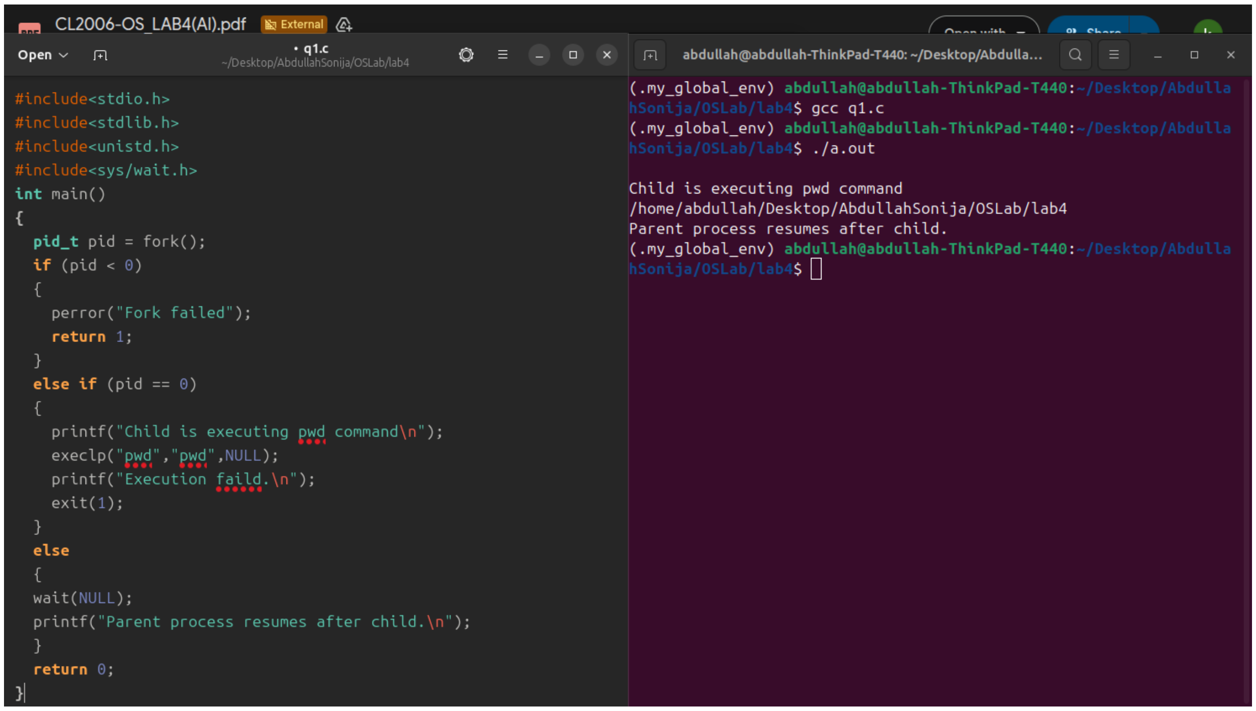
OS Lab 4

Q1:

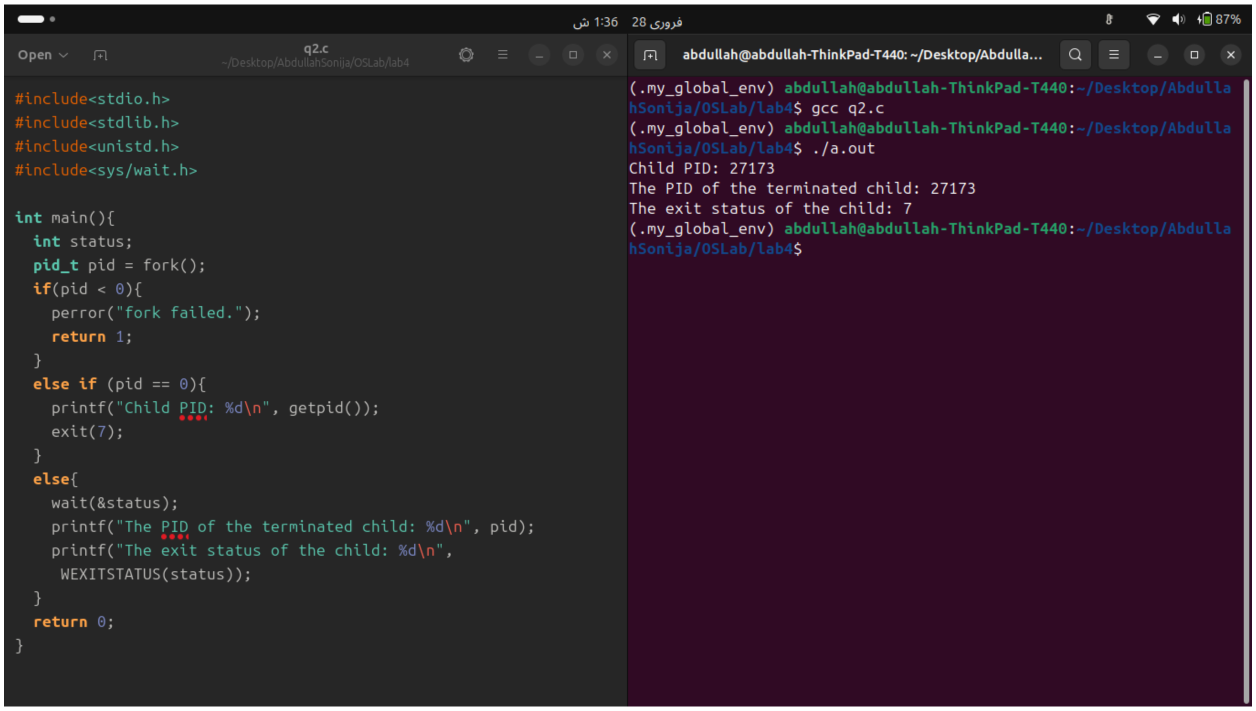

CL2006-OS_LAB4(AI).pdf
External
Open
• q1.c
~/Desktop/AbdullahSonija/OSLab/lab4
abdullah@abdullah-ThinkPad-T440: ~/Desktop/Abdulla...

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>
int main()
{
  pid_t pid = fork();
  if (pid < 0)
  {
    perror("Fork failed");
    return 1;
  }
  else if (pid == 0)
  {
    printf("Child is executing pwd command\n");
    execlp("pwd","pwd",NULL);
    printf("Execution faild.\n");
    exit(1);
  }
  else
  {
  wait(NULL);
  printf("Parent process resumes after child.\n");
  }
  return 0;
}
```

```
(.my_global_env) abdullah@abdullah-ThinkPad-T440:~/Desktop/Abdulla
hSonija/OSLab/lab4$ gcc q1.c
(.my_global_env) abdullah@abdullah-ThinkPad-T440:~/Desktop/Abdulla
hSonija/OSLab/lab4$ ./a.out

Child is executing pwd command
/home/abdullah/Desktop/AbdullahSonija/OSLab/lab4
Parent process resumes after child.
(.my_global_env) abdullah@abdullah-ThinkPad-T440:~/Desktop/Abdulla
hSonija/OSLab/lab4$
```

Q2:


1:36 ش
28 فروری
87%
Open
q2.c
~/Desktop/AbdullahSonija/OSLab/lab4
abdullah@abdullah-ThinkPad-T440: ~/Desktop/Abdulla...

```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/wait.h>

int main(){
  int status;
  pid_t pid = fork();
  if(pid < 0){
    perror("fork failed.");
    return 1;
  }
  else if (pid == 0){
    printf("Child PID: %d\n", getpid());
    exit(7);
  }
  else{
    wait(&status);
    printf("The PID of the terminated child: %d\n", pid);
    printf("The exit status of the child: %d\n",
     WEXITSTATUS(status));
  }
  return 0;
}
```

```
(.my_global_env) abdullah@abdullah-ThinkPad-T440:~/Desktop/Abdulla
hSonija/OSLab/lab4$ gcc q2.c
(.my_global_env) abdullah@abdullah-ThinkPad-T440:~/Desktop/Abdulla
hSonija/OSLab/lab4$ ./a.out
Child PID: 27173
The PID of the terminated child: 27173
The exit status of the child: 7
(.my_global_env) abdullah@abdullah-ThinkPad-T440:~/Desktop/Abdulla
hSonija/OSLab/lab4$
```

Q3:

- **Total processes:** 4 (The original parent + 3 children).
- **printf() executions:** 4 times (Once for every process).
- Every process will print **x=9**.
- **Predictability:** No. The OS scheduler decides which process runs first. You might see the parent print before the child, or vice versa.

Q4:

- **Processes created:** 2 (Parent and 1 child).
- **Which runs date:** The child process.
- **"Exec Failed" condition:** It will only print if execlp fails (e.g., the date command isn't found). If exec succeeds, the child's memory is overwritten.
- **“Finish" prints:** Once. Only parent reaches that line.
- **Does execlp() create a new process?** No.

Q5:

- **What value will the parent print?** 5.
- **What does wait(&status) return?** The PID of the terminated child.
- **What happens if wait() is removed?** The parent may finish before the child, potentially creating a zombie process.
- **What if child uses exit(10) instead?** The parent will print 10.
- **Why do we use WEXITSTATUS(status):** To extract the specific exit code from the full status integer provided by the kernel.

Q6:

- **Which prints first? parent or child?** The parent. (The child is delayed by the sleep command).
- **Why is sleep(3) used?** To ensure the parent finishes its task first, often to demonstrate orphan processes.
- **What does getppid() return?** The Process ID of the Parent.
- **Can parent terminate before child?** Yes.
- **What happens if parent terminates early?** The child becomes an orphan and is adopted by the init or systemd process (PID 1).

Q7:

- **After how many seconds will "Time is up!" print?** 3 seconds.
- **Which signal is generated?** SIGALRM.

- **What happens if signal(SIGALRM, handler) is removed?** The program will terminate abruptly because the default action for SIGALRM is to kill the process.
- **Why is while(1) used?** To keep the process running so it stays alive long enough to receive the signal.
- **What does alarm(3) do?** It schedules a kernel-level timer to send a signal to the process after 3 seconds.