1-Code examples (Lab5) :

1-

Exercise 1: Using fork() in C

Write a C program that demonstrates process creation using the fork() system call.

Code Example:

```
#include <stdio.h>
#include <unistd.h>
int main() {
pid_t pid = fork();
if (pid == 0) {
printf("This is the child process. PID: %d\n", getpid());
} else if (pid > 0) {
printf("This is the parent process. PID: %d\n", getpid());
} else {
printf("Fork failed!\n");
}
return 0;
}
```

Task: Compile the program

```
gcc process_creation.c -o process_creation
./process_creation
```

Explanation:

This code is used to create new processes and check if its a child ,parent or a failed fork

#include <unistd.h> The headers areincluded because they contain the declarations for the fork(), getpid(), and pid_t type.

pid_t pid = fork(); fork returns 0 if its a child process , >0 if its a parent process and -1 if the fork failed

2-

Exercise 5: Role of the Linker

Write two separate C files, then compile and link them together.
File 1: " file1.c ":

```
#include <stdio.h>
void hello() {
printf("Hello from file1!\n");
}
```

File 2: " file2.c ":

```
void hello();
int main() {
hello();
return 0;
}
```

Explanation:
This example shows how we link 2 files using 'linker'
file2.c calls the hello() function , which is defined in file1.c .
The Linker's job is to connect this call to the function definition when creating the final program.

3-

Exercise 6: Role of the Loader

Write a simple program and inspect the libraries it uses with ldd
Simple Program:

```
#include <stdio.h>
int main() {
printf("This is a simple program.\n");
return 0;
}
```

Compile the program:

```
gcc simple_program.c -o simple_program
```

Use ldd to list the dynamic libraries:

```
ldd simple_program
```

Explanation:
This example is used to show what the 'loader' do
Running ldd on it lists the shared libraries it needs .
The Loader is responsible for loading these libraries into memory when the program starts

2-Loader and Linker:

1-Loader is responsible for starting programs,It takes the executable file from the hard disk, places it into the main memory, initializes the registers, and tells the CPU to begin executing the instructions.
2-Linker runs during the compilation process. Its main job is to combine independent object files (.o) and libraries into a single executable file. It performs Symbol Resolution (connecting function calls to their definitions) and Relocation (assigning memory addresses to code sections).

3-makefile:

Makefile

```
# Build the fork program
all: fork_app

fork_app: main.c
	gcc main.c -o fork_app

# Run the program
run: fork_app
	./fork_app

# Clean up files
clean:
	rm -f fork_app
```