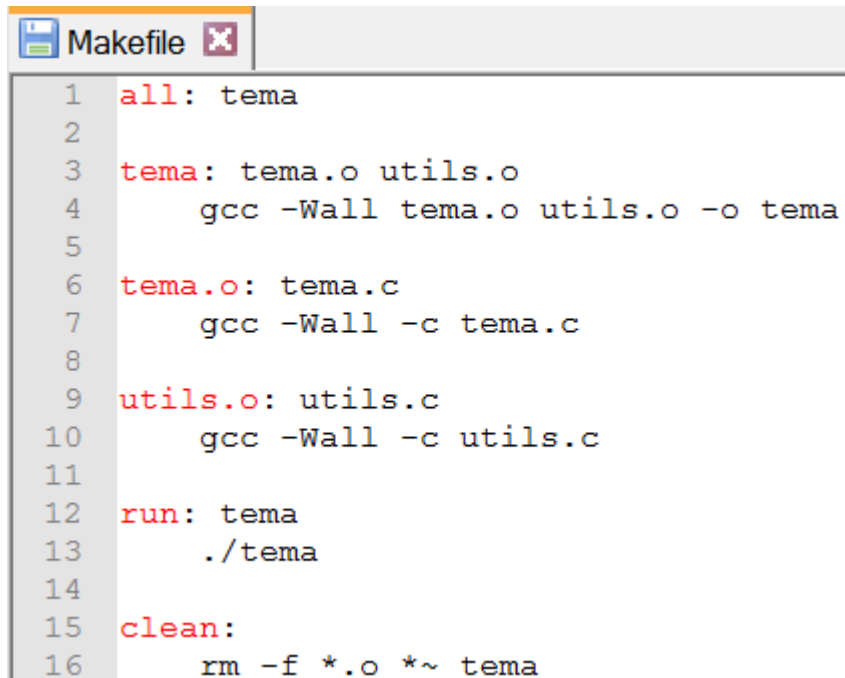
# MEMORY MANAGEMENT USING MAKEFILES



```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  #include "utils.h"
5
6  #define N 9
7
8  int main(void)
9  {
10     int i, v[] = {10, -20, 30, 9, 7, 8, 11, 5, -2, 100};
11
12     printf("Values: ");
13     for (i = 0; i <= N; ++i) {
14         printf("%d ", v[i]);
15     }
16     printf("\n");
17
18     printf("Values greater than %d: %d\n", MIN_VAL, vect_gt(v, N, MIN_VAL));
19     printf("Values less than %d: %d\n", MAX_VAL, vect_lt(v, N, MAX_VAL));
20
21     return 0;
22 }
```



# MEMORY MANAGEMENT USING MAKEFILES

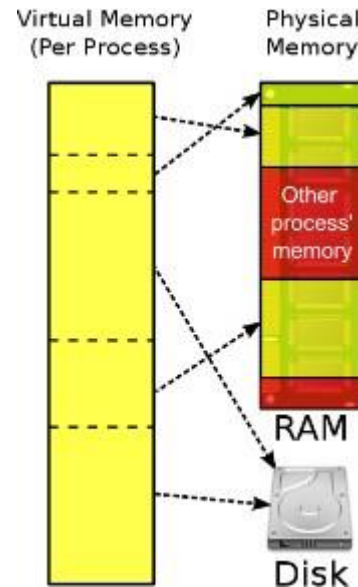


```
1 all: tema
2
3 tema: tema.o utils.o
4     gcc -Wall tema.o utils.o -o tema
5
6 tema.o: tema.c
7     gcc -Wall -c tema.c
8
9 utils.o: utils.c
10    gcc -Wall -c utils.c
11
12 run: tema
13    ./tema
14
15 clean:
16    rm -f *.o *~ tema
```



# VIRTUAL MEMORY

- Mechanism implicitly used by the operating system kernel to implement an efficient memory management policy, but also to map the address space of a process: files, memory, devices.



# VIRTUAL MEMORY

- Consider the following simple program which does nothing but return 0.

```
#include <stdio.h>
/**
 * main - Entry point of the program
 *
 * Return: 0 every time.
 */
int main(void)
{
    return (0);
}
```

- On compiling and then linking we get an object file (main.o) and an executable (main).

```
$gcc -Wall -Wextra -Werror main.c -c
$gcc -o main main.o
```





# VIRTUAL MEMORY

- The size tool is used to analyze the object file.

```
$size -A main.o
main.o :
section            size      addr
.text              11         0
.data              0         0
.bss               0         0
.comment           44         0
.note.GNU-stack    0         0
.eh_frame          56         0
Total             111
```

- The inspection of the text section is done using objdump.

```
$objdump -M intel -j .text -d main.o

main.o:          file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
   0: 55                      push    rbp
   1: 48 89 e5                mov     rbp, rsp
   4: b8 00 00 00 00         mov     eax, 0x0
   9: 5d                      pop     rbp
  a: c3                      ret
```



# VIRTUAL MEMORY

- In the following program we are dynamically allocating the memory using malloc and then run a loop infinitely (so that we can track its memory info).

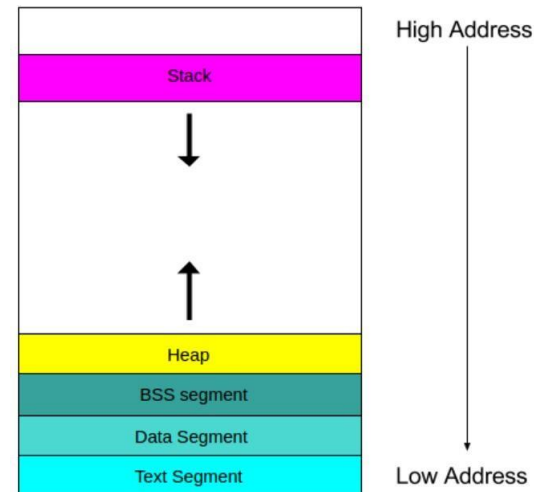
```
#include <stdio.h>
#include <stdlib.h>
/**
 * main - Entry point of the program
 *
 * Return: 0 every time.
 */
int main(void)
{
    char *p = (char*)malloc(sizeof(char));

    while (1)
    {
        sleep(1);
    }
    return (0);
}
```



# VIRTUAL MEMORY

- Each time we run a process the **pid** changes and the memory address used by **stack** and **heap** too. On fetching the process id **pid** and then checking the maps file under the proc fs.
- The main process id is received using pgrep. To visualize the mapping, the command `cat /proc/your_pid/maps` is used and you will see heap addresses like 01331000 and stack addresses which have the prefix 7ffed47.
- Both the stack and the heap can grow based on the way program handles the function calls and memory allocations.
- The virtual memory lookslike:



## FILE SYSTEM (FAT, NTFS, EXT3)

- FAT is used in MS-DOS, Windows '95, Windows '98.
- It contains the boot sector, FAT region, Root directory region and data region.
- A FAT table is a list of entrances mapped per clusters.
- FAT is used for embedded systems and USB flat formatting.



## FILE SYSTEM (FAT, NTFS, EXT3)

- NTFS is used in Windows 200, Windows XP, Windows 2003, Vista and Windows 7.
- Replaces FAT.
- Is enhanced by the use of access control lists (access rights), file compression, file system encryption.
- Ntfs-3g is a driver used for Linux

```
$ mount /dev/sda5 /mnt/sda5 -t ntfs-3g
```



# FILE SYSTEM (FAT, NTFS, EXT3)

- EXT (Extended File System) is the native Linux system.
- Uses indexed allocation
- Is involved in the file system check (fsck)
- Checks the consistency of a file system



# VIRTUALIZATION

- Most commands implemented in linux-utils package are basically wrappers for various Linux system calls.

```
#define _GNU_SOURCE
#include <sys/utsname.h>
#include <sched.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/mount.h>
#include <assert.h>

#define errExit(msg)    do { perror(msg); exit(EXIT_FAILURE);} while (0)

static void mnt_rdonly(const char *dir) {
    assert(dir);

    // mount --bind /bin /bin
    if (mount(dir, dir, NULL, MS_BIND|MS_REC, NULL) < 0)
        errExit(dir);

    // mount --bind -o remount,ro /bin
    if (mount(NULL, dir, NULL, MS_BIND|MS_REMOUNT|MS_RDONLY|MS_REC, NULL)
        errExit(dir);
}

static void mnt_tmpfs(const char *dir) {
    assert(dir);

    // mount -t tmpfs -o size=100m tmpfs
    if (mount(NULL, dir, "tmpfs", 0, NULL) < 0)
        errExit(dir);
}

int main(int argc, char **argv) {
    int i;

    if (unshare(CLONE_NEWNS /*| CLONE_NEWNET | CLONE_NEWIPC */) < 0)
        errExit("unshare");

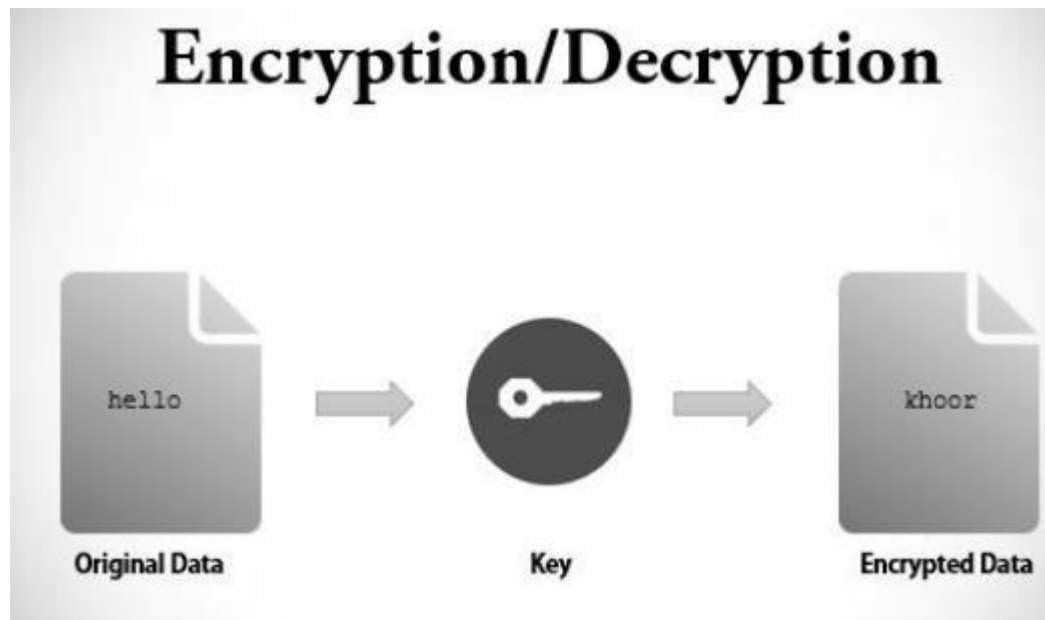
    // mount --make-rslave /
    if (mount(NULL, "/", NULL, MS_SLAVE | MS_REC, NULL) < 0)
        errExit("mount slave");
    mnt_rdonly("/bin");
    mnt_rdonly("/sbin");
    mnt_rdonly("/lib");
    mnt_rdonly("/lib64");
    mnt_rdonly("/usr");
    mnt_rdonly("/etc");
    mnt_tmpfs("/home/netblue");

    chdir("/");
    execlp("/bin/bash", "/bin/bash", NULL);
    return 0;
}
```



# SECURITY

- **Computer security** is the protection of **computer** systems from theft or damage to their hardware, software or electronic data, as well as from disruption or misdirection of the services they provide.





# CAESAR CYPHER ALGORITHM

- For encryption and decryption, 3 is used as a key value.
- While encrypting the given string, 3 is added to the ASCII value of the characters.
- Similarly, for decrypting the string, 3 is subtracted from the ASCII value of the characters to print an original string.

```
#include <stdio.h>
int main() {
    int i, x;
    char str[100];
    printf("\nPlease enter a string:\t");
    gets(str);
    printf("\nPlease choose following options:\n");
    printf("1 = Encrypt the string.\n");
    printf("2 = Decrypt the string.\n");
    scanf("%d", &x);
    //using switch case statements
    switch(x) {
    case 1:
        for(i = 0; (i < 100 && str[i] != '\0'); i++)
            str[i] = str[i] + 3; //the key for encryption is 3 that is added to ASCII value
        printf("\nEncrypted string: %s\n", str);
        break;
    case 2:
        for(i = 0; (i < 100 && str[i] != '\0'); i++)
            str[i] = str[i] - 3; //the key for encryption is 3 that is subtracted to ASCII value
        printf("\nDecrypted string: %s\n", str);
        break;
    default:
        printf("\nError\n");
    }
    return 0;
}
```



## EXERCISES (1/3)

- 1) Based on the code from the chapter Memory management using Makefiles, we want to compile the theme by programming using the Makefile file. Run the *make* command. Run the *make* command again. Has any order been executed?

Change the value of the `MIN_VAL` macro to the `utils.h` file. Run the *make* command again. Why does not the executable file update?

Change the *Makefile* file to get the object files (with the `.o` extension) also to keep in mind the header files (with the `.h` extension) that they depend on. (2 points)



## EXERCISES (2/3)

- RSA is another method for encrypting and decrypting the message. It involves public key and private key, where the public key is known to all and is used to encrypt the message whereas private key is only used to decrypt the encrypted message. (3points)
- It has mainly 3 steps:

### 1: Creating Keys

- Select two large prime numbers  $x$  and  $y$
- Compute  $n = x * y$ , where  $n$  is the modulus of private and the public key
- Calculate totient function,  $\phi(n) = (x - 1)(y - 1)$
- Choose an integer  $e$  such that  $e$  is coprime to  $\phi(n)$  and  $1 < e < \phi(n)$ .  $e$  is the public key exponent used for encryption
- Now choose  $d$ , so that  $d \cdot e \bmod \phi(n) = 1$ , i.e.,  $d$  is the multiplicative inverse of  $e$  in mod  $\phi(n)$



## EXERCISES (3/3)

### 2: Encrypting Message

- Messages are encrypted using the Public key generated and is known to all.
- The public key is the function of both  $e$  and  $n$  i.e.  $\{e, n\}$ .
- If  $M$  is the message(plain text), then ciphertext
- $C = M^e \pmod n$

### 3: Decrypting Message

- The private key is the function of both  $d$  and  $n$  i.e.  $\{d, n\}$ .
- If  $C$  is the encrypted ciphertext, then the plain decrypted text  $M$  is
- $M = C^d \pmod n$

