HW1: programming exercise

This assignment practices basics of UNIX programming and system calls. Your program should consist of a *main* and the functions detailed below.

1)
Write the function ***pid_t my_fork (void)***, which does the followings:
- Call fork()
- If fork() succeeds (namely, its return value is non-negative), return the value returned by fork().
- Else (namely, fork() failed, from any reason), print an error message, and exit the process.

2)
Write the function: ***void print_pids (int fd, short unsigned int N, short unsigned int G)***, where

For each $g=0, 1, \ldots, G$, the function should print N^g lines in the following format:
My pid is p. My generation is g.
Where:

- *g* is the *generation* of the process: the generation of the original (first) process is 0. If the generation of a process is *i*, then the generation of its child process is *i+1*.
- *p* is the pid of the process which prints the line.

For instance, the line may be:
My pid is 1038. My generation is 0.

Note that the pids are arbitrary, and are depended upon the processes' scheduling at your machine. In particular, the operating system may allocate a new process the same pid of an old terminated process.
The function should print its output to the file, whose file descriptor is the input *fd*.

Limitations and hints:

- The function should use **a single loop**.
- Each process iterates exactly ***N* times** over the loop.
- Each process prints a single line.
- Use *my_fork()*, not *fork()*.
- No prints neither calls to *my_fork()* are allowed outside the loop.
- Each process should print its line only **after all its children printed their lines.**

3)
Write the function: **void count_lines (short unsigned int G)**.
This function counts the number of lines in the file *out.txt* (to which we will write output, as described below); and then prints **to the screen** output which follows the following format:
Number of lines by processes of generation 2 is 9
Number of lines by processes of generation 1 is 3
Number of lines by processes of generation 0 is 1

Limitations and hints:

- The function should use **a single loop**.
- The function should generate either *G-1* or *G* new processes.

- Each process should print a single line: the line referring to generation *g* should be printed by a process of generation *g*.
- You may use the commands *system()*, *grep* and *wc*.
- The output should be written in **decreasing** generation order (as in the sample above).

4. Write the main function: **int main (int argc, char* argv[])**, which:
* opens a file named *out.txt* (for simplicity, we'll use always the same output file name). If the file already exists, its content will be re-written by the program. Else, it will be created.
* Calls *print_pids* with the descriptor of *out.txt* and with argv[1], argv[2] as N, G respectively.
* Calls *count_lines()* to check the results of *print_pids()*.

5. Write a makefile which compiles the code to an executable named *OS*.

<u>Example</u>
Running the following lines
> *make*
> *./OS 3 2*
Should:
- Generate the file named *out.txt* which you'll have in Moodle.
- print to the screen the following lines:
  Number of lines by processes of generation 2 is 9
  Number of lines by processes of generation 1 is 3
  Number of lines by processes of generation 0 is 1

You may assume that the input to *main* is correct.
The HW should accurately follow the submission instructions in the course's site.

<u>HW2 – theoretical questions</u>

<u>1.</u> Write two advantages for using a multiple-threaded process for performing a task over using a few processes.

<u>2.</u> What are the pros and cons of user-level threading vs. kernel-level threading?

<u>3.</u>
Consider a system running 3 I/O-bound processes, P1, P2 and P3. Each of them needs the CPU for 1 ms to make a request to an I/O device, which is served within 6 ms. This is done iteratively – namely, after the I/O request is fully served, each of them need again the CPU for 1 ms and so on.
Each of these processes calls a unique I/O device – namely, the I/O requests of T1, T2 and T3 may be served simultaneously.
The system also runs a single CPU-bound process, T4, which requires 100 ms of CPU, and doesn't use I/O devices.
A context switch (*ctxw)* takes 1 ms, in which the CPU performs only the ctxw, and can process nothing else.
We define as "CPU utilization" the percentage of cycles in which the CPU is processing a process - ie, the ratio between the number which the CPU is processing (i.e. not in idle / context switch mode), and the total number of cycles.
We assume that all processes arrive together, at time 0.

A) The scheduling policy is Preemptive Shortest Job First, and the quantum is 1 ms. Namely, after 1 ms, the scheduler schedules either another process, or the currently-running process, to run for the next 1 ms. What is the CPU utilization? What is the turnaround of T4?

B) In order to increase the CPU utilization, the quantum was increased to *n* ms, where *n* is a natural number. A currently-running process runs for *n* ms, unless it is blocked due to an I/O call.
What is the minimal *n* required for achieving a utilization of 70%? What is the turnaround of T4 in that case?

C) Repeat B when the desired utilization is at least 90%.

4.
In this question we consider the code in file ***HW2_19.c***.

Assume that no signal is lost – that is, every single is caught and handled by **every** process which may catch it.

It's given that during the time a process waits for 1[sec], the system finishes printing to the screen the outputs of all the other processes.

Answer the following American questions, and **explain your answer**. There can be more than one correct option for every question.

1. Upon running the program (without typing anything), which of the following outputs is possible?

A) Nothing is printed.

B)^CProcess number 3 caught one
Process number 2 caught one
Process number 1 caught one
Process number 0 caught one

C)^CProcess number 0 caught one
Process number 1 caught one
Process number 2 caught one
Process number 3 caught one

^CProcess number 0 caught one
Process number 1 caught one
Process number 2 caught one

^CProcess number 2 caught one
Process number 1 caught one
Process number 0 caught one

D) All the previous answers are wrong.

2. Yossi runs the program, and then presses ctrl+C. Which of the following outputs is possible?

A)^CProcess number 3 caught one Process number 2 caught one Process number 1 caught one Process number 0 caught one Process number 2 caught one Process number 1 caught one Process number 0 caught one Process number 1 caught one Process number 0 caught one Process number 0 caught one	B)^CProcess number 0 caught one Process number 1 caught one Process number 2 caught one Process number 3 caught one Process number 0 caught one Process number 1 caught one Process number 2 caught one Process number 0 caught one Process number 1 caught one Process number 0 caught one
C) ^CProcess number 3 caught one Process number 1 caught one Process number 2 caught one Process number 0 caught one Process number 2 caught one Process number 0 caught one Process number 1 caught one Process number 1 caught one Process number 0 caught one Process number 0 caught one	D)^CProcess number 0 caught one Process number 1 caught one Process number 3 caught one Process number 0 caught one Process number 1 caught one Process number 2 caught one Process number 2 caught one Process number 1 caught one Process number 0 caught one Process number 0 caught one
E) ^CProcess number 2 caught one Process number 1 caught one Process number 3 caught one Process number 0 caught one Process number 1 caught one Process number 2 caught one Process number 2 caught one Process number 1 caught one Process number 0 caught one Process number 2 caught one	F) ^CProcess number 0 caught one Process number 1 caught one Process number 2 caught one Process number 0 caught one Process number 1 caught one Process number 2 caught one Process number 3 caught one Process number 1 caught one Process number 0 caught one Process number 2 caught one

5.

At time 0 arrive processes 1, 2, 3, 4, requiring processing times 5,4,3,2 respectively. Assume that process 1 arrived first, process 2 arrived 2nd, and so on.

Calculate the mean turnaround when using:

A) FIFO
B) RR with quanta=1
C) SJF

6.

Part I

At time 0 arrive processes 1, 2, 3, 4, requiring processing times 2,3,4,5 respectively. Assume that process 1 arrived first, process 2 arrived 2nd, and so on.

Calculate the mean turnaround when using:

A) FIFO

B) RR with quanta=1
C) SJF

Part II
At time 0 arrive processes 1, 2, 3, 4,5 ,6 which require the following processing time: 3,2,5,10,2,1.
(the leftmost number is the process which arrives first).
For each of the following algorithms, calculate the average turnaround.

A. FIFO
B. RR with quanta = 1
C. RR with quanta = 2
D. SJF

7.
Consider a system with a preemptive RR scheduling with quanta = 1. The RR scheduler uses a cyclic pointer, which points to the next process to be executed. The pointer is initiated to 1, and is incremented by 1. If the pointed process doesn't need the CPU, the pointer is incremented again, until it reaches a process, which needs the CPU.
All processes arrive at time 0. Neglect the context switch time.
The processes use CPU and I/O alternately as follows:
Process 1: 1,3,1,3,3,1
Process 2: 3,2,3,1,3,2
Process 3: 1,4,1,3,2
For instance, 1 requires 1 CPU cycle, then 3 I/O cycles, then again 1 CPU cycle and so on.
The system has 3 I/O devices, which may work on parallel (each serves a single request).

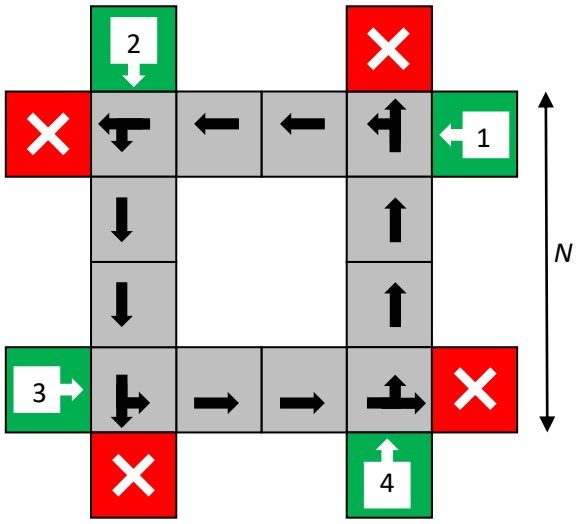
A. Plot a Gantt table.
   I. What is the utilization of the CPU and of the I/O devices until all the processes finish? For the utilization of the I/Os you have to divide the total number of I/O cycles by (3 times the number of cycles until all processes are done)
   II. What is the average turnaround time?
B. The system has a single I/O device, which uses priority-based preemptive scheduling, where the priority is defined by the process ID (process 1 before process 2, process 2 before process 3). Repeat A.

8.
Consider an OS with n processes, where process p_i is generated in time $t = i$. For instance, process 0 is generated in time $t = 0$, process 1 in time $t = 1$ … and process p_{n-1} is generated in time $t = n - 1$.
Each process requires exactly $3 \cdot n$ CPU cycles, and outputs for the first time after 3 cycles. Neglect the time required for output.

A. Suggest a scheduling policy, which minimizes the average response time. Justify your answer.
B. Is there any scheduling policy which minimizes both the average response time and the average turnaround time? Justify your answer.

HW3

In this assignment you will practice multi-thread programming combined with mutual exclusion.
Assume that you've been asked by the MOT (Ministry of Transportation) to build a simulator of a traffic circle that will help them to predict the congestion in a junction, according to some parameters. The parameters are detailed (in capitalized letters) below.
The simulation structure is as follows:



The traffic circle is built of square units where each one of them may contain one car at most at any given time. The size of the square's side is N.
In the circle's corners, there are four "car generators" (green units). Each car generator generates a new car once in a randomized time, which is picked uniformly at random between MIN_INTER_ARRIVAL_IN_NS and MAX_INTER_ARRIVAL_IN_NS [ns] (nano-seconds).
A car that has been generated enters the circle only once the square to which it wants to enter AND the square before it are free.
A car tries to progress to the next square every INTER_MOVES_IN_NS [ns]. Then, if the next square is free, the car moves there; else, it stays in its current slot, until the next square is free.
Also in the circle's corners are 4 sinks. Each car moves around the circles. When a car progresses from a square before a sink, it disappears from the simulation with probability FIN_PROB; and continues moving around the circle with probability 1 - FIN_PROB. A car may vanish in a sink only after it has finished moving along at least one side of the square.
The simulation takes SIM_TIME seconds.
10 times during the simulation, the program should take a snapshot of the circle - namely, check which squares have cars on them; and then print the results like in the following example (for N=5):

```
* *
 @@@*
*@@@*
 @@@
* *     *
```

Where * denotes a car. The (N-1)x(N-1) squares in the middle of the traffic circles are denoted by @. A free square is denoted by a blank.
For relaxing the demands, the snapshot does not have to be consistent. Namely, cars may continue to progress (or try to progress) when another thread takes the snapshot.
Detailed requirements are in the next page.

Requirements

- Your system must use a mutex per each square.
- Cars should be able to progress simultaneously in different squares in the circle, as long as they don't use the same square.
- When a call to lock the mutex fails, you should check the returned value, for identifying whether it happened because the mutex is already locked, or because an error occurred. In the latter case, you should finish the simulation with an appropriate error message.
- Similarly, you should check the returned value when creating a new thread (pthread_create()).
- You should initialize all the mutexes at the beginning of the program, and destroy all of them when the program finishes from any reason.
- Minimize the sections of code which require mutual exclusion.
- The program should use a single process, with multiple threads.
- The program should avoid deadlocks.
- The program should print nothing beside of the 10 prints of the circle along the simulation. However, you may print blank lines before and/or after each print of the circle, so as to make the prints clearer.
- Note, that using extreme values for the parameters may yield strange results. Eg, if you generate new cars too frequently, the operating system may fail to allocate resources for opening new threads for them. Below are recommended values for the parameters:

```
#define N 5
#define FIN_PROB 0.1
#define MIN_INTER_ARRIVAL_IN_NS 8000000
#define MAX_INTER_ARRIVAL_IN_NS 9000000
#define INTER_MOVES_IN_NS       100000
#define SIM_TIME 2
```

- When you wanna a little break from the work, you may read about **DATACH ADASH BA'ACHAB**.