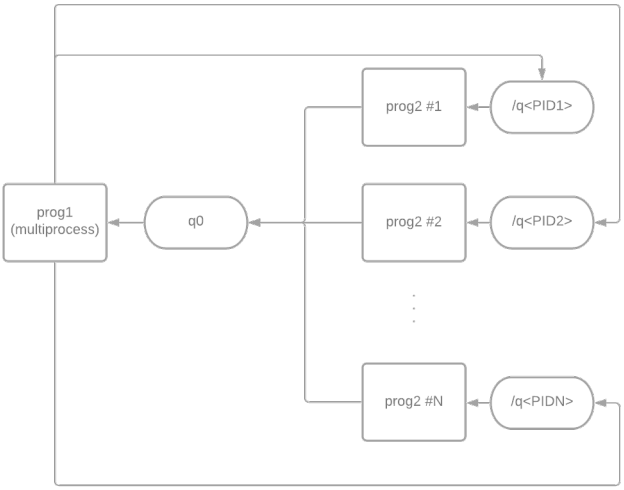
# Task 2: POSIX Queues

## 1 Task




Write two programs: `prog1` and `prog2`. All communication between these programs must use POSIX message queues.
`prog1`:

- Takes two paramaters `q0_name` - name of the `q0` queue, `t` - integer in range [100, 2000] - sleep interval
- If `q0` doesn't exist it creates it, otherwise exits with error.
- Parent process receives messages via `q0` and handles two types of messages: `register <PID>` (without `<>`) which creates a child process opening message queue named `/q<PID>` (without `<>`) for communication with process `<PID>` and `status <PID> <value>` (without `<>`). Any other messages are ignored. `status` messages should have higher priority than `register` messages
- All child processes (in an infinite loop) sleep `t` milliseconds and push `check status` message to its respective queue.
- On SIGINT, `prog1` closes and unlinks all opened or created queues.

`prog2`:

- Takes two parameteres: `q0_name` - name of the queue used for communication with `prog1` (`q0`), `t` - integer in range [100, 2000] - sleep interval
- If `q0` doesn't exist the program prints a message that the given queue doesn't exist and exits
- Otherwise the program pushes `register <PID>` (without `<>`) message to `q0` and opens a queue named `/q<PID>` (without `<>`) for communication with `prog1`.
- In in infinite loop, the process randomizes (with 50% probability) its value - either `0` or `1` and sleeps for `t` milliseconds
- When `check status` message appears in its queue, the process pushes a `status <PID> <value>` (without `<>`) message to `q0` and falls back to randomizing value
- When SIGINT is sent, `prog2` closes all open queues.
- Multiple `prog2` can be opened at the same time.

# 2 Stages

## 2.1 Lab part

- Stage 1 - **2p** - `prog1` only takes `q0_name` argument, creates a queue, pushes any message to the queue, reads the message from the queue, prints it, closes and ulinks the queue.
- Stage 2 - **2p** - `prog1` receives `register` messages from `q0` but only prints its content on receive, `prog2` pushes register message to `q0`, closes the queue and exits. At this stage `prog1` no longer unlinks the queue and doesn't read from it.
- Stage 3 - **3p** - `prog1` creates corresponding queues on `register` message, each child process opens its queue, pushes single `check status` message and exits. `prog2` waits for messages on its queue, prints its content and exits. All queues are closed on exit and processes are properly waited.

## 2.2 Home part

- Stage 4 - **3p** - Full functionality of `prog1`, `prog2` waits for a `check status` message for up to `t` milliseconds (instead of sleeping), randomizes its value, pushes value correctly on `check status` message,
- Stage 5 - **3p** - Fully functional `prog2` and priority handling
- Stage 6 - **1p** - SIGINT is properly handled

# 3 UPLOAD

Please upload your solution to: `/home2/samba/sobotkap/unix/` You have to upload each stage immediately after finishing it. You upload to a network share via `ssh.mini.pw.edu.pl` server:

```
scp user.etapX.tar.bz2 user@ssh.mini.pw.edu.pl:/home2/samba/sobotkap/unix/
```

Please name your stages files according to the schema: `LOGIN.etapN.tar.bz2(.gz)`

All programs must build using single `make` (with no arguments) command

# 4 THE STATEMENT

By decree 27/2020 of University Rector you must add the following statement to the uploads:

---

I declare that this piece of work which is the basis for recognition of achieving learning outcomes in the OPS course was completed on my own. [First and last name] [Student record book number (Student ID number)]

---

Please add it as comment at the beginning of each source file you upload. Replace square brackets with your data.