SCT211-0010/2022 MAURICE.M.MACHARIA

CAT 2

1. What is a process?

**A process is an instance of a running program**

2. State the UNIX command for viewing the current processes

**ps (process status)**

3. Define the following process properties;

a. TTY

**Terminal type associated with the process**

b. PID

**Process ID. It is a unique numerical identifier assigned by the operating system to each running process and is used to manage and control processes, such as terminating a specific process or sending signals to it**

c. PPID

**Parent process ID . It is the PID of the process that created the current process.**

d. UID

**User ID that this process belongs to (the person running it). It is a numerical identifier assigned to each user on the system**

e. ARGS

**ARGS stands for Arguments. It refers to the command-line arguments that were used to launch the process.**

4. How are processes born in UNIX?

**Processes are created when a user or another process requests a new process to be created. This is done by duplicating an existing process using a system call called `fork()`. The new process created is called the child process, and the existing process that initiated the `fork()` system call is called the parent process .After the `fork()` system call, the child process and the parent process are two separate processes that can run independently of each other. The child process is a copy of the parent process, including its memory and program code. However, the child process has its own unique process ID (PID) and runs in its**

**own process context . Once the child process is created, it can execute a different program using the exec() system call or perform other operations, such as reading from a file or communicating with other processes.**

5. Define the following types of processes;

a. Daemon

**Is a program with a unique purpose .They are utility programs that run silently in the background to monitor and take care of certain subsystems to ensure that the operating system runs properly .**

b. Zombie

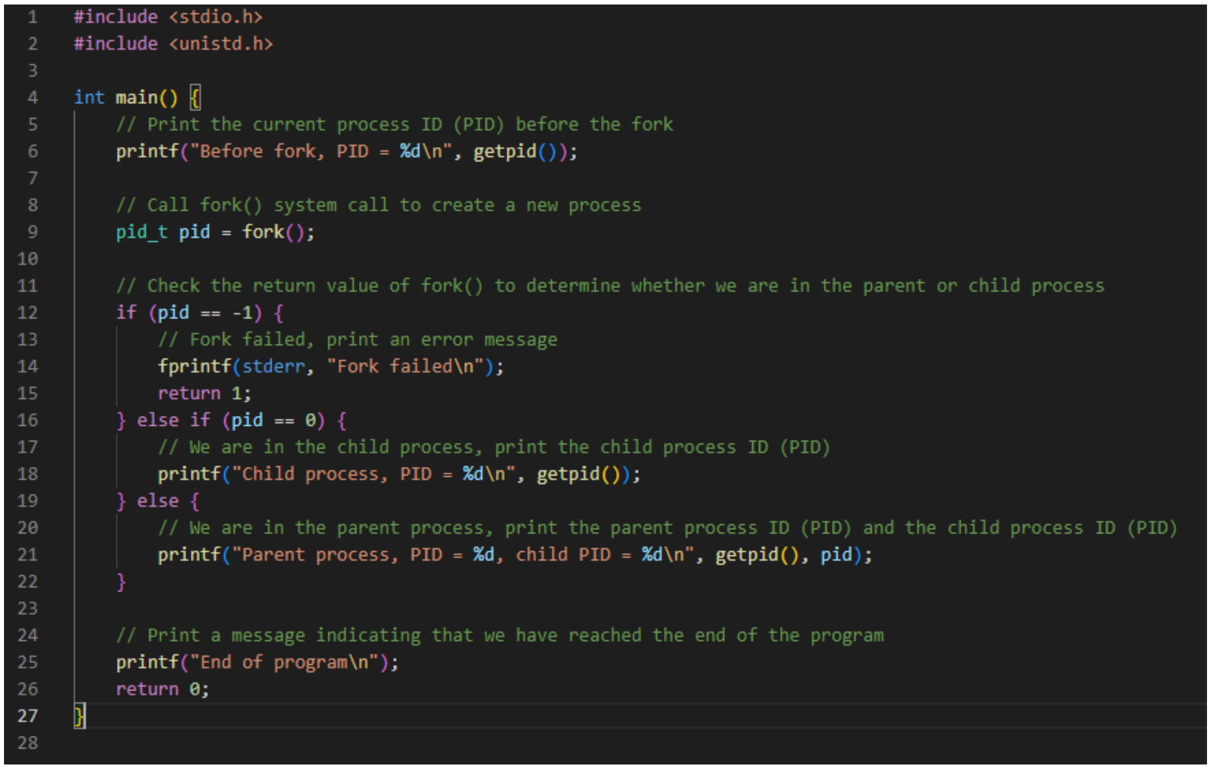
**When a child process dies, the parent process is informed so that it can do some clean up like freeing up memory etc. However, child process goes into zombie state if the parent process is not aware of its death. For the parent, the child still exists but the child process is actually dead .**

c. Orphaned

**A computer process whose parent process has finished or terminated, though it remains running itself.**

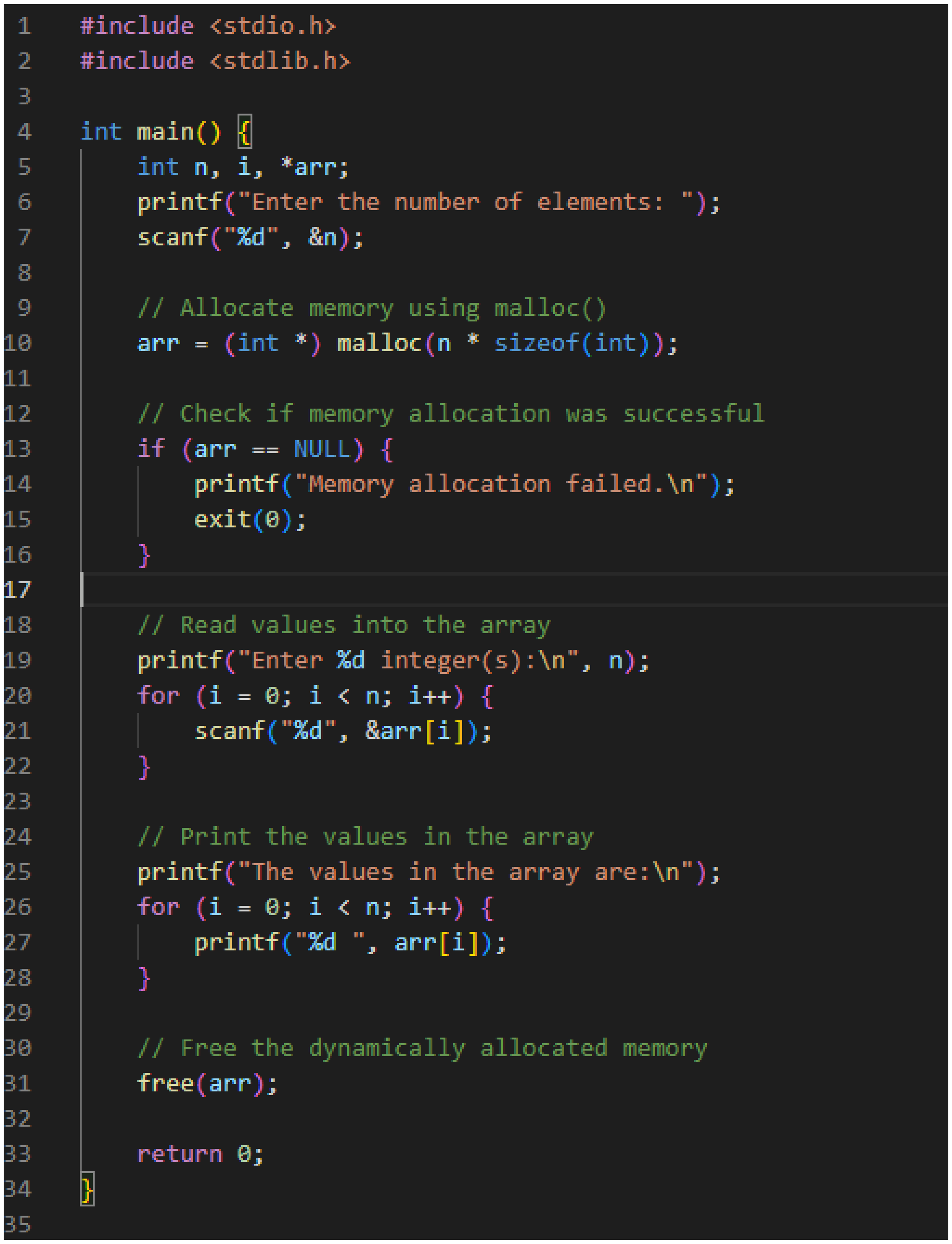
6. Write a program to demonstrate that fork is called once but returns twice. Use comments

and a short description to explain your code.



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

int main() {
    // Print the current process ID (PID) before the fork
    printf("Before fork, PID = %d\n", getpid());

    // Call fork() system call to create a new process
    pid_t pid = fork();

    // Check the return value of fork() to determine whether we are in the parent or child process
    if (pid == -1) {
        // Fork failed, print an error message
        fprintf(stderr, "Fork failed\n");
        return 1;
    } else if (pid == 0) {
        // We are in the child process, print the child process ID (PID)
        printf("Child process, PID = %d\n", getpid());
    } else {
        // We are in the parent process, print the parent process ID (PID) and the child process ID (PID)
        printf("Parent process, PID = %d, child PID = %d\n", getpid(), pid);
    }

    // Print a message indicating that we have reached the end of the program
    printf("End of program\n");
    return 0;
}

```

7. Write a program to demonstrate dynamic memory allocation using malloc? With explanations.



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

int main() {
    int n, i, *arr;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Allocate memory using malloc()
    arr = (int *) malloc(n * sizeof(int));

    // Check if memory allocation was successful
    if (arr == NULL) {
        printf("Memory allocation failed.\n");
        exit(0);
    }

    // Read values into the array
    printf("Enter %d integer(s):\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Print the values in the array
    printf("The values in the array are:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }

    // Free the dynamically allocated memory
    free(arr);

    return 0;
}

```