General Notes (if needed)

- FIFO is the main IPC mechanism used throughout the project.
- FIFO is also called a named pipe, it's different from regular pipes in three things:
    1. It's a special type of file that resides on the filesystem. So, anyone with the path (and with enough permissions) can open the FIFO either for reading, writing, or both.
    2. It synchronizes the two processes opening them. That is:
        - If a process opens it for reading, an no process has opened it for writing, it will block until a process opens it for writing.
        - If a process doesn't want to block, it can open it for both reading and writing, like what happens in:
            - shmFifo in scheduler.c
            - processFifo in scheduler.c
- Writing to FIFOs doesn't block. Reading blocks.

Design Choices of the IPC communication.

- Between the process generator and the scheduler, there are three FIFOs (aka, named pipes), these are:
    1. processArrivalFifo
       Through this one is sent the processes data from the process generator to the scheduler at their arrival time.
    2. eoArrivalFifo (end of arrival fifo)
       The process generator sends a 0 here after it sends a process to the scheduler, if there aren't any more processes arriving at this time.
       It sends a 1 to indicate that there are still more to come at the same arrival time, so that the scheduler wont't spawn any process till it received all the processes arriving at this time.
    3. ackFifo (acknowledgement fifo)
       After the scheduler receives every process, it writes something to this fifo, acknowledging the process receiving, whereas the process generator will be blocking to read the acknowledgement, before proceeding to send any other process.

- Between the processes and the scheduler there are two FIFOs:
    1. processFifo
       The process sends its pid to this fifo when it terminates, wherefore the process generator unblocks reading from this fifo.
    2. shmFifo
       The scheduler sends the process, upon its creation, the key to the shared memory here.

Broad Design Choices

Algorithms' structure

There are three algorithms implemented for the scheduler: - HPF (non-preemptive) - SRTN - RR

Every one of them has almost the same structure.

An argument is pased to the scheduler, based upon which it calles one of the three functions named after the algorithms.

The main part of all the functions: the select() call.
Read more about select() in linux's manual:
https://man7.org/linux/man-pages/man2/select.2.html

But briefly, the select() calls waits(blocks) on multiple file descriptors to read from. When any of the file descriptors it's waiting on unblocks (that is, recieves something), the select() call unblocks, and we can know which file descriptor can be read from.

So , the main loop in every algorithm does the following:

- Checks for termination's condition.

- Then calls select(). It waits to read from two different FIFOs:
    - processArrivalFifo (process_generator writes here when a new process arrives).
    - processFifo (running processes write here when they terminate)
    - ***ONLY IN RR:*** the select() call has a timeout, this timeout is set to the quantum. So, when the select() doesn't receive on either of the two FIFOs, and the qauntum passes, it times out and returns.

      We always check afterwards the amount left in the timeout, because the select() calls modifies the structure we pass to it of the type ***timeval***, and sets its value to the remaining time.

      If the amount left is 0, then we know the quantum has passed, and we stop the running process to run another if there are any available.

Everything about processes

1. When processes are started, they are passed the shm key to their allocated memory generated by the scheduler, through the shmFifo.

2. Processes are passed, as the first argument their runtime, and as the second the size of memory allocated (the size of the shared memory region).
They don't communicate with the parent any further, or the clock, regarding the remaining time. They store the runtime in a variable they keep decrementing every second.

3. Processes are stopped (preempted) by sending them a SIGTSTP signal. In the process's file, *process.c*, we set up a signal handler for this signal. The process checks whether the parent, *scheduler.c*, is the one that sent it this signal, and only stops when it is.

4. Processes are continued when the parent sends them a SIGCONT signal, and only the parent, again, can continue them.

5. When processes terminate, they write their pid to the aforementioned processFifo, on which the scheduler is blocking for reading.

Everything about memory

1. The buddy allocator system represents the memory as a binary tree, see the video link in the comment description of the function *buddyAllcater()* in *scheduler.c* file.

2. The allocated memory are, as we said, shared memory regions, with keys randomly generated by the scheduler and passed to the processes upon their creation.

3. Every process gets its memory allocated when it ***enters*** the system, that is, when it ***arrives*** to the scheduler, not when it's started.

4. When a process arrives:

- if there's enough memory, its memory is allocated and the process is placed in the ready queue.
- if there's not, it's placed in the waiting queue.

5. Whenever a process terminates, the scheduler calls the *allocateAllYouCan()* function, which, evidently enough from the name, goes through the waiting queue, and allocates all it can, and moves the process for which memory is allocated to the ready queue.
