Q1)

```c
int main()
{
    pid_t id;
    for (int i = 0; i < 3; ++i)
    {
        id = fork();
        if (id < 0)
        {
            fprintf(stderr, "Failed to fork.\n");
            exit(-1);
        }
    }
    printf("[PID] %d\n", getpid());

    return 0;
}
```

Q2)

1. thread_info
2. stack
3. usage
4. flag
5. ptrace
6. on_cpu
7. wake_entry
8. recent_used_cpu
9. wake_cpu
10. on_req

Q3)

4 processes are created

Q4)

100, 250, 250, 250

Q5)

```c
int main()
{
    pid_t id;
    id = fork();
    if (id < 0)
    {
        printf("[ERROR] Creating new process.\n");
        return 1;
    }
    // First child
    if (id == 0)
    {
        execl("/bin/ls", "/bin/ls", "-al", NULL);
    }
    else
    {
        id = fork();
        if (id < 0)
        {
            printf("[ERROR] Creating new process.\n");
            return 1;
        }
        // Second child
        if (id == 0)
        {
            execl("/bin/ps", "ps", "aux", NULL);
        }
        // Parent wait
        else
        {
            wait(NULL);
            wait(NULL);
        }
    }
    return 0;
}
```

Q6)

```c
int main()
{
    pid_t id;
    key_t key;
    int q_id;
    char msg[100] = "I hear and I forget. I see and I remember. I
 do and I understand.";

    key = ftok("mykey", 65);
    q_id = msgget(key, 0666 | IPC_CREAT);

    id = fork();
    if (id < 0)
    {
        printf("[ERROR] Creating new process.\n");
        return 1;
    }
    // First child
    if (id == 0)
    {
        printf("[PID] %d [SENDING]: %s\n", getpid(), msg);
        // Send the message
        msgsnd(q_id, &msg, sizeof(msg), 0);
    }
    else
    {
        id = fork();
        if (id < 0)
        {
            printf("[ERROR] Creating new process.\n");
            return 1;
        }
        // Second child
        if (id == 0)
        {
            // Receive the message
```

```
            msgrcv(q_id, &msg, sizeof(msg), 1, 0);
            printf("[PID] %d [RECEIVED] %s\n", getpid(), msg);
        }
        // Parent wait
        else
        {
            wait(NULL);
            wait(NULL);
        }
    }
    return 0;
}
```

Q7)

```
int main()
{
    char *file_to_read = "in.txt";
    char *file_to_write = "out.txt";
    int write_fd, read_fd; // File descriptors

    // Open files
    read_fd = open(file_to_read, O_RDONLY);
    write_fd = open(file_to_write, O_WRONLY | O_CREAT);

    if (read_fd == -1 || write_fd == -1)
    {
        fprintf(stderr, "Failed to open the files.\n");
        exit(-1);
    }
    char c;
    int bytes;
    do
    {
        // Read a byte
        bytes = read(read_fd, &c, sizeof(c));
```

```
        // Duplicate
        char out[2] = {c, c};
        write(write_fd, &out, sizeof(out));

    } while (bytes > 0);

    // Close
    close(read_fd);
    close(write_fd);

    return 0;
}
```